Task1 rpt?

(1)

After HBase create table Create data by self(two vehicle, four repair, one or two owner)

Task2

Function API?(select)

1. Start hadoop
2. $HBASE\_HOME/bin/start-hbase.sh &jps
3. $HBASE\_HOME/bin/hbase shell

Rpt command(exit)

946a0aa27690bf46acd687f3a7cea0a

Create table

//create ‘table\_name’

Create ‘COURSEWORK’, ‘SUBJECT’, ’SUBMISSION’, ’FILE’

//insert column

//alter ‘table\_name’, {NAME=>’column\_name’, VERSION=>1}

alter ‘table\_name’, {NAME=>’STUDENT’, VERSION=>1}

List

//show table names

The structure of column family, show property of column

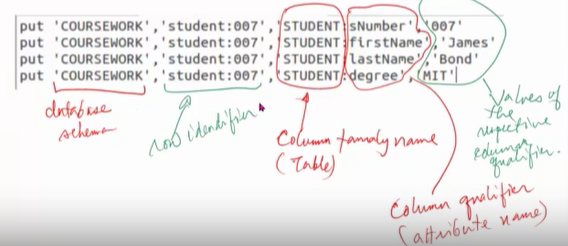
describe ’table\_name’

Put data, insert data

//put ‘table\_name’, ’row identifier’, ‘column family:first\_attribute’, ‘value’

Put ‘table\_name’, ‘student:007’, ‘STUDENT:snumber’, ‘007’

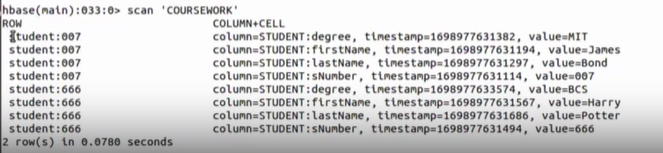




To view data

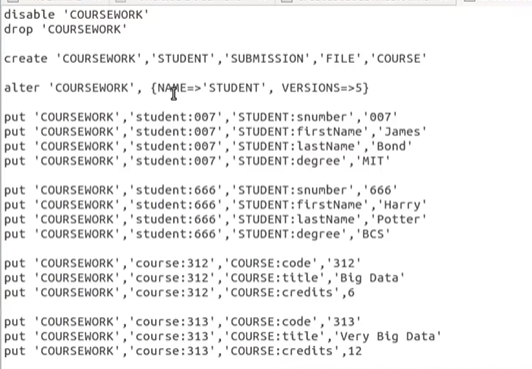
Scan ‘table\_name’

Scan ‘table\_name’, {COLUMN=>’STUDENT:firstname’}

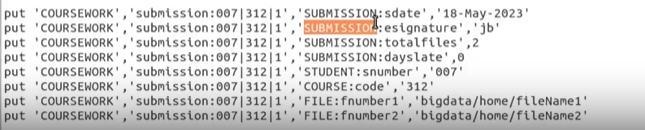
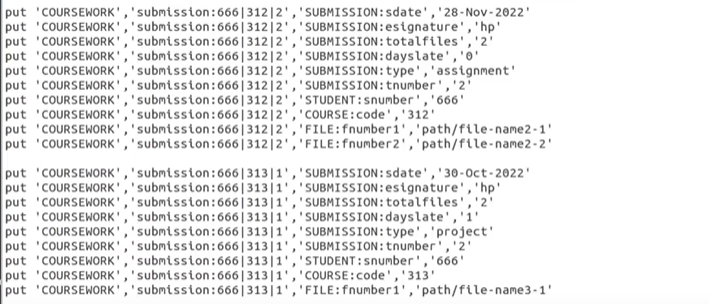


Script:

createStdSubmission.hb



Submission is a association class



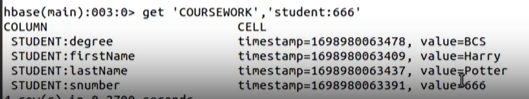
RUN .hb

// source(‘path’)

source(‘/home/bigdata/.../createsubmission.hb’)

Query search

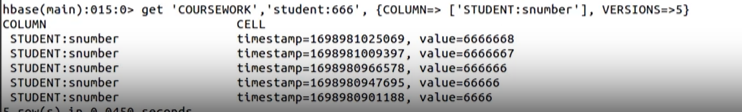
//get ‘table\_name’ ‘row\_id’



Only view snumber

// //get ‘table\_name’ ‘row\_id’, {COLUMN=>’STUDENT:snumber’}

About version 29:40



alter ‘table\_name’, {NAME=>’STUDENT’, VERSION=>1}

To change version

alter ‘table\_name’, {NAME=>’STUDENT’, VERSION=>5}

Scan&get

-scan get more

- get one(need row key)

Clear content, but still exit table

Truncate ‘table\_name’

Disable ‘table\_name’

Drop ‘table\_name’

Task3 Pig

Start Hadoop

- ./start-hadoop.sh

New terminal to start pig

- $PIG\_HOME/bin/pig

Then ‘grunt>’

PUT data file into HDFS

- $HADOOP\_HOME/bin/hadoop fs -put file\_name.txt

READ file into HDFS

- $HADOOP\_HOME/bin/hadoop fs -cat file\_name.txt

REMOVE file in HDFS

- -rm file\_name.txt

Create a pig latin container by loading data from HDFS

- orders = load ‘/path/...file\_name.txt’ using PigStotage(‘,’);

// order - container name

//PigStorage - get data down from hadoop

//(‘,’) - separate it using , because data is separated using ,

Comment

Block

/\*

...

...

\*/

Line

-- ... ...

- describe container\_name;

- dump container\_name;

Download data from HDFS, and describe the schema

- container\_name = load ‘/path/...file\_name.txt’ using PigStotage(‘,’)

> as (item:chararray, customer:chararray, quantity:int, year:int, month:int, day:int);

- dump container\_name;

- describe container\_name; (will get schema)

Save scrip as file, then execute in **HDFS**

- $PIG\_HOME/bin/pig -f /path.../file\_name.pig

store

Read in key-value pair

load data into hdfs first

- -put ...

- -ls (to check)

- -cat (to view)

Download map from HDFS, describe the schema

- container\_name = load ‘/path/.../file\_name.txt’ as (personal:map[]);

- describe c\_n;

- dump c\_n;

- describe c\_n;

Create bag, contain values({value1},{value2})

Download a bag(multi-value object) from HDFS into PigStorage

- c\_n = load’/path/.../file\_name.txt’ as (firstName:chararray, hobbies:bag{t:(hobby:chararray)});

//t - is the name of bag

- dump c\_n;

- describe c\_n;

Download a nested structure from HDFS into PigStorage

- nested=load’/path/.../file\_name.txt’ as (firstName:chararray, cars:bag{t:(manufacturer:chararray, rego:chararray)});

- dump

- describe

Manipulate and processing data

How to project attributes from a Pig Latin container

- new\_c\_n = foreach c\_n generate c\_attribute;

- items = foreach orders generate item;

Creating a distinct item from items container

- distinctItems = distinct items;

Generate multiple tuples

- dates = foreach orders generate day, month, year;

Filter orders where the quantity is greater than 100

- bigorder = filter orders by quantity > 100;

Find Null value

nullOrder = filter orders by quantity is null;

Split orders into

orders2016 if year == 2016,

orders 2018 if year == 2018,

otherOrder otherwise;

dump orders 2016;

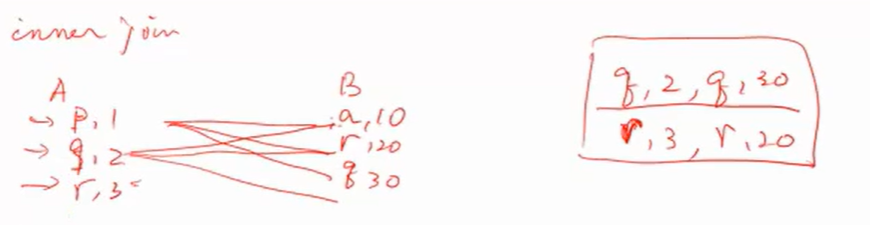
dump orders 2018;

dump orders otherOrder;

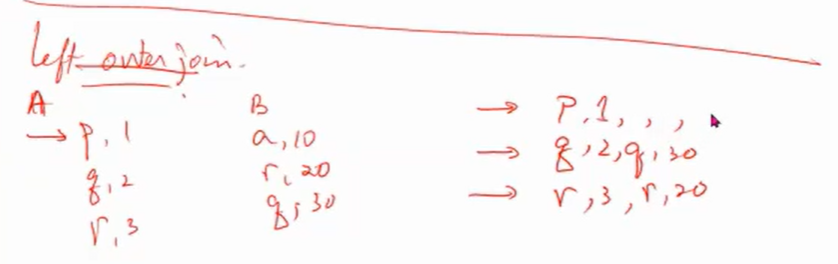
Inner join

- items = load’/path...’ using PigStorage(‘,’) as (item:chararray, price:float);

- ordersJoinItems = join orders by item, items by item



Left outer join



- leftOuter = join items by item left outer, orders by item; (what items ordered in orders)

Non-equal join (items not ordered), if item are not equal join, if equal not join

- notOrderedItems = filter leftOuter by orders::item is null;

- ItemNotOrdered = foreach notOrderedItem generate items::item, items::price;

Cross join (cartesion product)

- newItems = load’path’ using PigStorage(‘,’) as (item:chararray, price:float);

- crossjoin = cross items, new items;

- result = filter crossjoin by items::price > new items::price

Grouping and aggregation functions, group by

- ordergrp = group orders by items;

- itemcnt = foreach ordergrp generate group, COUNT(orders.item);

Roll up and CUBE

- ordercube = cube orders by CUBE(item, customer);

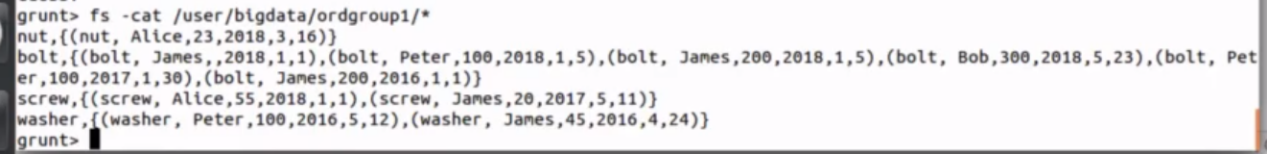
- cntcube = foreach ordercube generate group, count(cube.item);

How to save the results in HDFSw

-store ordergrp into ‘/user/bigdata/ordergroup1’ using PigStorage(‘,’);(pig)

- fs -cat /user/bigdata/ordergroup1/\*(hdfs)

- fs -cat /user/bigdata/ordergroup1/\*(pig)



1. See from bank first, create container to get data loaded, then filter equal to construction
2. Leftouter join
3. Generate container of all the bank account number in japan, group by and count
4. Join, count

Spark :quit

Start hadoop

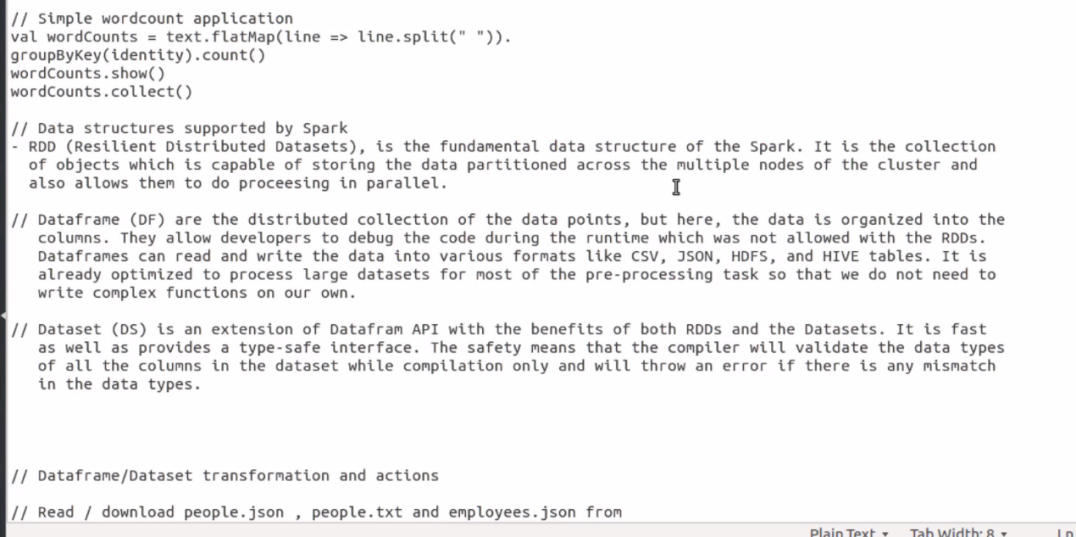
Go to spark home

- $SPARK\_HOME/bin/spark-shell --master local[\*]

- scala>

To view spark version

> spark



Create data command

> val myRange1 = spark.range(20).toDF(“number”)

// val is value, range is a function generate 20 int number 0-19, toDF convert it to DataFrame(column name an attribute)

- the name of column is number, the value is integer(describe as spark sql)

>val myRange2 = spark.range(18).toDF(“number”)

> myRange2.show()

> val myRange3 = myRange1.except(myRange2)

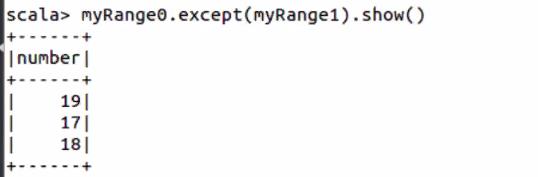
> myRange3.show()

Show content

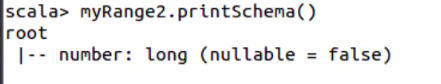
>myRange0.show()

Minus range1 - range2

> myRange1.except(myRange2).show()



> myRange2.printSchema()



Eco system component of the hadoop

Read text file

-val text = spark.read.textFile(‘file\_name.txt’)

-text.printSchema()

- text.show(104,false) // see content

- text.count()

- text.first()

- text.filter(line => line.ccontains(”Spark”)).count()

- text.map(line => line.split(“ ”).size).reduce((a,b) => if (a > b) a else b)

- val wordCounts = text.flatMap(line => line.split(“ ”)).groupByKey(identtity),count()

// flatMap - break file into line by line

Read a json file into a dataframe

- val df = spark.read.json(“file\_name.json”)

- df.show()

- df.printSchema()

Some basic relational operations

- df.select($”name”, $”age”+2).show() //$ means value of

-df.filter($”age”>21).show()

- df.groupBy(“age”).count().show()

- df.createOrReplaceTempView(“People”) //use sql

- val sqlDF = spark.sql(“select \* from People”)

Reading into a dataset, need to introduce column and data type

- case class Person(name:String, age:Long) // Seq - insert into

- val cdDS = Seq(Person(“Andy”,32)).toDS()

- cdDS.show()

- cdDS.printSchema()

- cdDS.select($”name”).show()

Read text file(from **HDFS**) into dataframe

- val peopleDF = spark.sparkContext.textFile(“file\_name.txt”).

map(\_.split(“,”)).

map(attributes => Person (attributes(0), attribute(1).trim.toInt)).

toDF()

peopleDF.show()

Read it as a dataset

- val peopleDS = spark.sparkContext.textFile(“file\_name.txt”).

map(\_.split(“,”)).

map(attributes => Person (attributes(0), attribute(1).trim.toInt)).

toDS()

peopleDS.show()

Read a json into a dataset or dataframes

- case class Employee(name:String, salary:Long)

- val ds = spark.read.json(“file\_name.json”).as[Employee]

Read text file into RDDs from HDFS

- val itemsRDD= sc.textFile(“file\_name.txt”)

Transform the dataset items RDD into a new RDD data pairs

- val pairs = itemsRDD.map(s => (pairs.split(“,”)(0), (s.split(“,”)(1).toFloat)))

Perform a collection action ‘collect’ to retrieve all the elements of the itemsRDD

- pairs.collect()

Perform a collection action ‘reduceByKey’ to return the result

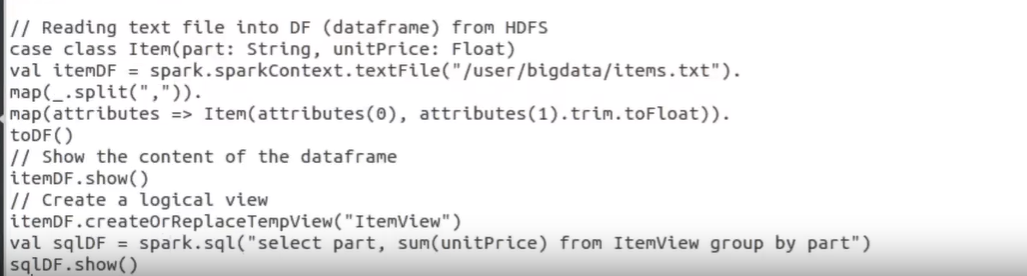
- val result = pairs.reduceByKey((a,b) => a + b)

Collect the result

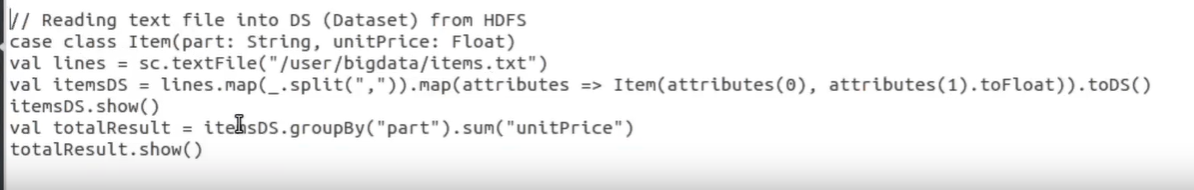
- result.collect()

- result.foreach(println)

Reading text file into DF(dataframe) from HDFS



Reading text field into DS(Dataset) from HDFS



Three different data structure

RDD(Resilient Distributed Datasets. It is the fundamental data structure of the Spark. It is the collection of objects which is capable of storing the data partitioned across the multiple nodes of the cluster and also it allows them to do processing in parallel.)

Dataframes(dataframes are the distributed collection of the data points. But here, the data is organized into the **named columns**. They allow developer to debug the code during the runtime which was not allowed with the RDDs. Dataframes can read and write the data into various format like CSV, JSON, AVRO, HDFS, and HIVE tables. It is already optimised to process large datasets for most of the pre-processing task so that we do not need to write complex functions on our own)

Dataset is an extension of Dataframes API with the benefits of both RDDs and the Datasets. It is fast as well as provides a type-safe interface. Type safety means that the compiler will validate the data types of all the columns in the dataset while compilation only and will throw an erroe if there is any mismatch in the data types.