```
'Types Transformation in Spark'
--------------------------

1. Map          --> Realtime
2. FlatMap      --> Realtime
3. Filter       --> Realtime
4. join         --> Realtime
5. groupbykey   --> Realtime
6. reduceByKey  --> Realtime
7. aggregateByKey
8. mapPartition
9. mapPartitionWithIndex
10. coalsec     --> Realtime
11. repartition  --> Realtime
12. cogroup
13. union
14. union all
15. distinct
16. sortBy
17. intersect
18. cartesian


Key-value
1. aggregateByKey
2. reduceByKey
3. groupByKey
4. sortByKey
5. join
6. cogroup


----------------28 th May --------------------------------------

val edata = sc.textFile("file:///home/cloudera/emp.txt")
val ddata = sc.textFile("file:///home/cloudera/dept.txt")

val edata_pair = edata.map{ x =>
val w = x.split(",")
val eno = w(0).toInt
val ename = w(1)
val sal = w(2).toInt
val gendar = w(3)
val dno = w(4).toInt
(dno,sal)
}

val ddata_pair =ddata.map { x=>
val w = x.split(",")
val dno = w(0).toInt
val dname = w(1)
val dloc = w(2)
(dno,dloc)
}

val edata_pair_join_ddata_pair = edata_pair.join(ddata_pair)

val result = edata_pair_join_ddata_pair.map{ x =>
val dno = x._1
val sal = x._2._1.toInt()
val loc = x._2._2
val avgsal = sal.sum/sal.size
(dno,loc,avgsal)
}
```

```
val result = edata_pair_join_ddata_pair.map { x =>
val dno = x._1
val sal = x._2._1
val loc = x.2._2
(dno,sal,loc)
}



----DISTINCT TRANSFORMATION

scala> val data1= sc.parallelize(1 to 10)
data1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[19] at parallelize at <console>:27

scala>
data2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[20] at parallelize at <console>:27

scala> val result = data1.union(data2)
result: org.apache.spark.rdd.RDD[Int] = UnionRDD[21] at union at <console>:31

scala> result.collect()
res11: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20)


'* In Spark Union transformation combine 2 data sets and allow duplicate values*'

scala> val result = data1.union(data2).distinct()
result: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[25] at distinct at <console>:31

scala> result.collect()
res12: Array[Int] = Array(4, 16, 8, 12, 20, 13, 1, 17, 9, 5, 14, 6, 18, 10, 2, 19, 15, 11, 3, 7)

---Its Wide transformation

res15: String =
(4) MapPartitionsRDD[33] at distinct at <console>:31 []
 |  ShuffledRDD[32] at distinct at <console>:31 []
 +-(4) MapPartitionsRDD[31] at distinct at <console>:31 []
    |  UnionRDD[30] at union at <console>:31 []
    |  ParallelCollectionRDD[19] at parallelize at <console>:27 []
    |  ParallelCollectionRDD[20] at parallelize at <console>:27 []

scala> val resultsort = result.sortBy( x => x,false)
resultsort: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[38] at sortBy at <console>:33

scala> resultsort.collect()
res16: Array[Int] = Array(20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)

scala> val resultsort = result.sortBy( x => x,true)
resultsort: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[43] at sortBy at <console>:33

scala> resultsort.collect()
res17: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20)

'Intersect '

scala> val data1= sc.parallelize(1 to 10)
data1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[44] at parallelize at <console>:27

scala> val data2 = sc.parallelize (5 to 20)
data2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[45] at parallelize at <console>:27

scala>
scala> val result = data1.intersection(data2)
result: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[51] at intersection at <console>:31

scala> result.collect().foreach(println)
6
8
10
7
9
5

scala>

'CROSS PRODUCT '

scala> val x = sc.parallelize(List(1,2,3,4,5),2)
x: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[60] at parallelize at <console>:27

scala> val y = sc.parallelize(List('a','b','c','d','e','f'),2)
y: org.apache.spark.rdd.RDD[Char] = ParallelCollectionRDD[61] at parallelize at <console>:27

scala> x.cartesian(y).collect().foreach(println)
(1,a)
(1,b)
(1,c)
(2,a)
(2,b)
(2,c)
(1,d)
(1,e)
(1,f)
(2,d)
(2,e)
(2,f)
(3,a)
(3,b)
(3,c)
(4,a)
(4,b)
(4,c)
(5,a)
(5,b)
(5,c)
(3,d)
(3,e)
(3,f)
(4,d)
(4,e)
(4,f)
(5,d)
(5,e)
(5,f)

scala>

'Cogroup ' -- If Join and group operates on Same column then use Cogroup

        select dno,avg(sal)
        from emp e join dept d
        on e.deptno = d.deptno
        group by deptno;

Data Should be in Key-value pairs

val a = sc.parallelize(List((1,'a'),(2,'a'),(3,'a'),(4,'a'),(2,'c'),(3,'c'),(3,'a')),2)
val b = sc.parallelize(List((1,'b'),(2,'b'),(3,'b'),(4,'b')),2)

'Actions'
---------
1. collect
2. saveAsSequenceFile -- Recently introduced
3. saveAsObjectFile
4. count








'count'
scala> val x = List(1,2,3,4,5)
x: List[Int] = List(1, 2, 3, 4, 5)

scala> val r = sc.parallelize(x);
r: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[75] at parallelize at <console>:29

scala> r.count
res23: Long = 5

scala> r.first
res24: Int = 1


scala> r.take(2);
res28: Array[Int] = Array(1, 2)

'Difference between reduce and reduceByKey'

reduceByKey is a transformation
recude is an action

If you are using reduceByKey your data should be in Key-value
If you are using reduce action key,value is not mandatory


Reduce :

select sum(sal),avg(sal),count(sal) from emp ;



val rrd1 = sc.parallelize(List(10,20,30,40,50,60,70,80,90),2)


rdd1.sum // All Partitions data will be collected into local, sum executed at local no parallel
procession

rdd1.reduce(_+_) // It executed at cluster all separtlely for each partition


---------------------29 May


Spark SQL API Enables spark core functionality as well as sequel api functionality.


Rules for Spark SQL API .

1. Spark SQL Api Requiresd Sql context object - SQLContext
```

```
2. Data should be in structured format
3. Proper Schema for your data


SQLContext + Structured Data + Proper Schema = Data Frame

Data Frame is a SQL Table

'How to create a Data frame'
----------------------------

select dno,loc,avg(sal),max(sal),min(sal) from emp e join dept d
where e.dno=d.dno
group by dno,dloc


import org.apache.spark.sql.SQLContext

val sqlContext = new sqlContext(sc)

val emp = sc.textFile("file:///home/cloudera/emp.txt")
val dept = sc.textFile("file:///home/cloudera/dept.txt")

case class Employee (eno:Int,ename:String,sal:Int,gendar:String,dno:Int)
case class Department (dno:Int,dname:String,dloc:String)


val edata = emp.map{ x =>
val w = x.split(",")
val eno = w(0).toInt
val ename=w(1)
val sal = w(2).toInt
val gendar = w(3)
val dno =w(4).toInt
Employee (eno,ename,sal,gendar,dno)
}


val ddata = dept.map{ x =>
val w = x.split(",")
val dno = w(0).toInt
val dname=w(1)
val dloc = w(2)
Department (dno,dname,dloc)
}

--//converting RDD to DataFrame

import sqlContext.implicits._

val edf = edata.toDF
val ddf = ddata.toDF

edf.show()
ddf.show()

edf.registerTempTable("empview")
ddf.registerTempTable("deptview")


val eresult= sqlContext.sql("select d.dno,d.dloc,avg(sal) as AVG_SAL ,max(sal) MAX_SAL
,min(sal) MIN_SAL,count(*) COUNT_SAL from empview e join deptview d on e.dno=d.dno group by
d.dno,d.dloc")

val eresult= sqlContext.sql("select d.dno,d.dloc from empview e join deptview d on e.dno=d.dno")
```

```
In Spark 1.x --> SQL Contex allows only DRL  (Data Retrieval only select statement)
In Spark 2.x --> we can write DDL Statements , DML Statements


Spark 1.x (1.6.0/1.6.2/1.6.3)                   Spark2.x (2.1.0 / 2.3x)

1. Spark core Contxt (RDD)                      1. In Spark 2.x only 1 context Spark Session
2. SQLContext (SQL Api, supports DRL)                   Spark Session -- Spark Context,SQL
context , SSC

3. Hive Context ( Spark + Hive )
    ( Integration,
        Support DDL,DML,DRL

    )
4. Spark StreamingContext
    (
    SSC , Supports DStream

    )



'Spark + Hive Integration using Hive Context '
------------------------

1. Copy


------------------------31st May ---------------------------

'How to enable Hive context in Spark 2.x' --****** V Imp

Method :
enablehivesupport

Warehouse Directory :


cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/


val spark= SparkSession
            .builder()
            .appName("Spark Hive Example")
            .config("spark.sql.warehouse.dir","warehouseLocation")
            .enablehivesupport
            .getOrCreate()

'Why DataFrames are very Powerful'


DataFrame = RDD + Catalyst Optimizer + DAG + in Memory

DataSet = RDDs + Catalyst Optimizer + CPU Caches

CBO = Cost Based Opitimizer

Catalyst Optimzer internally uses CBO


Catalyst Optimzer -- Read more about it


Catalog represents a serializer .
```

```
serialization and De-serialization -- How to read and write a data.


'Different serialization avaialble in Spark '
    Spark supports Different types of serialization
        1. Java serialization --default
        2. Kyro serialization -- Advance to Java
        3. avro serialization -- Read and write data of avro types
        4. Sequence serialization -- Read and write data of squence serialization
        5. Parquet serialization --Read and write parquet serialization

How to enable

val conf = new SparkConf()
            .set("spark-serializer","org.apache.spark.serializer.KryoSerizer")

It re-commendable to use kryo serialization for performance



'#Fetching the Data from RDS using Spark DataFrames'

RDS


    cloud

        mysql ---------------*****----------------mysql
                                                    |
                                                    |
                                                    |
                                                  Spark


            mysql --> mysql via AWS Virtual private cloud

            Spark cannot read the data from cloude because of 2 reasons
            1. Firewalls
            2. DNS

            Spark read data from local mysql

Services --> Database --> RDS (Relational Data Services)
    --> Launch DB Instance
    --> MySQL

    * Check the check box --> only enable options eligible for free usage type

    -> Next

    --> Check If Free Usage

Settings

    1. DB Instance Identifier -- Unique Name for the database (mysqldb)
    2. username and password

Configure Advanced settings

1. Default VPC --> Keep it default
2. Public
    yes --> Access outside of the cloud
    no  --> Access outside of the cloud
3. Database option

    Database Name : testdb
    Port No       : 3304

Launch

Install Client Machines

1. SQL WorkBench

    Download generic package for all systems


copy End Point information

Enabling -- In bound and outbound connection

--> Click Security Group
    --> Inbound
    --> Add Rule

    Type : MySQL/Aurora
    Source : Anywhere

    Save

    --> Outbound type

    Edit --> Add Rule --> MySQL/Aurora
    Desitination --> Anywhere

Step no : 3 Open SQL WorkBench

Give DB Name
MySQL JDBC Jar Download -- Platform Independent
Manage Driver --> add Driver
Copy URL and replace with AWS information

'1-June'

Reading the data from RDS using Spark SQL Engine



IntelliJ
    File Menu
        Project structured
            Modules
                Dependencies
                    Right Most corner
                        Jars
                            mysql



'4-June'

'How to Handle csv file format, JSON file format and xml file formats using spark sql'

Spark packages (2.x)

csv
xml
json
html
rcv
etc
```

```
import spark.implicits._ --> To convert RDD To DataFrame

1.x

//registerTempTable --> Available upto that session

2.x

createOrReplaceTempView == registerTempTable
createOrReplaceGlobalTempView --> Available even after closing the spark sql

'Handling csv file '

import org.apache.spark.sql.SparkSession


object SparkCsv {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder.master("local[*]").appName("SparkCsv").getOrCreate()
    //local represents local machine & star represents no of reources i.e utilising all resources
    //.getOrCreate => creating a application or using already existed application
    //.appName => creating an application with name "csvExample_1"

    //creating 2 contex sparkContext and sqlContext
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext

    import spark.implicits._
    //Creating DataFrame using SQLContext Reading data)
    val df = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option(
    "inferschema","true").load("C:\\Users\\NaraVish\\Desktop\\#Personal\\#Interview
    Documents\\filformatsinspark\\Police_Department_Incidents.csv")
    //.option("header","true") => Use first line of all files as Header
    //.option("inferschema","true") => Automatically infer data types

    df.show()
    //df.select("Category").distinct.collect().foreach(println)
    df.createOrReplaceTempView("sfpd") //Creating Temp table
    // sqlContext.sql("select Category from sfpd").collect().foreach(println)
    //top 10 results
    //sqlContext.sql("SELECT Resolution , count(Resolution) as rescount FROM sfpd group by
    Resolution order by rescount desc limit 10").collect().foreach(println)
    val t = sqlContext.sql("select Category,count(Category) as catcount from sfpd group by
    Category order by catcount desc limit 10")
    t.show()
    t.map(t=> "column 0: "+ t(0)).collect().foreach(println)
    spark.stop()

  }
}


'Handling of Json files using spark SQL'

Json is inbuilt feature of spark


sqlContext.read.json("Json_path")


import org.apache.spark.sql.SparkSession

object JsonExample {
  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder.master("local[*]").appName("JsonExample").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext

    //Converting RDD to Data Frame
    import spark.implicits._
    import spark.sql
    val df = sqlContext.read.json("C:\\Users\\NaraVish\\Desktop\\#Personal\\#Interview
    Documents\\filformatsinspark\\world_bank.json")
    df.printSchema() //printing schema
   df.createOrReplaceTempView("jsondata_one")
    df.show()

    //val result =   sqlContext.sql("select url,totalamt,abc.* from jsondata_one " + "lateral
    view explode(theme_namecode) as abc")
    val result =   sqlContext.sql("select _id from jsondata_one")
    result.show(10)

//    println(result)


    //result.write.format("com.databricks.spark.csv").option("header","true").save(
    "C:\\Users\\sonirai\\Desktop\\Hadoop GV\\Spark\\SparkSQL\\datasets\\jsontocsv")
    spark.stop()
  }
}


'How to handle XML Files in Spark SQL '



import org.apache.spark.sql.SparkSession
object XMLExample {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder.master("local[*]").appName("xmlExample_1").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    val df = sqlContext.read.format("com.databricks.spark.xml").option("rootTag","books").option
    ("rowTag","book").load("C:\\Users\\NaraVish\\Desktop\\#Personal\\#Interview
    Documents\\filformatsinspark\\books.xml")
    df.show()
    //    df.printSchema()
    //df.createOrReplaceTempView("jsondata_one")
    //sqlContext.sql("select url,totalamt,abc.code,abc.name from jsondata_one " +
    //"lateral view explode(theme_namecode) as abc").show(5)
    //    result.write.format("com.databricks.spark.csv").option("header","true").save(
    "C:\\Kalyan\\POC\\Spark\\spark_datasets\\json_data\\jsontocsv")
    spark.stop()
  }

}


Major Sources of Data Frame :

1. RDS
2. CSV
3. JSON
4. XML
```

Spark Core is faster than Spark SQL Because Spark Sql another layer on Spark Core

DATA frame = Schema + Structured + RDD + Cost Based Optimzer

Data Set = Schema + Structured + RDD + Cost Based Optimzer + CPU Caching

Spark SQL writes the data to Driver Node.

Two issues with above

1. Memory Issues
2. Storage issue in Driver Node

Data Frame is subset of Data Set.

CPU Cache :

NVDR

```
        ------------------------------
       |     Spark SQL
       |----------------------------
       |     Spark Core
       |------------------------
       |  HDFS
        --------------------------------
```

Data Set Using one more serializer encoder (along with Java or Kyro)
It is 10 times faster than kyro serializer

4040 :

Data Frame DSL : Domain Specific Language

Read About Encoder serializer

RDDs :

1. Functional Programming
2. Type Safe

DataFrames

1. Relational
2. Catalyst Query Optimization
3. Tungsten direct/packed RAM
4. JIT Code Generation
5. Sorting/shuffling without De-serialization

CPU Cache is also called Tungsten in-memory in Spark

Data Frame Operations

1. First
    It returns the first row
        df.first() --> Action in RDD internally
2. Take --> It randomly collects the data
    It you want to display first n number of rows use take
        df.take(8)

3. Head()

Returns the first n rows in the form of rows and arrray Format.

```
df.head(5)
```

4. collect
    Returns an array that contains all of rows in this DataFrame

    Its similar to head(all),if the data is too large, Its not re-commendable

```
df.collect()
```

Take randomly takes the record with ordering , head first sorts and show the record so degrade the performance

5. count

    Returns the number of rows in the DataFrame

```
df.count()
```

6. show

    Displays the table on scree

```
df.show(5)
```

7. printSchema

'DSL' --> Domain Specific Language

Is the Native Language of Data Frame. Its is faster.

Its only avaialble for Retrieval purpose

```
import org.apache.spark.sql.SQLContext

val sqlContext = new sqlContext(sc)

val emp = sc.textFile("file:///home/cloudera/emp.txt")
val dept = sc.textFile("file:///home/cloudera/dept.txt")

case class Employee (eno:Int,ename:String,sal:Int,gendar:String,dno:Int)
case class Department (dno:Int,dname:String,dloc:String)


val edata = emp.map{ x =>
val w = x.split(",")
val eno = w(0).toInt
val ename=w(1)
val sal = w(2).toInt
val gendar = w(3)
val dno =w(4).toInt
Employee (eno,ename,sal,gendar,dno)
}


val ddata = dept.map{ x =>
val w = x.split(",")
val dno = w(0).toInt
val dname=w(1)
val dloc = w(2)
Department (dno,dname,dloc)
}

--//converting RDD to DataFrame

import sqlContext.implicits._

val edf = edata.toDF
val ddf = ddata.toDF

ddf.show()
edf.collect()
edf.take(10)
edf.count()
edf.first()


---DSL

edf.select($"eno",$"ename",$"sal",$"gendar",$"dno").show()
edf.select($max("sal").show()

*** msck repair table

val dataset = Seq(1,2,3).toDS()
dataset.show()

case class Person(name:String , age :Int)

val personDS = Seq (Person("Max",33),Person("Adam",32),Person("Muller",62)).toDS()
personDS.show()

---
val rdd = sc.parallelize(Seq((1,"Spark"),(2,"databricks")))
val integerDS= rdd.toDS()
integerDS.show()

--- converting DF to Data Set
case class Company (name : String,foundingYear :Int,numEmployees :Int)

val inputSeq = Seq(Company("ABC",1998,310),Company("XYZ",1998,310),Company("NOP",1998,310))
val df = sc.parallelize(inputSeq).toDF()

val companyDS = df.as[Company]
```

```
companyDS.show()


--- 4 steps in 1.x

import org.apache.spark.sparkContext
import org.apache.spark.SQLContext

val sc = new sparkContext

val sqlContext= new SQLContext(sc)







--**Download this file winutils.exe**

C : \
create Directory winutils
    create bin Directory bin
        copy winutils.exe






Spark Important

1. What is Spark
2. Why Spark is faster than Error
3. Difference between Hadoop and Spark
4. What is RDD
5. What is lenience
6. How Fault tolerance works in Spark
7. RDD - RESILIENT DISTRIBUTED DATATASET
8. What is RDD and what is not RDD
9. RDD Property
    a. List of Partititions
    b. Compute Functions
    c. If you call function in RDD it will perform for all elements not to specific
    d. List of Dependencies
    e. Main function of RDD is keep track of operation not the data.
    f. logical plan != Actual plan
10. Dependencies
    a. Narrow Dependencies
    b. Wide Dependencies
11. Intermediate data is cached but not the RDDs itself


----------------------------------------------------------------


Google --> Apache kafka

kafka.apache.org

Download

0.10.2.1

scala 2.11  --> kafka_2.11.0.10.2.1.tgz  --> Download this
```

```
**** msck repair

Apache Kafka


        Web Services 2000-03
        JMS (Java Messaging Services) 2005-07
                -- only 1 queue but not distributed
        TIBCO / Web Methods / Web Speher / Active MQ / Rabbit MQ -- >2008 (Web brokers)
                --n queues but only 1 queue is distributed
        Apache Kafka
                -- Uses a queue as well as broker which uses n queues and all are distributed

Apache Kafka is Independent System , its not dependent on Hadoop.
Its runs without Hadoop. But we can integrate Kafka to Hadoop or Kafka to spark.


Apache Flume Architecture :

    1. Flume is part of Hadoop
    2. Apache flume is a real time log processing (Data Ingestion) technique , it contains
    Components like
        a. Agents
        b. source
        c. sink
        d. channel
        e. Event
        f. Interceptor
        g. channel selector




Source      --> Aggent --> Events E1,E2 -- Sink
                        -----channel------------

** Learn Flume Architecture in Detail

Drawback : if we are selecting multiple Sourcess and multiple sinks then flume agent will fail
There is no backup for Source,sink and channel.


If we doing more sensitive data then flume is not usefull.

1.7 Flume has secondary channel

Kafka is less security.

Other Systems :
    1. Event Hub (Kafka + Security) -- Microsoft
    2. AWS - kenesis (Kafka + Security)


Kafka Architecture :

    1. Producer --no limits for sources
    2. consumer --no limits for targets
    3. Kafka Cluster
        a. Kafka Brokers
            i. Topic (logical, phyically it called partitions)

Each Topic contains N number of Partitions
Kafka maintains order using index called offset.
Kafka Replication
```

Producers

1. IOT
2. WebApp
3. NoSQL DBs
4. RDBMS
5. Sensor Data (Telemetric System)

Producers

1. IOT
2. WebApp
3. NoSQL DBs
4. Spark
5. Hadoop
6. Filesystem
7. cloud
8. RDBMS

'Communication between Producers to Kafka broker'

By Defaultly communication between producer to Kafka broker is synchronous communication

sync communication : Any system that sends acknowledge to source is sync system

Its not re-commendable as it increases throughput

'Asynchrononus communication'

There is no communication between producer and kafka broker.

Mostly it will be Asynchrononus .

Realtime Asynchrononus communication is used .

Disadvantage with Kafka :

Very Less Security
dependent on ZOOKEEPER

In Latest version of Kafka 1.x there is no Dependencies on zookeer
(bootstap servers)

kafka_2.11-1.1.0.tgz

```
[cloudera@quickstart ~]$ cd kafka_2.11-1.1.0
[cloudera@quickstart kafka_2.11-1.1.0]$ ls
bin  config  libs  LICENSE  NOTICE  site-docs
[cloudera@quickstart kafka_2.11-1.1.0]$ ls -l
total 52
drwxr-xr-x 3 cloudera cloudera  4096 Mar 23 15:54 bin
drwxr-xr-x 2 cloudera cloudera  4096 Mar 23 15:54 config
drwxr-xr-x 2 cloudera cloudera  4096 Jun 11 20:26 libs
-rw-r--r-- 1 cloudera cloudera 28824 Mar 23 15:51 LICENSE
-rw-r--r-- 1 cloudera cloudera   336 Mar 23 15:51 NOTICE
drwxr-xr-x 2 cloudera cloudera  4096 Mar 23 15:54 site-docs
[cloudera@quickstart kafka_2.11-1.1.0]$
```

In Kafka Bin Directory contains list of all kafka services

Producer start & stop
borker start and stop
consumer start and stop etc

```
By Defaultly all the services available in bin directory takes default properties

In config directory contains default properties like server.properties , zookeer.properties,
producer.properties & consumer.properties


--------------14th June-----------------------------

server properties --> Retension policy (Default 24*7 = 168 Hours)

Topic is replicated and it is taken care of cluster

IN ZOOKER.PROPERITES file contains 2 imp properties

1. Data Directory (dataDir)
    Stores Zooker log information ,
    Default path of log directory is /tmp/zookeer
2. clientPort
    The Default port no of zooker is 2181

dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181


QuorumPeerMain --> This means your zookeeper Running (Internally using 2181)




How to kill the service := kill -9 Process_id



1. Change port no in zookeeper properties
2. root user
    start zookeeper server
    kafka/bin
    ./zookeeper-server-start.sqh  ../config/zookeeper.properties
3.
server.properties file contains 4 major parameter

    1. broker.id
    2. broker ip address (listener)
    3. broker log Directory
    4. zooker ip address and port no


9092 is default port no of a kafka broker

4. Start zookeeper service

    ./kafka-server-start.sh ../config/server.properties &

5. jps

    kafka services will be started

6.

[cloudera@quickstart bin]$ cd
[cloudera@quickstart ~]$ cd kafka_2.11-1.1.0
[cloudera@quickstart kafka_2.11-1.1.0]$ cd bin/
[cloudera@quickstart bin]$
```

```
./kafka-topics.sh --create --zookeeper localhost:2182 --partitions 2  --replication-factor 1
--topic test_20180613

./kafka-topics.sh --list --zookeeper localhost:2182

[cloudera@quickstart bin]$ pwd
/usr/lib/zookeeper/bin


[cloudera@quickstart bin]$ ./zkCli.sh -server localhost:2182
Connecting to localhost:2182
2018-06-12 20:18:53,474 [myid:] - INFO  [main:Environment@100] - Client environment:zookeeper.
version=3.4.5-cdh5.12.0--1, built on 06/29/2017 11:30 GMT
2018-06-12 20:18:53,479 [myid:] - INFO  [main:Environment@100] - Client environment:host.name=
quickstart.cloudera
2018-06-12 20:18:53,479 [myid:] - INFO  [main:Environment@100] - Client environment:java.version
=1.7.0_67
2018-06-12 20:18:53,484 [myid:] - INFO  [main:Environment@100] - Client environment:java.vendor=
Oracle Corporation
2018-06-12 20:18:53,484 [myid:] - INFO  [main:Environment@100] - Client environment:java.home=/
usr/java/jdk1.7.0_67-cloudera/jre
2018-06-12 20:18:53,484 [myid:] - INFO  [main:Environment@100] - Client environment:java.class.
path=/usr/lib/zookeeper/bin/../build/classes:/usr/lib/zookeeper/bin/../build/lib
/*.jar:/usr/lib/zookeeper/bin/../lib/slf4j-log4j12.jar:/usr/lib/zookeeper/bin/../lib/slf4j-log4j1
2-1.7.5.jar:/usr/lib/zookeeper/bin/../lib/slf4j-api-1.7.5.jar:/usr/lib/zookeeper/bin/../lib/netty
-3.10.5.Final.jar:/usr/lib/zookeeper/bin/../lib/log4j-1.2.16.jar:/usr/lib/zookeeper/bin/../lib/jl
ine-2.11.jar:/usr/lib/zookeeper/bin/../zookeeper-3.4.5-cdh5.12.0.jar:/usr/lib/zookeeper/bin/../sr
c/java/lib/*.jar:/usr/lib/zookeeper/bin/../conf:
2018-06-12 20:18:53,484 [myid:] - INFO  [main:Environment@100] - Client
environment:java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib
2018-06-12 20:18:53,484 [myid:] - INFO  [main:Environment@100] - Client
environment:java.io.tmpdir=/tmp
2018-06-12 20:18:53,484 [myid:] - INFO  [main:Environment@100] - Client
environment:java.compiler=<NA>
2018-06-12 20:18:53,484 [myid:] - INFO  [main:Environment@100] - Client environment:os.name=Linux
2018-06-12 20:18:53,484 [myid:] - INFO  [main:Environment@100] - Client environment:os.arch=amd64
2018-06-12 20:18:53,486 [myid:] - INFO  [main:Environment@100] - Client
environment:os.version=2.6.32-573.el6.x86_64
2018-06-12 20:18:53,486 [myid:] - INFO  [main:Environment@100] - Client
environment:user.name=cloudera
2018-06-12 20:18:53,486 [myid:] - INFO  [main:Environment@100] - Client
environment:user.home=/home/cloudera
2018-06-12 20:18:53,486 [myid:] - INFO  [main:Environment@100] - Client
environment:user.dir=/usr/lib/zookeeper/bin
2018-06-12 20:18:53,488 [myid:] - INFO  [main:ZooKeeper@438] - Initiating client connection,
connectString=localhost:2182 sessionTimeout=30000
watcher=org.apache.zookeeper.ZooKeeperMain$MyWatcher@5cd4927f
Welcome to ZooKeeper!

2018-06-12 20:18:53,639 [myid:] - INFO
[main-SendThread(localhost:2182):ClientCnxn$SendThread@975] - Opening socket connection to
server localhost/127.0.0.1:2182. Will not attempt to authenticate using SASL (unknown error)
JLine support is enabled
2018-06-12 20:18:53,679 [myid:] - INFO
[main-SendThread(localhost:2182):ClientCnxn$SendThread@852] - Socket connection established,
initiating session, client: /127.0.0.1:33792, server: localhost/127.0.0.1:2182
2018-06-12 20:18:53,706 [myid:] - INFO
[main-SendThread(localhost:2182):ClientCnxn$SendThread@1235] - Session establishment complete
on server localhost/127.0.0.1:2182, sessionid = 0x163f6fde80d0004, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2182(CONNECTED) 0]
```

```
*/

[zk: localhost:2182(CONNECTED) 2] ls brokers/
Command failed: java.lang.IllegalArgumentException: Path must start with / character
[zk: localhost:2182(CONNECTED) 3] ls /brokers
[seqid, topics, ids]
[zk: localhost:2182(CONNECTED) 4] ls /brokers/ids
[0]
[zk: localhost:2182(CONNECTED) 5] ls /brokers/topics
[test_20180613]
[zk: localhost:2182(CONNECTED) 6] ls /brokers/seqid
[]


--Start Producer as normal user

There are two types of producers in kafka
    1. console producer (Its default producer)
    2. custom producer (End User create custom producer by using producer api option)


'Step 5'
./kafka-console-producer.sh --broker-list localhost:9092 --topic test_20180613

'Step 6'

    Two Types of consumer
        1. console consumer (Its Default consumer)
        2. custom consumer (End user create custom consuer by using api consumer api)

./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test_20180613
--from-beginning


----------------14th June------------------



./zookeeper-server-start.sh ../config/zookeeper.properties &


cp server.properties server_2.properties
  350  gedit server.properties
  351  cp server.properties server_1.properties
  352  gedit server.properties
  353  gedit server_1.properties
  354  gedit server_2.properties
  355  mkdir /tmp/kb0
  356  mkdir /tmp/kb1
  357  jps
  358  exit
  359  ls
  360  ./zookeeper-server-start.sh ../config/zookeeper.properties &
  361  jps


  ./kafka-server-start.sh ../config/server_1.properties &
  368  ./kafka-server-start.sh ../config/server_2.properties &
  369  jps
  370  ./kafka-server-start.sh ../config/server_1.properties &
  371  ./kafka-server-start.sh ../config/server_2.properties &

[zk: localhost:2182(CONNECTED) 0] ls /brokers
[seqid, topics, ids]
[zk: localhost:2182(CONNECTED) 1] ls /brokers/ids
[2, 1, 0]
```

```
[zk: localhost:2182(CONNECTED) 2]



'Command to Identify who is the leader and who is slave'

/kafka-topics.sh --desc --zookeeper localhost:2182 --topic test_20180614

0 --> Means Leader
1 --> Follower 1
2 --> Follower 2

Isr means in-sync-replica


Partitions wise Leader is selected not topic wise

-----------------------------------------------------------------------


Custom Producer




MySQL DB --> Custom Producer --> Kafka Broker   --> Console
                (producer api)

import java.util.Properties;
import java.sql.*;
import kafka.javaapi.producer.Producer; // send method
import kafka.producer.KeyedMessage;  // Serializer
import kafka.producer.ProducerConfig; // configuration where is my broker list

public class JdbcProducer{
public static void main(String[] args) throws ClassNotFoundException,SQLException{
      Properties  props = new Properties();
      props.put("zk.connect","localhost:2182");
      props.put("serializer.class","kafka.serializer.StringEncoder");
      props.put("metadata.broker.list","localhost:9092");
      ProducerConfig config = new ProducerConfig(props);
      Producer producer = new Producer(config);
      try{
           class.forName("com.mysql.jdbc.Driver")
           Connection con = DriverManager.getConnection(
           "jdbc:mysql://localhost:3306/test","root","root");

           // test is the DB name
           Statement stmt = con.createStatement();
           ResultSet rs = stmt.executeQuery("select * from emp");
           while(rs.next())
                         producer.send(new KeyedMessage("test",rs.getString(1)+" "+rs.getString(
                         2)));
            con.close()
            }catch (Exception e){
            System.out.println(e)
            }
            }
            }

Key Serializer
Value Serializer


we need to copy mysql jar to kafka/lib
```

```
***


'How to start kafka '

./kafka-server-start.sh ../config/server.properties &


./kafka-topics.sh --create --zookeeper localhost:2182 --partitions 2  --replication-factor 1
--topic testodbc

./kafka-topics.sh --list --zookeeper localhost:2182

./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic testodbc --from-beginning


'Custom Consumer'

Major Consumer

    Kafka + Spark
    Kafka + Storm
    Kafka + Filesystem

'Integration of Kafka with Spark'

Kafka                       Spark

(Near Real Time)        (Batch + Inmemory)


                        Spark Streaming
                            (Micro Batch)

Spark Streaming internally using a concept of Micro batch processing

Batch --> Static and Historical Data , not transactional Data (Days - Months - Years )
Micro Batch --> Not a static data , -- Almost Transactional Data
                -- Seconds
                -- Minutes
                -- Rarely Hours

Spark Streaming internally using a concept of D-Streams

D-Streams --Discretized Data Streams

D-Streams internally working on RDDs only.

Spark Streaming always expects 2 threads
    1. Spark Streaming Context Object (ssc)
    2. Spark Context


P -- > Kafka --> Spark Streaming Context --> D Streams --> Spark Context

D-Streams works Inmemory and reside in Worker Node


spark-shell --master local[2]

import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

val ssc = new StreamingContext(sc,Seconds(10))
val lines = ssc.socketTextStream("localhost",9999)
val words = lines.flatMap(x => x.split(" "))
val pairs = words.map(x => (x,1))
val res = pairs.reduceByKey(_+_)
res.print()
ssc.start

nc -lk 9999

nc --> net cat Server
-lk --> localhost
9999 --> port no


'Integration of Kafka to Spark '

                                          KF
Console                         --> (topic : Input) --> SSC      -->        SC
Producer                                                                    |
                                                                            |
--Console Consumer <-- Output   <-- (topic :output )            <--         SC Custom Producer


1. Start the ZooKeeper

kafka/bin
    ./zookeeper-server-start.sqh  ../config/zookeeper.properties

2. Start Kafka Broker

kafka/bin
    ./zookeeper-server-start.sh ../config/server.properties

3. Create topics

./kafka-topics.sh --create -zookeeper localhost:2182 --paritnitions 1 -replication-factor 1
topic-input
./kafka-topics.sh --create -zookeeper localhost:2182 --paritnitions 1 -replication-factor 1
topic-output

4. Start console producer

kafka/bin
./kafka-console-producer.sh --broker-list localhost:9092 --topic input

'* Kafka Utils establish between kafka broker to Spark Streaming context*'

5.


'*RDDs --> Partitions --> Records*'

spark-shell --master local[2]

import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

val ssc = new StreamingContext(sc,Seconds(10))
val lines = ssc.socketTextStream("localhost",9999)
val words = lines.flatMap(x => x.split(" "))
val pairs = words.map(x => (x,1))
val res = pairs.reduceByKey(_+_)
res.print()
ssc.start
```

```
nc -lk 9999
----------------------------------------------------------------------
kafka - Spark Streaming Example
1) please start zookeeper
   ./bin/zookkeeper-server-start.sh config/zookeeper.properties &
2) please start kafka-Broker
   ./bin/kafka-server-start.sh config/server.properties &
3) spark-shell --master local[2]
4) create two topics mytopic,results
   ./bin/kafka-topics.sh --create --zookeeper localhost:2182 --partitions 1
    --replication-factor 1 --topic mytopic

    ./bin/kafka-topics.sh --create --zookeeper localhost:2182 --partitions 1
    --replication-factor 1 --topic results

    ./bin/kafka-topics.sh --list --zookeeper localhost:2182

5) please start console producer with mytopic
   ./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic mytopic

6) create another to write the spark streaming output data(topic name result)

7) please start console consumer to read the data from result topic.
   ./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic results
   --from-beginning

   spark-shell --master local[2]
8) import org.apache.spark.streaming.StreamingContext
   import org.apache.spark.streaming.Seconds
   val ssc = new StreamingContext(sc, Seconds(5))
   import org.apache.spark.streaming.kafka.KafkaUtils
   val kafkaStream = KafkaUtils.createStream(ssc,"localhost:2182",
   "spark-streaming-consumer-group",Map("mytopic" ->5))
   val lines = kafkaStream.map(x => x._2.toUpperCase)
   val words = lines.flatMap(x => x.split(" "))
   val pairs = words.map(x => (x,1))
   val res = pairs.reduceByKey(_+_)
   import org.apache.kafka.clients.producer.ProducerConfig
   import java.util.HashMap
   import org.apache.kafka.clients.producer.KafkaProducer
   import org.apache.kafka.clients.producer.ProducerRecord
   res.foreachRDD(rdd =>
       rdd.foreachPartition(partition =>
        partition.foreach{
         case(w:String,cnt:Int)=>{
          val x = w+"\t"+cnt
          val props = new HashMap[String,Object]()
          props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"localhost:9092")
          props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
          "org.apache.kafka.common.serialization.StringSerializer")
          props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
          "org.apache.kafka.common.serialization.StringSerializer")

import org.apache.kafka.clients.producer.KafkaProducer
   import org.apache.kafka.clients.producer.ProducerRecord
val producer = new KafkaProducer[String,String](props)
val message = new ProducerRecord[String,String]("results",null,x)
          producer.send(message)

         }
        }
       )
   )

   8)please start streaming context
   ssc.start
```

```
Accumulator --21st June
Broadcast Variable --21st Jun
Spark Memory Management  -- 22rd June

Sqoop (Mon and Tue) --25th,26th June
Flume (Wed and Thurs) --27th,28th June
Pig   (Friday , Mond, Tue) --29th,2nd and 3rd July
MR (Wed to Friday) -- 4th,5th,6th July
Oozie (Mon & Tue ) --9th July , 10th July
Hbase (Wed & Thurs) --12th ,12th July
MongoDB (Frid and Mon ) -- 13th,16th July

Hive SerDe & UDF (Tues) --17th July
GIT and Bitbucket (Wed) --18th July
Challenging Prod (Thursday & Friday ) --19th & 20th July


'Shard variables in Spark'

In Spark two types of shared variables
    1. Broadcast variables
    2. Acumulators

1. Broadcast Variables

    Broadcast variables are immutable

    Broadcast variables are readonly(immutable)
    Broadcast variables are fault tolerant.
    Broadcast variables should fit in Mememory
    Broadcast variables are distributed to the cluster
    Broadcast varialbe are not re-commendable production (Same concept avaialble in MR, We call
    it has distributed cache (Map Join))


How to create Broadcast variable

    If you want to create Broadcast variable we require 2 methods
    1. sc.Broadcast
    2. convert static table to a HashMap table using a method called collectAsMap


sc.Broadcast method to initialize the Broadcast variable in Executor in-memory

collectAsMap Method converts a scala object to or RDD to a HashMap Table



val hoods = Seq((1,"Mission"),(2,"Soma"))
val checkins = Seq((234,1),(567,2))
val hoodsRDD=sc.parallelize(hoods)
val checkRDD=sc.parallelize(checkins)
val broadcastedHoods=sc.broadcast(hoodsRDD.collectAsMap)

val checkinsWithHoods = checkRDD.mapPartitions({
row =>
row.map(x => (x._1,x._2,broadcastedHoods.value.getOrElse(x._2,-1))) // -1 means descending order
},preservesPartitioning=true)// to make it as narrow transformation

checkinsWithHoods.take(5)


// Accumulator

Accumulator are broadcast variable. It will collect performance counter of each and every worker
```

node basic statstics of executor performance.
and Submitted back to driver node.

Major Accumulator will be taken care by administrator.

It will take performance counters (map Reduce : Counting with Counters= Accumators in Spark)

performance counters means stats about the executor

Accumulator collects performance counter for each executor to Driver node

'How to create an Accumulator'

In Spark Core Context a method called Accumulator and it expected Initial value of counter.

```
scala> val accum= sc.Accumulator[Int] = 0
<console>:1: error: ';' expected but '=' found.
       val accum= sc.Accumulator[Int] = 0
                                      ^

scala> val accum= sc.accumulator(0)
accum: org.apache.spark.Accumulator[Int] = 0

scala> sc.parallelize(Array(1,2,3,4)).foreach( x => accum +=x)

scala> accum.value
res3: Int = 10
```

'Memory Management is Spark'

In Spark Memory is splitted into 3 core Components

1. Reserved Memory
2. User Memory
3. Spark Memory
    Further Divided into 2 splits
    a. Executor Memory
    b. Storage Memory

1. Reserved Memory

    To Start a Spark Application we required a certain amount of Memory and that memory is called as reserved memory
    and default size of reserved Mememory is 300 MB

2. User Memory

    The Deverloper or End User start creating RDDs , Partitions , Transformation , Actions internally using a memory block called User Memory.

    This Mememory belong to Driver Node.

    25% of Whole Memory (After substraction of Reserved Memory 300)

3. Spark Memory -- Worker Node

    Remaining 75% Allocated to Spark

```
    Spark 1.x :     Executor = Storage = 50% of Spark Memory
    Spark 2.x :     ?
```

    Further Divided into 2 splits
    a. Executor Memory -- only to execute the job
    b. Storage Memory --Store the intermediate data

```
Checkpoint Service ?

RDD.cache()
RDD.persist()


Both Represents Storage Memory

Cache                                    Persist

Default it is stored in         1. MEMORY_ONLY
Storage Memory -- RAM           2. DISK_ONLY (Worker node Disk)

MEMORY_ONLY                     3. MEMORY_AND_DISK (70% in Memeory and 30% in Disk) --this is configurable

                                4. MEMORY_AND_DISK_SER
                                5. MEMORY_ONLY_SER


spark.exit() --> All Data will be

RDD.cache() --> It can store in any one of the worker node


cache Never use in Real time

 2 more options in latest release

MEMORY_AND_DISK_2
MEMORY_ONLY_2

uncache()
unpersist()

Drawback of Spark 1.x is storage 50% is a bottleneck

In Spark

--------------------------------------------------------------END OF SPARK------

Scala Programs

scala_practice (1).txt
Details
Activity
scala_practice (1).txt
Sharing Info
k
d

J
R
+8
General Info
Type
Text
Size
12 KB (12,342 bytes)
Storage used
0 bytesOwned by someone else
Location
Scala
Owner
kumar K
```

```
Modified
24 Mar 2018 by Soni Rai
Created
24 Mar 2018
Description
Add a description
Download permissions
Viewers can download


Scala Practice Exercises
========================
Ex:1)val l = List(10,20,30,40,50,60,70,80,90,100)
     //find elements greater than 40
     val result = l.filter(x => x>40)

val l = 1 to 10 toList
val x = 1 to 10 toArray
val y = 1 to 100 by 2

Ex:2)val names = List("abc","def","ghi")
     //convert all elements to uppercase
     val result = names.map(x => x.toUpperCase)
     val result = names.map(_.toUpperCase)
     names.

Ex:3) val x = List(10,20,30,40,50)
     //sum of all elemets in the list using for loop
     var tot =0
     for(i<-x)
     tot+=i
     tot

Ex:4)val lst = List(1,2,3,4,5,6,7,8,9,10)
     //4th element to remaining elements
     val r1 = lst.slice(3,lst.size)
     //1st element to 5th element
     val r2 = lst.slice(0,5)
     //3rd element to 8th element
     val r3 = lst.slice(2,8)

Ex:5)val x = List(10,20,30)
     5::x
     x::35 //not possible but its possible in ListBuffers

    val x = 1 to 5 toList
    val y = 6 to 10 toList
    x:::y
    x::y Or x++y

Ex:6)val x = List(10,20,30,40,50,60)
     val y = List(70,80,90)
     x++y
     val z = y++x

Ex:7)val x = List(10,20,30,40)
     x++List(50)

Ex:8)val t=("Ravi",35,"Mtech","Hyd")
     val name= t._1
     val age = t._2
     val qual = t._3
     val loc = t._4

Ex:9)var x= Map("x" -> "abc","y"->"def")
     x("x")
     x+=(z->"ghi")
     x

Ex:10)//Transformations Map,FlatMap and Filter
     val x = List(10,20,30,40,50)
     val y = x.map(x => x+100)
     val z = y.filter(x => x>120)
     (or) val r = x.map(x => x+100).filter(v => v>120)

Ex:11)val name = List("rAvI","rani","VANi","VeNu")
      val result = name.map{ x =>
      val w = x.trim()
      val fc = w.substring(0,1).toUpperCase
      val rc = w.substring(1).toLowerCase
      fc+rc
      }

Ex:12)val  sal = List(10000,20000,30000,40000,50000)
     //net salary tax 10% and hra 20%
     val nets = sal.map{x =>
     val tax = x*10/100
     val hra = x*20/100
     val net = x+hra-tax
     net
     }

Ex:13)val name = List("rAvI","rani","VANi","VeNu")
      val result = name.map{ x =>
      val w = x.trim()
      val fc = w.slice(0,1).toUpperCase
      val rc = w.slice(1,w.size).toLowerCase
      fc+rc
      }

Ex:14)//Diff between map and flatmap
        val l = List(List(1,2,3),List(3,4),List(1,3,5,6),List(1,2,3))
        //using map transformation
        val r = l.map(x => x.sum)
        val res = r.sum
        //using flatMap transformation
        val res = l.flatMap(x => x).sum

Ex:15)val lines = List("I love hadoop","I Love Spark","I love Spark and Hadoop","Spark is Great")
      //((I,1), (love,1), (hadoop,1), (I,1), (Love,1), (Spark,1), (I,1), (love,1), (Spark,1), (
      and,1), (Hadoop,1), (Spark,1), (is,1), (Great,1))
      val rmap = lines.map(x => x.split(" "))
      val rfmap = lines.flatMap(x => x.split(" ")).map(x => (x,1))

Ex:16)val recs = List("101,Anil,100000,m,11","102,amala,50000,f,12","103,giri,60000,m,11",
"104,girija,90000,f,13","105,Mani,10000,m,12")
      //select sex,sum(sal) from emp group by sex;
      recs.foreach(println)
      val arr = recs.map(x => x.split(","))
      val pair = arr.map(x => (x(3),x(2).toInt))

Ex:17)val data = List("100,200,300","800,200,300,400,500","10000,30000","900,1000,5000")

Ex:18)val l = List(List(1,2,3),List(3,4),List(1,2,3,4),List(1,2,3,4,5))
      val result = l.filter(x => x.size>2)

Ex:19)val recs = List("101,Anil,100000,m,11","102,amala,50000,f,12","103,giri,60000,m,11",
"104,girija,90000,f,13","105,Mani,10000,m,12")
      //select * from emp where gendar = "m";
      val m = recs.filter(x => x.contains("m"))

      val males = recs.filter{ x =>
```

```
           val gendar = x.split(",").toLowerCase
           gendar == "m"
           }
     males.foreach(println)

val recs = List("101,Anil,100000,m,11","102,amala,50000,f,12","103,giri,60000,m,11",
"104,girija,90000,f,13","105,Mani,10000,m,12")

val result = recs.map{x=>
val w = x.split(",")
val dno = w(4).toInt
val ename = w(1)
val sal = w(2).toInt
val gendar = w(3)
(dno,(ename,sal,gendar))
}
Ex:20)//Conditional Transformations
      val lst = List(10000,30000,90000,20000,60000,80000)
      lst.sum
      lst.size
      val avg = lst.sum/lst.size
      val result = lst.map(x => if(x>=avg) "Above AVG Sal" else "Below AVG Sal")

val recs = List("101,Anil,100000,m,11","102,amala,50000,f,12","103,giri,60000,m,11",
"104,girija,90000,f,13","105,Mani,10000,m,12")

val result = recs.map{x=>
val w = x.split(",")
val sal = w(2).toInt
val ssum = sal.sum
val scnt = sal.size
val savg = ssum/scnt
val r = w.map(x => if(x>=savg) "Above AVG Sal" else "Below AVG Sal")
 r
}

Ex:21)val a =100
      val b =250
      val c =150

      if(a>b) a else if(b>c)b else if(a>c)a else c

Ex:22)val dno = List(11,12,13,11,11,11,12,13,12,12,13,14,15,11)
      val dname = dno.map{x =>
      if(x==11) "Marketing" else if(x==12) "HR" else if(x==13) "Finance" else "Others"
      }

Ex:23)val gendar = "m"
      val result = gendar match{
      case "m" => "Male"
      case "f" => "Female"
      case other => "Unknown"
      }

Ex:24)val gendar ="f"
      val result = (gendar == "m") match{
      case true => "Male"
      case false => "Female"
      }

Ex:25)val sal = List(10000,30000,90000,20000,60000,80000)
      val r =sal.map{ x =>
      var grade =" "
      if (x>=70000) grade = "A" else if(x>=50000) grade ="B" else if(x>=30000) grade ="C" else
      grade ="D"
      grade
      }

Ex:26)val sal = List(10000,30000,90000,20000,60000,80000)
      val res = sal.map{x =>
      var grade =" "
      (x>=70000) match{
      case true => grade ="A"
      case false => (x>=50000) match{
      case true => grade ="B"
      case false => (x>=30000) match{
      case true => grade ="C"
      case false => grade = "D"
      }
     }
    }
    grade
    }

Ex:27)//using match and if combination
      val sal = List(10000,30000,90000,20000,60000,80000)
      val res = sal.map{x =>
      var grade =" "
      (x>=70000) match {
      case true => grade ="A"
      case false => if(x>=50000) grade="B" else if(x>=30000) grade="C" else grade = "D"
      }
     grade
    }

Ex:28) val records = List("101,Anil,40000,m,11","102,aMaLA,80000,F,12","103,ManI,10000,m,13",
"104,GIri,45000,m,14","105,SuReSH,60000,f,12","106,SiRI,90000,M,15")
       //name --->first char is upper and remaining are lower.
       //sal ---> calculate net salary(net =sal+hra-tax)hra=20% and tax=10%
       //sal --->sal grades like A,B,C,D
       //gendar ---> m/M and f/F convert as Male and Female
       //dno ---> 11-Marketing,12-HR,13-Finance,others

       val results = records.map{x =>
       val w = x.split(",")
       val id = w(0)
       val name = w(1).trim()
       var sal = w(2).toInt
       var gendar = w(3)
       val dno = w(4).toInt
       val fc = name.slice(0,1).toUpperCase
       val rc = name.slice(1,name.size).toLowerCase
       val newname = fc+rc
       gendar = if(gendar.toUpperCase=="M") "Male" else "Female"
       val tax = sal*0.1
       val hra = sal*0.2
       val net = sal+hra-tax
       var grade = " "
       if(net>=70000) grade="A" else if(net>=50000) grade ="B" else if(net>=30000) grade ="C"
       else "D"
       grade
       val dname = dno match{
       case 11 => "Marketing"
       case 12 => "HR"
       case 13 => "Finance"
       case other => "Others"
       }
     val newList = List(id,newname,sal.toString,hra.toString,tax.toString,net.toString,grade,
     gendar,dno.toString,dname)
     newList.mkString(",")
    }
```

```
Ex:29) val emp = Array("101,aaaa,30000,m,11","102,bbbb,50000,f,12","103,hhhh,60000,m,11",
"104,qqqq,80000,f,11")
       //select gendar,sum(sal) from emp group by gendar;
       val pair1 = emp.map{x =>
       val w = x.split(",")
       val gendar = w(3)
       val sal = w(2).toInt)
       (gendar,sal)
       }

Ex:30)val emp = Array("101,aaaa,30000,m,11","102,bbbb,50000,f,12","103,hhhh,60000,m,11",
"104,qqqq,80000,f,11")
      //select dno,sum(sal) from emp group by gendar;
       val pair1 = emp.map{x =>
       val w = x.split(",")
       val dno = w(4)
       val sal = w(2).toInt)
       (dno,sal)
       }

Ex:31)val emp = Array("101,aaaa,30000,m,11","102,bbbb,50000,f,12","103,hhhh,60000,m,11",
"104,qqqq,80000,f,11")
      //select dno,gend,sum(sal) from emp group by gendar;
       val pair1 = emp.map{x =>
       val w = x.split(",")
       val dno = w(4)
       val gend = w(3)
       val sal = w(2).toInt)
       val mykey = (dno,gend)
       (mykey,sal)
       }

Ex:32)//Making the records into structures
      i)Tuple
      ii)case class

      emp = Array("101,aaaa,30000,m,11","102,bbbb,50000,f,12","103,hhhh,60000,m,11",
      "104,qqqq,80000,f,11")
      emp.foreach(println)

      val recs = emp.map{ x =>
      val w = x.split(",")
      val id = w(0).toInt
      val name = w(1)
      val sal = w(2).toInt
      val gend = w(3)
      val dno = w(4).toInt
      (id,name,sal,gend,dno)
      }
      recs.foreach(println)

      val pair4 = recs.map(x => (x._4,x._3.toInt))

      //select sum(sal) from recs;

      val tsum = recs.map(x => x._3).sum

Ex:33)val textdata = "    I    Love    Spark    "
      val word = textdata.trim()
      val s = word.split(" ")
      val w = s.filter(x => x!="")
      val result = w.mkString(" ")

Ex:34)val post = List("I Love Spark    ","you    Love    hadoop","hadoop and spark are great
big data    systems")
      val result = post.map{x =>
      val w = x.trim().split(" ")
      val words = w.filter(x => x!="")
      words.mkString(" ")
     }
     result.foreach(println)

Ex:35)//case class
     case class Samp(a:Int,b:Int,c:Int)
     val s1 = Samp(10,20,30)
     val s2 = Samp(1,2,3)
     val s3 = Samp(100,200,300)
     val s = List(s1,s2,s3)

Ex:36)//case class
     case class Emp(id:Int,name:String,sal:Int,gendar:String,dname:String)
     val e = emp.map{x =>
     val w = x.split(",")
     val id = w(0).toInt
     val name = w(1)
     val sal = w(2).toInt
     val gendar = w(3)
     val dno = w(4).toInt
     val dname = dno match{
     case 11 => "Marketing"
     case 12 => "HR"
     case 13 => "Finance"
     case other => "Others"
     }
     val result = Emp(id,name,sal,gendar,dname)
     result
    }
    e.foreach(println)
    val pair5 = e.map(x =>(x.dname,x.sal))

Ex:37)//Functions
     def f:String = "Hello World"
      f

Ex:38) def f ={
        val x = "hello"
        val y = x.toUpperCase
         y
         }

Ex:39) def fx(a:Int):Int = a+100
        fx(10)

Ex:40)case class Empl(id:Int,name:String,sal:Int,sex:String,dno:Int,
dname:String,grade:String)

     def FirstUpper(x:String):String={
      val w = x.trim()
      val fc = w.slice(0,1).toUpperCase
      val rc = w.slice(1,w.size).toLowerCase
      val name = fc+rc
      name
     }

     def gend(x:String):String={
      if(x.toUpperCase == "M") "Male" else "Female"
     }

     def grade(x:Int):String={
       if(x>=70000) "A" else if(x>=50000) "B" else if(x>=30000) "C" else "D"
     }
     def dept(x:Int):String ={
      val dname = x match{
      case 11 => "Marketing"
      case 12 => "HR"
      case 13 => "Finance"
      case other => "Others"
      }
     dname
    }

     def toEmp(line:String):Empl={
      val w = line.split(",")
      val id = w(0).toInt
      val name = FirstUpper(w(1))
      val sal = w(2).toInt
      val gendar = gend(w(3))
      val dno = w(4).toInt
      val dname = dept(dno)
      val grad = grade(sal)
      val e = Empl(id,name,sal,gendar,dno,dname,grad)
      e
     }

     toEmp("201,ANiL,80000,m,11")

     val emps = emp.map(x =>toEmp(x))
     emps.foreach(println)

Ex:41)def isMale(x:String):Boolean={
       x.toUpperCase == "M"
       }
       val lst = List("m","M","f","F","M","f","M")
       val males = lst.filter(x => isMale(x))
       val females = lst.filter(x => !isMale(x))

       val m = emp.filter(x =>isMale(x.split(",")(3)))
       val f = emp.filter(x =>!isMale(x.split(",")(3)))

       val m = emps.filter(x => isMale(x.gendar.slice(0,1)))
       val f = emps.filter(x => !isMale(x.gendar.slice(0,1)))

Ex:42)def power(x:Int,n:Int):Int={
       if(n>=1) x*power(x,n-1)
       else 1
       }

Ex:43)def fact(x:Int):Int={
       if(x>1) x*fact(x-1)
       else 1
       }


--- Spark Core


Ex:1)val r1 = List((11,10000),(11,20000),(12,30000),(12,40000),(13,50000))
     val r2 = List((11,"hyd"),(12,"bang"),(13,"hyd"))
     val rdd1 = sc.parallelize(r1)
     val rdd2 = sc.parallelize(r2)
     val j = rdd1.join(rdd2)
     j.collect.foreach(println)
     val citysalpair = j.map{x =>
                             val city = x._2._2
                 val sal  = x._2._1
                     (city,sal)
                }
    citysalpair.collect.foreach(println)
    val result = citysalpair.reduceByKey(_+_)
    result.collect.foreach(println)

Ex:2)val e = List((11,30000,10000),(11,40000,20000),(12,50000,30000),
(13,60000,20000),(12,80000,30000))
     val ee = sc.parallelize(e)
     ee.collect.foreach(println)
     rdd2.collect.foreach(println)
     val j2 = ee.join(rdd2)//error because both structures in key and value.
     val e3 = ee.map{x =>
                val dno = x._1
            val sal = x._2
            val bonus = x._3
            (dno,(sal,bonus))
                }
    val j3 = e3.join(rdd2)
    j3.collect.foreach(println)
    val pair = j3.map{x =>
                val sal= x._2._1._1
            val bonus = x._2._1._2
            val tot = sal+bouus
            val city = x._2._2
            (city,tot)
            }
    pair.collect.foreach(println)
    val result2 = pair.reduceByKey(_+_)
    result2.collect.foreach(println)

Ex:3)emp file
         cat > emp
         101,aaaa,70000,m,12
     102,bbbb,90000,f,12
     103,cccc,10000,m,11
     104,dddd,40000,m,12
     105,eeee,70000,f,13
     106,ffff,80000,f,13
     107,gggg,90000,m,14
     108,hhhh,10000,f,14
     109,iiii,30000,m,11
     110,jjjj,60000,f,14
     111,kkkk,90000,m,15
     112,llll,10000,m,15
     dept file
     cat > dept
     11,marketing,hyd
     12,hr,del
     13,finance,hyd
     14,admin,del
     15,accounts,hyd
     //move the files from lfs to hdfs.
     hadoop fs -mkdir /sparkcore
     hadoop fs -put emp /sparkcore
     hadoop fs -put dept /sparkcore
     hadoop fs -ls /saprkcore
     val emp=sc.textFile("/user/cloudera/sparkcore/emp")
     val dept = sc.textFile("/user/cloudera/sparkcore/dept")
     emp.collect.foreach(println)
     dept.collect.foreach(println)
     val e = emp.map{ x =>
             val w = x.split(",")
             val dno = w(4).toInt
             val id = w(0)
             val name = w(1)
             val sal = w(2).toInt
             val sex = w(3)
             val info = id+","+name+","+sal+","+sex
             (dno,info)
                          //(dno,(id,name,sal,gendar))
             }









    e.collect.foreach(println)
    val d = dept.map{ x =>
            val w = x.split(",")
            val dno = w(0).toInt
            val dname = w(1)
            val city = w(2)
            val info =dname+","+city
            (dno,info)
            }
    d.collect.foreach(println)
    val ed = e.join(d)
    ed.collect.foreach(println)
    val ed2 = ed.map{x =>
              val einfo = x._2._1
              val dinfo = x._2._2
              val info = einfo+","+dinfo
              info
              }
    ed2.collect.foreach(println)
    ed2.saveAsTextFile("/user/cloudera/sparkcore/res1")

Ex:4)emp
     dept
     val ednosal = emp.map{ x =>
                   val w = x.split(",")
                   val dno = w(4)
                   val sal = w(2).toInt
                   (dno,sal)
                   }
    ednosal.collect.foreach(println)
     val dnocity = dept.map{ x =>
                   val w = x.split(",")
                   val dno = w(0)
                   val city = w(2)
                   (dno,city)
                   }
    dnocity.collect.foreach(println)

    val edj = ednosal.join(dnocity)
    edj.collect.foreach(println)

    val result = edj.map{ x =>
                 val city = x._2._2
                 val sal = x._2._1
                 (city,sal)
                 }
    val res = result.reduceByKey(_+_)
    res.collect.foreach(println)

    //how to get no of partitions?
      r1.partitions.size
Ex:5) cat > comment
I love spark
```

```
I love hadoop
I love hadoop and spark
Spark is super speed
     hadoop fs -put comment /sparkcore
     val data = sc.textFile("/user/cloudera/sparkcore/comment")
     data.collect.foreach(println)
     val word = data.flatMap(" ")
     val pair = word.map(x => (x,1))
     val result = pair.reduceByKey(_+_)
     result.collect.foreach(println)

Ex:6)cat > comment2
spark      Spark      spark
hadoop      spark Hadoop   HADOOP
HADOOP    Hadoop

def rmSpace(x:String):String={
"   I   love      SpARk   "

val line =x.trim()
val w = line.split(" ")
val words = w.filter(x => x!="").map(x => x.toLowerCase)
words.mkString(" ")
}
val lines = sc.textFile("/user/cloudera/sparkcore/comment2")
val data = lines.map(x => rmSpace(x))
val arr = data.flatMap(x => x.split(" "))
val pair = arr.map(x => (x,1))
val result = pair.reduceByKey(_+_)
result.collect.foreach(println)

Ex:7) val data = sc.textFile("/user/cloudera/sparkcore/emp")
      data.collect
      //select dno,sex,sum(sal) from emp
       //group by dno,sex;
       val result = data.map{ x =>
                  val w = x.split(",")
                  val dno =w(4)
                  val sex = w(3)
                  val sal = w(2).toInt
                  val mkey=(dno,sex)
                  (mkey,sal)
                  }
    result.collect.foreach(println)
    val fina_res = result.reduceByKey(_+_)
    final_res.collect.foreach(println)
    //no.of unique blocks(for files) = no.of partitions(for ram)
Ex:8)
val data1= sc.textFile("/user/cloudera/sparkcore/emp",2)
//data1.partitions.size
val r1 = data1.map(x => x.split(","))
val r2 = r1.map(x => (x(3), x(2).toInt))
//r2.persist
val res1 = r2.reduceByKey(_+_)
val res2 = r2.reduceByKey(Math.max(_,_))
val res3 = r2.reduceByKey(Math.min(_,_))
res1.saveAsTextFile("/user/cloudera/sparkcore/res1")
res2.saveAsTextFile("/user/cloudera/sparkcore/res2")
res3.saveAsTextFile("/user/cloudera/sparkcore/res3")
val tres1 = res1.map(x => x._1+"\t"+x._2)
tres1.collect
tres1.saveAsTextFile("/user/cloudera/sparkcore/tres1")

Ex:9) val data = sc.textFile("/user/cloudera/sparkcore/emp")
      val arr = data.map(x => x.split(","))
      val pair1 = arr.map(x => (x(4),x(2).toInt))
      pair1.collect
      val grp = pair1.groupByKey()
      grp.collect
      val res = grp.map{ x =>
          val dno = x._1
          val cb = x._2
          val tot = cb.sum
          val cnt = cb.size
          val avg = tot/cnt
          val max = cb.max
          val min = cb.min
          val r = dno+","+tot+","+cnt+","+avg+","+max+","+min
          r
          }
          res.collect.foreach(println)




val pairs = sc.parallelize(Array(("a", 3), ("a", 1), ("b", 7),("b", 2), ("a", 5)))
    //0 is initial value, _+_ inside partition, _+_ between partitions
    val resaggregateByKey = pairs.aggregateByKey(0)(_+_,_*_)
    resaggregateByKey.collect().foreach(println)




val a = sc.parallelize(List((1,'a'),(2,'a'),(3,'a'),(4,'a'),(1,'c'),(2,'c'),(3,'c'),(4,'c')), 2)
    val b = sc.parallelize(List((1,'b'),(2,'b'),(3,'b'),(4,'b')), 2)
    a.cogroup(b).sortByKey(false).collect().foreach(println)
```

```
Ex:10)data.collect
      arr.collect
      arr.persist
      val pair2 = arr.map(x =>((x(4),x(3)),x(2).toInt))
      pair2.collect
      val grp2 = pair2.groupByKey()
      grp2.collect.foreach(println)
      val result2 = grp2.map{x =>
             val k = x._1
             val dno = k._1
             val sex = k._2
             val cb = x._2
             val tot = cb.sum
             val cnt = cb.size
             val avg = tot/cnt
             val max = cb.max
             val min = cb.min
             (dno,sex,tot,cnt,avg,max,min)
             }
    result2.collect.foreach(println)

Ex:11)select sum(sal),avg(sal),count(*),max(sal),min(sal) from emp;
      data.collect
      arr.collect
      val sal = arr.map(x =>x(2).toInt)
      sal.collect

      val tot = sal.sum
      val cnt = sal.count
      val avg:Int = (tot/cnt).toInt
      val max = sal.max
      val min = sal.min

      or
      val tot = sal.reduce(_+_)
      val cnt = sal.count
      val avg = tot/cnt
      val max = sal.reduce(Math.max(_,_))
      val min = sal.reduce(Math.min(_,_))

Ex:12)val rrd1 = sc.parallelize(List(10,20,30,40,50,60,70,80,90),2)
partition1 ---> List(10,20,30,40,50)
partition2 ---> List(60,70,80,90)
rdd1.sum// all partitions data will be collected into local,
sum executed at local(no parallel processing)
rrd1.reduce(_+_)
the above operation excuted at cluster level separately for
each partition
partition1 result =150
partition2 result =300
these independent results will be collected into any one slave of spark cluster.

List(150,300)=450

Ex:13)//select sum(sal) from emp where sex='m'
     def isMale(x:String)={
     val w = x.split(",")
     val sex = w(3).toLowerCase
     sex =="m"
     }
     val males = data.filter(x => isMale(x))
     val sals = males.map(x => x.split(",")(2).toInt)
     sals.reduce(_+_)
     //select sum(sal) from emp where sex='f'
     //select max(sal) from emp where sex='m'
     //select min(sal) from emp where sex='m'

14)scala> val parallel = sc.parallelize(1 to 9, 3)
   parallel.mapPartitionsWithIndex( (index: Int, it: Iterator[Int]) =>
 it.toList.map(x => index + ", "+x).iterator).collect


--- Spark Transformation

******************************Spark-Core***********************************
******************************Sprark-Template******************************
import org.apache.spark.sql.SparkSession
object ${NAME} {
def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("${NAME}").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
     val spark = SparkSession.builder.master("local[*]").appName("${NAME}").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
spark.stop()
}
}
*******************************************************************************

1)aggregateByKeyTransformation
------------------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */

import org.apache.spark.sql.SparkSession

object aggregateByKeyTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("aggregateByKeyTransformation"
    ).config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("aggregateByKeyTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    //Example for reduceByKey
    //    val pairs = sc.parallelize(Array(("a", 3), ("a", 1), ("b", 7), ("a", 5)))
    //    val resReduceByKey = pairs.reduceByKey(_+_)
    //    resReduceByKey.collect().foreach(println)
    //Example-1 for aggregateByKey
    val pairs = sc.parallelize(Array(("a", 3), ("a", 1), ("b", 7),("b", 2), ("a", 5)))
    //0 is initial value, _+_ inside partition, _+_ between partitions
    val resaggregateByKey = pairs.aggregateByKey(0)(_+_,_*_)
    resaggregateByKey.collect().foreach(println)
    //Example-2 for aggregateByKey
    //    import scala.collection.mutable.HashSet
    //    //the initial value is a void Set. Adding an element to a set is the first
    //    //_+_ Join two sets is the  _++_
    //    val sets = pairs.aggregateByKey(new HashSet[Int])(_+_, _++_)
    spark.stop()
  }
}

2)cartesianTransformation
--------------------------
package SparkCore

/**
```

```
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object cartesianTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("cartesianTransformation").
    config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("cartesianTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val x = sc.parallelize(List(1,2,3,4,5),2)
    val y = sc.parallelize(List('a','b','c','d','e','f'), 2)
    x.cartesian(y).collect().foreach(println)
    spark.stop()
  }
}

3)coalesceTransformation
-------------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object coalesceTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("coalesceTransformation").
    config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("coalesceTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val x = sc.parallelize(Array(1,2,3,4,5),3)
    val y = x.coalesce(2,false)
    println(y.getNumPartitions)
    spark.stop()
  }
}

4)cogroupTransformation
-----------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object cogroupTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("cogroupTransformation").
    config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("cogroupTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val a = sc.parallelize(List((1,'a'),(2,'a'),(3,'a'),(4,'a'),(1,'c'),(2,'c'),(3,'c'),(4,'c'
    )), 2)
    val b = sc.parallelize(List((1,'b'),(2,'b'),(3,'b'),(4,'b')), 2)
    a.cogroup(b).sortByKey(false).collect().foreach(println)
    spark.stop()
  }
}

5)distinctTransformation
------------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object distinctTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("distinctTransformation").
    config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("distinctTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    //example-1 with files
    System.setProperty("hadoop.home.dir", "C:\\winutils");
    val data = sc.textFile("D:\\Sprak-Training\\spark_datasets\\duplicate_data.csv")
    val result = data.distinct()
    result.collect().foreach(println)
    //example-2 using parallelize
//    val data1 = sc.parallelize(1 to 10)
//    val data2 = sc.parallelize(5 to 20)
//    val result1 = data1.union(data2).distinct().sortBy(x =>x,false)
//    result1.collect().foreach(println)
    spark.stop()
  }
}

6)filterTransformation
----------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object filterTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("filterTransformation").config
    ("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("filterTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    System.setProperty("hadoop.home.dir", "C:\\winutils");
    val dataset = sc.textFile("D:\\Sprak-Training\\spark_datasets\\sample_data.csv")
    val result =dataset.map(x=>(x.split(",")))
    val filtereddata = result.map(x=>((x(2).toInt))).filter(x=>x<25)
    filtereddata.collect().foreach(println)
    spark.stop()
  }
}

7)flatMapTransformation
------------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object flatMapTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("MapTransformation").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("MapTransformation").getOrCreate
    ()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    System.setProperty("hadoop.home.dir", "C:\\winutils");
    val dataRDD = sc.textFile("D:\\Sprak-Training\\spark_datasets\\favourite_animals.csv")
    val res = dataRDD.flatMap(x => x.split(","))
    res.collect().foreach(println)
    spark.stop()
  }
}

8)groupByKeyTransformation
--------------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object groupByKeyTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("groupByKeyTransformation").
    config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("groupByKeyTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    System.setProperty("hadoop.home.dir", "C:\\winutils");
    val data = sc.textFile("D:\\Sprak-Training\\spark_datasets\\sample_baby_names.csv")
    val header = data.first()
    val rows = data.filter(row => row != header).map(x=>x.split(","))
    val namesToCountries = rows.map(x=>(x(2),x(1)))
    namesToCountries.groupByKey().collect().foreach(println)
    spark.stop()
  }
}

9)intersectionTransformation
-----------------------------
```

```
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object intersectionTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("intersectionTransformation").
    config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("intersectionTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    System.setProperty("hadoop.home.dir", "C:\\winutils");
    val data1 = sc.parallelize(1 to 10)
    val data2 = sc.parallelize(5 to 20)
    val result = data1.intersection(data2)
    result.collect().foreach(println)
    spark.stop()
  }
}

10)joinTransformation
----------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object joinTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("joinTransformation").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("joinTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val x = sc.parallelize(Array(("a",1),("b",2)))
    val y = sc.parallelize(Array(("a",3),("a",4),("b",5)))
    //Inner Join
    val innerjoin = x.join(y)
    println("Inner Join :" + innerjoin.collect().mkString(","))
    //Left Outer Join
    val leftjoin = x.leftOuterJoin(y)
    println("Left Outer Join :" + leftjoin.collect().mkString(","))
    //Right Outer Join
    val rightjoin = x.rightOuterJoin(y)
    println("Right Outer Join :" + rightjoin.collect().mkString(","))
    //Full Outer Join
    val fulljoin = x.rightOuterJoin(y)
    println("Full Outer Join :" + fulljoin.collect().mkString(","))
    spark.stop()
  }
}

11)mapPartitionsTransformation
-------------------------------
```

```
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object mapPartitionsTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("mapPartitionsTransformation"
    ).config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("mapPartitionsTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val x = sc.parallelize(Array(1,2,3),2)
    def f(i:Iterator[Int])={(i.sum,42).productIterator}
    val y = x.mapPartitions(f)
    y.collect().foreach(println)
    spark.stop()
  }
}

12)mapPartitionsWithIndexTransformation
---------------------------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object mapPartitionsWithIndexTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName(
    "mapPartitionsWithIndexTransformation").config("spark.sql.warehouse.dir",
    "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName(
    "mapPartitionsWithIndexTransformation").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val x = sc.parallelize(Array(1,2,3),2)
    def f(partitionIndex:Int,i:Iterator[Int])={
      (partitionIndex,i.sum).productIterator
    }
    val y = x.mapPartitionsWithIndex(f)
    y.collect().foreach(println)
    spark.stop()
  }
}

13)mapTransformation
--------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object mapTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("mapTransformation").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("mapTransformation").getOrCreate
    ()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    System.setProperty("hadoop.home.dir", "C:\\winutils");
    val dataRDD = sc.textFile("D:\\Sprak-Training\\spark_datasets\\animals.csv")
    val res = dataRDD.flatMap(x => x.split(","))
    res.collect().foreach(println)
    spark.stop()
  }
}

14)reduceByKeyTransformation
----------------------------

package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object reduceByKeyTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("reduceByKeyTransformation").
    config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("reduceByKeyTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    System.setProperty("hadoop.home.dir", "C:\\winutils")
    val data = sc.textFile("D:\\Sprak-Training\\spark_datasets\\sample_baby_names.csv")
    val header = data.first()
    val rows = data.filter(row => row != header).map(x=>x.split(","))
    val names = rows.map(x=>(x(1),x(4).toInt)).reduceByKey((v1,v2) => v1+v2)
    names.collect().foreach(println)
    spark.stop()
  }
}

15)repartitionAndSortWithinPartitionsTransformation
---------------------------------------------------

package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object repartitionAndSortWithinPartitionsTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName(
    "repartitionAndSortWithinPartitionsTransformation").config("spark.sql.warehouse.dir",
    "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName(
    "repartitionAndSortWithinPartitionsTransformation").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql


    spark.stop()
  }
}

16)repartitionTransformation
-----------------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object repartitionTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("repartitionTransformation").
    config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("repartitionTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val x = sc.parallelize(Array(1,2,3,4,5),3)
    val y = x.repartition(2)
    println(y.getNumPartitions)
    spark.stop()
  }
}

17)sampleTransformation
-----------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object sampleTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("sampleTransformation").config
    ("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("sampleTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    System.setProperty("hadoop.home.dir", "C:\\winutils");
    val dataset = sc.textFile("D:\\Sprak-Training\\spark_datasets\\sample_data.csv")
    val result = dataset.sample(true,0.5)//sample(withReplacement: Boolean, fraction: Double,
    seed: Long = Utils.random.nextLong)
    result.collect().foreach(println)
    spark.stop()
  }
}

18)sortByKeyTransformation
---------------------------
```

```
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object sortByKeyTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("sortByKeyTransformation").
    config("spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().
    getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("sortByKeyTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    System.setProperty("hadoop.home.dir", "C:\\winutils")
    val data = sc.textFile("D:\\Sprak-Training\\spark_datasets\\sample_baby_names.csv")
    val header = data.first()
    val rows = data.filter(row => row != header).map(x => x.split(","))
    val names = rows.map(x => (x(1), x(4).toInt)).sortByKey(false) //false-descending order,true
    -Ascending order(default)
    names.collect().foreach(println)
    spark.stop()
  }
}

19)unionTransformation
-----------------------
package SparkCore

/**
  * Created by Kalyan on 5/26/2017.
  */
import org.apache.spark.sql.SparkSession

object unionTransformation {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("unionTransformation").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("unionTransformation").
    getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val data1 = sc.parallelize(1 to 10)
    val data2 = sc.parallelize(5 to 20)
    val result = data1.union(data2)
    result.collect().foreach(println)
    spark.stop()
  }
}


----- Spark Submit Commands

./bin/spark-submit --class org.apache.spark.examples.SparkPi \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 4g \
    --executor-memory 2g \
    --executor-cores 4 \
    --queue default \
    lib/spark-examples*.jar


The key configurations to run a Spark job on a YARN cluster are:

master – Determines how to run the job.
As we want for this blog review to execute Spark in YARN, the ‘yarn’ value
has been selected for the example above.  The other options available include
‘mesos’ and ‘standalone’.

deploy-mode – We selected ‘cluster’ to run the above SparkPi example within the cluster.
To run the problem outside of the cluster, then select the ‘client’ option.

driver-memory – The amount memory available for the driver process. In a YARN cluster
Spark configuration the Application Master runs the driver.

executor-memory – The amount of memory allocated to the executor process.

executor-cores – the total number of cores allocated to the executor process

queue – The YARN queue name on which this job will run.
If you have not already defined queues to your cluster,
it is best to utilize the ‘default’ queue.


Broadcast Variable & Accumulators
---------------------------------
1)Broadcast Variables
---------------------

As part of this topic we will see details about Broadcast Variables

At times we need to pass (broadcast) some information to all the executors
It can be done by using broadcast variables
A broadcast variable can be of preliminary type or it could be a hash map
Here are few examples
   Single value – Common discount percent for all the products
   Hash map – look up or map side join
When very large data set (fact) is tried to join with smaller data set (dimension), broadcasting
 dimension can have considerable performance improvement.
Broadcast variables have to be immutable.



Define problem statement and design the application
---------------------------------------------------
Get total revenue per department for each day

Again we will be using our retail_db database
Please refer for data model
Department name is in departments
To get department name we need to join these tables
order_items
products
categories
departments
We will first join products, categories and departments
get product_id and department_name
broadcast data which contain product_id and department_name
Then join orders and order_items
Perform simple join between orders and order_items
As part of join look up into hashmap with product_id to get department_name
Use reduceByKey to compute revenue for each date and department

Step-1)//Read products, categories and departments
```

```
       val departments = sc.textFile(inputPath + "/departments")
       val categories = sc.textFile(inputPath + "/categories")
       val products = sc.textFile(inputPath + "/products")

Step-2)//Join products, categories and departments
      val departmentsMap = departments.map(rec => (rec.split(",")(0).toInt, rec.split(",")(1)))
      val categoriesMap = categories.map(rec => (rec.split(",")(0).toInt, rec.split(",")(1).toInt))
      val productsMap = products.map(rec => (rec.split(",")(1).toInt, rec.split(",")(0).toInt))
      val productCategories = productsMap.join(categoriesMap)
      val productCategoriesMap = productCategories.map(rec => (rec._2._2, rec._2._1))
      val productDepartments = productCategoriesMap.join(departmentsMap)

Step-3)//Build hash map and create broadcast variable
      val productDepartmentsMap = productDepartments.map(rec => (rec._2._1, rec._2._2)).distinct
      val bv = sc.broadcast(productDepartmentsMap.collectAsMap())

Step-4)//Read orders and order_items
      val orders = sc.textFile(inputPath + "/orders")
      val orderItems = sc.textFile(inputPath + "/order_items")

Step-5)//Filter for completed orders and extract required fields from orders
      val ordersCompleted = orders.filter(rec => (rec.split(",")(3) == "COMPLETE")).map(rec => (rec.split(",")(0).toInt, rec.split(",")(1)))

Step-6)//Extract required fields from order_items while looking up into hash map to get department name
      val orderItemsMap = orderItems.map(rec =>
      (rec.split(",")(1).toInt, (bv.value.get(rec.split(",")(2).toInt).get,rec.split(",")(4).toFloat)))

Step-7)//Join orders and order_items and compute daily revenue for each product
      val ordersJoin = ordersCompleted.join(orderItemsMap)
      val revenuePerDayPerDepartment = ordersJoin.map(rec => ((rec._2._1, rec._2._2._1), rec._2._2._2)).reduceByKey((acc, value) => acc + value)

Step-8)//Save output to file system of your choice
       revenuePerDayPerDepartment.sortByKey().saveAsTextFile(outputPath)

Accumulators
------------
In this topic we will see details about accumulators

It is important to perform some counts as part of application for
  unit testing
  data quality
These counters cannot be global variables as part of the program
Instead we need to use accumulator which will be managed by spark
Accumulators will be passed to executors and scope is managed across all the executors or executor tasks
We will see

how accumulators are implemented
issues with accumulators


Implementation of accumulators
------------------------------


Take our program AvgRevenueDaily
Add as many accumulators as you desire

Create variable like val ordersCompletedAccum = sc.accumulator(0, "ordersCompleted count")
Update the appropriate Spark API RDD function to increment accumulator (see the sample code below)
Compile the program
```

```
Build jar and ship it to the remote cluster
Run on remote cluster
Open spark history server and review the appropriate executor task in which accumulators are
implemented
You will see counter as part of history server

Step-1) val ordersRDD = sc.textFile(inputPath + "/orders")
        val orderItemsRDD = sc.textFile(inputPath + "/order_items")

Step-2)val ordersCompletedAccum = sc.accumulator(0, "ordersCompleted count")
       val ordersFilterInvokedAccum = sc.accumulator(0, "orders filter invoked count")

Step-3)val ordersCompleted = ordersRDD.
          filter(rec => {
          ordersFilterInvokedAccum += 1
          if (rec.split(",")(3) == "COMPLETE") {
          ordersCompletedAccum += 1
        }
        rec.split(",")(3) == "COMPLETE"
      })

      val ordersAccum = sc.accumulator(0, "orders count")
      val orders = ordersCompleted.
      map(rec => {
        ordersAccum += 1
        (rec.split(",")(0).toInt, rec.split(",")(1))
      })

      val orderItemsMapAccum = sc.accumulator(0, "orderItemsMap count")
      val orderItemsMap = orderItemsRDD.
      map(rec => {
        orderItemsMapAccum += 1
        (rec.split(",")(1).toInt, rec.split(",")(4).toFloat)
      })

      val orderItemsValuesAccum = sc.accumulator(0, "reduceByKey values count")
      val orderItems = orderItemsMap.
      reduceByKey((acc, value) => {
        orderItemsValuesAccum += 1
        acc + value
      })

Step-4)val ordersJoin = orders.join(orderItems)
       val ordersJoinMap = ordersJoin.map(rec => (rec._2._1, rec._2._2))

Step-5)val revenuePerDay = ordersJoinMap.aggregateByKey((0.0, 0))(
      (acc, value) => (acc._1 + value, acc._2 + 1),
      (total1, total2) => (total1._1 + total2._1, total1._2 + total2._2))

      val averageRevenuePerDay = revenuePerDay.
      map(rec => (rec._1, BigDecimal(rec._2._1 / rec._2._2).
        setScale(2, BigDecimal.RoundingMode.HALF_UP).toFloat))

      val averageRevenuePerDaySorted = averageRevenuePerDay.
      sortByKey()

      averageRevenuePerDaySorted.
      map(rec => rec._1 + "," + rec._2).
      saveAsTextFile(outputPath)

Cache()
-------
ex: rdd.cache() //default StorageLevel is MEMORY_LEVEL
ex: rdd.uncache()
```

Persist()
----

ex : rdd.persist(StorageLevel.<type>)

1)MEMORY_ONLY
In this storage level, RDD is stored as deserialized Java object in the JVM. If the size of RDD is greater than memory, It will not cache some partition and recompute them next time whenever needed. In this level the space used for storage is very high, the CPU computation time is low, the data is stored in-memory. It does not make use of the disk.

2)MEMORY_AND_DISK
In this level, RDD is stored as deserialized Java object in the JVM. When the size of RDD is greater than the size of memory, it stores the excess partition on the disk, and retrieve from disk whenever required. In this level the space used for storage is high, the CPU computation time is medium, it makes use of both in-memory and on disk storage.

3)MEMORY_ONLY_SER
This level of Spark store the RDD as serialized Java object (one-byte array per partition). It is more space efficient as compared to deserialized objects, especially when it uses fast serializer. But it increases the overhead on CPU. In this level the storage space is low, the CPU computation time is high and the data is stored in-memory. It does not make use of the disk.

4)MEMORY_AND_DISK_SER
It is similar to MEMORY_ONLY_SER, but it drops the partition that does not fits into memory to disk, rather than recomputing each time it is needed. In this storage level, The space used for storage is low, the CPU computation time is high, it makes use of both in-memory and on disk storage.

5) DISK_ONLY
In this storage level, RDD is stored only on disk. The space used for storage is low, the CPU computation time is high and it makes use of on disk storage.

ex : rdd.unpersist()

---- Spark SQL

```
import org.apache.spark.sql.SparkSession

object csvExample_1 {
  case class sfpd(auctionid: String, bid: Float, bidtime: Float, bidder: String, bidderrate:
  Integer, openbid: Float, price: Float, item: String, daystolive: Integer)
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("csvExample_1").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("csvExample_1").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    val df = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option(
    "inferschema","true").load(
    "C:\\Kalyan\\POC\\Spark\\spark_datasets\\Police_Department_Incidents.csv")
    df.show()
    df.select("Category").distinct.collect().foreach(println)
    df.createOrReplaceTempView("sfpd")
    sqlContext.sql("select Category from sfpd").collect().foreach(println)
// top 10 results
```

```
    sqlContext.sql("SELECT Resolution , count(Resolution) as rescount FROM sfpd group by
    Resolution order by rescount desc limit 10").collect().foreach(println)
    val t = sqlContext.sql("select Category,count(Category) as catcount from sfpd group by
    Category order by catcount desc limit 10")
    t.show()
    t.map(t=> "column 0: "+ t(0)).collect().foreach(println)
    spark.stop()
  }
}

==================================================================================================
=============================================================
package SparkSQL.SparkSQL_CSV

/**
  * Created by Kalyan on 4/20/2017.
  */

import org.apache.spark.sql.SparkSession

object csvExample_2 {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("csvExample_2").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("csvExample_2").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    val df = sqlContext.read.format("com.databricks.spark.csv").option("header","false").option(
    "inferschema","true")
      .load("C:\\Kalyan\\POC\\Spark\\spark_datasets\\ebay.csv")
    df.show()
    df.select($"_c0".alias("auctionid"),$"_c1".alias("bid"),$"_c2".alias("bidtime"),$"_c3".alias
    ("bidder"),$"_c4".alias("bidderrate")
    ,$"_c5".alias("openbid"),$"_c6".alias("price"),$"_c7".alias("item"),$"_c8".alias(
    "daystolive")).show()
//    sqlContext.sql("select")
    spark.stop()
  }
}
==================================================================================================
==============================================================
package SparkSQL.SparkSQL_CSV

/**
  * Created by Kalyan on 4/20/2017.
  */

import org.apache.spark.sql.SparkSession

object csvExample_3 {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("csvExample_3").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("csvExample_3").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    val df = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option(
    "inferschema","true")
      .load("C:\\Kalyan\\POC\\Spark\\spark_datasets\\cars.csv")
    df.show()
    spark.stop()
  }
}
==================================================================================================
============================================================
package SparkSQL.SparkSQL_CSV

/**
  * Created by Kalyan on 4/20/2017.
  */

import org.apache.spark.sql.SparkSession

object csvExample_4 {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("csvExample_4").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("csvExample_4").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    val df = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option(
    "infereschema","true")
      .load(
      "C:\\Kalyan\\POC\\Spark\\spark_datasets\\csv_data\\spark-csv-master\\src\\test\\resources\\
      ages-alternative-malformed.csv")
    df.show()
    spark.stop()
  }
}
=================================================================================================
============================================================
package SparkSQL.SparkSQL_CSV

/**
  * Created by Kalyan on 4/28/2017.
  */

import org.apache.spark.sql.SparkSession

object csvExample_5 {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("csvExample_5").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("csvExample_5").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val df = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option(
    "inferschema","true").load("C:\\Users\\Kalyan\\Desktop\\sampledata.csv")
    df.show()
    spark.stop()
  }
}
=================================================================================================
============================================================
package SparkSQL.SparkSQL_JSON

/**
  * Created by Kalyan on 4/20/2017.
  */

import org.apache.spark.sql.SparkSession

object jsonExample_1 {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("jsonExample_1").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("jsonExample_1").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    val df = sqlContext.read.json(
    "C:\\Kalyan\\POC\\Spark\\spark_datasets\\json_data\\world_bank.json")
//    df.printSchema()
    df.createOrReplaceTempView("jsondata_one")
    sqlContext.sql("select url,totalamt,abc.code,abc.name from jsondata_one " +
      "lateral view explode(theme_namecode) as abc").show(5)
//    result.write.format("com.databricks.spark.csv").option("header","true").save(
"C:\\Kalyan\\POC\\Spark\\spark_datasets\\json_data\\jsontocsv")
    spark.stop()
  }
}

================================================================================================
===========================================================
package SparkSQL.SparkSQL_XML

/**
  * Created by Kalyan on 4/20/2017.
  */

import org.apache.spark.sql.SparkSession

object xmlExample_1 {
  def main(args: Array[String]) {
    //val spark = SparkSession.builder.master("local[*]").appName("jsonExample_1").config(
    "spark.sql.warehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark = SparkSession.builder.master("local[*]").appName("jsonExample_1").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    val df = sqlContext.read.format("com.databricks.spark.xml").option("rootTag","books").option
    ("rowTag","book")
    .load("C:\\Kalyan\\POC\\Spark\\spark_datasets\\books.xml")
    df.show()
//    df.printSchema()
    //df.createOrReplaceTempView("jsondata_one")
    //sqlContext.sql("select url,totalamt,abc.code,abc.name from jsondata_one " +
      //"lateral view explode(theme_namecode) as abc").show(5)
//    result.write.format("com.databricks.spark.csv").option("header","true").save(
"C:\\Kalyan\\POC\\Spark\\spark_datasets\\json_data\\jsontocsv")
    spark.stop()
  }
}
================================================================================================
=====================================================




Spark SQL
========================

Example-1
=========
step 1)import org.apache.spark.sql.SQLContext
       val sqlContext = new SQLContext(sc)
       import sqlContext.implicits._//to convert RDD's into DataFrames implicitly

step 2)case class Sample(a:Int,b:Int,c:Int)
       val s1 = Sample(10,20,30)
       val s2 = Sample(1,2,3)
       val s3 = sample(100,200,300)
       val s4 = Sample(1000,2000,3000)

       val data = sc.parallelize(List(s1,s2,s3,s4))
       data.collect
       val x = data.map(v => v.a+v.b+v.c).collect

step 3) val df = data.toDF //if your rdd having schema its eligible for DataFrame.
        df.show()

step 4) df.registerTempTable("sample")

step 5) val result = sqlContext.sql("select * from sample)
        val result1 = sqlContext.sql("select a,b from sample")
        result.show
        result.printSchema()
        result1.show()
        val result2 = sqlContext.sql("select a,b,c,a+b+c as total from sample")
        result2.show()


Example-2
=========
step 1) creating a file locally emp
        101,aaa,10000,m,11
        102,bbb,20000,f,12
        103,ccc,30000,m,13
        104,ddd,40000,f,11
        105,eee,50000,m,12
        106,fff,60000,f,13

step 2) creating a directory in hdfs
        hdfs dfs -mkdir /user/cloudera/spark_sql/

        hdfs dfs -put emp /user/cloudera/spark_sql/

step 3) loading the data into spark
        val data = sc.textFile("/user/cloudera/spark_sql/emp")
        data.count
        data.take(1)

step 4) case class Emp(id:Int,name:String,sal:Int,gendar:String,dno:Int)

step 5) def toEmp(x:String)={
            val w = x.trim().split(",")
            val id = w(0).toInt
            val name = w(1)
            val sal = w(2).toInt
            val gendar = w(3)
            val dno = w(4).toInt
            val empdetails = Emp(id,name,sal,gendar,dno)
            empdetails
            }

                        //To check the function is working or not(toEmp)
            val rec = "101,kalyan,70000,m,12"
            val i = toEmp(rec)
            i.name
            i.sal

step 6) val einfo = data.map(x => toEmp(x)) //its RDD and having schema
            einfo.foreach(println)

step 7) convert einfo to DF.
        val edf = einfo.toDF
        edf.show()
        edf.printSchema()

step 8) create temp table
        edf.registerTempTable("edf")
```

```
val result = sqlContext.sql("select dno,gendar,sum(sal)
                              from edf group by dno,gendar")
result.show()


step 9) //RDD style
        einfo.filter(x => x.gendar == "m").collect
        //sql style
        val result = sqlContext.sql("select * from edf where gendar='m'")
        result.show()

        //select gendar,sum(sal) from emp
        //RDD style
        val pair = einfo.map(x=>(x.gendar,x.sal))
        val result1 = pair.reduceByKey(_+_)
        result1.collect
        //sql style

        val result2 = sqlContext.sql("select gendar,sum(sal) as total_sal from edf group by
        gendar").show


Example 3)
=========
//RDD style of multiple aggregations
val pair = einfo.map(x=>(x.sex,x.sal))
val grp = pair.groupByKey()
val res = grp.map{x =>
           val gendar = x._1
           val cb = x._2
           val tot = cb.sum
           val cnt = cb.size
           val avg = tot/cnt
           val max = cb.max
           val min = cb.min
           (sex,tot,cnt,avg,max,min)
           }

//sql style
val result = sqlContext.sql("select gendar,sum(sal) as tot,count(*) as cnt,avg(sal),max(Sal) as
max,min(sal) as min from edf group by gendar").show()
//sql statement always returns DF.

val result1 = sqlContext.sql("select dno,gendar,sum(sal) as tot,count(*) as
cnt,avg(sal),max(Sal) as max,min(sal) as min from edf group by gendar").show()


Example 4)//create a file emp2
201,kiran,14,m,90000
202,kalyani,12,f,100000
203,anushka,11,f,80000
204,kalyan,13,m,70000

step 1) hdfs dfs -put emp2 /user/cloudera/spark_sql/

step 2) val data2 = sc.textFile("/user/cloudera/spark_sql/emp2")

step 3) val emp2 = data2.map{x=>
                 val w = x.split(",")
                 Emp(w(0).toInt,w(1),w(4).toInt,w(3),w(2).toInt)
                 }
        emp2.collect().foreach(println)

step 4) val edf2 = emp2.toDF
step 5) edf2.registerTempTable("edf2")

step 6) val result = sqlContext.sql("select * from edf union all select * from edf2").show()
            result.registerTempTable("emp_result")
            val result1 =
sqlContext.sql("select gendar,sum(sal) from emp_result group by gendar").show()


Example 5)//create dept file
11,marketing,hyd
12,hr,bang,
13,finance,hyd
14,sales,pune

step 1) hdfs dfs -put dept /user/cloudera/spark_sql/

step 2) val ddata = sc.textFile("/user/cloudera/spark_sql/dept")

step 3)case class Dept(dno:Int,dname:String,loc:String)

step 4)val dept = ddata.map{x=>
                 val w = x.split(",")
                 val dno = w(0).int
                 val dname =w(1)
                 val loc = w(2)
                 Dept(dno,dname,loc)
                 }
                 dept.collect().foreach(println)

step 5) val ddf = dept.toDF
step 6) ddf.registerTempTable("dept")
step 7) val result = sqlContext.sql("select loc,sum(sal) as tot  from edf e join dept d
on(e.dno= d.dno) group by loc").show()


====================using spark sql connect to
hive=================================================================================
Example 1)
step 1) import org.apache.spark.sql.hive.HiveContext

step 2) val hc = new HiveContext(sc)

step 3) val dbname = hc.sql("create database mysparkdb")

step 4)val usedbname = hc.sql("use mysparkdb")

step 5) val table = hc.sql("create table emp(eno int,name string,sal int,gendar string,dno int)
row format delimited fields terminated by ',')")

step 6) val load= hc.sql("load data inpath "/user/cloudera/SparkSql/emp"
 into table emp")

step 7) val result = hc.sql("select * from emp").show()


Spark SQL
========================

Example-1
=========
step 1)import org.apache.spark.sql.SQLContext
       val sqlContext = new SQLContext(sc)
       import sqlContext.implicits._//to convert RDD's into DataFrames implicitly

step 2)case class Sample(a:Int,b:Int,c:Int)
       val s1 = Sample(10,20,30)
       val s2 = Sample(1,2,3)
       val s3 = sample(100,200,300)
       val s4 = Sample(1000,2000,3000)

       val data = sc.parallelize(List(s1,s2,s3,s4))

       data.collect
       val x = data.map(v => v.a+v.b+v.c).collect

step 3) val df = data.toDF //if your rdd having schema its eligible for DataFrame.
        df.show()

step 4) df.registerTempTable("sample")

step 5) val result = sqlContext.sql("select * from sample)
        val result1 = sqlContext.sql("select a,b from sample")
        result.show
        result.printSchema()
        result1.show()
        val result2 = sqlContext.sql("select a,b,c,a+b+c as total from sample")
        result2.show()


Example-2
=========
step 1) creating a file locally emp
        101,aaa,10000,m,11
        102,bbb,20000,f,12
        103,ccc,30000,m,13
        104,ddd,40000,f,11
        105,eee,50000,m,12
        106,fff,60000,f,13

step 2) creating a directory in hdfs
        hdfs dfs -mkdir /user/cloudera/spark_sql/

        hdfs dfs -put emp /user/cloudera/spark_sql/

step 3) loading the data into spark
        val data = sc.textFile("/user/cloudera/spark_sql/emp")
        data.count
        data.take(1)

step 4) case class Emp(id:Int,name:String,sal:Int,gendar:String,dno:Int)

step 5) def toEmp(x:String)={
            val w = x.trim().split(",")
            val id = w(0).toInt
            val name = w(1)
            val sal = w(2).toInt
            val gendar = w(3)
            val dno = w(4).toInt
            val empdetails = Emp(id,name,sal,gendar,dno)
            empdetails
            }

                        //To check the function is working or not(toEmp)
            val rec = "101,kalyan,70000,m,12"
            val i = toEmp(rec)
            i.name
            i.sal

step 6) val einfo = data.map(x => toEmp(x)) //its RDD and having schema
            einfo.foreach(println)

step 7) convert einfo to DF.
```

```
        val edf = einfo.toDF
        edf.show()
        edf.printSchema()

step 8) create temp table
        edf.registerTempTable("edf")

val result = sqlContext.sql("select dno,gendar,sum(sal)
                              from edf group by dno,gendar")
result.show()


step 9) //RDD style
        einfo.filter(x => x.gendar == "m").collect
        //sql style
        val result = sqlContext.sql("select * from edf where gendar='m'")
        result.show()

        //select gendar,sum(sal) from emp
        //RDD style
        val pair = einfo.map(x=>(x.gendar,x.sal))
        val result1 = pair.reduceByKey(_+_)
        result1.collect
        //sql style

        val result2 = sqlContext.sql("select gendar,sum(sal) as total_sal from edf group by
        gendar").show


Example 3)
=========
//RDD style of multiple aggregations
val pair = einfo.map(x=>(x.sex,x.sal))
val grp = pair.groupByKey()
val res = grp.map{x =>
           val gendar = x._1
           val cb = x._2
           val tot = cb.sum
           val cnt = cb.size
           val avg = tot/cnt
           val max = cb.max
           val min = cb.min
           (sex,tot,cnt,avg,max,min)
           }

//sql style
val result = sqlContext.sql("select gendar,sum(sal) as tot,count(*) as cnt,avg(sal),max(Sal) as
max,min(sal) as min from edf group by gendar").show()
//sql statement always returns DF.

val result1 = sqlContext.sql("select dno,gendar,sum(sal) as tot,count(*) as
cnt,avg(sal),max(Sal) as max,min(sal) as min from edf group by gendar").show()


Example 4)//create a file emp2
201,kiran,14,m,90000
202,kalyani,12,f,100000
203,anushka,11,f,80000
204,kalyan,13,m,70000

step 1) hdfs dfs -put emp2 /user/cloudera/spark_sql/

step 2) val data2 = sc.textFile("/user/cloudera/spark_sql/emp2")

step 3) val emp2 = data2.map{x=>
                 val w = x.split(",")
                 Emp(w(0).toInt,w(1),w(4).toInt,w(3),w(2).toInt)
                 }
        emp2.collect().foreach(println)

step 4) val edf2 = emp2.toDF

step 5) edf2.registerTempTable("edf2")

step 6) val result = sqlContext.sql("select * from edf union all select * from edf2").show()
            result.registerTempTable("emp_result")
            val result1 =
sqlContext.sql("select gendar,sum(sal) from emp_result group by gendar").show()


Example 5)//create dept file
11,marketing,hyd
12,hr,bang,
13,finance,hyd
14,sales,pune

step 1) hdfs dfs -put dept /user/cloudera/spark_sql/

step 2) val ddata = sc.textFile("/user/cloudera/spark_sql/dept")

step 3)case class Dept(dno:Int,dname:String,loc:String)

step 4)val dept = ddata.map{x=>
                 val w = x.split(",")
                 val dno = w(0).int
                 val dname =w(1)
                 val loc = w(2)
                 Dept(dno,dname,loc)
                 }
                 dept.collect().foreach(println)

step 5) val ddf = dept.toDF
step 6) ddf.registerTempTable("dept")
step 7) val result = sqlContext.sql("select loc,sum(sal) as tot  from edf e join dept d
on(e.dno= d.dno) group by loc").show()


====================using spark sql connect to
hive===================================================================================
Example 1)
step 1) import org.apache.spark.sql.hive.HiveContext

step 2) val hc = new HiveContext(sc)

step 3) val dbname = hc.sql("create database mysparkdb")

step 4)val usedbname = hc.sql("use mysparkdb")

step 5) val table = hc.sql("create table emp(eno int,name string,sal int,gendar string,dno int)
row format delimited fields terminated by ','")

step 6) val load= hc.sql("load data inpath "/user/cloudera/SparkSql/emp"
 into table emp")

step 7) val result = hc.sql("select * from emp").show()

Example 2)
{"name":"kalyan","age":33}
{"name":"aparna","city":"hyd"}
{"name":"Akhil","age":5,"city":"nrt"}
{"name":"Akshaya","age":2}
```

```
step 3) create table jraw(line string);

step 4) load data local inpath 'sample_json' into table jraw;

step 5) select get_json_object(line,'$.name') from jraw;

        select get_json_object(line,'$.name'),
get_json_object(line,'$.age'),get_json_object(line,'$.city') from jraw;

step 6)select x.* from jraw
       lateral view json_tuple(line,'name','age','city') x as n,a,c;
```

```
       //By using Spark SQL
step 1) hadoop fs -put sample_json /user/cloudera/spark_sql/

step 2) val df = sqlContext.read.json("/user/cloudera/Spark_Sql/sample_json")

step 3)df.show()



Example 3)sample_json2
{"name":"Kalyan","age":33,"wife":{"name":"aparna","age":28,"city":"hyd"},
"city":"hyd"}
{"name":"anil","age":32,"wife":{"name":"praveena","qual":"bsc","city":"cpt"},
"city":"nrt"}

//hive
create table jraw2(line string)
load data inpath '/user/cloudera/Spark_Sql/sample_json2' into table jraw2;

create table json2(name string,age int,wife string,city string);

insert into table json2
select x.* from jraw2
lateral view json_tuple(line,'name','age','wife','city') x as n,a,w,c;

select * from json2;

select name,get_json_object(wife,'$.name'),age,get_json_object(wife,'$.age'),
get_json_object(wife,'$.qual'),city,get_json_object(wife,'$.city') from json2;
```

```
//using Spark Sql

val data = sqlContext.read.json("/user/cloudera/Spark_Sql/sample_json2")
data.show()
data.collect.map(x=> x(3))
data.collect.map(x=> x(3)(1))
data.collect.map(x=> x(3)(2))


Example 4)Handling xml files using hive and spark sql
<rec><name>Kalyan</name><age>33</age></rec>
<rec><name>Aparna</name><gendar>F</gendar></rec>
<rec><name>Anil></name><age>33</age><gendar>M</gendar></rec>


hc.sql("use mysparkdb")
hc.sql("create table xraw(line string)")
hc.sql("load data inpath 'xml1' into table xraw")
hc.sql("create table xinfo(name string,age int,gendar string)
row format delimited fields terminated by ','")
hc.sql("insert into table xinfo
        select xpath_string(line,'rec/name'),xpath_int(line,'rec/age'),
xpath_string(line,'rec/gendar') from xraw")


--- Spark To MYSQL

mysql to spark
--------------
package SparkSQL.RDS
/**
  * Created by Kalyan on 5/31/2017.
  */
import org.apache.spark.sql.SparkSession
object mysqlConnect {
  def main(args: Array[String]) {
    //val spark =
    SparkSession.builder.master("local[*]").appName("coalesceTransformation").config("spark.sql.w
    arehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark =
    SparkSession.builder.master("local[*]").appName("coalesceTransformation").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import java.util.Properties
    val mtable = "emp"
    val hostname="jdbc:mysql://mysql.c5ghyblpskst.ap-south-1.rds.amazonaws.com:3306/mysqldb"
    val mprp = new Properties()
    mprp.setProperty("driver","com.mysql.jdbc.Driver")
    mprp.setProperty("user","musername")
    mprp.setProperty("password","mpassword")
    val odf = spark.read.jdbc(hostname,mtable,mprp)
    odf.show()
    spark.stop()
  }
}
```

```
-------------------------------------
oracle to spark
---------------
package SparkSQL.RDS

/**
  * Created by Kalyan on 5/31/2017.
  */

import org.apache.spark.sql.SparkSession

object oracleConnect {
  def main(args: Array[String]) {
    //val spark =
    SparkSession.builder.master("local[*]").appName("coalesceTransformation").config("spark.sql.w
    arehouse.dir", "/home/hadoop/work/warehouse").enableHiveSupport().getOrCreate()
    val spark =
    SparkSession.builder.master("local[*]").appName("coalesceTransformation").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    import java.util.Properties
    val otable = "dept"
    val hostname="jdbc:oracle:
    thin://@oracle.c5ghyblpskst.ap-south-1.rds.amazonaws.com:1521/oracledb"
    val oprp = new Properties()
    oprp.setProperty("driver","oracle.jdbc.OracleDriver")
    oprp.setProperty("user","ousername")
    oprp.setProperty("password","opassword")

    val odf = spark.read.jdbc(hostname,otable,oprp)
    odf.show()
//    val result = odf.where("DEPTNO>20")
//    result.write.format("csv").option("header","true").save("D:\\Sprak-Training\\oracleoutput")

    spark.stop()
  }
}

---- DATA SETS IN SPARK

ex 1)
val dataset = Seq(1, 2, 3).toDS()
dataset.show()

ex 2)
case class Person(name: String, age: Int)

val personDS = Seq(Person("Max", 33), Person("Adam", 32), Person("Muller", 62)).toDS()
personDS.show()

ex 3)
val rdd = sc.parallelize(Seq((1, "Spark"), (2, "Databricks")))
val integerDS = rdd.toDS()
integerDS.show()

ex 4)
case class Company(name: String, foundingYear: Int, numEmployees: Int)
val inputSeq = Seq(Company("ABC", 1998, 310), Company("XYZ", 1983, 904), Company("NOP", 2005,
83))
val df = sc.parallelize(inputSeq).toDF()

val companyDS = df.as[Company]
companyDS.show()
```

```
ex 5)
val rdd = sc.parallelize(Seq((1, "Spark"), (2, "Databricks"), (3, "Notebook")))
val df = rdd.toDF("Id", "Name")

val dataset = df.as[(Int, String)]
dataset.show()

ex 6)
val wordsDataset = sc.parallelize(Seq("Spark I am your father", "May the spark be with you",
"Spark I am your father")).toDS()
val groupedDataset = wordsDataset.flatMap(_.toLowerCase.split(" "))
                                 .filter(_ != "")
                                 .groupBy("value")
val countsDataset = groupedDataset.count()
countsDataset.show()

ex 7)
case class Employee(name: String, age: Int, departmentId: Int, salary: Double)
case class Department(id: Int, name: String)

case class Record(name: String, age: Int, salary: Double, departmentId: Int, departmentName:
String)
case class ResultSet(departmentId: Int, departmentName: String, avgSalary: Double)

val employeeDataSet1 = sc.parallelize(Seq(Employee("Max", 22, 1, 100000.0), Employee("Adam",
33, 2, 93000.0), Employee("Eve", 35, 2, 89999.0), Employee("Muller", 39, 3, 120000.0))).toDS()
val employeeDataSet2 = sc.parallelize(Seq(Employee("John", 26, 1, 990000.0), Employee("Joe",
38, 3, 115000.0))).toDS()
val departmentDataSet = sc.parallelize(Seq(Department(1, "Engineering"), Department(2,
"Marketing"), Department(3, "Sales"))).toDS()

val employeeDataset = employeeDataSet1.union(employeeDataSet2)

def averageSalary(key: (Int, String), iterator: Iterator[Record]): ResultSet = {
   val (total, count) = iterator.foldLeft(0.0, 0.0) {
       case ((total, count), x) => (total + x.salary, count + 1)
   }
   ResultSet(key._1, key._2, total/count)
}

val averageSalaryDataset = employeeDataset.joinWith(departmentDataSet, $"departmentId" ===
$"id", "inner")
                                            .map(record => Record(record._1.name, record._1.age,
                                            record._1.salary, record._1.departmentId,
                                            record._2.name))
                                            .filter(record => record.age > 25)
                                            .groupBy($"departmentId", $"departmentName")
                                            .avg()

ex 8)
import org.apache.spark.sql.functions._

val wordsDataset = sc.parallelize(Seq("Spark I am your father", "May the spark be with you",
"Spark I am your father")).toDS()
val result = wordsDataset
               .flatMap(_.split(" "))                // Split on whitespace
               .filter(_ != "")                      // Filter empty words
               .map(_.toLowerCase())
               .toDF()                               // Convert to DataFrame to perform
               aggregation / sorting
               .groupBy($"value")                    // Count number of occurences of each word
               .agg(count("*") as "numOccurances")
               .orderBy($"numOccurances" desc)       // Show most common words first
result.show()

--- Spark Streaming
```

```
import java.util.Properties;
import java.sql.*;
import kafka.javaapi.producer.Producer;
import kafka.producer.KeyedMessage;
import kafka.producer.ProducerConfig;

public class JdbcProducer{
public static void main(String[] args) throws ClassNotFoundException,SQLException{
      Properties  props = new Properties();
      props.put("zk.connect","localhost:2181");
      props.put("serializer.class","kafka.serializer.StringEncoder");
      props.put("metadata.broker.list","localhost:9092");
      ProducerConfig config = new ProducerConfig(props);
      Producer producer = new Producer(config);
      try{
           class.forName("com.mysql.jdbc.Driver")
           Connection con = DriverManager.getConnection(
           "jdbc:mysql://localhost:3306/test","root","root");
           Statement stmt = con.createStatement();
           ResultSet rs = stmt.executeQuery("select * from emp");
           while(rs.next())
                         producer.send(new KeyedMessage("test",rs.getString(1)+"
                         "+rs.getString(2)));
            con.close()
            }catch (Exception e){
            System.out.println(e)
            }
            }
            }



spark-shell --master local[2]

import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._

val ssc = new StreamingContext(sc,Seconds(10))
val lines = ssc.socketTextStream("localhost",9999)
val words = lines.flatMap(x => x.split(" "))
val pairs = words.map(x => (x,1))
val res = pairs.reduceByKey(_+_)
res.print()
ssc.start

nc -lk 9999
---------------------------------------------------------------------
kafka - Spark Streaming Example
1) please start zookeeper
   ./bin/zookkeeper-server-start.sh config/zookeeper.properties &
2) please start kafka-Broker
   ./bin/kafka-server-start.sh config/server.properties &
3) spark-shell --master local[2]
4) create two topics mytopic,results
   ./bin/kafka-topics.sh --create --zookeeper localhost:2182 --partitions 1
    --replication-factor 1 --topic mytopic

    ./bin/kafka-topics.sh --create --zookeeper localhost:2182 --partitions 1
    --replication-factor 1 --topic results

    ./bin/kafka-topics.sh --list --zookeeper localhost:2182

5) please start console producer with mytopic
   ./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic mytopic
```

```
6) create another to write the spark streaming output data(topic name result)

7) please start console consumer to read the data from result topic.
   ./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092
    --topic results --from-beginning

   spark-shell --master local[2]
8) import org.apache.spark.streaming.StreamingContext
   import org.apache.spark.streaming.Seconds
   val ssc = new StreamingContext(sc, Seconds(5))
   import org.apache.spark.streaming.kafka.KafkaUtils
   val kafkaStream = KafkaUtils.createStream(ssc,"localhost:2182",
   "spark-streaming-consumer-group",Map("mytopic" ->5))
   val lines = kafkaStream.map(x => x._2.toUpperCase)
   val words = lines.flatMap(x => x.split(" "))
   val pairs = words.map(x => (x,1))
   val res = pairs.reduceByKey(_+_)
   import org.apache.kafka.clients.producer.ProducerConfig
   import java.util.HashMap
   import org.apache.kafka.clients.producer.KafkaProducer
   import org.apache.kafka.clients.producer.ProducerRecord
   res.foreachRDD(rdd =>
       rdd.foreachPartition(partition =>
        partition.foreach{
         case(w:String,cnt:Int)=>{
          val x = w+"\t"+cnt
          val props = new HashMap[String,Object]()
          props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"localhost:9092")

          props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.common.seriali
          zation.StringSerializer")

          props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.common.serializa
          tion.StringSerializer")

import org.apache.kafka.clients.producer.KafkaProducer
   import org.apache.kafka.clients.producer.ProducerRecord
val producer = new KafkaProducer[String,String](props)
val message = new ProducerRecord[String,String]("results",null,x)
          producer.send(message)


         }
        }
       )
   )

   8)please start streaming context
   ssc.start
```

Spark Streaming

```
Ex 1) spark-shell --master local[2]
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
val ssc = new StreamingContext(sc,Seconds(10)) //10 seconds is microbatch period
val data = ssc.scoketTextStream("localhost",9999)
val word = data.flatMap(x => x.split("\\W+"))
val pairs = word.map(x => (x,1))
val res = pairs.reduceByKey(_+_)
res.print()
ssc.start //streaming started
```

```
Ex 2)Sliding and Windowing
pairs.persist()
pairs.reduceByKeyAndWindow(_+_,20,40)//20 seconds for sliding and 40 seconds for windowing
//sliding period is greater than equal to micro batch period.
```

```
--- SQOOP -------------

sqoop.txt
Details
Activity
sqoop.txt
Sharing Info
k

J
R

+7
General Info
Type
Text
Size
10 KB (9,920 bytes)
Storage used
0 bytesOwned by someone else
Location
Sqoop
Owner
kumar K
Modified
18 Feb 2018 by kumar K
Opened
18:39 by me
Created
18 Feb 2018
Description
Add a description
Download permissions
Viewers can download
All selections cleared All selections cleared

Creating the tables in MySql DB
--------------------------------------
CREATE TABLE cities (
id INTEGER UNSIGNED NOT NULL,
country VARCHAR(50),
city VARCHAR(150),
PRIMARY KEY (id)
);
```

```
INSERT INTO cities(id,country,city) VALUES (1, "USA", "Palo Alto");
INSERT INTO cities(id,country,city) VALUES (2, "Czech Republic", "Brno");
INSERT INTO cities(id,country,city) VALUES (3, "USA", "Sunnyvale");

CREATE TABLE countries(
country_id INTEGER UNSIGNED NOT NULL,
country VARCHAR(50),
PRIMARY KEY (country_id)
);

INSERT INTO countries(country_id,country) VALUES (1, "USA");
INSERT INTO countries(country_id,country) VALUES (2, "Czech Republic");

CREATE TABLE normcities (
id INTEGER UNSIGNED NOT NULL,
country_id INTEGER UNSIGNED NOT NULL,
city VARCHAR(150),
PRIMARY KEY (id),
FOREIGN KEY (country_id) REFERENCES countries(country_id)
);

INSERT INTO normcities(id,country_id,city) VALUES (1, 1, "Palo Alto");
INSERT INTO normcities(id,country_id,city) VALUES (2, 2, "Brno");
INSERT INTO normcities(id,country_id,city) VALUES (3, 1, "Sunnyvale");

CREATE TABLE visits (
id INTEGER UNSIGNED NOT NULL,
city VARCHAR(50),
last_update_date DATETIME NOT NULL,
PRIMARY KEY (id),
KEY (last_update_date)
);

INSERT INTO visits(id,city,last_update_date) VALUES(1, "Freemont", "1983-05-22 01:01:01");
INSERT INTO visits(id,city,last_update_date) VALUES(2, "Jicin", "1987-02-02 02:02:02");

---To find out user name MySql
1)select USER();
2)select CURRENT_USER();

Providing the permissions to HDFS home directory
--------------------------------------------------
hdfs dfs -chmod -R 777 /user/hadoop;

Importing the data from MySql DB to HDFS,Hive and HBase
--------------------------------------------------------



1)Importing the data from mysql db to your HDFS if the table does not have a primary key and
using warehuse-dir.

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table cities
--warehouse-dir /cities_output/ -m 1;

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table countries
--warehouse-dir /cities_output/ -m 1;

2)Importing the data from mysql db to your HDFS if the table does not have a primary key.

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table emp --target-dir
/sqoop -m 1;

3)Importing the data from mysql db to your HDFS if the table have primary key.

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table users
```

```
--target-dir /sqoop;

4)Importing the data from mysql db to your HDFS with compression.

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table cities
--target-dir /cities_compress --compress;

5)Importing the data from mysql db to your HDFS with where condition.

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table emp
--target-dir /some_emp --where "empname='XYZ'";

6)Importing the data from mysql db to your HDFS with direct condition.

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table users
--target-dir /fast_load --direct -m 1;

7)Importing the data from mysql db to your HDFS with overriding the type.

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table users
--target-dir /map_col --map-column-java user_id=Long;

8)Importing the data from mysql db to your HDFS applying parallelism

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table cities
--target-dir /sqoop_para --num-mappers 10;

9)Importing the data from mysql db to your HDFS --handling of null values

sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table cities
--target-dir /handling_null --null-string '\\N' --null-non-string '\\N';

10)Importing the all tables from mysql db to your HDFS

sqoop import-all-tables -connect jdbc:mysql://localhost:3306/test -username root
--warehouse-dir /test_db;

11)Importing the some tables from mysql db to your HDFS

sqoop import-all-tables -connect jdbc:mysql://localhost:3306/test -username root
--warehouse-dir /test_dbase --exclude-tables emp,employee,users,empl,nyse;

12)Importing  the data from 2 tables in mysql db to your HDFS.

sqoop import -connect jdbc:mysql://localhost:3306/test -username root --query 'SELECT normcities
.id,countries.country, normcities.city FROM normcities JOIN countries USING(country_id) WHERE $
CONDITIONS'  --split-by id  --target-dir /norm_cities;

--boundary-query "select min(id), max(id) from normcities"

13)Importing  the data from 2 tables and applying boundary conditions in mysql db to your HDFS.

sqoop import -connect jdbc:mysql://localhost:3306/test -username root --query 'SELECT normcities
.id,countries.country, normcities.city FROM normcities JOIN countries USING(country_id) WHERE $
CONDITIONS'  --split-by id  --target-dir /norm_cities_boundary --boundary-query "select
min(id), max(id) from normcities";


14)Importing the data from mysql db to hive if the table have primary key.

sqoop import --connect jdbc:mysql://localhost:3306/test -username root --table users
--hive-table users --hive-import --target-dir /sqoop/hive/users;

15)Importing the data from mysql db to hive if the table not have primary key.

sqoop import --connect jdbc:mysql://localhost:3306/test -username root --table employee
```

```
--hive-table users --hive-import --target-dir /sqoop/hive/employee -m 1;
```

16)Importing the data from mysql db to hbase if the table not have primary key.

```
sqoop import --connect jdbc:mysql://localhost:3306/test --username root  --table empl
--hbase-table employeeHBase --column-family info --hbase-row-key empid --hbase-create-table -m 1;
```

17)Importing the data from mysql db to hbase if the table have primary key.

```
sqoop import --connect jdbc:mysql://localhost:3306/test --username root  --table empl
--hbase-table employeeHBase --column-family info --hbase-row-key empid --hbase-create-table;
```

18)If table have primary key and import only few columns of MySQL table into HBase table.

```
sqoop import --connect jdbc:mysql://localhost:3306/test --username root --table cities
--hbase-table empdet --columns id,country --column-family empf1 --hbase-create-table
```

Note : Column names specified in --columns attribute must contain the primary key column.

19)If table doesn't have primary key then choose one column as a hbase-row-key. Import all the column of MySQL table into HBase table.

```
sqoop import --connect jdbc:mysql://localhost:3306/test --username root --table emp
--hbase-table empwopk --column-family f1 --hbase-row-key empid --hbase-create-table -m 1;
```

20)If table doesn't have primary key then choose one column as a hbase-row-key. Import only few columns of MySQL table into HBase table.

```
sqoop import --connect jdbc:mysql://localhost:3306/test --username root--table emp
--hbase-table empwopkfc --columns empid,empname --column-family f1 --hbase-row-key empid
--hbase-create-table -m 1;
```

Note: Column name specified in hbase-row-key atribute must be in columns list. Otherwise command will execute successfully but no records are inserted into hbase.

21)Let us consider a MySQL table empdemo which have two columns name,address. The table empdemo doesn't have primary key or unique key column.

Records of empdemo:
____

```
create table empdemo(
    -> name varchar(10),
    -> address varchar(20)
    -> );

insert into  empdemo(name,address) values ('abc','123'),('sqw','345'),('abc','125'),('sdf',
'1234'),('aql','23dw');

name    address
----------------
abc     123
sqw     345
abc     125
sdf     1234
aql     23dw

sqoop import --connect jdbc:mysql://localhost:3306/test --username root --table empdemo
--hbase-table empdemo --column-family ed --hbase-row-key name --hbase-create-table -m 1;
```

Note:
 Only 4 records are visible into HBase table instead of 5. In above example two rows have same value 'abc' of name column and value of this column is used as a HBase row key value. If record having value 'abc' of name column come then thoes record will inserted into HBase table. Next time, another record having the same value 'abc' of name column come then thoes column will

overwrite the value previous column.

Above problem also occured if table have composite primary key because the one column from composite key is used as a HBase row key.

22)Saving the output format in a sequence file.

```
sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table cities
--warehouse-dir /cities_output1/ --as-sequencefile -m 1;
```

23)Saving the output format in a avrodata file.

```
sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --table cities
--warehouse-dir /cities_output2/ --as-avrodatafile -m 1;
```

24)Protecting your password.

Sqoop execution that will read the password from standard input:

```
sqoop import -connect jdbc:mysql://localhost:3306/test -username root  -P --table cities
--warehouse-dir /cities_output2/ --as-avrodatafile -m 1;
```

Reading the password from a file:

```
sqoop import -connect jdbc:mysql://localhost:3306/test -username root  --password-file <path of
the file contains pwd> --table cities --warehouse-dir /cities_output2/ --as-avrodatafile -m 1;
```

Exporting the data from HDFS to MySql DB
----

1)Exporting the data from HDFS db to your MySql DB if the table does not have a primary key.

```
sqoop export  --connect jdbc:mysql://localhost:3306/test --username root --table emp
--export-dir /sqoop/part-m-00000 --input-fields-terminated-by ',' -m 1;
```

2)Exporting the data from HDFS db to your MySql DB if the table does not have a primary key.

```
sqoop export  --connect jdbc:mysql://localhost:3306/test --username root --table emp
--export-dir /sqoop/part-m-00000 --input-fields-terminated-by ',';
```

SQOOP PRACTICE.

```
1) sqoop list-databases --connect "jdbc:mysql://quickstart.cloudera:3306" --username retail_dba
--password cloudera

2) sqoop list-databases --connect "jdbc:mysql://quickstart.cloudera:3306" --username retail_dba
-P

3) sqoop list-tables --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username
retail_dba --password cloudera

4) sqoop eval --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username retail_dba
--password cloudera --query "select count(1) from order_items"

5) sqoop import-all-tables -m 12 --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --as-avrodatafile
--warehouse-dir=/user/hive/warehouse/retail_stage.db

6) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --as-textfile
--target-dir=/user/cloudera/departments

7) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --as-sequencefile
--target-dir=/user/cloudera/departments_seq
```

```
8) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --as-avrodatafile
--target-dir=/user/cloudera/departments

9) sqoop import-all-tables --num-mappers 1 --connect
"jdbc:mysql://quickstart.cloudera:3306/retail_db" --username=retail_dba --password=cloudera
--hive-import --hive-overwrite --create-hive-table --compress --compression-codec
org.apache.hadoop.io.compress.SnappyCodec

10) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --target-dir
/user/cloudera/departments -m 2 --boundary-query "select 2, 8 from departments"--columns
department_id,department_name

  sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
  --username=retail_dba --password=cloudera --table departments --target-dir
  /user/cloudera/departments -m 2 --boundary-query "select
  min(department_id),max(department_id) from departments"

sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username=retail_dba
--password=cloudera --table departments --target-dir /user/cloudera/departments -m 2
--boundary-query "select 2,8 from departments where department_id <>1000 "


11) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --query="select * from orders join order_items on
orders.order_id = order_items.order_item_order_id where \$CONDITIONS" --target-dir
/user/cloudera/order_join --split-by order_id --num-mappers 4

12 )sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --target-dir
/user/hive/warehouse/retail_ods.db/departments --append --fields-terminated-by '|'
--lines-terminated-by '\n' --num-mappers 1

13) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --target-dir
/user/hive/warehouse/retail_ods.db/departments --append --fields-terminated-by '|'
--lines-terminated-by '\n' --split-by department_id

Note :- Importing table with out primary key using multiple threads (split-by).When using split-
by, using indexed column is highly desired.If the column is not indexed then performance will be
 bad because of full table scan by each of the thread.

14) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --target-dir
/user/hive/warehouse/retail_ods.db/departments --append --fields-terminated-by '|'
--lines-terminated-by '\n' --split-by department_id --where "department_id > 7"

15) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --target-dir
/user/hive/warehouse/retail_ods.db/departments --append --fields-terminated-by '|'
--lines-terminated-by '\n' --check-column "department_id" --incremental append --last-value 7

16) sqoop job --create sqoop_job -- import --connect "jdbc:
mysql://quickstart.cloudera:3306/retail_db" --username=retail_dba --password=cloudera --table
departments  --target-dir /user/hive/warehouse/retail_ods.db/departments --append
--fields-terminated-by '|' --lines-terminated-by '\n' --check-column "department_id"
--incremental append --last-value 1000

sqoop job --list

sqoop job --exec sqoop_job

17) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --fields-terminated-by '|'
```

```
--lines-terminated-by '\n' --hive-home /user/hive/warehouse/retail_ods.db --hive-import
--hive-overwrite --hive-table departments

18) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --fields-terminated-by '|'
--lines-terminated-by '\n' --hive-home /user/hive/warehouse --hive-import --hive-table
mysparkdb.departments_test --create-hive-table

19)--Connect to mysql and create database for reporting database
--user:root, password:cloudera
mysql -u root -p
create database retail_rpt_db;
grant all on retail_rpt_db.* to retail_dba;
flush privileges;
use retail_rpt_db;
create table departments as select * from retail_db.departments where 1=2;

sqoop export --connect "jdbc:mysql://quickstart.cloudera:3306/retail_rpt_db" --username
retail_dba --password cloudera --table departments --export-dir
/user/hive/warehouse/retail_ods.db/departments --input-fields-terminated-by '|'
--input-lines-terminated-by '\n' --num-mappers 2 --batch

20) sqoop export --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" --username
retail_dba --password cloudera --table departments --export-dir
/user/hive/warehouse/retail_ods.db/departments --batch -m 1 --update-key department_id
--update-mode allowinsert

21) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments_test --target-dir
/user/hive/warehouse/retail_ods.db/departments_test --append --fields-terminated-by '|'
--lines-terminated-by '\n' --num-mappers 1  --null-string na --null-non-string -1

22) sqoop export --connect "jdbc:mysql://quickstart.cloudera:3306/retail_rpt_db" --username
root --password cloudera --table departments_test --export-dir
/user/hive/warehouse/retail_ods.db/departments_test --input-fields-terminated-by '|'
--input-lines-terminated-by '\n' --input-null-string na --input-null-non-string -1

23) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments --target-dir
/user/hive/warehouse/retail_ods.db/departments_new --append --fields-terminated-by '|'
--lines-terminated-by '\n' --num-mappers 1 --enclosed-by \"

24) sqoop import --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db"
--username=retail_dba --password=cloudera --table departments_new --target-dir
/user/hive/warehouse/retail_ods.db/departments_new1 --append --fields-terminated-by '|'
--lines-terminated-by '\n' --num-mappers 1 --escaped-by \@



1)sqoop list-databases --connect jdbc:mysql://localhost:3306 --username root

2)sqoop list-tables --connect jdbc:mysql://localhost:3306/test --username root

3)sqoop eval --connect jdbc:mysql://localhost:3306/test --username root --query "select
count(*) from cities"

4)-- Reference:
http://www.cloudera.com/content/cloudera/en/developers/home/developer-admin-resources/get-started
-with-hadoop-tutorial/exercise-1.html
sqoop import-all-tables -m 12 --connect jdbc:mysql://localhost:3306/test --username=root
--as-avrodatafile --warehouse-dir=/user/hive/warehouse/retail_stage.db

5)sqoop import --connect jdbc:mysql://localhost:3306/test --username root --table emp
--as-textfile --target-dir=/user/hadoop/emp_txt

6)sqoop import --connect jdbc:mysql://localhost:3306/test --username root --table emp
```

```
--as-sequencefile --target-dir=/user/hadoop/emp_seq

7)sqoop import --connect jdbc:mysql://localhost:3306/test --username root --table emp
--as-avrodatafile --target-dir=/user/hadoop/emp_avro


Note:- (This below comments are used for hive)
-- A file with extension avsc will be created under the directory from which sqoop import is
executed
-- Copy avsc file to HDFS location
-- Create hive table with LOCATION to /user/hadoop/emp_avro and TBLPROPERTIES pointing to avsc
file
hadoop fs -put sqoop_import_departments.avsc /user/cloudera

CREATE EXTERNAL TABLE departments
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
LOCATION 'hdfs:///user/cloudera/departments'
TBLPROPERTIES ('avro.schema.url'=
'hdfs://quickstart.cloudera/user/cloudera/sqoop_import_departments.avsc');


-- It will create tables in default database in hive
-- Using snappy compression
-- As we have imported all tables before make sure you drop the directories
-- Launch hive drop all tables
drop table departments;
drop table categories;
drop table products;
drop table orders;
drop table order_items;
drop table customers;

-- Dropping directories, in case your hive database/tables in consistent state
hadoop fs -rm -R /user/hive/warehouse/departments
hadoop fs -rm -R /user/hive/warehouse/categories
hadoop fs -rm -R /user/hive/warehouse/products
hadoop fs -rm -R /user/hive/warehouse/orders
hadoop fs -rm -R /user/hive/warehouse/order_itmes
hadoop fs -rm -R /user/hive/warehouse/customers

8) sqoop import-all-tables --num-mappers 1 --connect jdbc:mysql://localhost:3306/test
--username root
  --hive-import --hive-overwrite --create-hive-table --compress --compression-codec
  org.apache.hadoop.io.compress.SnappyCodec
  --outdir java_files

sudo -u hdfs hadoop fs -mkdir /user/cloudera/retail_stage
sudo -u hdfs hadoop fs -chmod +rw /user/cloudera/retail_stage
hadoop fs -copyFromLocal ~/*.avsc /user/cloudera/retail_stage

-- Boundary Query and columns
9) sqoop import jdbc:mysql://localhost:3306/test --username root
  --table departments --target-dir /user/hadoop/departments -m 2
  --boundary-query "select 2, 8 from departments limit 1"
  --columns department_id,department_name

-- query and split-by
sqoop import
  --connect jdbc:mysql://localhost:3306/test --username root
  --query="select * from orders join order_items on orders.order_id =
  order_items.order_item_order_id where \$CONDITIONS" \
  --target-dir /user/hadoop/order_join --split-by order_id \
  --num-mappers 4
```

```
-- Copying into existing table or directory (append)
-- Customizing number of threads (num-mappers)
-- Changing delimiter
sqoop import \
  --connect jdbc:mysql://localhost:3306/test --username root
  --table departments --target-dir /user/hive/warehouse/retail_ods.db/departments
  --append --fields-terminated-by '|' -lines-terminated-by '\n' --num-mappers 1
  --outdir java_files

-- Importing table with out primary key using multiple threads (split-by)
-- When using split-by, using indexed column is highly desired
-- If the column is not indexed then performance will be bad
-- because of full table scan by each of the thread
sqoop import \
  --connect jdbc:mysql://localhost:3306/test --username root
  --table departments --target-dir /user/hive/warehouse/retail_ods.db/departments \
  --append --fields-terminated-by '|' --lines-terminated-by '\n' --split-by department_id \
  --outdir java_files
```

Notes:-
--split-by : It is used to specify the column of the table used to generate splits for imports. This means that it specifies which column will be used to create the split while importing the data into your cluster. It can be used to enhance the import performance by achieving greater parallelism. Sqoop creates splits based on values in a particular column of the table which is specified by --split-by by the user through the import command. If it is not available, the primary key of the input table is used to create the splits.
Reason to use : Sometimes the primary key doesn't have an even distribution of values between the min and max values(which is used to create the splits if --split-by is not available). In such a situation you can specify some other column which has proper distribution of data to create splits for efficient imports.
--boundary-query : By default sqoop will use query select min(), max() from to find out boundaries for creating splits. In some cases this query is not the most optimal so you can specify any arbitrary query returning two numeric columns using --boundary-query argument.
Reason to use : If --split-by is not giving you the optimal performance you can use this to improve the performance further.

```
-- Getting delta (--where)
10)sqoop import --connect jdbc:mysql://localhost:3306/test --username root
  --table departments --target-dir /user/hive/warehouse/retail_ods.db/departments
  --append --fields-terminated-by '|' --lines-terminated-by '\n' --split-by department_id \
  --where "department_id > 7" --outdir java_files

-- Incremental load
11)sqoop import --connect jdbc:mysql://localhost:3306/test --username root
  --table departments --target-dir /user/hive/warehouse/retail_ods.db/departments
  --append --fields-terminated-by '|' --lines-terminated-by '\n' --check-column "department_id" \
  --incremental append --last-value 7 --outdir java_files

12) sqoop job --create sqoop_job
  --import --connect jdbc:mysql://localhost:3306/test --username root
  --table departments --target-dir /user/hive/warehouse/retail_ods.db/departments
  --append --fields-terminated-by '|' --lines-terminated-by '\n' --check-column "department_id" \
  --incremental append --last-value 7 --outdir java_files

sqoop job --list

sqoop job --show sqoop_job

sqoop job --exec sqoop_job

-- Hive related
-- Overwrite existing data associated with hive table (hive-overwrite)
13 )sqoop import --connect jdbc:mysql://localhost:3306/test --username root
  --table departments --fields-terminated-by '|' --lines-terminated-by '\n'
  --hive-home /user/hive/warehouse/retail_ods.db --hive-import --hive-overwrite
  --hive-table departments --outdir java_files
```

```
--Create hive table example
14)sqoop import --connect dbc:mysql://localhost:3306/test --username root
  --table departments --fields-terminated-by '|' --lines-terminated-by '\n'
  --hive-home /user/hive/warehouse --hive-import --hive-table departments_test
  --create-hive-table \
  --outdir java_files

--Connect to mysql and create database for reporting database
--user:root, password:cloudera
mysql -u root -p
create database retail_rpt_db;
grant all on retail_rpt_db.* to retail_dba;
flush privileges;
use retail_rpt_db;
create table departments as select * from retail_db.departments where 1=2;
exit;

--For certification change database name retail_rpt_db to retail_db
15 )sqoop export --connect dbc:mysql://localhost:3306/test --username root
       --table departments --export-dir /user/hive/warehouse/retail_ods.db/departments
       --input-fields-terminated-by '|' --input-lines-terminated-by '\n'
       --num-mappers 2 --batch --outdir java_files

16)sqoop export --connect jdbc:mysql://localhost:3306/test --username root
  --table departments --export-dir /user/hadoop/sqoop_import/departments_export
  --batch --outdir java_files -m 1 --update-key department_id --update-mode allowinsert

17)sqoop export --connect jdbc:mysql://localhost:3306/test --username root
 --table departments_test --export-dir /user/hive/warehouse/departments_test
  --input-fields-terminated-by '\001' --input-lines-terminated-by '\n'
  --num-mappers 2 --batch --outdir java_files --input-null-string nvl --input-null-non-string -1

--Merge process begins
hadoop fs -mkdir /user/cloudera/sqoop_merge

--Initial load
sqoop import \
  --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
  --username=retail_dba \
  --password=cloudera \
  --table departments \
  --as-textfile \
  --target-dir=/user/cloudera/sqoop_merge/departments

--Validate
sqoop eval --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
  --username retail_dba \
  --password cloudera \
  --query "select * from departments"

hadoop fs -cat /user/cloudera/sqoop_merge/departments/part*

--update
sqoop eval --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
  --username retail_dba \
  --password cloudera \
  --query "update departments set department_name='Testing Merge' where department_id = 9000"

--Insert
sqoop eval --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
  --username retail_dba \
  --password cloudera \
  --query "insert into departments values (10000, 'Inserting for merge')"

sqoop eval --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
  --username retail_dba \
  --password cloudera \
  --query "select * from departments"

--New load
sqoop import \
  --connect "jdbc:mysql://quickstart.cloudera:3306/retail_db" \
  --username=retail_dba \
  --password=cloudera \
  --table departments \
  --as-textfile \
  --target-dir=/user/cloudera/sqoop_merge/departments_delta \
  --where "department_id >= 9000"

hadoop fs -cat /user/cloudera/sqoop_merge/departments_delta/part*

--Merge
sqoop merge --merge-key department_id \
  --new-data /user/cloudera/sqoop_merge/departments_delta \
  --onto /user/cloudera/sqoop_merge/departments \
  --target-dir /user/cloudera/sqoop_merge/departments_stage \
  --class-name departments \
  --jar-file <get_it_from_last_import>

hadoop fs -cat /user/cloudera/sqoop_merge/departments_stage/part*

--Delete old directory
hadoop fs -rm -R /user/cloudera/sqoop_merge/departments

--Move/rename stage directory to original directory
hadoop fs -mv /user/cloudera/sqoop_merge/departments_stage
/user/cloudera/sqoop_merge/departments

--Validate that original directory have merged data
hadoop fs -cat /user/cloudera/sqoop_merge/departments/part*

--Merge process ends


============================================================================

                                MONGO DB

============================================================================

1)Please download MongoDB by using below link

http://www.mongodb.org/downloads

2)place the MongoDB.taz into download directory
3)Extract the MongoDB.taz using below command

tar xvf mongodb-linux-i686-3.0.0.tgz

4)create a data directory under downloads

sudo mkdir -p /mongo/data

5)Then start the MongoDB server by using below command

sudo mongodb-linux-i686-3.0.0/bin/mongod --dbpath /mongo/data/

6)Then Start MongoDB client by using below command

mongodb-linux-i686-3.0.0/bin/mongo
-----------------------------------------------------------------
```

```
================MongoDB Commands================================
1)create a db in MongoDB
The below command creates a new db in MongoDB

>use sample

2)if you want check current db name

>db

3)list of dbs

>show dbs

4)Inserting a record into collection(Table)
syntax- >db.collection name.save({})
db.mycol.save({name:''gvipl',interest:'ēricket'})

5)if you want looking for a record

>db.mycol.find()

6)if you want to looking for a one record

>db.mycol.findOne();

7)disply the list of tables in a db

>show collections;

8)Dropping a Database

db.dropDatabase();

9)Creating a collection

>db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000
} );

Field     Type           Description
capped   Boolean    (Optional) If true, enables a capped collection. Capped collection is a
collection fixed size collecction that automatically overwrites its
oldest entries when it reaches its maximum size. If you specify true, you need to specify size
parameter also.
autoIndexID Boolean (Optional) If true, automatically create index on _id field.s Default value
is false.
size    number          (Optional) Specifies a maximum size in bytes for a capped collection.
If If capped is true, then you need to specify this field also.
max number          (Optional) Specifies the maximum number of documents allowed in the capped
collection.

10)Dropping a Collection

db.COLLECTION_NAME.drop()

>db.mycol.drop();

11)Inserting a Document

db.COLLECTION_NAME.insert(document)

db.post.insert([ {title:'MOngoDB is no sql database', by:'Mongo DB',
url:'http://www.mongodb.org' ,tags:['mongodb','database','NoSQL'], likes:100 }]);

db.post.insert([ {title:'MOngoDB is no sql database', by:'Mongo DB',
url:'http://www.mongodb.org' ,tags:['mongodb','database','NoSQL'], likes:20 }]);

db.post.insert([ {title:'MOngoDB is no sql database', by:'Mongo DB',
url:'http://www.mongodb.org' ,tags:['mongodb','database','NoSQL'],
likes:50,comments:[{user:'user1',message: 'My first comment',dateCreated: new
Date(2013,11,10,2,35),like: 0}] }]);

12)db.post.find().pretty();

13)db.post.find({"by":"Mongo DB"}).pretty();

14)db.post.find({"likes":{$lt:50}}).pretty();

15)db.post.find({"likes":{$lte:50}}).pretty();

16)db.post.find({"likes":{$gt:50}}).pretty();

17)db.post.find({"likes":{$gte:50}}).pretty();

18)db.post.find({"likes":{$ne:50}}).pretty();

19)AND Condition
db.post.find({"by":"Mongo DB","title": "MongoDB Overview"}).pretty();

20)or condition
db.post.find({$or:[{"by":"Mongo DB"},{"title": "MongoDB Overview"}]}).pretty();

21)And ,or both condition at atime
db.post.find("likes": {$gt:10}, $or: [{"by": "Mongo DB"}, {"title": "MongoDB Overview"}]
}).pretty()

22)Updating a collection

db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)

>db.post.update({'likes':100},{$set:{'title':'New MongoDB Tutorial'}})

23)Multiple documents update

>db.post.update({'title':'MOngoDB is no sql database'},{$set:{'title':'New MongoDB
Tutorial'}},{multi:true})

24)save method

db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})

>db.post.save({"_id":ObjectId("550559e86a39df979dae4d80"),"title":"Mongo DB New Topic",
"by":"GVIPL Point"})

25)remove method

db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

db.post.remove({'title':'MongoDB Overview'})

db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)

db.post.remove({'title':'MongoDB Overview'},1);

if you want remove all documents

db.post.remove();

26)Projection of a fields

>db.post.find({},{"title":1,"_id":0});
```

```
27)Limit ()

db.COLLECTION_NAME.find().limit(NUMBER)

>db.post.find({},{"title":1,"_id":0}).limit(2);

28)Skip()

db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)

>db.post.find({},{"title":1,"_id":0}).limit(1).skip(1);

29)sort()

db.COLLECTION_NAME.find().sort({KEY:1})

>db.post.find({},{"title":1,"_id":0}).sort({"title":-1});

DataTypes in Mongo DB
--------------------

    String : This is most commonly used datatype to store the data. String in mongodb must be
    UTF-8 valid.

    Integer : This type is used to store a numerical value. Integer can be 32 bit or 64 bit
    depending upon your server.

    Boolean : This type is used to store a boolean (true/ false) value.

    Double : This type is used to store floating point values.

    Min/ Max keys : This type is used to compare a value against the lowest and highest BSON
    elements.

    Arrays : This type is used to store arrays or list or multiple values into one key.

    Timestamp : ctimestamp. This can be handy for recording when a document has been modified
    or added.

    Object : This datatype is used for embedded documents.

    Null : This type is used to store a Null value.

    Symbol : This datatype is used identically to a string however, it's generally reserved for
    languages that use a specific symbol type.

    Date : This datatype is used to store the current date or time in UNIX time format. You can
    specify your own date time by creating object of Date and passing day,             year
    into it.

    Object ID : This datatype is used to store the document’s ID.

    Binary data : This datatype is used to store binay data.

    Code : This datatype is used to store javascript code into document.

    Regular expression : This datatype is used to store regular expression

1)Indexing in Mongo DB

ensureIndex()

db.COLLECTION_NAME.ensureIndex({KEY:1})

>db.post.ensureIndex({"title":1})
```

```
>db.mycol.ensureIndex({"title":1,"description":-1})

2)Aggregation

aggregate()

db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

group by($group)

>db.post.aggregate([{$group : {_id : "$by", num_tutorial : {$sum : 1}}}])

sum($sum)

>db.post.aggregate([{$group : {_id : "$by", num_tutorial : {$sum : "$likes"}}}])

avg($avg)

>db.post.aggregate([{$group : {_id : "$by", num_tutorial : {$avg : "$likes"}}}])

min($min)

>db.post.aggregate([{$group : {_id : "$by", num_tutorial : {$min : "$likes"}}}])

max($max)

>db.post.aggregate([{$group : {_id : "$by", num_tutorial : {$max : "$likes"}}}])

push($push)--Inserts the value to an array in the resulting document.

>db.post.aggregate([{$group : {_id : "$by", url : {$push: "$url"}}}])

addToSet($addToSet)--   Inserts the value to an array in the resulting document but does not
create duplicates.

>db.post.aggregate([{$group : {_id : "$by", url : {$addToSet: "$url"}}}])

first($first)

>db.post.aggregate([{$group : {_id : "$by", first_url : {$first: "$url"}}}])

last($last)

>db.post.aggregate([{$group : {_id : "$by", last_url : {$last: "$url"}}}])


Analyze the Query Performence
-----------------------------
>db.collection.explain();
>db.collection.explian().help()

Importing the data from a file
------------------------------
>cd Downloads
>mongodb-linux-i686-3.0.0/bin/mongoimport --db sample --collection contacts --file test.json
>mongodb-linux-i686-3.0.0/bin/mongoimport --db sample --collection contacts --type csv
--headerline --contacts.csv

Aggregate Function Examples
--------------------------

1) >db.zipcode.aggregate( [
   { $group: { _id: "$state", totalPop: { $sum: "$pop" } } },
   { $match: { totalPop: { $gte: 10*1000*1000 } } }
] )
```

```
2)>db.zipcode.aggregate( [
   { $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } },
   { $group: { _id: "$_id.state", avgCityPop: { $avg: "$pop" } } }
] )

3)>db.zipcode.aggregate( [
   { $group:
      {
        _id: { state: "$state", city: "$city" },
        pop: { $sum: "$pop" }
      }
   },
   { $sort: { pop: 1 } },
   { $group:
      {
        _id : "$_id.state",
        biggestCity:  { $last: "$_id.city" },
        biggestPop:   { $last: "$pop" },
        smallestCity: { $first: "$_id.city" },
        smallestPop:  { $first: "$pop" }
      }
   },

   { $project:
    { _id: 0,
      state: "$_id",
      biggestCity:  { name: "$biggestCity",  pop: "$biggestPop" },
      smallestCity: { name: "$smallestCity", pop: "$smallestPop" }
    }
  }
] )

Mongo DB Relationships
-----------------------
1)Embaded Relationship
2)Referenced Relationship

db.user.find().pretty();
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "name": "gvipl",
   "contact": "987654321",
   "dob": "01-01-1991"
}

db.address.find().pretty()
{
   "_id":ObjectId("52ffc4a5d85242602e000000"),
   "building": "GVIPL",
   "pincode": 123456,
   "city": "Hyderabad",
   "state": "AP"
}

Embaded Relationship
--------------------
([
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "gvipl",
   "address": [
      {
         "building": "GVIPL1",
         "pincode": 123456,
         "city": "Hyderabad",
         "state": "AP"
      },
      {
         "building": "GVIPL2",
         "pincode": 456789,
         "city": "Hyderabad",
         "state": "TS"
      }]
}
]
)

>db.users.findOne({"name":"gvipl"},{"address":1})

Referenced Relationships
------------------------
{
   "_id":ObjectId("52ffc33cd85242f436000001"),
   "contact": "987654321",
   "dob": "01-01-1991",
   "name": "gvipl",
   "address_ids": [
      ObjectId("52ffc4a5d85242602e000000"),
      ObjectId("52ffc4a5d85242602e000001")
   ]
}

>var result = db.users.findOne({"name":"gvipl"},{"address_ids":1})
>var addresses = db.address.find({"_id":{"$in":result["address_ids"]}})

1)Backup and Restore
--------------------
Backup
------
>mongodump
>mongodump --host localhost --port 27017
>mongodump --out /data/backup/
>mongodump --collection myCollection --db test
>mongodump --host mongodb1.example.net --port 3017 --username user --password pass --out
/opt/backup/mongodump-2013-10-24

Restore
-------
syntax:-mongorestore --port <port number> <path to the backup>

>mongorestore dump-2013-10-25/
>mongorestore --host mongodb1.example.net --port 3017 --username user --password pass
/opt/backup/mongodump-2013-10-24


===========================================================================================

==========================================================================
                    HBASE
===========================================================================

[hadoop@localhost ~]$ sudo /sbin/service zookeeper-server start;
[sudo] password for hadoop
[hadoop@localhost ~]$ sudo /sbin/service hbase-master start;
[hadoop@localhost ~]$ sudo /sbin/service hbase-master status;

[hadoop@localhost ~]$ sudo service hbase-regionserver start;

[hadoop@localhost ~]$ hbase shell
```

```
-----------------------------------------------------------
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:8020/hbase</value>
</property>
<property>
<name>hbase.zookeeper.quorum</name>
<value>localhost:2181</value>
</property>
<property>
<name>hbase.cluster.distributed</name>
<value>true</value>
</property>
</configuration>

// Below command gives the all list of all hbase shell commands

1.hbase> help

//General hbase shell commands

1.status:-how cluster status. Can be ësummaryí, ësimpleí, or ëdetailedí. Thedefault is ësummaryí.

  hbase> status
  hbase> status ësimpleí
  hbase> status ësummaryí
  hbase> status ëdetailedí
2.version:-Output this HBase versionUsage

  hbase>version
3.whoami:shows the current hbase user.

  hbase>whoami

//Tables Management commands(DDL)

1.create:-Create table; pass table name, a dictionary of specifications percolumn family, and
optionally a dictionary of table configuration.

  hbase> create ët1', {NAME => ëf1', VERSIONS => 5}
  hbase> create ët1', {NAME => ëf1'}, {NAME => ëf2'}, {NAME => ëf3'}
  hbase> # The above in shorthand would be the following:
  hbase> create ët1', ëf1', ëf2', ëf3'
  hbase> create ët1', {NAME => ëf1', VERSIONS => 1, TTL => 2592000, BLOCKCACHE => true}
  hbase> create ët1', {NAME => ëf1', CONFIGURATION => {ëhbase.hstore.blockingStoreFilesí =>
  í10í}}

2.describe:-this command describe the hbase table

 hbase>describe '<table name>'

3.disable:-disable the specified table

 hbase>disable '<table name>

if you set a disable condition on a table that table cannot scanned.

 hbase>scan '<disabled table>'

4.disable_all:-Disable all of tables matching the given regex

 hbase>disable_all't.*'

5.is_disabled:-verifies Is named table disabled and it retuns true/false
```

```
 hbase>is_disabled '<table name>'

6.drop:-Drop the named table. Table must first be disabled

 hbase>drop 't1'

7.drop_all:-Drop all of the tables matching the given regex

 hbase>drop_all 't.*'

8.is_enabled:-verifies Is named table enabled and returns true/false

 hbase>is_enabled '<table name>'

9.exists:-Does the named table exist and returns true/false

 hbase>exists '<table name>'

10.list:-List all tables in hbase. Optional regular expression parameter could be used to
filter the output

 hbase>list
 hbase>list 'abc.*'

11.show_filters:-Show all the filters in hbase

 hbase>show_filters

12.alter:-alter column family schema; pass table name and a dictionary specifying new column
family schema.
Dictionaries are described on the main help command output.
Dictionary must include name of column family to alter.
For example, to change or add the ëf1' column family in table ët1' from current value to keep a
maximum of 5 cell VERSIONS, do:

First we need to disable the table

 hbase>alter 't1',NAME=>'f1',VERSIONS=>5

13.alter_status:-get the status of the alter command. Indicates the number of regions of the
table that have received the updated schema Pass table name

 hbase>alter_status<table_name>

//Data Manipulation commands(DML)

1.put:-Put a cell ëvalueí at specified table/row/column and optionally timestamp coordinates.
To put a cell value into table ët1' at
row ër1' under column ëc1' marked with the time ëts1', do

 hbase>create 'emp',{NAME=>'address'},{NAME=>'expinfo'}

       put <Table Name>,<row_id>,<family:col name>,<value>

 hbase>put 'emp','1','address:city','hyd'
 hbase>put 'emp','1','address:state','ap'
 hbase>put 'emp','1','address:country','india'
 hbase>put 'emp','1','expinfo:doj','02022012'
 hbase>put 'emp','1','expinfo:dol','

2.scan:-Scan a table; pass table name and optionally a dictionary of scanner specifications.
Scanner specifications may include one or more of:
TIMERANGE, FILTER, LIMIT, STARTROW, STOPROW, TIMESTAMP, MAXLENGTH,
or COLUMNS, CACHEIf no columns are specified, all columns will be scanned.
To scan all members of a column family, leave the qualifier empty as in ëcol_family:í.The
filter can be specified in two ways:
```

```
1. Using a filterString ñ more information on this is available in the Filter Language document
attached to the HBASE-4176 JIRA
2. Using the entire package name of the filter.Some examples:hbase> scan ë.META.í

 hbase>scan 'emp'

3.count:-count the no.of rows in a table.

 hbase>count 'em'

4.delete:-Put a delete cell value at specified table/row/column and optionally timestamp
coordinates. Deletes must match the deleted cellís
coordinates exactly. When scanning, a delete cell suppresses older versions. To delete a cell
from ët1' at row ër1' under column ëc1'
marked with the time ëts1', do:
'
 hbase>delete '<Table Name>','<row_id>','<family:colname>'
 hbase>delete 'emp','1','expinfo:dol'

5.deleteall:-Delete all cells in a given row; pass a table name, row, and optionally a column
and timestamp.

 hbase> deleteall ë<Table Name>','<row_id>'

6.get:-Get row or cell contents; pass table name, row, and optionally a dictionary of
column(s), timestamp, timerange and versions.

 hbase>get '<Table Name>','<row_id>'
 hbase>get 'emp','1'

7.incr:-Increments a cell ëvalueí at specified table/row/column coordinates.
To increment a cell value in table ët1' at row ër1' under column ëc1' by 1 (can be omitted) or
10 do

 hbase>incr '<Table_Name>','<row_id>','<family:colname>','<incrvalue>'
 hbase>incr 'emp','1,'expinfo:doj','1'

8.truncate:-Disables, drops and recreates the specified table

 hbase>truncate <Table_Name>
 hbase>truncate 'Employee'


-----------------------------------------SPARK INTELLIJ-----------------------

*********WordCount

package SparkCore

import org.apache.spark.sql.SparkSession

object WordCount {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder.master("local[*]").appName("word count").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    import spark.implicits._
    import spark.sql
    println("Success")


    val rawData =
    sc.textFile("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\wordcount.txt")

    println("Data SUCCESSFULL")
    //convert the lines into words using flatMap operation
    val words = rawData.flatMap(line => line.split(" "))



    //count the individual words using map and reduceByKey operation
    val wordCount = words.map(word => (word, 1)).reduceByKey(_ + _)


    println(wordCount.count())

    wordCount.foreach(println)

  //  wordCount.map(x=> println(x))

    wordCount.saveAsTextFile("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\output")

    spark.stop()

  }
}


-------------------------

***MapvsMapPartitions

package SparkCore

/*
 Map Transformation iterates each and every element and creates an RDD at each element level
In Case of Map Partition RDD is created per Partition

Map Partition contain a iterator object , it iterates each and every record from a given data
set.
And Finally creates an RDD per partitions


Map Partition with Index is similar to map partition and it takes one extra parameter called it
as index number.
If we are using map partition with index we are loosing spark functionality of spliiting the
data.

 */


import org.apache.spark.sql.SparkSession

object MapvsMapPartitions {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder().master("local[*]").getOrCreate()
    val sc = spark.sparkContext

// Map
    val m = sc.parallelize(Array(1,2,3))

    println(m.getNumPartitions)

    // Map Partitions
    val x = sc.parallelize(Array(1,2,3),1)

    println(x.getNumPartitions)
    // Mapr Partitions with Indix
    val y = sc.parallelize(Array(1,2,3),2)

    def f(partitionIndex:Int,i:Iterator[Int])={
      (partitionIndex,i.sum,42).productIterator
    }
    val z = y.mapPartitionsWithIndex(f)
    z.collect().foreach(println)

  }
}


----------------------------------------------------------------

****JOIN

package SparkCore

import org.apache.spark.sql.SparkSession


object Join {
  def main(args: Array[String]): Unit = {

    val spark= SparkSession.builder().master("local[*]").appName("Join").getOrCreate()
    val sc=spark.sparkContext

    val edata = sc.textFile("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\emp.txt")
    val ddata = sc.textFile("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\dept.txt")

    val edata_pair = edata.map{ x =>
      val w = x.split(",")
      val eno = w(0).toInt
      val ename = w(1)
      val sal = w(2).toInt
      val gendar = w(3)
      val dno = w(4).toInt
      (dno,(eno,ename,sal,gendar))
    }

    val ddata_pair =ddata.map { x=>
      val w = x.split(",")
      val dno = w(0).toInt
      val dname = w(1)
      val dloc = w(2)
      (dno,(dname,dloc))
    }

    val edata_pair_join_ddata_pair = edata_pair.leftOuterJoin(ddata_pair)


    edata_pair_join_ddata_pair.foreach(println)



//    edata_pair_join_ddata_pair.saveAsTextFile(
"C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\output")
    edata_pair_join_ddata_pair.toDebugString
    spark.stop()

  }

}

-----------------------------------------------------------------------
```

```
---RepartitionVsCoalesec

package SparkCore

import org.apache.spark.sql.SparkSession


object RepartitionVsCoalesec {

  /*
  1. Both coalesec and repartition enables the re assinging of the partitions at run time.
2. coalesec by defaultly shuffling is false
3. Re-partition by defaultly shuffling is true

we can swith off shuffing in repartition that will behave as coalesec
we can not switch on shuffing in coalesec

repartition is not avaialble in apache storm

coalesec is re-commendable

  */

  def main(args: Array[String]): Unit = {

    val spark= SparkSession.builder().master("local[*]").getOrCreate()

    val sc= spark.sparkContext

//    Example coalesec :

    val x = sc.parallelize(Array(1,2,3,4,5),3)
    val y = x.coalesce(2,false)
    println(y.getNumPartitions)

    //Example repartition :

    val repartList = sc.parallelize(Array(1,2,3,4,5),3)
    val repartListOutput = repartList.repartition(2)
    println(repartListOutput.getNumPartitions)


  }

}


---------------------------

************DifferentWaysToCreateRDD

package SparkCore


import org.apache.spark.sql.SparkSession

object DifferentWaysToCreateRDD {

  def main(args: Array[String]): Unit = {


    val spark = SparkSession.builder.master("local[*]").appName("word count").getOrCreate()
    val sc = spark.sparkContext
    /// Through Serialized Method
    val l =List(10,20,30)

    val rdd = sc.parallelize(l)

    rdd.foreach(println)

    /// From Existing RDD

    val newRdd = rdd.filter(x=> x>10)

    newRdd.foreach(println)

    // External Sources sc.TextFile

    val emp = sc.textFile("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\emp.txt")

  sc.stop()

  }

}

-----------------------------------------------------------------------------

*********CSVProcessing

package SparkPackage

import org.apache.spark.sql.SparkSession

object CSVProcessing {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder.master("local[*]").appName("SparkCsv").getOrCreate()
    //local represents local machine & star represents no of reources i.e utilising all resources
    //.getOrCreate => creating a application or using already existed application
    //.appName => creating an application with name "csvExample_1"

    //creating 2 contex sparkContext and sqlContext
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext

    import spark.implicits._
    //Creating DataFrame using SQLContext Reading data)
    val df = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option(
    "inferschema","true").load(
    "C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\Police_Department_Incidents.csv")
    //.option("header","true") => Use first line of all files as Header
    //.option("inferschema","true") => Automatically infer data types

    df.show()
    //df.select("Category").distinct.collect().foreach(println)
    df.createOrReplaceTempView("sfpd") //Creating Temp table
    // sqlContext.sql("select Category from sfpd").collect().foreach(println)
    //top 10 results
    //sqlContext.sql("SELECT Resolution , count(Resolution) as rescount FROM sfpd group by
    Resolution order by rescount desc limit 10").collect().foreach(println)
    val t = sqlContext.sql("select Category,count(Category) as catcount from sfpd group by
    Category order by catcount desc limit 10")
    t.show()
    t.map(t=> "column 0: "+ t(0)).collect().foreach(println)
    spark.stop()

  }
```

```
}

---------------------------------------------------------------------

***********ExcelExample

package SparkPackage

import org.apache.spark.sql.SparkSession

object ExcelExample {

  def main(args: Array[String]): Unit = {

    val spark= SparkSession.builder().master("local[*]").appName("ExcelExample").getOrCreate()

    val sc=spark.sparkContext
    val sqlContext=spark.sqlContext

    val df = sqlContext.read
      .format("com.crealytics.spark.excel")
      .option("useHeader", "true")
      .option("treatEmptyValuesAsNulls", "false")
      .option("inferSchema", "false")
      .option("startColumn", 0) // Optional, default: 0
      .option("endColumn", 99) // Optional, default: Int.MaxValue
      .option("timestampFormat", "dd-MON-YY HH:mm:ss") // Optional, default: yyyy-mm-dd hh:mm:ss
      [.fffffffff]
      .option("location", "C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\EMP.xlsx")
      .option("addColorColumns", "False")
      .option("sheetName", "EMP")
      .load("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\EMP.xlsx")

    //
/*
df.write
  .format("com.crealytics.spark.excel")
  .option("sheetName", "Daily")
  .option("useHeader", "true")
  .option("dateFormat", "yy-mmm-d") // Optional, default: yy-m-d h:mm
  .option("timestampFormat", "mm-dd-yyyy hh:mm:ss") // Optional, default: yyyy-mm-dd hh:mm:ss.000
  .mode("overwrite")
  .save("Worktime2.xlsx")
* */


    df.show(10)

  }

}

----------------------------------------------------------------

*************JsonProcessing

package SparkPackage

import org.apache.spark.sql.SparkSession

object JsonProcessing {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder.master("local[*]").appName("JsonExample").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext

    //Converting RDD to Data Frame
    import spark.implicits._
    import spark.sql
    val df = sqlContext.read.json(
    "C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\world_bank.json")
    df.printSchema() //printing schema
    df.createOrReplaceTempView("jsondata_one")
    df.show()



    val result =   sqlContext.sql("select url,totalamt,abc.* from jsondata_one " + "lateral
    view explode(theme_namecode) as abc")
    val result2 =   sqlContext.sql("select _id from jsondata_one")
    result.show(10)
    result2.show(10)
    //    println(result)


    //result.write.format("com.databricks.spark.csv").option("header","true").save(
    "C:\\Users\\sonirai\\Desktop\\Hadoop GV\\Spark\\SparkSQL\\datasets\\jsontocsv")
    spark.stop()
  }

}


-------------------------------------------------------

*******RDS_FromOracle

package SparkPackage

import org.apache.spark.sql.SparkSession

object RDS_FromOracle {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder().master("local[*]").appName("Oracle Example").getOrCreate()

    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    val empDF = spark.read.format("jdbc").option("url",
    "jdbc:oracle:thin:scoot/tiger@//HYRDSLVM0028.es.ad.adp.com:1521/cri02hyd").option("dbtable",
     "EMP")
      .option("user", "scott").option("password", "tiger").option("driver",
      "oracle.jdbc.driver.OracleDriver").load()

    empDF.show()

    empDF.registerTempTable("emp")

    val empDF2 = sqlContext.sql("select job,count(0) as jobcount from emp group by job order by
    job")

    empDF2.show(10)


  }


}
```

```
----------------------------------------------------------------------

*******XMLProcessing

package SparkPackage


import org.apache.spark.sql.SparkSession

object XMLProcessing {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder.master("local[*]").appName("xmlExample_1").getOrCreate()
    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext
    val df = sqlContext.read.format("com.databricks.spark.xml").option("rootTag","books").option
    ("rowTag","book").load("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\books.xml")
    df.show()
    //    df.printSchema()
    //df.createOrReplaceTempView("jsondata_one")
    //sqlContext.sql("select url,totalamt,abc.code,abc.name from jsondata_one " +
    //"lateral view explode(theme_namecode) as abc").show(5)
    //    result.write.format("com.databricks.spark.csv").option("header","true").save(
    "C:\\Kalyan\\POC\\Spark\\spark_datasets\\json_data\\jsontocsv")
    spark.stop()
  }



}

-----------------------------------------------------------------------------

SparkSQL

package SparkSQL

case class Sample(a: Int, b: Int, c: Int)

import org.apache.spark.sql.SparkSession


object SparkSQLExample1 {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder().master("local[*]").appName("SparkSQLExample").getOrCreate
    ()

    val sc = spark.sparkContext
    val sqlContext = spark.sqlContext

    import sqlContext.implicits._

    val s1 = Sample(10, 20, 30)
    val s2 = Sample(1, 2, 3)
    val s3 = Sample(100, 200, 300)
    val s4 = Sample(1000, 2000, 3000)

    val data = sc.parallelize(List(s1, s2, s3, s4))

    data.collect
    val x = data.map(v => v.a + v.b + v.c).collect
    x.foreach(println)
    val df = data.toDF //if your rdd having schema its eligible for DataFrame.
    df.show()

    df.registerTempTable("sample")

    val result = sqlContext.sql("select * from sample")
    val result1 = sqlContext.sql("select a,b from sample")
    result.show()
    result.printSchema()
    result1.show()
    val result2 = sqlContext.sql("select a,b,c,a+b+c as total from sample")
    result2.show()

    val result4 = sqlContext.sql("select a,b,c,a+b+c as total from sample")
    result4.show(10)

    val groupbySql = sqlContext.sql("select a,sum(b) from sample group by a")
    groupbySql.show(10)



  }


}


--------------------------------------------------------------------------

FirstDataFrame

package SparkSQL

import org.apache.spark.sql.SparkSession

case class Employee(eno: Int, ename: String, sal: Int, gendar: String, dno: Int)
case class Department(dno: Int, dname: String, dloc: String)

object FirstDataFrame {
  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder().master("local[*]").appName("FirstDataFrame").getOrCreate()
    val sc = spark.sparkContext
    val sQLContext=spark.sqlContext

    /*
    select dno,loc,avg(sal),max(sal),min(sal) from emp e join dept d
    where e.dno=d.dno
    group by dno,dloc

    */


    val emp = sc.textFile("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\emp.txt")
    val dept = sc.textFile("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\dept.txt")



    val edata = emp.map { x =>
      val w = x.split(",")
      val eno = w(0).toInt
      val ename = w(1)
      val sal = w(2).toInt
      val gendar = w(3)
      val dno = w(4).toInt
      Employee(eno, ename, sal, gendar, dno)
    }


    val ddata = dept.map { x =>
      val w = x.split(",")
      val dno = w(0).toInt
      val dname = w(1)
      val dloc = w(2)
      Department(dno, dname, dloc)
    }

    import spark.implicits._
    import spark.sqlContext

    val empDataFrame = edata.toDF()
    val deptDataFrame = ddata.toDF()



    empDataFrame.createGlobalTempView("empview")
    deptDataFrame.createGlobalTempView("deptview")


    val empdf= sqlContext.sql("select * from empview").toDF()
    val deptdf= sqlContext.sql("select * from deptview").toDF()
    val avg_sal_max_sal= sqlContext.sql("select d.dno,d.dloc,avg(sal) as AVG_SAL ,max(sal)
    MAX_SAL ,min(sal) MIN_SAL,count(*) COUNT_SAL from empview e join deptview d on e.dno=d.dno
    group by d.dno,d.dloc").toDF()

    val join= sqlContext.sql("select d.dno,d.dloc from empview e join deptview d on e.dno=d.dno"
    ).toDF()

    empdf.show()
    deptdf.show()
    join.show()
    avg_sal_max_sal.show()

    spark.stop()


  }

}


--------------------------------------------------------------------

FirstDataSet

package SparkSQL

import org.apache.spark.sql.SparkSession

//case class Employee(eno: Int, ename: String, sal: Int, gendar: String, dno: Int)
//case class Department(dno: Int, dname: String, dloc: String)

object FirstDataSet {
  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder().master("local[*]").appName("FirstDataFrame").getOrCreate()
    val sc = spark.sparkContext
    val sQLContext=spark.sqlContext

    /*
    select dno,loc,avg(sal),max(sal),min(sal) from emp e join dept d
    where e.dno=d.dno
    group by dno,dloc

    */


    val emp = sc.textFile("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\emp.txt")
    val dept = sc.textFile("C:\\Users\\NaraVish\\IdeaProjects\\SparkPractice\\Data\\dept.txt")



    val edata = emp.map { x =>
      val w = x.split(",")
      val eno = w(0).toInt
      val ename = w(1)
      val sal = w(2).toInt
      val gendar = w(3)
      val dno = w(4).toInt
      Employee(eno, ename, sal, gendar, dno)
    }


    val ddata = dept.map { x =>
      val w = x.split(",")
      val dno = w(0).toInt
      val dname = w(1)
      val dloc = w(2)
      Department(dno, dname, dloc)
    }

    import spark.implicits._
    import spark.sqlContext

    val empDataSet = edata.toDS
    val deptDataSet = ddata.toDS


    empDataSet.show(10)
    deptDataSet.show(10)


    empDataSet.select($"eno",$"ename",$"sal",$"gendar",$"dno").show()

    empDataSet.filter("eno=101").show(10)

    empDataSet.select($"(sal)").show()

    // Pending hwo to get Max
    //empDataSet.select($max"(sal)").show()



    spark.stop()


  }

}


----------------------------------------------------

package SparkSQL

case class Person(name: String, age: Int)
case class Company(name: String, foundingYear: Int, numEmployees: Int)
case class Employee(name: String, age: Int, departmentId: Int, salary: Double)
case class Department(id: Int, name: String)

case class Record(name: String, age: Int, salary: Double, departmentId: Int, departmentName:
String)
case class ResultSet(departmentId: Int, departmentName: String, avgSalary: Double)


import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._

object AnotherDataSets {

  def main(args: Array[String]): Unit = {

    val spark = SparkSession.builder().master("local[*]").appName("AnotherDataSets").getOrCreate
    ()

    val sc =spark.sparkContext
    val sqlContext=spark.sqlContext

    import spark.implicits._

    val dataset = sc.parallelize(Seq(1, 2, 3))
    dataset.toDS().show()
    /* -- Not workding in 2.0
    val dataset = sc.parallelize(Seq(1, 2, 3))







    val personDS = Seq(Person("Max", 33), Person("Adam", 32), Person("Muller", 62)).toDS()
    personDS.show()

    val rdd = sc.parallelize(Seq((1, "Spark"), (2, "Databricks")))
    val integerDS = rdd.toDS()
    integerDS.show()



    val inputSeq = Seq(Company("ABC", 1998, 310), Company("XYZ", 1983, 904), Company("NOP",
    2005, 83))
    val df = sc.parallelize(inputSeq).toDF()

    val companyDS = df.as[Company]
    companyDS.show()


    val rdd = sc.parallelize(Seq((1, "Spark"), (2, "Databricks"), (3, "Notebook")))
    val df = rdd.toDF("Id", "Name")

    val dataset = df.as[(Int, String)]
    dataset.show()


    val wordsDataset = sc.parallelize(Seq("Spark I am your father", "May the spark be with
    you", "Spark I am your father")).toDS()
    val groupedDataset = wordsDataset.flatMap(_.toLowerCase.split(" "))
      .filter(_ != "")
      .groupBy("value")
    val countsDataset = groupedDataset.count()
    countsDataset.show()
    val employeeDataSet1 = sc.parallelize(Seq(Employee("Max", 22, 1, 100000.0),
    Employee("Adam", 33, 2, 93000.0), Employee("Eve", 35, 2, 89999.0), Employee("Muller", 39,
    3, 120000.0))).toDS()
    val employeeDataSet2 = sc.parallelize(Seq(Employee("John", 26, 1, 990000.0),
    Employee("Joe", 38, 3, 115000.0))).toDS()
    val departmentDataSet = sc.parallelize(Seq(Department(1, "Engineering"), Department(2,
    "Marketing"), Department(3, "Sales"))).toDS()

    val employeeDataset = employeeDataSet1.union(employeeDataSet2)

    def averageSalary(key: (Int, String), iterator: Iterator[Record]): ResultSet = {
      val (total, count) = iterator.foldLeft(0.0, 0.0) {
        case ((total, count), x) => (total + x.salary, count + 1)
      }
      ResultSet(key._1, key._2, total / count)
    }

    val averageSalaryDataset = employeeDataset.joinWith(departmentDataSet, $"departmentId" ===
    $"id", "inner")
      .map(record => Record(record._1.name, record._1.age, record._1.salary,
      record._1.departmentId, record._2.name))
      .filter(record => record.age > 25)
      .groupBy($"departmentId", $"departmentName")
      .avg()



    val wordsDataset = sc.parallelize(Seq("Spark I am your father", "May the spark be with
    you", "Spark I am your father")).toDS()
    val result = wordsDataset
      .flatMap(_.split(" ")) // Split on whitespace
      .filter(_ != "") // Filter empty words
      .map(_.toLowerCase())
      .toDF() // Convert to DataFrame to perform aggregation / sorting
      .groupBy($"value") // Count number of occurences of each word
      .agg(count("*") as "numOccurances")
      .orderBy($"numOccurances" desc) // Show most common words first
    result.show()
    */

  }
}


--------------------------------------------------------------------------------
```