Apache Spark

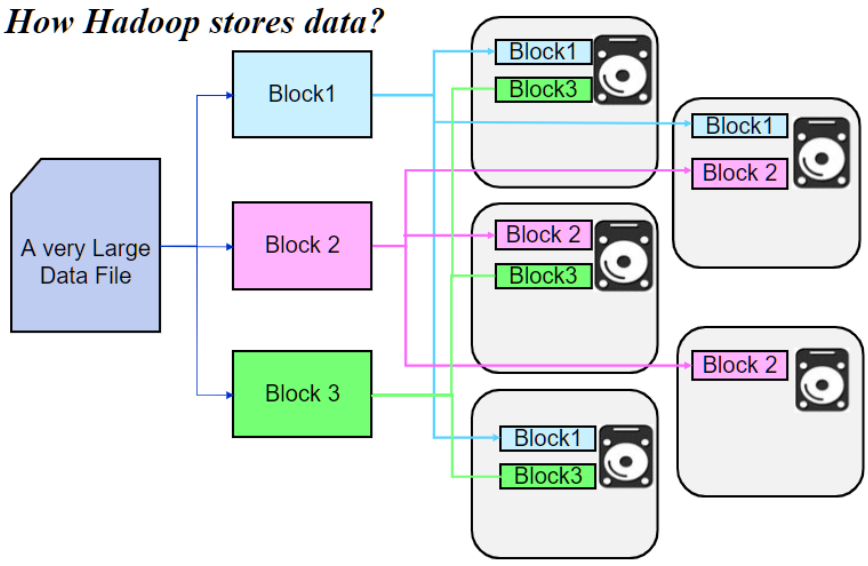
LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- What is Spark?
- Why Spark?
- Spark Vs Hadoop

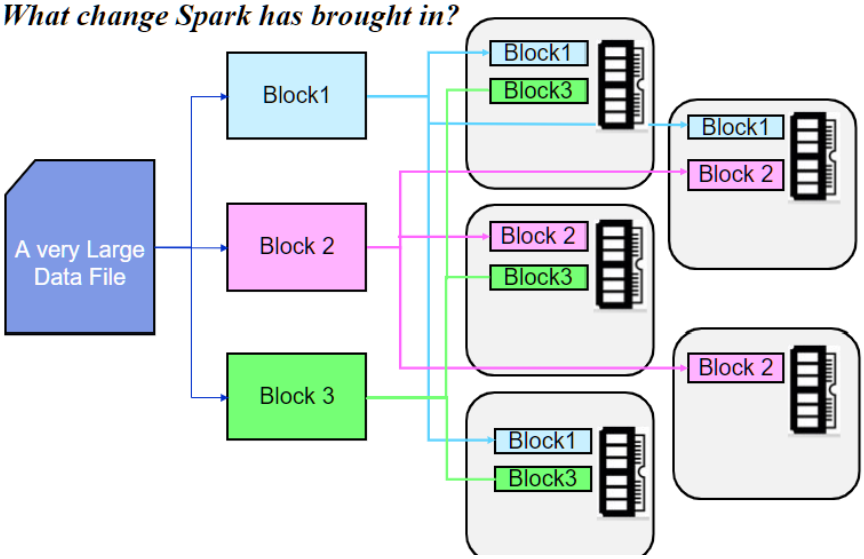
- Open source fast and general engine for large data processing.
- Developed in 2009 in UC Berkeley's AMP Lab, and open sourced in 2010 as an Apache project.
- Written in Scala.
- In memory processing framework.
- Provides high-level APIs in Java, Scala, Python and R.

- Unified framework to manage big data requirements.
- Provides high level operators such as filter, map, etc..
- 100x faster in memory, 10x faster on disk.
- Spark Shell
  - Interactive - for data exploration and testing.
  - Scala or Python
- Spark Applications
  - For large scale and data processing.
  - Scala, Python or Java



Spark Vs Hadoop

What change Spark has brought in?



	Spark	Hadoop
Introduction	▪ Faster and general purpose data processing engine. ▪ Handles batch as well real time processing.	▪ Processes structured and unstructured data that are stored in HDFS. ▪ Handles only batch processing.

	Spark	Hadoop
Speed	▪ Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. ▪ Reducing the number of read/write cycle to disk and storing intermediate data in-memory.	▪ Reads and writes from disk and that slows down the processing speed.

	Spark	Hadoop
Difficulty	▪ Provides high level operators. E.g. filter, map	▪ No high level operators. ▪ Need to hand code each and every operation.

Big Code

```
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

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

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

Tiny Code

```
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

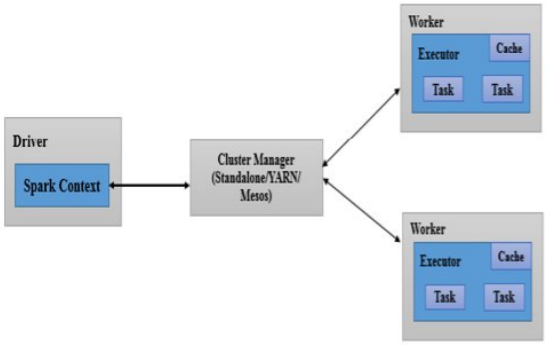
LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- Architecture Overview
- Spark Cluster setup – Pseudo distributed mode
- Spark Shell
- Spark Context

ARCHITECTURE OVERVIEW

- Master/Slave architecture with cluster manager and two daemons.
- Daemons are:
  - Master – Master/ Driver Process
  - Worker – Slave Pr



LEARNING OBJECTIVES

At the end of this unit, you should be able to:

Spark Eco System

- Spark SQL
- Spark Streaming
- Mlib (Machine Learning)
- GraphX
- SparkR

ARCHITECTURE OVERVIEW

- A spark cluster has single coordinator called driver and many distributed workers.
- Driver communicate with large number of distributed workers called executors.
- Cluster Manager is responsible for scheduling and allocating resources to a Spark Job.

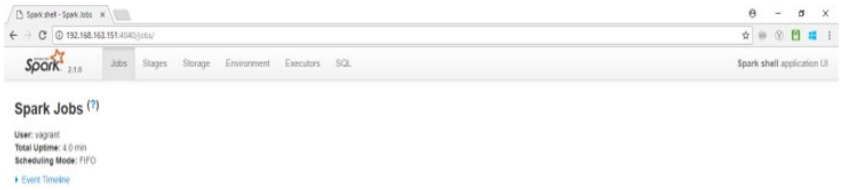
ARCHITECTURE OVERVIEW - ROLE OF DRIVER

- The driver program
  - Entry point of the Spark Shell (Scala, Python, and R).
  - Runs application main () function
  - Is the place where Spark Context is created.
- Responsible for:
  - Scheduling job, negotiating with the cluster manager.
  - Converting a user application into smaller execution units i.e. tasks.

- You can access running spark application information through default Web UI at port 4040.

ARCHITECTURE OVERVIEW - ROLE OF DRIVER

- **Spark shell application UI.**



ARCHITECTURE OVERVIEW - ROLE OF EXECUTORS

- Distributed agent.
- Responsible for:
  - Executing tasks.
  - Performing data processing.
  - Reading from and Writing data to external sources.
  - Storing computation results data in-memory, cache or disk.
  - Interacting with the storage systems.

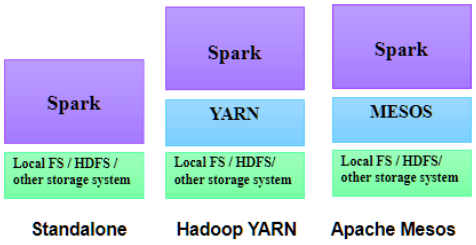
ARCHITECTURE OVERVIEW - ROLE OF CLUSTER MANAGER

- It is an external service.
- Responsible for:
  - acquiring resources and allocating them to a spark application.
  - allocation and deallocation of various physical resources such as CPU, memory, etc..,

ARCHITECTURE OVERVIEW - ROLE OF CLUSTER MANAGER

Three types of cluster managers:

- Standalone Cluster Manager
- Hadoop YARN
- Apache Mesos



SPARK SHELL

- Provides interactive shell for data exploration and testing (REPL).
- To start Spark Shell, type spark-shell command as shown below.

Note: REPL (Read/Evaluate/Print Loop)

```
vagrant@master:~$ spark-shell
```

```
Spark context Web UI available at http://192.168.163.151:4040
Spark context available as 'sc' (master = local[*], app id = local-1509877354288).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_77)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

SPARK CONTEXT

- At high level, every Spark application requires Spark Context.
- Spark Context is entry point to the Spark API.
- Driver program communicates with Spark through Spark Context.

```
Spark context available as 'sc' (master = local[*], app id = local-1509877354288).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/
```

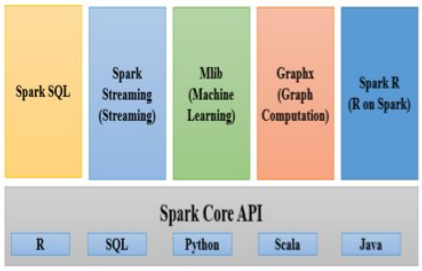
```
scala> sc.appName
res0: String = Spark shell

scala>
```

Spark Core(RDD):

- Fundamental data structure of Spark.
- RDD (Resilient Distributed Dataset), fault-tolerant collection of elements that can be operated on in parallel.

Spark SQL:

- Running SQL like queries on Spark data.



Spark Streaming:

- Scalable, fault-tolerant, high stream processing of live data streams.

MLib:

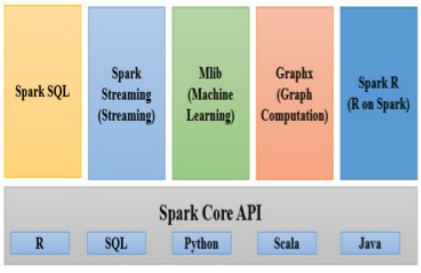
- Scalable practical machine learning.

Graphx:

- Graphs and graph-parallel computation.

SparkR:

- R package that provides a light-weight frontend to use Apache Spark from R.



LEARNING OBJECTIVES

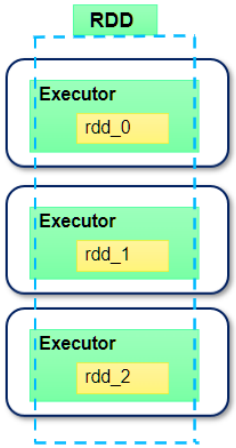
At the end of this unit, you should be able to:

- What is RDD?
- RDD Characteristics
- Partitions
- Read - Only
- RDD operations
- Creating an RDD

WHAT IS RDD?

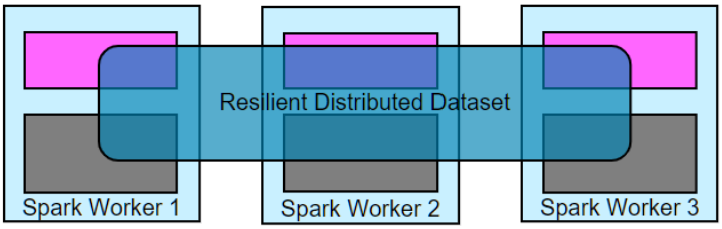
RDD (Resilient distributed dataset)

- **Fundamental data structure of Apache Spark.**
- **Resilient - In built fault tolerance. If something goes wrong reconstruct from source (Lineage).**
- **Distributed - data is distributed in memory across the worker nodes.**
- **Dataset - represents records of the data.**



WHAT IS RDD?

- **RDD is an immutable collection of objects which computes on the different node of the cluster.**
- **Dataset in RDDs are logically partitioned across many nodes for parallel computation.**



RDD CHARACTERISTICS

Partitions:

- **RDD represent data in memory.**
- **To handle huge volume of data**
  - data is divided in to partitions that are distributed to multiple machines called nodes.
  - process data in parallel.

RDD CHARACTERISTICS

Ready - Only:

- **RDDs are immutable.**
  - can not be modified in place.
  - to modify, there are only two types of operations.
    - Transformations
    - Actions

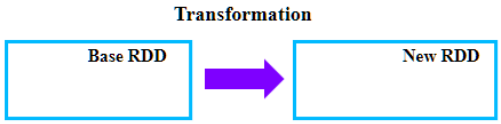
RDD OPERATIONS

Transformation:

- **Converts an RDD into another RDD.**
- **Transform records in dataset.**
- **Once a dataset is loaded into memory, you can perform chain of transformation before you see some results.**

Example:

- Filtering out only certain records.
- Extracting specific field.



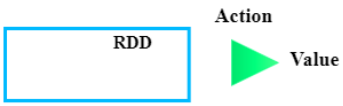
RDD OPERATIONS

Actions:

- **The actual data processing doesn't happen immediately.**
- **It happens only user requests a result.**

Example:

- **First 10 rows**
- **A Sum**
- **A Count**



CREATING AN RDD

Two ways to create an RDD.

- by loading an external dataset, or
- by distributing a collection of objects (e.g., a list or set).

CREATING AN RDD - USING PARALLELIZED COLLECTIONS

Parallelized Collections:

Objective: Creating an RDD using parallelized collection method of Spark Context.

Action:

```
val input = Array(1, 2, 3, 4, 5)
val output = sc.parallelize(input)
output.collect()    // To display output on the console
```

Note: collect() is an Action

Output:

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

scala> val output = sc.parallelize(input)
output: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:26

scala> output.collect()
res1: Array[Int] = Array(1, 2, 3, 4, 5)
```

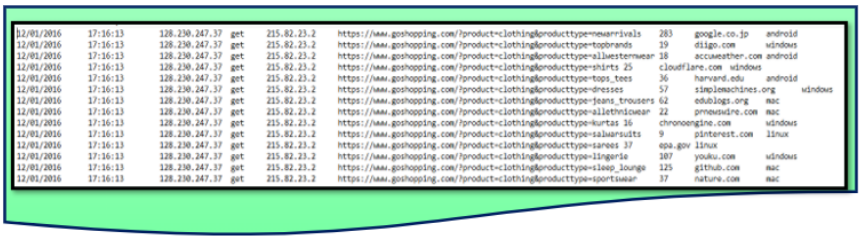
CREATING AN RDD – USING TEXT FILE

File based RDD:

TextFile RDD can be created using textFile method of Spark Context.

Input Data:

File Name: "goShopping_WebClicks.dat " (contains user activity)

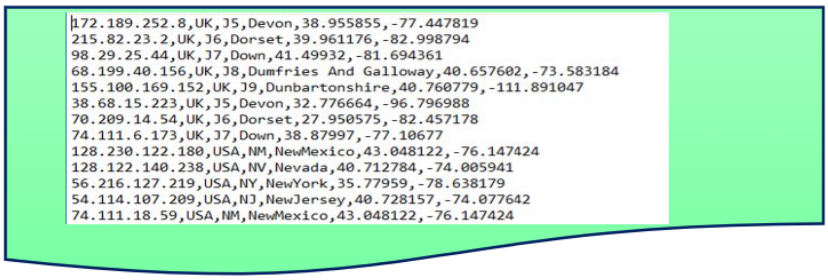


12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=newarrivals	283	google.co.jp	android
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=topbrands	19	diigo.com	windows
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=allwesternwear	18	accuweather.com	android
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=shirts	25	cloudflare.com	windows
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=tops_tees	36	harvard.edu	android
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=dresses	57	simplemachines.org	windows
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=jeans_trousers	62	edublogs.org	mac
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=allethniowear	22	prnewswire.com	mac
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=kurtas	16	chronoengine.com	windows
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=salwarsuits	9	pinterest.com	linux
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=sarees	37	epa.gov	linux
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=lingerie	107	youku.com	windows
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=sleep_lounge	125	github.com	mac
12/01/2016	17:16:13	128.230.247.37	get	215.82.23.2	https://www.goshopping.com/?product=clothing&producttype=sportswear	37	nature.com	mac

CREATING AN RDD – USING TEXT FILE

Input Data:

File Name: "goShopping_IpLookup.txt" (contains user IP details)



```
172.189.252.8,UK,J5,Devon,38.955855,-77.447819
215.82.23.2,UK,J6,Dorset,39.961176,-82.998794
98.29.25.44,UK,J7,Down,41.49932,-81.694361
68.199.40.156,UK,J8,Dumfries And Galloway,40.657602,-73.583184
155.100.169.152,UK,J9,Dunbartonshire,40.760779,-111.891047
38.68.15.223,UK,J5,Devon,32.776664,-96.796988
70.209.14.54,UK,J6,Dorset,27.950575,-82.457178
74.111.6.173,UK,J7,Down,38.87997,-77.10677
128.230.122.180,USA,NM,NewMexico,43.048122,-76.147424
128.122.140.238,USA,NV,Nevada,40.712784,-74.005941
56.216.127.219,USA,NY,NewYork,35.77959,-78.638179
54.114.107.209,USA,NJ,NewJersey,40.728157,-74.077642
74.111.18.59,USA,NM,NewMexico,43.048122,-76.147424
```

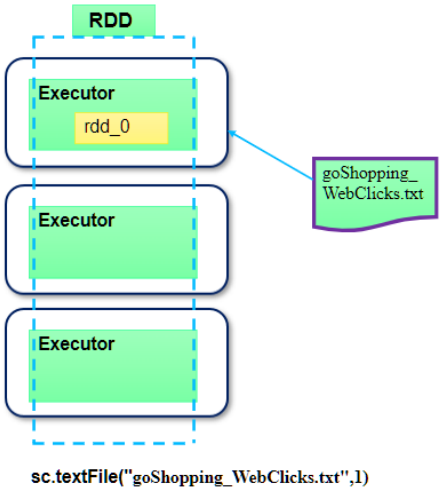
CREATING AN RDD – FILE PARTITIONING

Partitions - single file

- Partitions is done based on size.
- You can specify the number of partitions as shown below.

  textFile(filename, minPartitions)

- The default number of partitions: 2
- more number of partitions = more parallelization



sc.textFile("goShopping_WebClicks.txt",1)

CREATING AN RDD – USING TEXT FILE

Objective: Create an RDD for user activity (Use Local File System).

Action:

```
val LocalFSRdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
// URL of the local FS
LocalFSRdd.count() // count() is an action, counts the number of records.
```

Output:

```
scala> val LocalFSRdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
LocalFSRdd: org.apache.spark.rdd.RDD[String] = /home/vagrant/dataset/goShopping_WebClicks.dat MapPa
rtitionsRDD[1] at textFile at <console>:24

scala> LocalFSRdd.count()
res0: Long = 71492
```

CREATING AN RDD – USING TEXT FILE

Objective: Create an RDD for user activity (Use HDFS File System).

Action:

```
val HDFSRdd =
sc.textFile("hdfs://192.168.163.151:9000/data/goShopping_WebClicks.dat") // URL
of the HDFS path.
HDFSRdd.count()
```

Output:

```
scala> val HDFSRdd = sc.textFile("hdfs://192.168.163.151:9000/data/goShopping_WebClicks.dat")
HDFSRdd: org.apache.spark.rdd.RDD[String] = hdfs://192.168.163.151:9000/data/goShopping_WebClicks.d
at MapPartitionsRDD[3] at textFile at <console>:24

scala> HDFSRdd.count()
res1: Long = 71492
```

CREATING AN RDD – USING TEXT FILE

Objective: Create an RDD for user details and activity.

Action:

```
val FilesRdd = sc.textFile("/home/vagrant/dataset")
FilesRdd.count() //  count () is  an action.
```

Note: Directory dataset contains user activity and user details.

Output:

```
scala> val FilesRdd = sc.textFile("/home/vagrant/dataset")
FilesRdd: org.apache.spark.rdd.RDD[String] = /home/vagrant/dataset MapPartitionsRDD[7] at textFile at <console>:24

scala> FilesRdd.count()
res3: Long = 71515
```

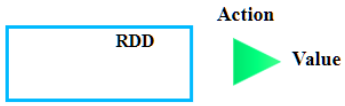
LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- Transformations
- Actions
- What is Pair RDDs
- Creating Pair RDDs
- Persisting RDDs
- Storage Level
- Accumulators
- Broadcast Variables

ACTIONS

- **Actions return values to the driver program.**
- **Common actions are:**
  - reduce(func)
  - collect()
  - count()
  - first()
  - take(n)
  - foreach(println)
  - saveAsTextFile(path)



ACTIONS – COLLECT()

Objective: Display the records of goShopping_WebClicks.dat file.

Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
rdd.collect()
```

Output:

```
scala> rdd.collect()
res4: Array[String] = Array("12/01/2016 17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com
/?product=clothing&producttype=newarrivals      283     google.co.jp    android ", "12/01/2016  17:16:13        128.230.24
7.37    get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=topbrands      19      diigo.comw
indows  ", "12/01/2016  17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothi
ng&producttype=allwesternwear   18      accuweather.com android ", "12/01/2016  17:16:13        128.230.247.37  get     21
5.82.23.2       https://www.goshopping.com/?product=clothing&producttype=shirts 25      cloudflare.com  windows ", "12/01/
2016    17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clothing&producttype=t
ops tees        36      harvard.edu     android ", "12/01/2016  17:16:13        128.230.247....
```

ACTIONS – COUNT()

Objective: Count the number of records of goShopping_WebClicks.dat file.

Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
rdd.count()
```

Output:

```
scala> val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
rdd: org.apache.spark.rdd.RDD[String] = /home/vagrant/dataset/goShopping_WebClicks.dat MapPartitionsRDD[13] at textFile at
 <console>:24

scala> rdd.count()
res6: Long = 71492
```

ACTIONS – TAKE(N)

Objective: Display first 2 records of goShopping_WebClicks.dat file.

Action:

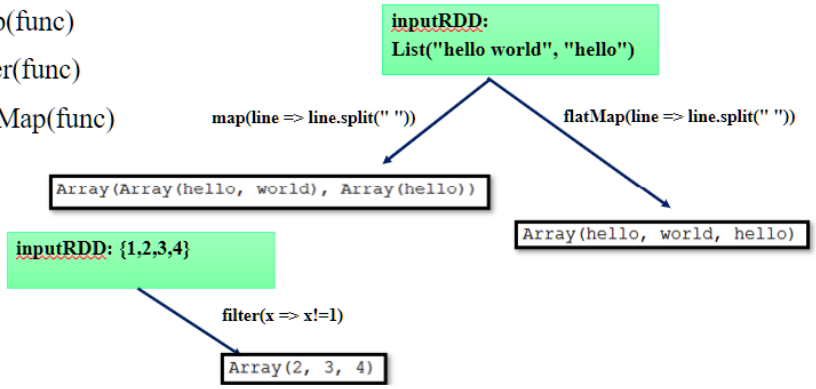
```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
rdd.take(2)
```

Output:

```
scala> rdd.take(2)
res0: Array[String] = Array("12/01/2016 17:16:13        128.230.247.37  get     215.82.23.2     https:/
/www.goshopping.com/?product=clothing&producttype=newarrivals   283     google.co.jp    android ", "12/
01/2016 17:16:13        128.230.247.37  get     215.82.23.2     https://www.goshopping.com/?product=clo
thing&producttype=topbrands     19      diigo.com       windows ")
```

TRANSFORMATIONS

- **Creating a new RDD from an existing RDD.**
- **Common transformations are:**
  - map(func)
  - filter(func)
  - flatMap(func)



TRANSFORMATIONS – MAP(FUNC)

Objective: Retrieve date, customer ip, time spent on shopping site .

Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
val split = rdd.map(line => line.split("\t"))
val result = split.map(field => (field(0),field(4),field(6)))
result.take(5).foreach(println)
```

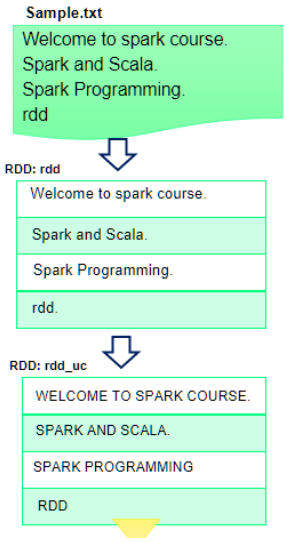
Output:

```
(12/01/2016,215.82.23.2,283)
(12/01/2016,215.82.23.2,19)
(12/01/2016,215.82.23.2,18)
(12/01/2016,215.82.23.2,25)
(12/01/2016,215.82.23.2,36)
```

LAZY EVALUATION

- **All transformations are lazy, in that they do not compute their results until an action is called.**

```
> val rdd = sc.textFile("sample.txt")

  val rdd_uc =rdd.map(line =>
  line.toUpperCase())

  rdd_uc.collect()
```

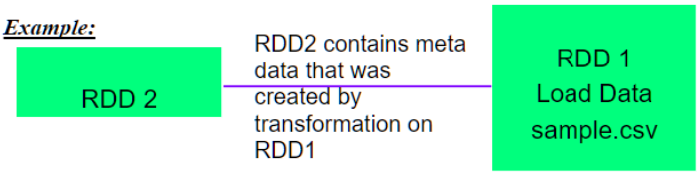


LAZY EVALUATION

RDD Lineage and toDebugString():

- **Spark maintains each RDDs lineage i.e. previous RDD which it depends.**
- **When an RDD is created, it just holds metadata.**
  - A transformation that created the RDD.
  - Its parent RDD from which it was created.

Example:

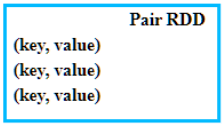


WHAT IS PAIR RDDS?

Special form of RDD.

Each element in an Pair RDD is a key-value pair.

Key-value can be of any type.



Common Pair RDDs are:

- groupByKey([numTasks])
- reduceByKey(func, [numTasks]):

Pair RDDs are useful:

- **sorting, grouping, etc..**

CREATING PAIR RDDS

To create Pair RDDs:

- get data in the form of key, value pair.

Commonly used functions to create Pair RDDs:

- groupByKey
- reduceByKey

PAIR RDDS – GROUPBYKEY([NUMTASKS])

Objective: Display the list of IP addresses for each country .

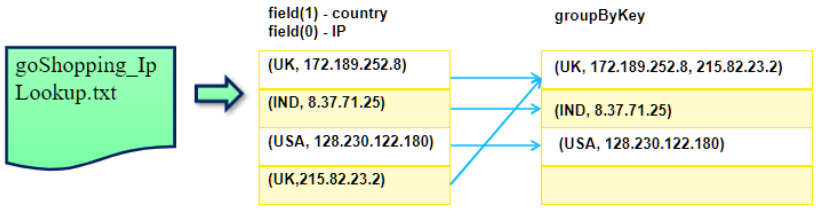
Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_IpLookup.txt")
val split = rdd.map(line => line.split(","))
val result = split.map(field => (field(1),field(0)))
result.groupByKey.take(10).foreach(print)
```

Output:

```
scala> result.groupByKey.take(10).foreach(print)
(UK,CompactBuffer(172.189.252.8, 215.82.23.2, 98.29.25.44, 68.199.40.156, 155.100.169.152, 38.68.15.223
, 70.209.14.54, 74.111.6.173))(IND,CompactBuffer(8.37.71.25, 8.37.71.69, 8.37.71.9, 8.37.71.57))(USA,Co
mpactBuffer(128.230.122.180, 128.122.140.238, 56.216.127.219, 54.114.107.209, 74.111.18.59, 8.37.70.170
, 8.37.70.77, 8.37.70.112, 8.37.70.226, 8.37.70.99, 8.37.71.43))
```

PAIR RDDS – GROUPBYKEY([NUMTASKS])



PAIR RDDS – REDUCEBYKEY(FUNC, [NUMTASKS])

Objective: Find total time spent on shopping site by each customer.

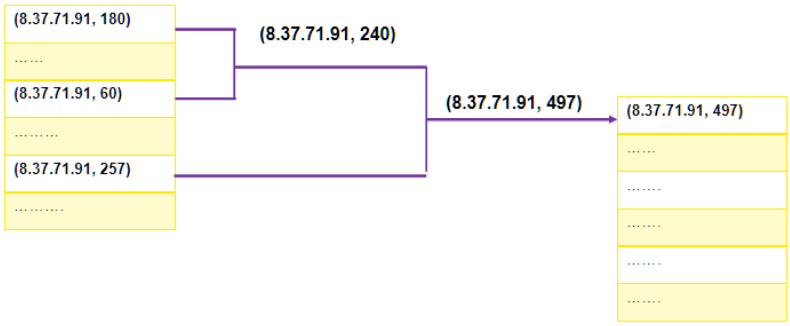
Action:

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
val split = rdd.map(line => line.split("\t"))
val result = split.map(field => (field(4),field(6).toInt)). reduceByKey((v1,v2) => v1 + v2)
result.take(5).foreach(println)
```

Output:

```
(38.68.15.223,253943)
(56.216.127.219,203646)
(70.209.14.54,1406717)
(128.230.122.180,14112)
(54.114.107.209,559909)
```

PAIR RDDS – REDUCEBYKEY(FUNC, [NUMTASKS])



PERSISTING RDDS

- **One of the important features of Spark is is persisting (or caching) a dataset in memory across operations.**

- **Persisting an RDD:**
  - each node stores any partitions of it that it computes in memory and reuses them in other actions on that dataset.
  - allows future actions to be much faster.
  - use persist() or cache() methods.

- **The difference between cache() and persist():**
- cache() == default storage level (MEMORY_ONLY).
- persist() == specify various storage levels.

STORAGE LEVELS

Storage Level	Meaning
MEMORY_ONLY	▪ Default level. ▪ Store RDD as deserialized Java objects in the JVM
MEMORY_AND_DISK	▪ Store RDD as deserialized Java objects in the JVM. ▪ If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
DISK_ONLY	▪ Store the RDD partitions only on disk.

PERSISTING RDDS - EXAMPLE

```
val rdd = sc.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat")
import org.apache.spark.storage.StorageLevel
rdd.persist(StorageLevel.MEMORY_ONLY)
```

ACCUMULATORS

Accumulators:

- **Variables that are used for aggregating information across the executors.**
- **Simple syntax for aggregating values from worker nodes back to the driver program.**

Use Case:

- **Find out the number of blank logs (blank lines) .**
- **Number of times the network failed.**
- **Number of times zero sales were recorded.**

ACCUMULATORS

Objective: Find the number of blank lines present in "goShopping_IpLookup.txt" .

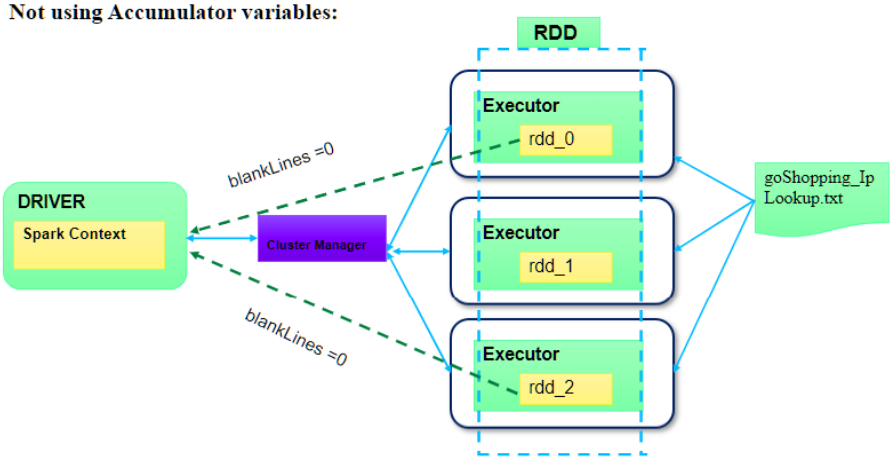
Input Data: "goShopping_IpLookup.txt"

```
172.189.252.8,UK,J5,Devon,38.955855,-77.447819

215.82.23.2,UK,J6,Dorset,39.961176,-82.998794

98.29.25.44,UK,J7,Down,41.49932,-81.694361
```

Number of blank lines: 2

ACCUMULATORS

Not using Accumulator variables:



ACCUMULATORS

Using Accumulator variables:

Action:

```
val blankLines = sc.accumulator(0)
sc.textFile("/home/vagrant/dataset/goShopping_IpLookup.txt").foreach{line =>if (line.length() == 0) blankLines += 1}
println("Blank lines: " + blankLines)
```

```
scala> val blankLines = sc.accumulator(0)
warning: there were two deprecation warnings; re-run with -deprecation for details
blankLines: org.apache.spark.Accumulator[Int] = 0

scala> sc.textFile("/home/vagrant/dataset/goShopping_IpLookup.txt").foreach{line =>if (line.length() == 0)
 blankLines += 1}

scala> println("Blank lines: " + blankLines)
Blank lines: 2
```

BROADCAST VARIABLES

Broadcast Variables:

- **To keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.**
- **Use SparkContext.broadcast(v) to create broadcast variable.**
- **Variable v acts as a broadcast variable.**
- **value() can be used to access the broadcast variable.**
- **Useful when tasks across multiple stages need the same data.**

BROADCAST VARIABLES

Objective:

Categorize the user IP as type "AccidentVisit" if time spent is less than 60 secs , else as type " PurposeVisit ".

" goShopping_WebClicks.dat "

BROADCAST VARIABLES

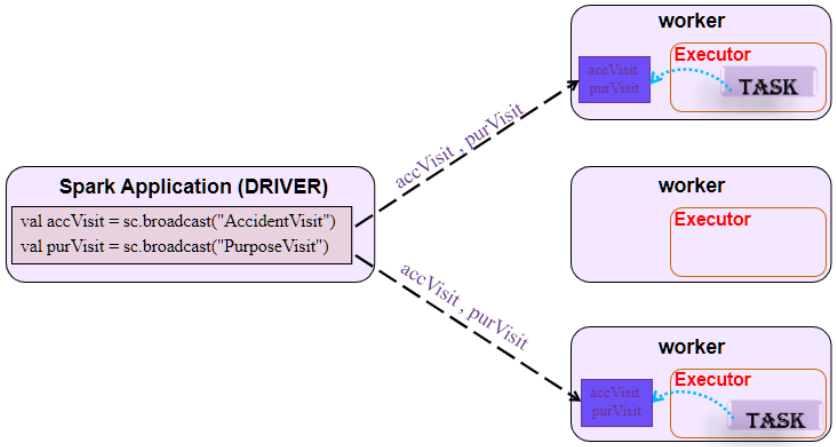
Action:

```
val accVisit = sc.broadcast("AccidentVisit")
val purVisit = sc.broadcast("PurposeVisit")
val rdd = sc.textFile("/home/vagrant/data/goShopping_WebClicks.dat",2)
val split = rdd.map(line => line.split("\t"))
val result = split.map(field =>
(field(4),if(field(6).toInt>=60){accVisit.value}else{purVisit.value}))
result.take(3).foreach(println)
```

Output:

```
scala> result.take(3).foreach(println)
(215.82.23.2,AccidentVisit)
(215.82.23.2,PurposeVisit)
(215.82.23.2,PurposeVisit)
```

BROADCAST VARIABLES



LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- What is Spark SQL?
- Overview
- Spark SQL
- Data Frames.
- Comparison between RDD, DataFrames and DataSets APIs .
- DataSets.

OVERVIEW – SPARK SQL

Click to enter the content

- Structured data processing.
- Information about the structure of both data and computation.
- Extra information is used to perform optimizations.
- Ways to interact with Spark SQL:
  - SQL
  - Dataset API

SPARKSESSION

- SparkSession
  - Entry point into all functionality in Spark.
- Use SparkSession.builder() to create SparkSession.

```
import org.apache.spark.sql.SparkSession
val spark = SparkSession
  .builder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()
// For implicit conversions like converting RDDs to DataFrames
import spark.implicits._
```

SPARK SQL

Spark SQL uses:

- To execute SQL queries.
- To read data from an existing Hive installation.

LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- What is DataFrames
- SparkSession
- Creating DataFrames
- Structure File
- Existing RDD
- Spark API's

WHAT IS DATAFRAMES?

- distributed collection of data organized into named columns.
- equivalent to a table in a relational database.
- can be constructed from
  - structured data files,
  - existing RDDs.
- DataFrame API is available in Scala, Java, Python, and R.

CREATING DATAFRAMES – USING STRUCTURE FILE

Dataset:

employee.json

```
{"empno":1001,"empname":"John","designation":"TL"}
{"empno":1002,"empname":"James","designation":"SSE"}
{"empno":1003,"empname":"Smith","designation":"AM"}
{"empno":1004,"empname":"Scott","designation":"SE"}
{"empno":1005,"empname":"Joshi","designation":"TL"}
{"empno":1006,"empname":"Jack","designation":"SSE"}
```

Objective: To create DataFrames.
Action:

```
val df = spark.read.json("/home/vagrant/dataset/employee.json")
df.show()
```

QUERYING DATAFRAMES

Two ways to query DataFrames:

- Untyped DataFrame Operations
- Apply SQL method
  - You can apply sql queries to query the data.

CREATING DATAFRAMES – USING STRUCTURE FILE

Output:

```
scala> val df = spark.read.json("/home/vagrant/dataset/employee.json")
df: org.apache.spark.sql.DataFrame = [designation: string, empname: string ... 1 more field]

scala> df.show()
+-----------+-------+-----+
|designation|empname|empno|
+-----------+-------+-----+
|         TL|   John| 1001|
|        SSE|  James| 1002|
|         AM|  Smith| 1003|
|         SE|  Scott| 1004|
|         TL|  Joshi| 1005|
|        SSE|   Jack| 1006|
+-----------+-------+-----+
```

UNTYPED DATAFRAME OPERATIONS

Objective: Select only the empname column.

Action:

```
df.select("empname").show()
```

Output:

```
scala> df.select("empname").show()
+-------+
|empname|
+-------+
|   John|
|  James|
|  Smith|
|  Scott|
|  Joshi|
|   Jack|
+-------+
```

UNTYPED DATAFRAME OPERATIONS

Objective: Select everybody, but increment the empno by 1.

Action:

```
df.select(df("empname"), df("empno") + 1).show()
```

Output:

```
scala> df.select(df("empname"), df("empno") + 1).show()
+-------+-----------+
|empname|(empno + 1)|
+-------+-----------+
|   John|       1002|
|  James|       1003|
|  Smith|       1004|
|  Scott|       1005|
|  Joshi|       1006|
|   Jack|       1007|
+-------+-----------+
```

UNTYPED DATAFRAME OPERATIONS

Objective: Select employee details where empno is greater than 1003.

Action:

```
df.filter(df("empno") > 1003).show()
```

Output:

```
scala> df.filter(df("empno") > 1003).show()
+-----------+-------+-----+
|designation|empname|empno|
+-----------+-------+-----+
|         SE|  Scott| 1004|
|         TL|  Joshi| 1005|
|        SSE|   Jack| 1006|
+-----------+-------+-----+
```

UNTYPED DATAFRAME OPERATIONS

Objective: Count employee by empname.

Action:

```
df.groupBy("empname").count().show()
```

Output:

```
scala> df.groupBy("empname").count().show()
+-------+-----+
|empname|count|
+-------+-----+
|  Scott|    1|
|  James|    1|
|   Jack|    1|
|  Smith|    1|
|  Joshi|    1|
|   John|    1|
+-------+-----+
```

DATAFRAMES – RUNNING SQL QUERIES

Objective: Display employee details.
Action:

```
val df = spark.read.json("/home/vagrant/dataset/employee.json")
df.createOrReplaceTempView("employee") // Create temporary view
val sqlDF = spark.sql("SELECT * FROM employee")
sqlDF.show()
```

Output:

```
+-----------+-------+-----+
|designation|empname|empno|
+-----------+-------+-----+
|         TL|   John| 1001|
|        SSE|  James| 1002|
|         AM|  Smith| 1003|
|         SE|  Scott| 1004|
|         TL|  Joshi| 1005|
|        SSE|   Jack| 1006|
+-----------+-------+-----+
```

CREATING DATAFRAMES – USING EXISTING RDDS

Dataset:
goShopping_WebClicks.dat
Objective: Display first five records of goShopping_WebClicks.txt
Action:

```
case class Clicks(date: String, time: String, hostIp:String,
csmethod:String,customerip:String,
url:String,timespent:String,redirectedFrom:String,deviceType:String)

val webclicksDF =
spark.sparkContext.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat").
map(_.split("\t")).map(attributes =>
Clicks(attributes(0),attributes(1),attributes(2),attributes(3),attributes(4),attributes(5)
,attributes(6),attributes(7),attributes(8))).toDF()
```

CREATING DATAFRAMES – USING EXISTING RDDS

Action:

```
webclicksDF.createOrReplaceTempView("clicks")
val ClicksDF = spark.sql("SELECT * from clicks limit 5")
ClicksDF.show()
```

Output:

```
+----------+--------+--------------+--------+-----------+--------------------+---------+---------------+----------+
|      date|    time|        hostIp|csmethod| customerip|                 url|timespent| redirectedFrom|deviceType|
+----------+--------+--------------+--------+-----------+--------------------+---------+---------------+----------+
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|      283|   google.co.jp|   android|
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|       19|      diigo.com|   windows|
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|       18|accuweather.com|   android|
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|       25| cloudflare.com|   windows|
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|       36|    harvard.edu|   android|
+----------+--------+--------------+--------+-----------+--------------------+---------+---------------+----------+
```

SPARK APIS

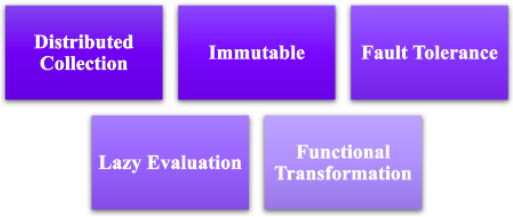
Spark provides three APIs.

- RDD
- DataFrames
- DataSets

SPARK APIS - RDD

- Main abstraction
- Distributed collection of elements, operated in parallel.

RDD Features:



SPARK APIS - RDD

RDD Limitations:

- No inbuilt optimization engine.
  - Developers responsibility to optimize each RDD based on its attributes.
- Handling structured data.
  - Unlike DataFrames, RDDs does not infer the ingested data schema. Users need to specify it.

SPARK APIS - DATAFRAMES

- distributed collection of data organized into named columns.

DataFrames Features:

- Uses catalyst optimizer for optimization.
- Hive Compatibility: Run unmodified hive queries.

SPARK APIS - DATAFRAMES

DataFrames Limitations:

- Compile-time type safety:
  - DataFrames API does not support compile time safety.

Example:

Dataset:

employee.json

```
{"empno":1001,"empname":"John","designation":"TL"}
{"empno":1002,"empname":"James","designation":"SSE"}
{"empno":1003,"empname":"Smith","designation":"AM"}
{"empno":1004,"empname":"Scott","designation":"SE"}
{"empno":1005,"empname":"Joshi","designation":"TL"}
{"empno":1006,"empname":"Jack","designation":"SSE"}
```

SPARK APIS - DATAFRAMES

Example:

Code:

```
val dataframe = spark.read.json("/home/vagrant/dataset/employee.json")
dataframe.filter("salary > 10000").show
```

It gives run time error instead of compile time error.

Note: Salary is not a member of employee.json

```
scala> dataframe.filter("salary > 10000").show
org.apache.spark.sql.AnalysisException: cannot resolve '`salary`' given input columns: [designation, empname, empno]; line 1 pos 0;
'Filter ('salary > 10000)
+- Relation[designation#131,empname#132,empno#133L] json
```

SPARK APIS - DATASETS

- Extension to DataFrames.
- Provides a type-safe programming interface.
- Strongly-typed, immutable collection of objects that are mapped to a relational schema.

DataSet Features:

- Best of both RDD and DataFrames.
- Provides compile time safety.

SPARK APIS - DATASETS

Compile time safety example:

```
case class Person(name: String, age: Long)
val caseClassDS = Seq(Person("James", 32)).toDS()
val result = caseClassDS.filter(p => p.age > 25)
result.show()
```

Output:

```
scala> result.show()
+-----+---+
| name|age|
+-----+---+
|James| 32|
+-----+---+
```

SPARK APIS - DATASETS

Compile time safety example:

```
val result = caseClassDS.filter(p => p.salary > 25)
```

Note: Salary is not a member of person class. It gives compile time error.

Output:

```
scala> val result = caseClassDS.filter(p => p.salary > 25)
<console>:27: error: value salary is not a member of Person
       val result = caseClassDS.filter(p => p.salary > 25)
                                              ^
```

LEARNING OBJECTIVES

At the end of this unit, you should be able to:

- Creating DataSets
- Using toDS()
- Using DataFrames
- Functional Transformations

CREATING DATASETS – USING TODS()

- Two ways to create DataSets.
  - Using toDS()
  - Using DataFrames

Using toDS():
Dataset:
goShopping_WebClicks.dat

Objective: Display first five records of goShopping_WebClicks.dat

SPARK APIS – DATASETS

Action:

```
case class Clicks(date: String, time: String, hostIp:String,
csmethod:String,customerip:String,
url:String,timespent:String,redirectedFrom:String,deviceType:String)

val webclicksDS =
spark.sparkContext.textFile("/home/vagrant/dataset/goShopping_WebClicks.dat").
map(_.split("\t")).map(attributes =>
Clicks(attributes(0),attributes(1),attributes(2),attributes(3),attributes(4),attributes(5)
,attributes(6),attributes(7),attributes(8))).toDS()

webclicksDS.take(5).foreach(println) // Apply RDD operations
```

SPARK APIS – DATASETS

Output:

```
scala> webclicksDS.take(5).foreach(println)
Clicks(2/11/2016,17:16:13,128.230.247.37,GET,215.82.23.2,https://www.goshopping.com/?product=CLOTHING&productType=NewArrivals,283,google
.co.jp,android)
Clicks(2/11/2016,17:16:13,128.230.247.37,GET,215.82.23.2,https://www.goshopping.com/?product=CLOTHING&productType=TopBrands,19,diigo.com
,Windows)
Clicks(2/11/2016,17:16:13,128.230.247.37,GET,215.82.23.2,https://www.goshopping.com/?product=CLOTHING&productType=AllWesternWear,18,accu
weather.com,android)
Clicks(2/11/2016,17:16:13,128.230.247.37,GET,215.82.23.2,https://www.goshopping.com/?product=CLOTHING&productType=Shirts,25,cloudflare.c
om,Windows)
Clicks(2/11/2016,17:16:13,128.230.247.37,GET,215.82.23.2,https://www.goshopping.com/?product=CLOTHING&productType=Tops_Tees,36,harvard.e
du,android)
```

CREATING DATASETS – USING DATAFRAMES

Using DataFrames:
Dataset:
goShopping_WebClicks.dat

Objective: Display first five records of goShopping_WebClicks.dat

CREATING DATASETS – USING DATAFRAMES

Action:

```
import org.apache.spark.sql.types._;

val date=StructField("date",DataTypes.StringType)
val time=StructField("time",DataTypes.StringType)
val hostIp=StructField("hostIp",DataTypes.StringType)
val csmethod=StructField("csmethod",DataTypes.StringType)
val customerip=StructField("customerip",DataTypes.StringType)
val url=StructField("url",DataTypes.StringType)
val timespent=StructField("timespent",DataTypes.StringType)
val redirectedFrom=StructField("redirectedFrom",DataTypes.StringType)
val deviceType=StructField("deviceType",DataTypes.StringType)

val fields =
Array(date,time,hostIp,csmethod,customerip,url,timespent,redirectedFrom,deviceT
ype)
val schema = StructType(fields)
```

CREATING DATASETS – USING DATAFRAMES

Action:

```
case class Clicks(date: String, time: String, hostIp:String,
csmethod:String,customerip:String,
url:String,timespent:String,redirectedFrom:String,deviceType:String)


val webClicksDs= spark.read.schema(schema).option("delimiter",
"\t").csv("/home/vagrant/dataset/goShopping_WebClicks.dat").as[Clicks]

webClicksDs.createOrReplaceTempView("clicks")

val results = spark.sql("SELECT * FROM clicks limit 5")

results.show()
```

CREATING DATASETS – USING DATAFRAMES

Output:

```
+----------+--------+--------------+--------+-----------+--------------------+---------+---------------+----------+
|      date|    time|        hostIp|csmethod| customerip|                 url|timespent| redirectedFrom|deviceType|
+----------+--------+--------------+--------+-----------+--------------------+---------+---------------+----------+
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|      283|   google.co.jp|   android|
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|       19|      diigo.com|   windows|
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|       18|accuweather.com|   android|
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|       25| cloudflare.com|   windows|
|12/01/2016|17:16:13|128.230.247.37|     get|215.82.23.2|https://www.gosho...|       36|    harvard.edu|   android|
+----------+--------+--------------+--------+-----------+--------------------+---------+---------------+----------+
```

DATASETS – FUNCTIONAL TRANSFORMATIONS

Functional Transformations:

- The Dataset API supports functional transformations(filter, map) much like the RDD API.
- Transform one Dataset into another Dataset
- These operations have compile-time type safety.

DATASETS - FUNCTIONAL TRANSFORMATIONS

Objective: Display users details who have spent less than 100.

Action:

```
val results = webClicksDs. filter($"timespent"< "100")

results.show()
```

CREATING DATASETS – USING DATAFRAMES

Output:

```
scala> results.show()
+---------+--------+---------------+---------+---------------+--------------------+---------+--------------------+----------+
|     date|    time|         hostIp|csmethod|     customerip|                 url|timespent|      redirectedFrom|deviceType|
+---------+--------+---------------+---------+---------------+--------------------+---------+--------------------+----------+
|2/11/2016|17:16:50|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|       10|          devhub.com|   android|
|2/11/2016|17:16:50|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        1|           opera.com|       mac|
|2/11/2016|17:16:50|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        0|            loc.gov|   android|
|2/11/2016|17:16:50|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        1|networksolutions.com|       mac|
|2/11/2016|17:16:50|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        0|            nba.com|       mac|
|2/11/2016|17:16:50|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|       10|          github.io|       mac|
|2/11/2016|17:16:59|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        1|            ucoz.ru|     linux|
|2/11/2016|17:16:59|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        0|            mapy.cz|     linux|
|2/11/2016|17:16:59|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|       10|       pinterest.com|       mac|
|2/11/2016|17:16:59|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        1|        geocities.jp|     linux|
|2/11/2016|17:16:59|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        1|     stumbleupon.com|     linux|
|2/11/2016|17:17:12|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        1|           ucoz.com|   Windows|
|2/11/2016|17:17:12|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        1|         storify.com|       mac|
|2/11/2016|17:17:12|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        1|         unesco.org|     linux|
|2/11/2016|17:17:12|128.230.247.37|      GET|128.122.140.238|https://www.gosho...|        1|           pbs.org|   Windows|
|2/11/2016|18:02:51|128.230.247.37|      GET|   74.111.6.173|https://www.gosho...|       10|       cbslocal.com|       mac|
```

DATASETS - FUNCTIONAL TRANSFORMATIONS

Objective: Display only the customerip.
Action:

```
val results = webClicksDs. map(_.customerip)
results.show()
```

Output:

```
scala> results.show()
+-------------+
|        value|
+-------------+
|215.82.23.2|
|215.82.23.2|
|215.82.23.2|
|215.82.23.2|
|215.82.23.2|
|215.82.23.2|
|215.82.23.2|
|215.82.23.2|
|215.82.23.2|
|215.82.23.2|
|215.82.23.2|
```