



Apache Spark™



Error/
Warning



Information



Flashback



Class Exercise

Table of Content

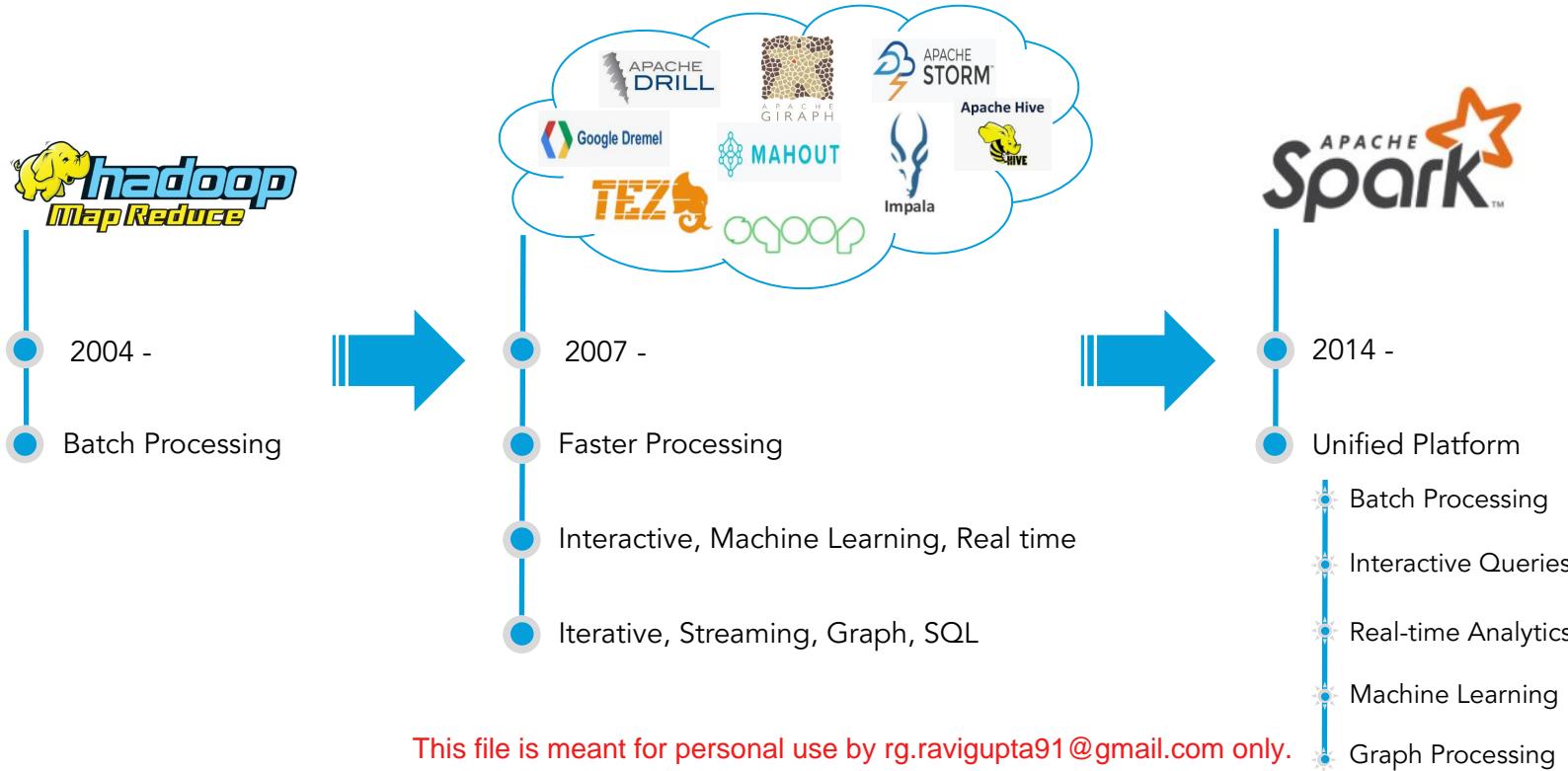
- What is Apache Spark
- MapReduce vs Spark
- Apache Spark Ecosystem
- History of Apache Spark
- Spark Pillars
 - Resilient Distributed Dataset (RDD)
 - Direct Acyclic Graph (DAG)
- RDD Operations
- Spark Cluster Architecture
- Spark Application Lifecycle
- Terminologies

Apache Spark is a unified analytics engine for
large-scale data processing and machine learning

Unified Analytics Engine

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Evolution of Unified Analytics Engine



Data Growth

- Worldwide data is expected to hit 175 zettabytes by 2025
- 90 ZB of this data will be from IoT devices in 2025
- 71% of enterprises reported that unstructured data was growing “much faster” than other business data
- 80% of data will be unstructured by 2025
- On top of business documents, video and audio are added new content such as social media, IoT, streaming and geo data
- There will be 4.8 billion internet users by 2022, up from 3.4 billion in 2017
- 90% of all data in existence today was created in the past two years
- More than 200 billion devices are projected to be generating data in the IoT trend

Measurement of Data

Abbreviation	Unit	Equivalent
B	Byte	8 Bits
KB	Kilobyte	1024 Bytes
MB	Megabyte	1024 Kilobytes
GB	Gigabyte	1024 Megabytes
TB	Terabyte	1024 Gigabytes
PB	Petabyte	1024 Terabytes
EB	Exabyte	1024 Petabytes
ZB	Zettabyte	1024 Exabytes
YB	Yottabyte	1024 Zettabytes
BB	Brontobyte	1024 Yottabytes

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Apache Spark



- Open Source from Apache Software Foundation
- Flexibility – Scala, Python, Java, SQL and R
- In-memory, Distributed and Parallel Processing
- Batch and Streaming
- Ease of use
- Better Analytics – SQL queries, ML algorithms
- Runs on Standalone Cluster, Apache Hadoop YARN, Apache Mesos, Kubernetes
- Connects to hundreds of data sources

Building Apache Spark



Total Lines :	1,445,443	Code Lines :	918,529	Percent Code Lines :	63.5%	
Number of Languages :	14	Total Comment Lines :	344,272	Percent Comment Lines :	23.8%	
		Total Blank Lines :	182,642	Percent Blank Lines :	12.6%	
Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
Scala	678,509	207,413	23.4%	124,395	1,010,317	<div style="width: 69.9%;"></div> 69.9%
Python	100,366	70,821	41.4%	30,735	201,922	<div style="width: 14.0%;"></div> 14.0%
Java	77,881	33,546	30.1%	16,055	127,482	<div style="width: 8.8%;"></div> 8.8%
SQL	25,279	12,611	33.3%	5,265	43,155	<div style="width: 3.0%;"></div> 3.0%
R	16,917	14,716	46.5%	3,709	35,342	<div style="width: 2.4%;"></div> 2.4%
XML	8,793	1,387	13.6%	471	10,651	<div style="width: 0.7%;"></div> 0.7%
JavaScript	4,744	1,136	19.3%	604	6,484	<div style="width: 0.4%;"></div> 0.4%
shell script	3,577	2,224	38.3%	983	6,784	<div style="width: 0.5%;"></div> 0.5%
CSS	1,265	225	15.1%	280	1,770	<div style="width: 0.1%;"></div> 0.1%
HTML	1,025	54	5.0%	55	1,134	<div style="width: 0.1%;"></div> 0.1%
Ruby	201	75	27.2%	67	343	<div style="width: 0.0%;"></div> 0.0%
DOS batch script	36	26	41.9%	17	79	<div style="width: 0.0%;"></div> 0.0%
C	19	20	51.3%	10	49	<div style="width: 0.0%;"></div> 0.0%
Make	17	21	55.3%	9	47	<div style="width: 0.0%;"></div> 0.0%
Totals	918,529	344,275	182,655	1,445,559		

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

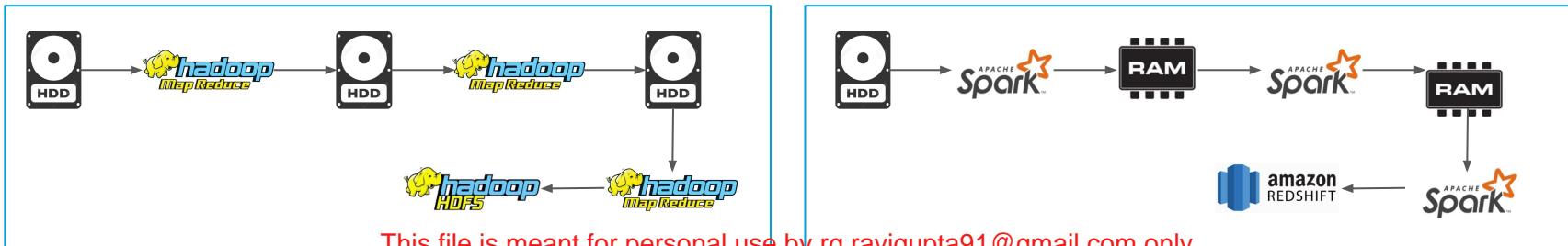
Source : https://www.openhub.net/p/apache-spark/analyses/latest/languages_summary

MapReduce vs Spark

Factors	Apache Spark	MapReduce
Speed	10 – 100 times faster	Faster than traditional systems
Written using	Scala (69.9%), Python (14%), Java (8.8%), SQL (3%), R (2.4%)	Java
Data Processing	Batch / Iterative / Interactive / Real-time / Graph	Batch
Complexity	Compact and Simple to use	Complex and Lengthy
Cache	In-memory/Disk Caching	More Disk IO and Minimal Caching

Beyond MapReduce

- MapReduce - Batch Processing
- MapReduce - No support for Real-time, Interactive, In-memory, Machine Learning and Graph data
- Spark - To handle all workloads. It offers
 - In-memory data caching - scan HDD only once, then all IO operations performed in RAM
 - Efficient Pipeline
 - Lazy Evaluation - Execution will not start until an action is triggered



This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Apache Spark Ecosystem

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Apache Spark Ecosystem

Spark SQL
+
DataFrames

Batch
+
Streaming

ML Lib
+
Machine Learning

GraphX

Spark Core API

Scala

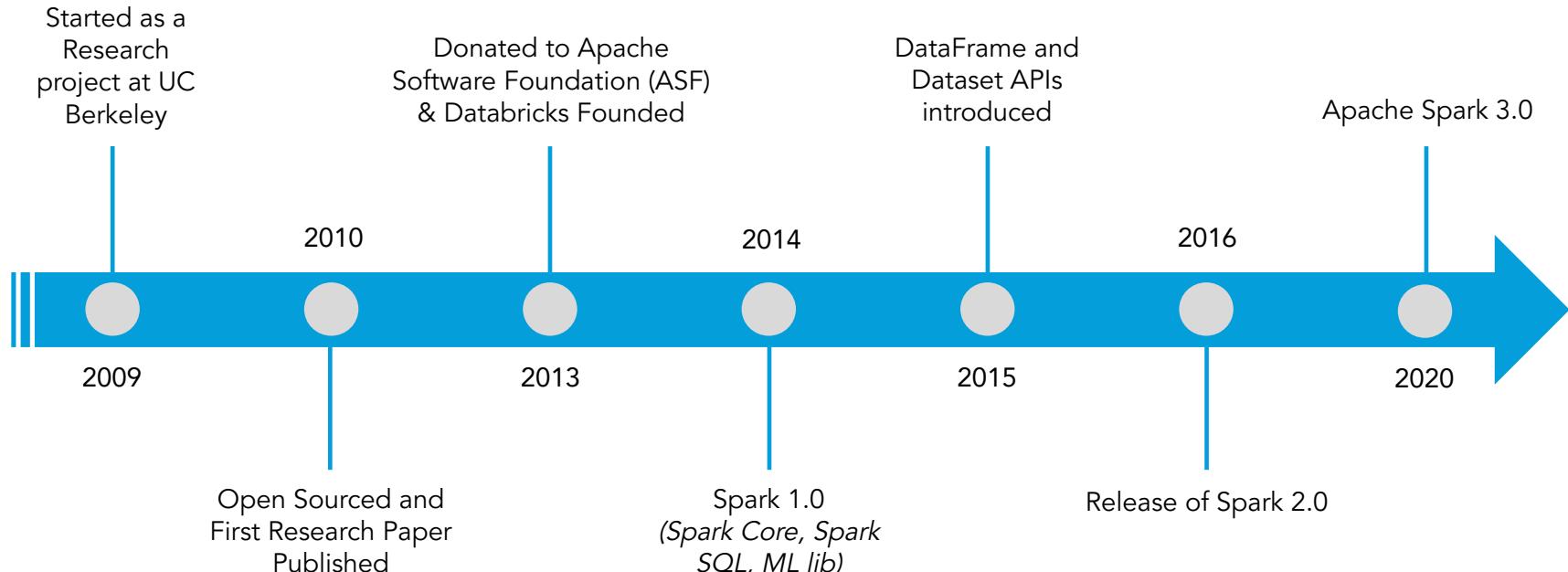
Python

SQL

Java

R

Apache Spark History



Spark - Pillars

- Resilient Distributed Dataset (RDD)
- Direct Acyclic Graph (DAG)

Resilient Distributed Dataset (RDD)

- Data read from a file will form a single RDD

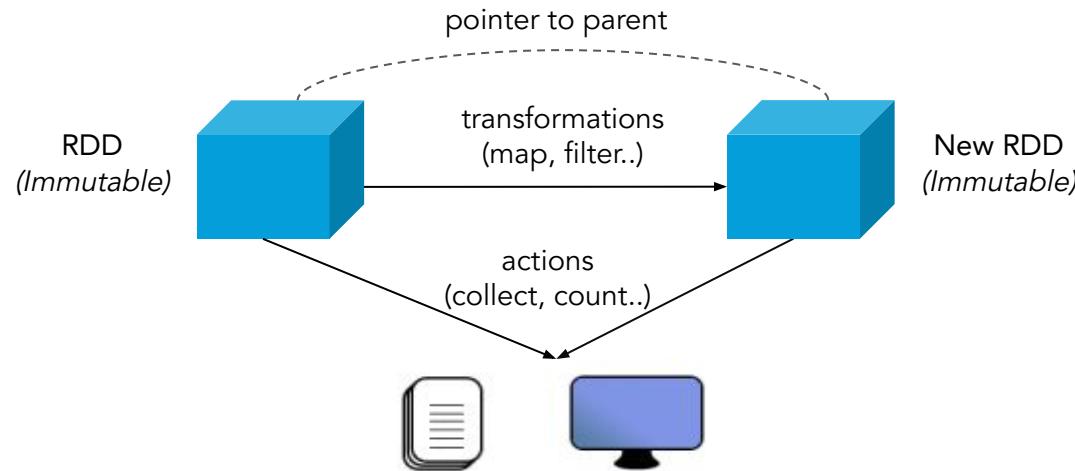
```
1 creditcardRDD = sc.textFile("/FileStore/tables/SalesTraining.csv")
2
3 print("Number of records :", creditcardRDD.count())
```

- RDDs are divided into logical partitions, which are stored on different worker machines of the cluster
- RDD is an immutable, distributed across nodes
- Fault tolerant – using DAG (Lineage)
- Partitions can be persisted in-memory or disk
- Lazily Evaluated and Executed

```
1 creditcardRDD.collect()
```

Resilient Distributed Dataset (RDD)

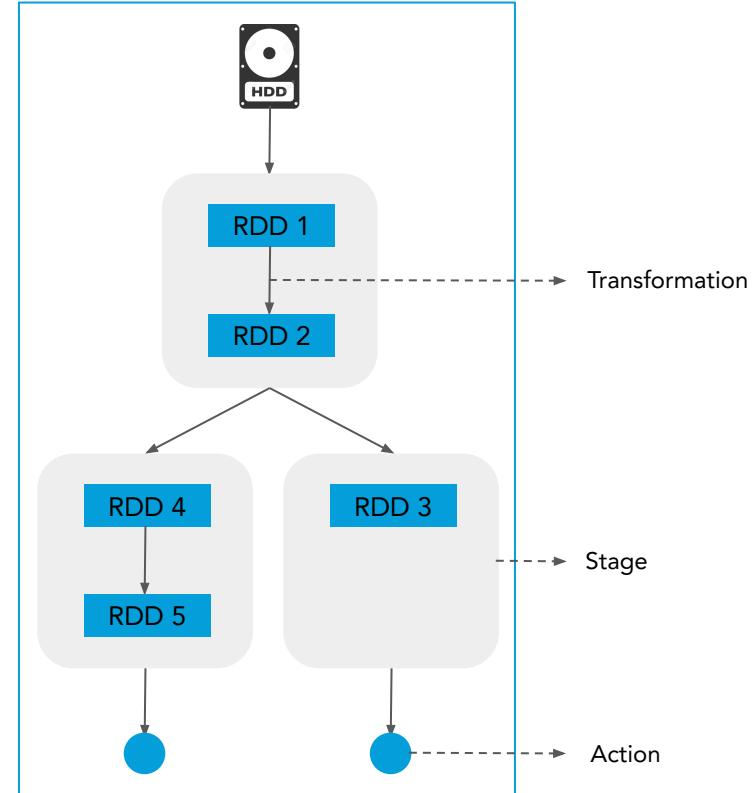
- Resilient (Fault Tolerant) – when the data in memory is lost, it can be recreated
- Distributed - stored in memory across the worker nodes of the cluster
- Dataset - file/created programmatically
- RDDs are the fundamental unit of data in Spark



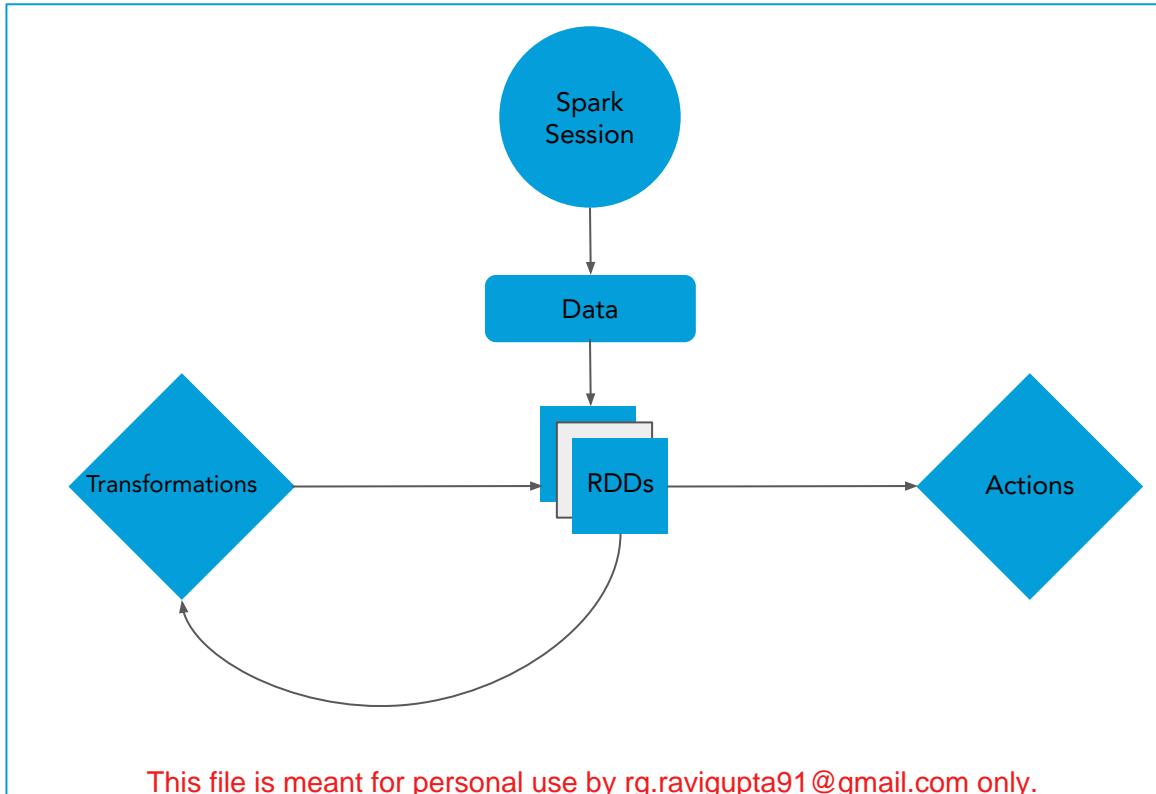
This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Direct Acyclic Graph (DAG)

- Sequence of computations performed on data
 - Node - Where RDD partitions resides
 - Edge - Transformation of top of data
 - Acyclic - Can't return to the old partition
 - Direct - Transformation is an action that transitions data partition state (from A to B)
- Lineage - Shows dependency between RDDs



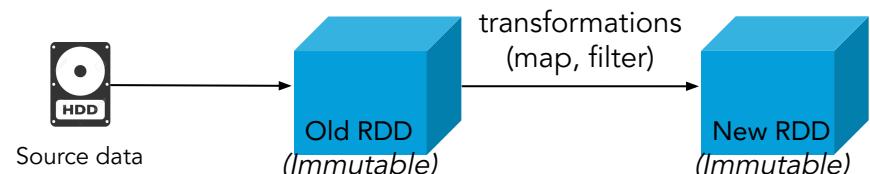
Spark Program Lifecycle



This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

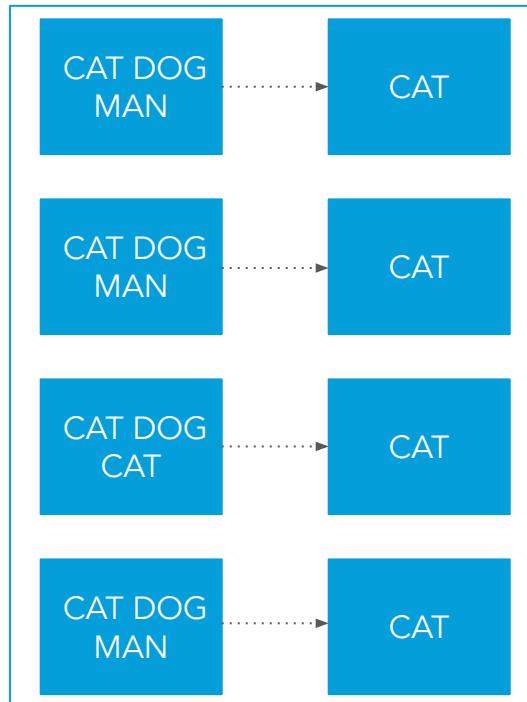
RDD Operations - Transformations

- Transformations create a new RDD from an existing RDD
- RDDs are immutable
 - Data in an RDD can't be changed
 - Transformations are performed in sequence to modify the data as needed
- Common Transformations
 - map() - creates a new RDD by performing a function on each record of the parent RDD
 - filter() - applies a boolean function on each record of the parent RDD to include/exclude records and creates a resultant RDD
- Types
 - Narrow Transformations
 - Wide Transformations

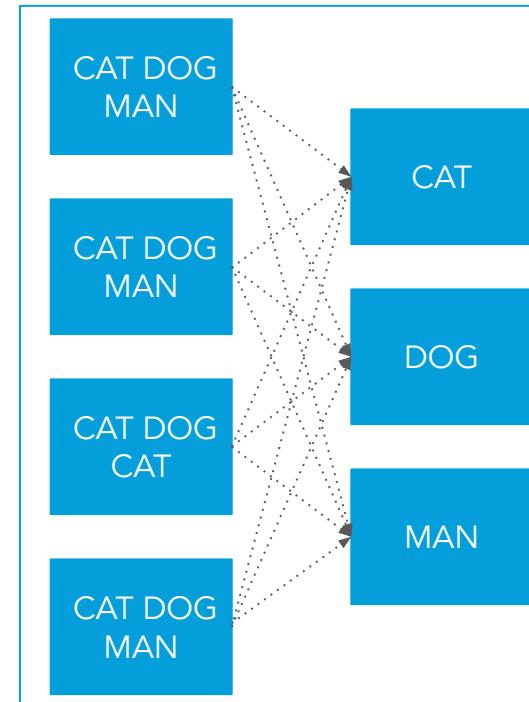


Transformations

Narrow Transformation (filter)



Wide Transformation (reduceByKey)



Transformations

- The following table lists some of the common *transformations* supported by Spark

Transformation	Meaning
<code>map(func)</code>	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i>
<code>filter(func)</code>	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true
<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item)
<code>coalesce(numPartitions)</code>	Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset

Transformations



Transformation	Meaning
<code>mapPartitions(func)</code>	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T
<code>mapPartitionsWithIndex(func)</code>	Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T
<code>groupByKey([numTasks])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs
<code>reduceByKey(func, [numTasks])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type <code>(V,V) => V</code> . Like in groupByKey, the number of reduce tasks is configurable through an optional second argument

This file is meant for personal use by rg.ravigupta91@gmail.com only.

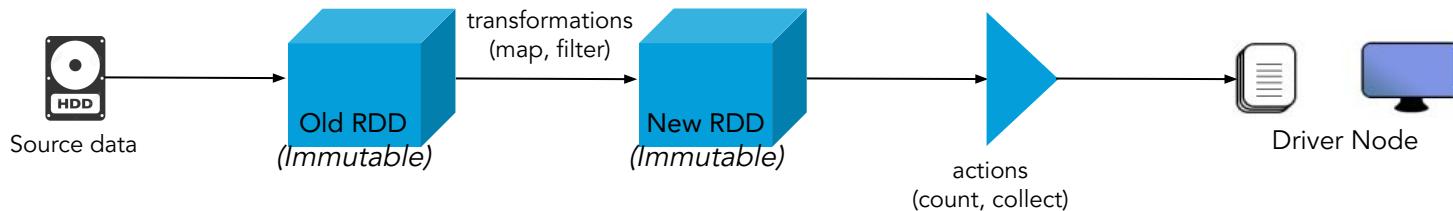
Sharing or publishing the contents in part or full is liable for legal action.

Transformations

Transformation	Meaning
<code>repartition(<i>numPartitions</i>)</code>	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network
<code>aggregateByKey(<i>zeroValue</i>)(<i>seqOp</i>, <i>combOp</i>, [<i>numTasks</i>])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value
<code>sortByKey([<i>ascending</i>], [<i>numTasks</i>])</code>	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument
<code>distinct([<i>numTasks</i>])</code>	Return a new dataset that contains the distinct elements of the source dataset

RDD Operations - Actions

- Actions return a value to the Driver program after running a computation on the dataset
- Some common actions are
 - count() - returns the number of elements
 - take(n) - returns an array to the Driver program having first 'n' elements
 - collect() - returns an array of all the elements
 - saveAsTextFile(filename) - save the elements to text file



Actions

- The following table lists some of the common *actions* supported by Spark

Actions	Meaning
reduce(<i>func</i>)	Aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data
count()	Return the number of elements in the dataset
first()	Return the first element of the dataset (similar to take(1))
take(<i>n</i>)	Return an array with the first <i>n</i> elements of the dataset

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Actions

Actions	Meaning
takeSample(<i>withReplacement</i> , <i>num</i> , [<i>seed</i>])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs
takeOrdered(<i>n</i> , [<i>ordering</i>])	Return the first <i>n</i> elements of the RDD using either their natural order or a custom comparator
saveAsTextFile(<i>path</i>)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call <code>toString</code> on each element to convert it to a line of text in the file
saveAsSequenceFile(<i>path</i>) (Java and Scala)	Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDs of key-value pairs that implement Hadoop's Writable interface

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Actions



Actions	Meaning
saveAsObjectFile(<i>path</i>) (Java and Scala)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using <code>SparkContext.objectFile()</code>
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key

A Sample Spark Application



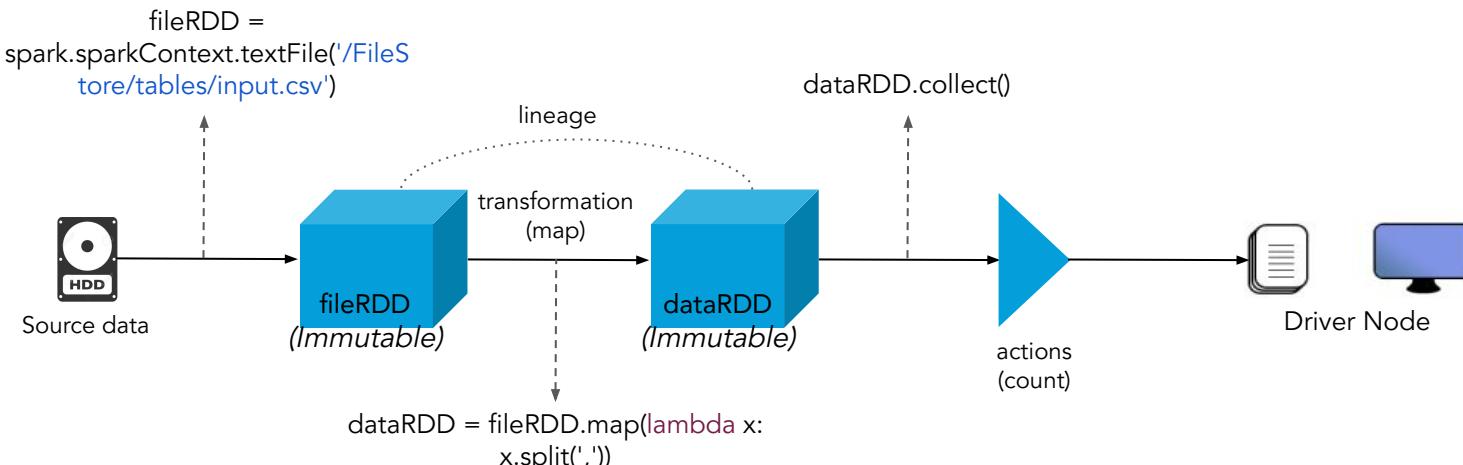
```
1 from operator import add..... → Import library  
2  
3 #fileRDD = sc.textFile('/FileStore/tables/input.csv')..... → RDDs  
4 fileRDD = spark.sparkContext.textFile('/FileStore/tables/input.csv')  
5  
6 print(type(fileRDD))  
7  
8 dataRDD = fileRDD.map(lambda x: x.split(','))..... → Transformation  
9 dataRDD.collect()  
10  
11 reduceRDD = dataRDD.map(lambda x: (x[1],1)).reduceByKey(add)..... → Transformation  
12  
13 dataRDD.filter(lambda x: 'Text' in x[1]).collect()..... → Transformation & Action  
14  
15 reduceRDD.collect()  
16 reduceRDD.cache()..... → In-memory  
17  
18 reduceRDD.count()..... → Action
```

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

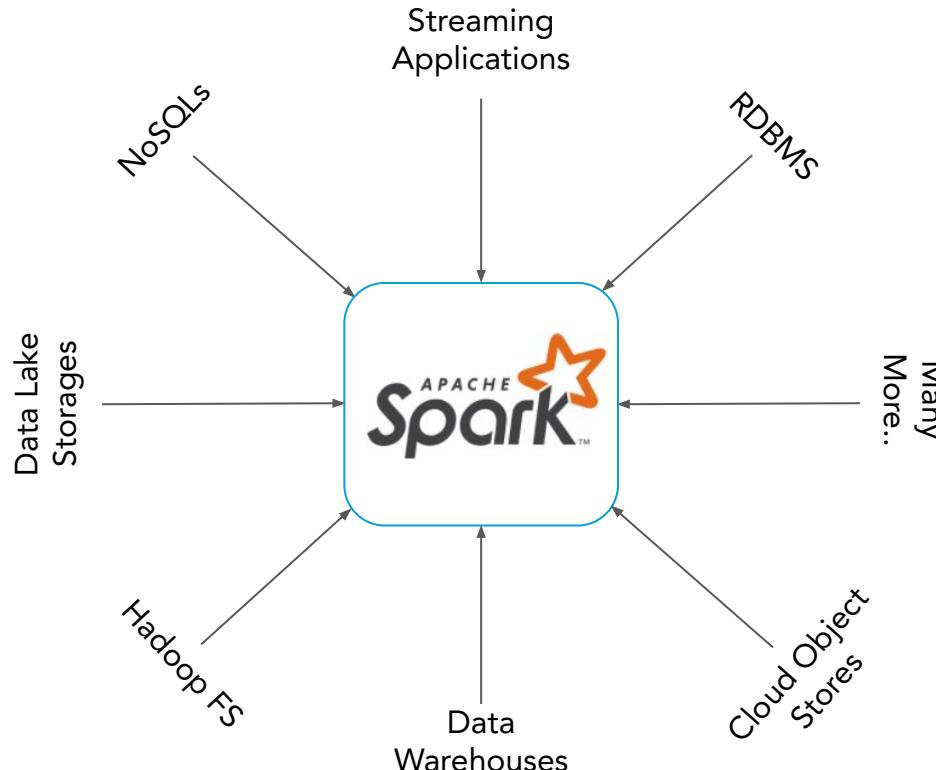
RDD Lineage

- When a transformation is called, Spark does not execute it immediately (lazy evaluation), instead it creates a lineage
- Lineages in the form of DAG, used by the Driver program to perform sequence of computations on data



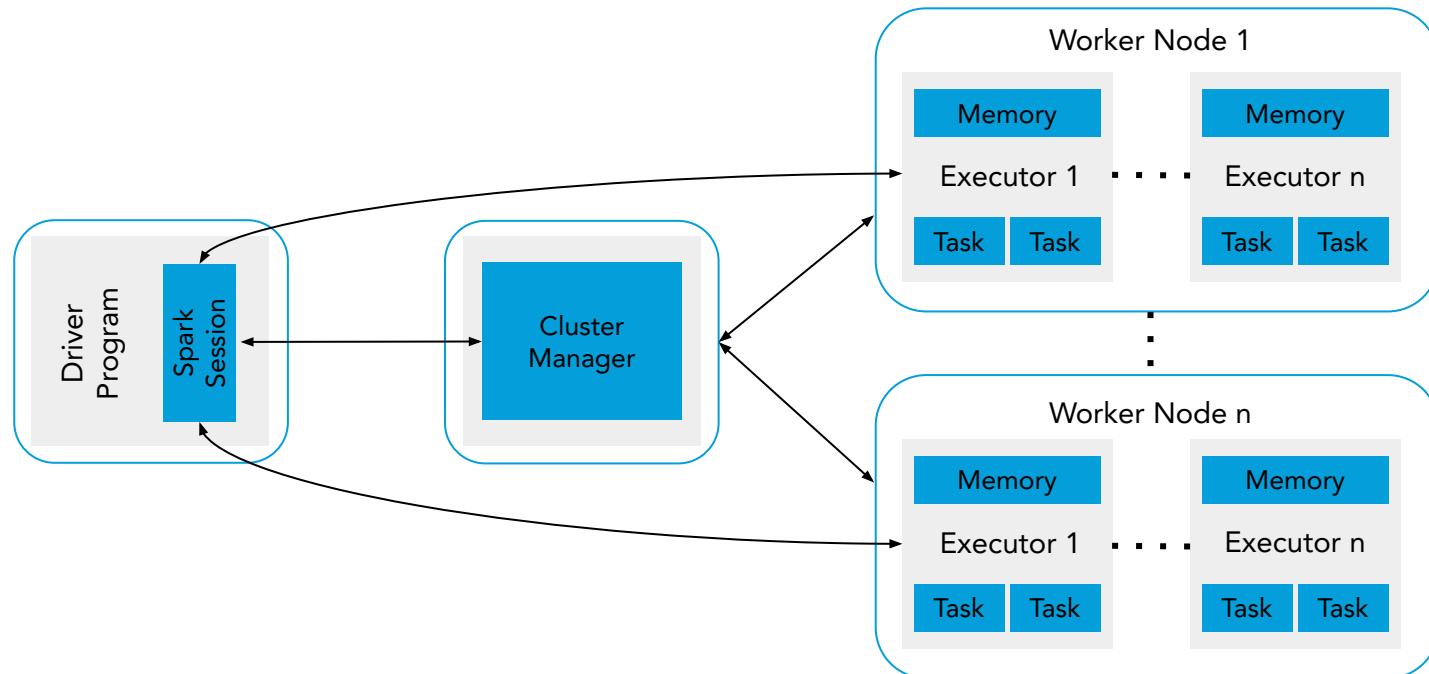
This file is meant for personal use by rg.ravigupta91@gmail.com only.
 Sharing or publishing the contents in part or full is liable for legal action.

Spark - Data Pipelines



This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Spark Cluster Architecture



This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Apache Spark Architecture

- Master-Slave Architecture
- One central coordinator called Spark Driver, which communicates with Workers
- Each Worker Node consists one or more Executor(s) responsible for executing the tasks (Transformations, Actions) assigned by the Driver program
- Working combination of Driver and Executors makes the Spark application
- The Spark application relies on Cluster Manager for allocating Worker Nodes. The Cluster Manager can be:
 - Apache Yarn
 - Standalone Spark Cluster
 - Apache Mesos
 - Kubernetes

Apache Spark Architecture - Driver Program



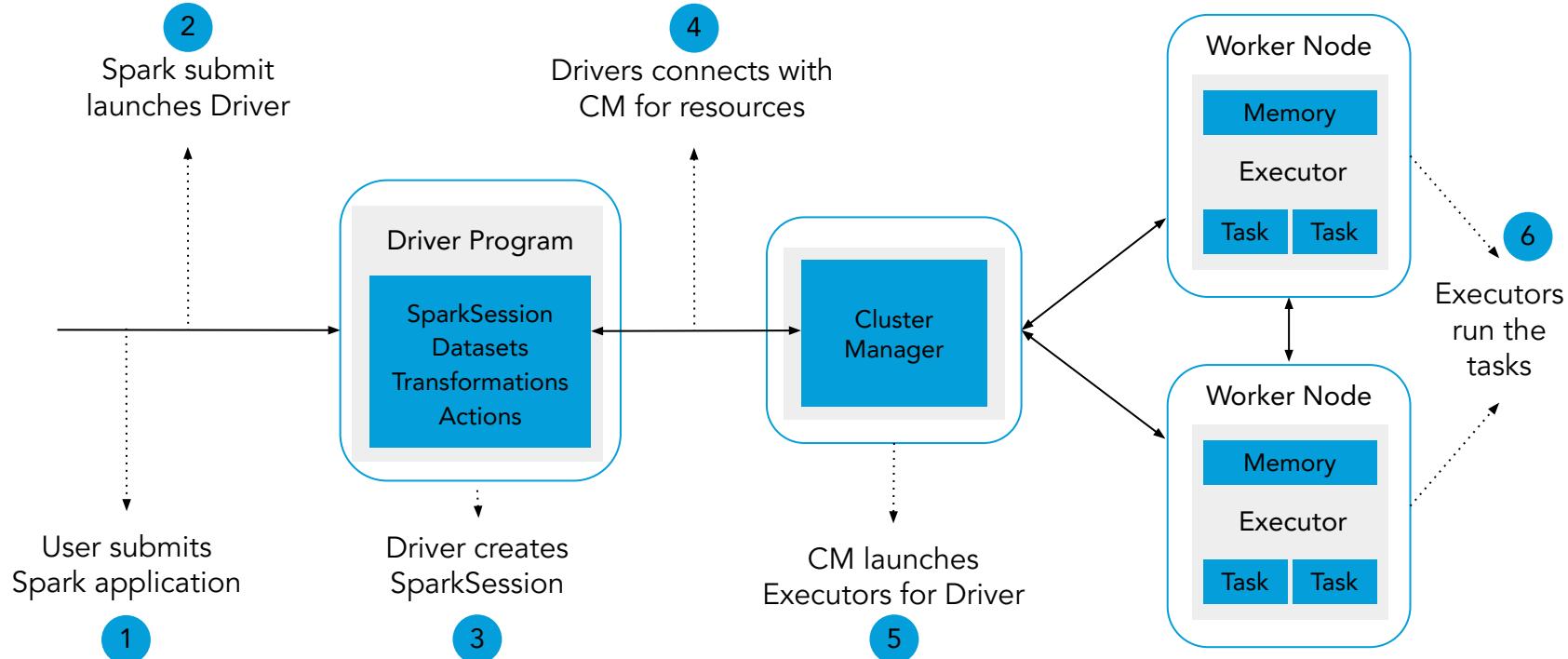
- The Driver program creates SparkSession object, which allocates and coordinates tasks of the Spark application
- Converts the code to tasks and determines number of tasks
- Helps creating DAG, lineage, logical and physical plans
- Connects with Cluster Manager, which allocates resources (in Worker node) for the Spark application
- Acquires Executors in Worker Nodes
- Sends the Application code and tasks to the Executors to run
- Works with Executors for completing the Spark application

Apache Spark Architecture - Executors



- Executors resides in the Worker node
- Each Executor has its own computation resources for executing the tasks
- Worker node can have one or more Executors
- Responsible for executing the tasks assigned by the Driver program
- Distributed Parallel Processing - Each Executor runs the code individually and parallelly
- Sends the output to the Driver program
- Caches the data in-memory, which enables faster processing
- Upon completion of the Spark application, all the assigned Executors are released

Spark Application - Life Cycle



When Driver main() exits, all the resources are released

This file is meant for personal use by rg.ravigupta91@gmail.com only.
 Sharing or publishing the contents in part or full is liable for legal action.

Terminologies

- Driver Program
 - Java Virtual Machine (JVM) allocated to start the execution of main()
 - Creates SparkSession
- SparkSession
 - Tells how to access the cluster components
- Cluster Manager
 - Service to manage and allocate the resources on the cluster
- Deploy Mode
 - cluster - Driver program runs in one of the cluster Worker nodes
 - client - Driver runs outside of cluster

Terminologies

- Worker Node
 - A place, where Executor resides
- Executor
 - JVM launched by the Worker node to execute the tasks from Driver
 - Will have computation resources
 - Keeps data in memory or disk when reading from external resources
- Job
 - Collection of Stages
- Stage
 - Each Job can have one or more Stages
 - Stages are executed sequentially
 - Each Stage has tasks
- Task
 - Unit of work, which is sent to the Executor
- Partition
 - Data unit what will be handled parallelly

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.



Databricks



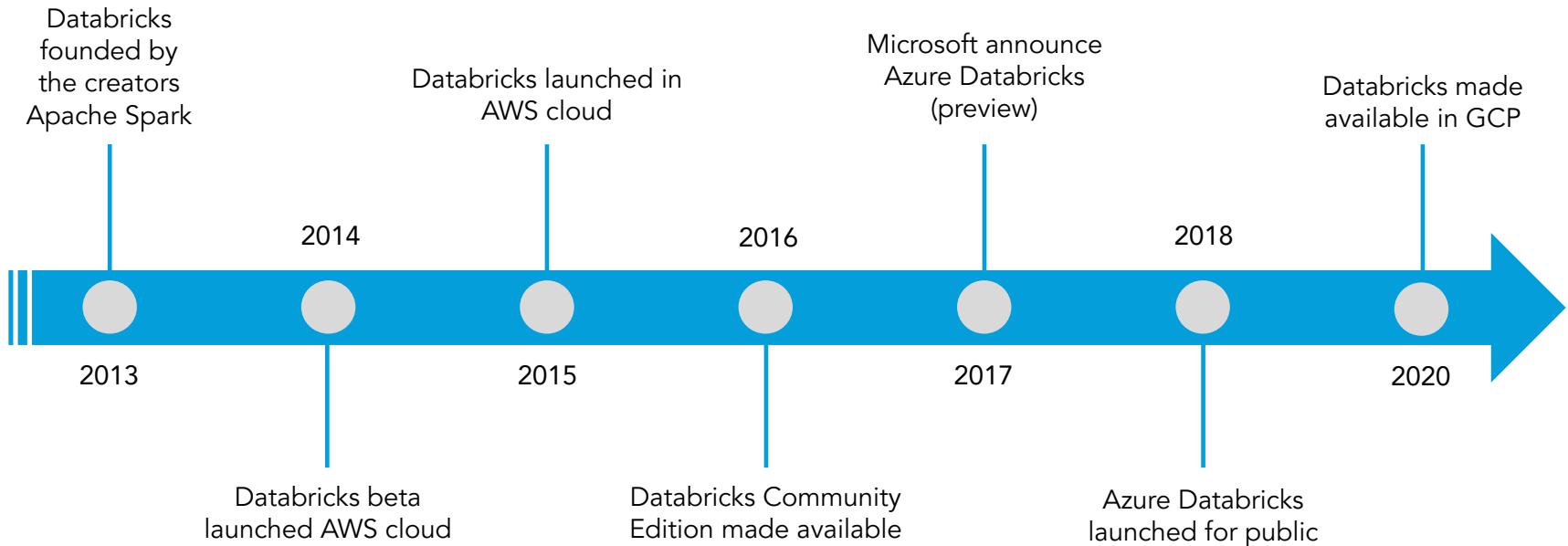
Table of Content

- What is Databricks?
- Databricks History
- Working with Databricks Community Edition
 - Spin up a Cluster
 - Create a new Notebook
 - Writing Spark Programs
 - Uploading files to DBFS
- How Spark Works in the Background?
- Class Exercise

Databricks

- Databricks is headquartered in San Francisco, California and was founded by the creators of open source projects, including Apache Spark, Delta Lake, MLflow and Koalas
- Cloud based Data Analytics platform for processing and transforming large quantities of data
- Apache-Spark based platform
- Runs a distributed system behind the scenes to enable distributed parallel processing for handling massive workloads
- Available in AWS, Azure and GCP
- Supports commonly used programming languages like Scala, Python, R, and SQL
- Integrates easily with other Cloud services
- Supports various file formats, connects with hundred of data sources
- Growing community, extensive documentation and support
- Thousands of global customers

Databricks History





Databricks

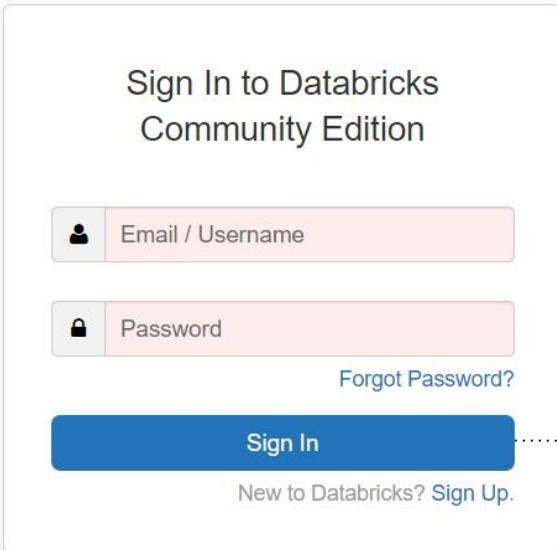
Community Edition



This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Databricks - Sign Up

1



The image shows the Databricks Community Edition sign-in page. At the top left is the Databricks logo with the text "Community Edition". Below it is the heading "Sign In to Databricks Community Edition". There are two input fields: one for "Email / Username" with a person icon and another for "Password" with a lock icon. Below the password field is a "Forgot Password?" link. A large blue "Sign In" button is at the bottom left. To its right, a dotted arrow points to the right, with the text "Click 'Sign In'" next to it. At the bottom left of the page are links for "Privacy Policy" and "Terms of Use".

Sign In to Databricks
Community Edition

Email / Username

Password

Forgot Password?

Sign In

New to Databricks? [Sign Up](#).

Privacy Policy | Terms of Use

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Databricks - Sign Up

2



databricks Platform Solutions Learn Customers Partners Company

Try Databricks for free

An open and unified data analytics platform for data engineering, data science, machine learning, and analytics.

From the original creators of Apache Spark™, Delta lake, MLflow, and Koalas.

Databricks trial:

- Collaborative environment for data teams to build solutions together.
- Interactive notebooks to use Apache Spark™, SQL, Python, Scala, Delta Lake, MLflow, TensorFlow, Keras, Scikit-learn and more.
- Available as a 14-day full trial in your own cloud, or as a lightweight trial hosted by Databricks.

Used by:

Please tell us about yourself

First Name: *

Last Name: *

Company *

Company Email *

Title *

Phone Number

Keep me informed with occasional updates about Databricks and related open source products

By Clicking "Get Started For Free", you agree to the [Privacy Policy](#).

GET STARTED FOR FREE

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Fill all the boxes and
select
"GET STARTED FREE"

Databricks - Sign Up

3



Platform Solutions Learn Customers Partners Company

Try Databricks

AN OPEN AND UNIFIED DATA ANALYTICS PLATFORM FOR DATA ENGINEERING, MACHINE LEARNING, AND ANALYTICS

From the original creators of Apache Spark™, Delta Lake, MLflow, and Koalas

Select a platform

DATABRICKS PLATFORM - FREE TRIAL

For businesses

- Collaborative environment for Data teams to build solutions together
- Unlimited clusters that can scale to any size, processing data in your own account
- Job scheduler to execute jobs for production pipelines
- Fully collaborative notebooks with multi-language support, dashboards, REST APIs
- Native Integration with the most popular ML frameworks(scikit-learn, TensorFlow, Keras,...), Apache Spark™, Delta Lake, and MLflow
- Advanced security, role-based access controls, and audit logs
- Single Sign On support
- Integration with BI tools such as Tableau, Qlik, and Looker
- 14-day full feature trial(excludes cloud charges)

COMMUNITY EDITION

For students and educational institutions

- Single cluster limited to 15GB and no worker nodes
- Basic notebooks without collaboration
- Limited to 3 max users
- Public environment to share your work

GET STARTED

By clicking 'Get Started' for the Community Edition, you agree to the [Databricks Community Edition Terms of Service](#).

CHOOSE YOUR CLOUD

Azure aws Google Cloud

Please note that Azure Databricks is provided by Microsoft and is subject to Microsoft's terms.

By clicking on the "aws" button to get started, you agree to the [Databricks Terms of Service](#).

By clicking on the "Google Cloud" button to get started, you agree to the [Databricks Terms of Service](#).

Click
"GET STARTED"

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.



Time to check your email!

Thank you for signing up. Now it's time to validate your email address.
Please check the email you provided for next steps.

Databricks - Sign Up

5



Welcome to Databricks! Please verify your email address. Inbox x

Databricks noreply@databricks.com via amazoneses.com
to me ▾

12:34 PM (5 minutes ago) Star Reply More

Welcome to Databricks Community Edition!

Databricks Community Edition provides you with access to a free micro-cluster as well as a cluster manager and a notebook environment - ideal for developers, data scientists, data engineers and other IT professionals to get started with Spark.

We need you to verify your email address by clicking on [this link](#). You will then be redirected to Databricks Community Edition!

Get started by visiting: <https://community.cloud.databricks.com/login.html?resetpassword&username=balamurugan.shss%40gmail.com&expiration=-60000&token=1074c22f1640595623c82f1d53a8290e1f3a4e18>

If you have any questions, please contact feedback@databricks.com.

- The Databricks Team

Click the link

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Reset Password

Please enter your new password: *

Please confirm your new password: *

Reset password

1. Enter the “Password”
2. Confirm the “Password”

Click
“Reset Password”

Welcome to Databricks



Upgrade ?

Welcome to databricks

Explore the Quickstart Tutorial

Spin up a cluster, run queries on preloaded data, and display results in 5 minutes.

Import & Explore Data

Drop files or [click to browse](#)

Create a Blank Notebook

Create a notebook to start querying, visualizing, and modeling your data.

Common Tasks

- New Notebook
- Create Table
- New Cluster
- New Job
- New MLflow Experiment
- Import Library
- Read Documentation

Recents

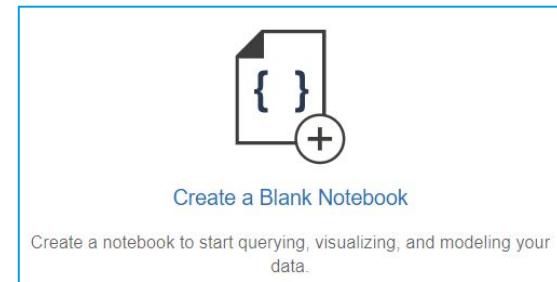
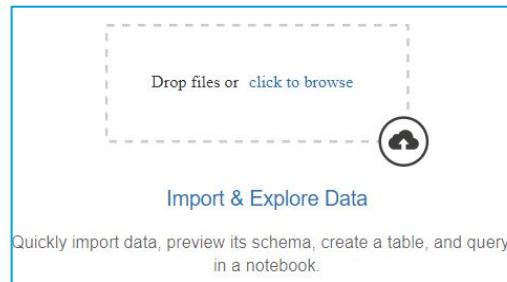
Recent files appear here as you work.

What's new in v3.46

Databricks Status
[View latest release notes](#)

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Working in Databricks



Spin up a cluster

Load the data

Create a notebook

- ★ Community Edition helps us to create a single node cluster. Both Driver and Executors will use the same node

- ★ Databricks support various file formats like csv, json, xml, text. It can also connect to various data sources

- ★ Databricks provide Jupyter Notebooks to write coding in Spark SQL, Python, Scala, R and Java

Spin up a cluster

1

A screenshot of a data science platform's interface. On the left is a dark sidebar with various icons: a square, a plus sign, a document, a file folder, a magnifying glass, a triangle, a square with a circle, a cluster icon (which is highlighted with a blue border), and a list icon. The main area has a light gray header with the text "Welcome to data" and a red icon of three stacked cubes. Below the header is a large button with a white document icon containing curly braces {}, and a lightbulb icon. A blue link "Explore the Quickstart Tutorial" is next to it. A text box says "Spin up a cluster, run queries on preloaded data, and display results in 5 minutes." Below this is a "Common Tasks" section with a list of items: "New Notebook", "Create Table", "New Cluster" (which is also highlighted with a blue border), "New Job", "New MLflow Experiment", "Import Library", and "Read Documentation".

Welcome to data

Explore the Quickstart Tutorial

Spin up a cluster, run queries on preloaded data, and display results in 5 minutes.

Common Tasks

- New Notebook
- Create Table
- New Cluster
- New Job
- New MLflow Experiment
- Import Library
- Read Documentation

Create
"New Cluster"

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Spin up a cluster

2



Create Cluster

New Cluster

Cancel Create Cluster

0 Workers: 0.0 GB Memory, 0 Cores, 0 DBU
1 Driver: 15.3 GB Memory, 2 Cores, 1 DBU

Cluster Name: greatlearning

Databricks Runtime Version: Runtime: 8.2 (Scala 2.12, Spark 3.1.1)

Note: Databricks Runtime 8.x uses Delta Lake as the default table format. [Learn more](#)

Instance: Free 15GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. [For more configuration options, please upgrade your Databricks subscription.](#)

Instances: [Spark](#)

Availability Zone: us-west-2c

A screenshot of the Databricks web interface for creating a new cluster. The main form is titled 'New Cluster'. It has fields for 'Cluster Name' (containing 'greatlearning'), 'Databricks Runtime Version' (set to 'Runtime: 8.2 (Scala 2.12, Spark 3.1.1)'), and 'Availability Zone' (set to 'us-west-2c'). There are also sections for 'Note' about Delta Lake and a message about free memory. A sidebar on the left contains various navigation icons.

- 1.Enter the cluster name
- 2.Choose the Databricks Runtime Version
- 3.Choose Availability Zone
- 4.Click “Create Cluster”
- 5.It creates a single node cluster with 15.3 GB memory, 2 cores
- 6.It takes a few minutes to create the cluster

Spin up a cluster

3

A screenshot of the Databricks Cluster Overview page. On the left is a vertical sidebar with icons for Clusters, Databases, Tables, Events, and Jobs. The main area shows a cluster named "training" in green, indicating it is active. Below the cluster name are buttons for Edit, Clone, Restart, Terminate, and Delete. A horizontal navigation bar includes Configuration (selected), Notebooks, Libraries, Event Log, Spark UI, Driver Logs, Metrics, Apps, and Spark Cluster UI - Master. Under Configuration, there are sections for Databricks Runtime Version (8.2), Driver Type (Community Optimized), and Instance. The Instance section contains a note about free memory and upgrade options. At the bottom are tabs for Instances, Spark, JDBC/ODBC, and Permissions. The Availability Zone is set to us-west-2c.

Clusters /

training

▪ Edit ▪ Clone ▪ Restart ▪ Terminate ▪ Delete

Configuration Notebooks Libraries Event Log Spark UI Driver Logs Metrics Apps Spark Cluster UI - Master

Databricks Runtime Version

8.2 (includes Apache Spark 3.1.1, Scala 2.12)

Driver Type

Community Optimized 15.3 GB Memory, 2 Cores, 1 DBU

Instance

Free 15GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For [more configuration options](#), please [upgrade your Databricks subscription](#).

Instances Spark JDBC/ODBC Permissions

Availability Zone [?](#)

us-west-2c

Cluster is ready

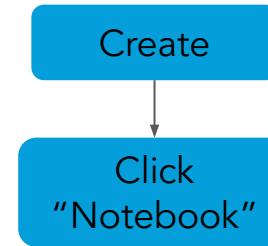
This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Create Notebook

1

A screenshot of the Databricks workspace interface. On the left is a sidebar with various icons and labels: 'databricks' (logo), 'Data Science & E...', 'Create' (highlighted with a white background), 'Workspace' (highlighted with a dark background), 'Repos', 'Recents', 'Search', 'Data', 'Clusters', and 'Jobs'. The main area is titled 'Workspace' with a 'Home' button. A dropdown menu is open under 'Workspace' with options: 'Workspace' (selected and highlighted with a grey background), 'Shared', 'Notebook' (highlighted with a dark background), 'Table', and 'Cluster'.

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.



Create Notebook

2



Create Notebook

Name

Default Language

Cluster

1. Enter the Notebook name
2. Choose the language.
Databricks CE support Python, Scala, R and SQL
3. Choose the Cluster. In CE, you can have only one cluster
4. Click "Create"

Create Notebook

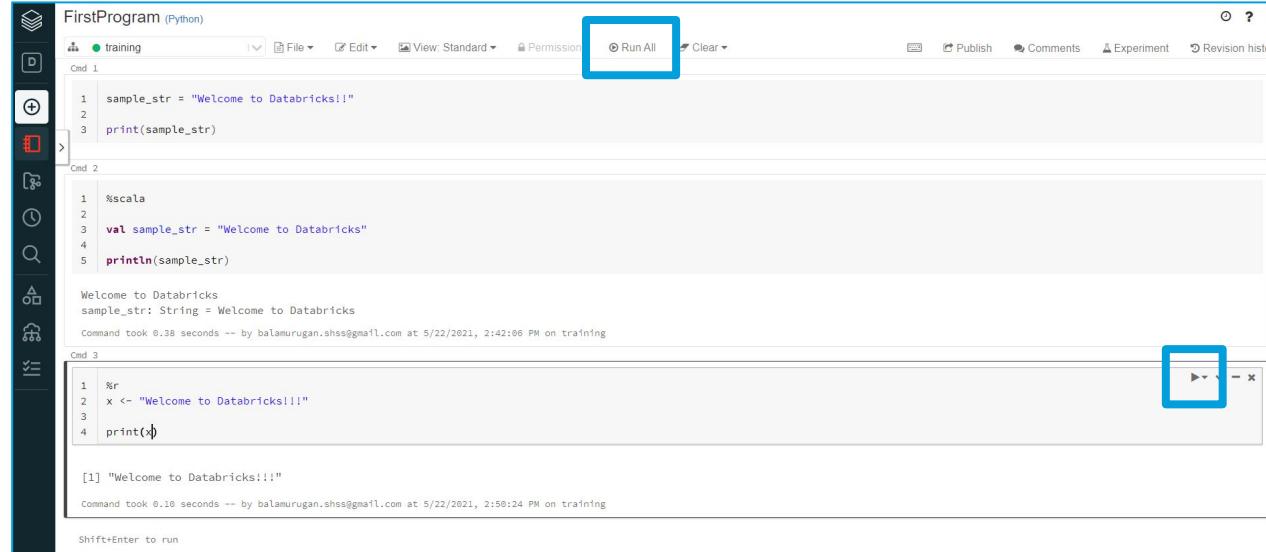
3

A screenshot of a Jupyter Notebook interface. The title bar says "FirstProgram (Python)". The main area shows a single code cell with the number 1 in it. Below the cell is a text input field with the placeholder "Shift+Enter to run". On the left side, there is a vertical toolbar with various icons for file operations like new, open, save, and delete, as well as other notebook-specific functions. The interface has a clean, modern look with a light gray background and white text.

Write your code here

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Write your first program



```

FirstProgram (Python)
Cmd 1
1 sample_str = "Welcome to Databricks!!"
2
3 print(sample_str)

Cmd 2
1 %scala
2
3 val sample_str = "Welcome to Databricks"
4
5 println(sample_str)

Welcome to Databricks
sample_str: String = Welcome to Databricks
Command took 0.38 seconds -- by balamurugan.shss@gmail.com at 5/22/2021, 2:42:06 PM on training

Cmd 3
1 %r
2 x <- "Welcome to Databricks!!!"
3
4 print(x)

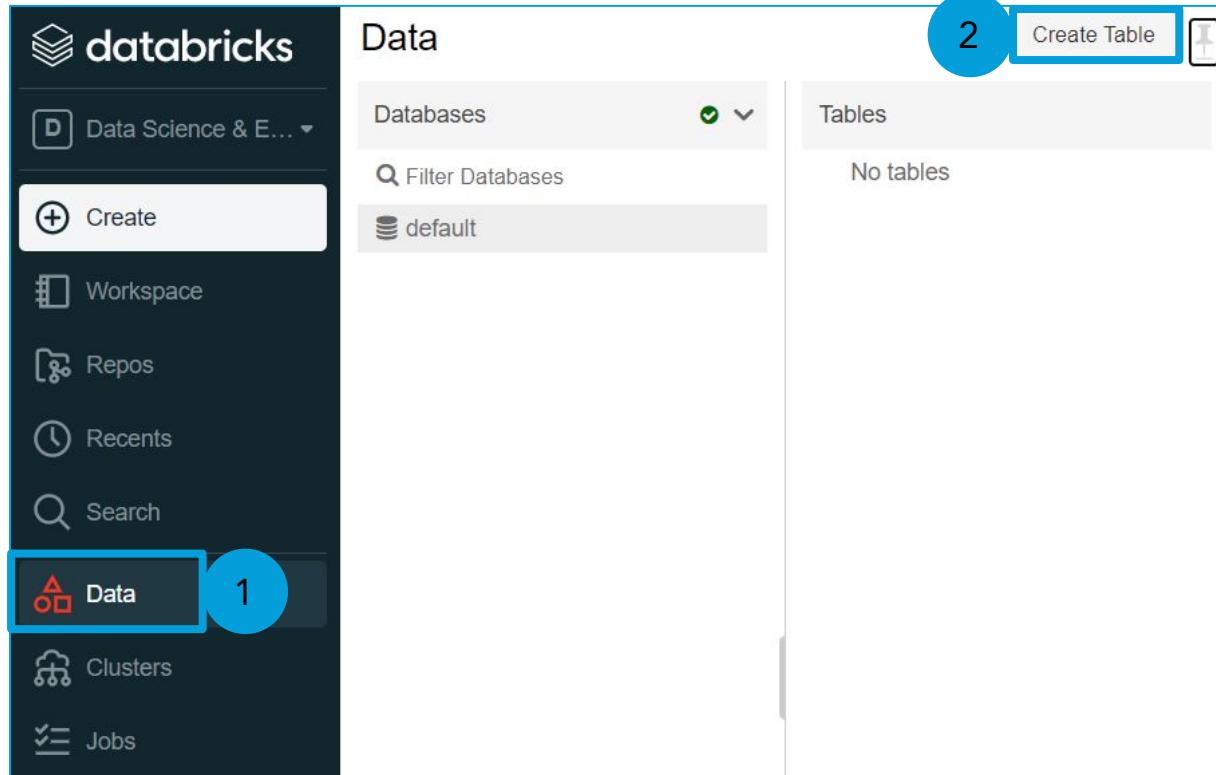
[1] "Welcome to Databricks!!!"
Command took 0.10 seconds -- by balamurugan.shss@gmail.com at 5/22/2021, 2:50:24 PM on training
  
```

Shift+Enter to run

- Control + Enter => will execute a single cell
- Shift + Enter => will execute a single cell + creates a new blank cell
- “Run All” will execute all the cells (highlighted in blue)
- “Run” will execute a single cell
- %python - python, %scala - scala, %sql - Spark SQL, %r - R

This file is meant for personal use by rg.ravigupta91@gmail.com only.
 Sharing or publishing the contents in part or full is liable for legal action.

Loading file(s) to DBFS



The screenshot shows the Databricks interface. On the left, a sidebar menu includes options like 'Data Science & E...', 'Create', 'Workspace', 'Repos', 'Recents', 'Search', 'Data' (which is highlighted with a blue box and has a blue circle with '1' over it), 'Clusters', and 'Jobs'. The main panel is titled 'Data' and contains two tabs: 'Databases' (selected, showing 'default' under 'Filter Databases') and 'Tables' (showing 'No tables'). A 'Create Table' button is located at the top right of the main panel, with a blue circle and '2' over it.

Loading files to
Databricks File System
(DBFS)

- 1.Click "Data" tab
- 2.Click "Create Table"

Loading file(s) to DBFS

Create New Table

Data source ?

Upload File S3 DBFS Other Data Sources Partner Integrations

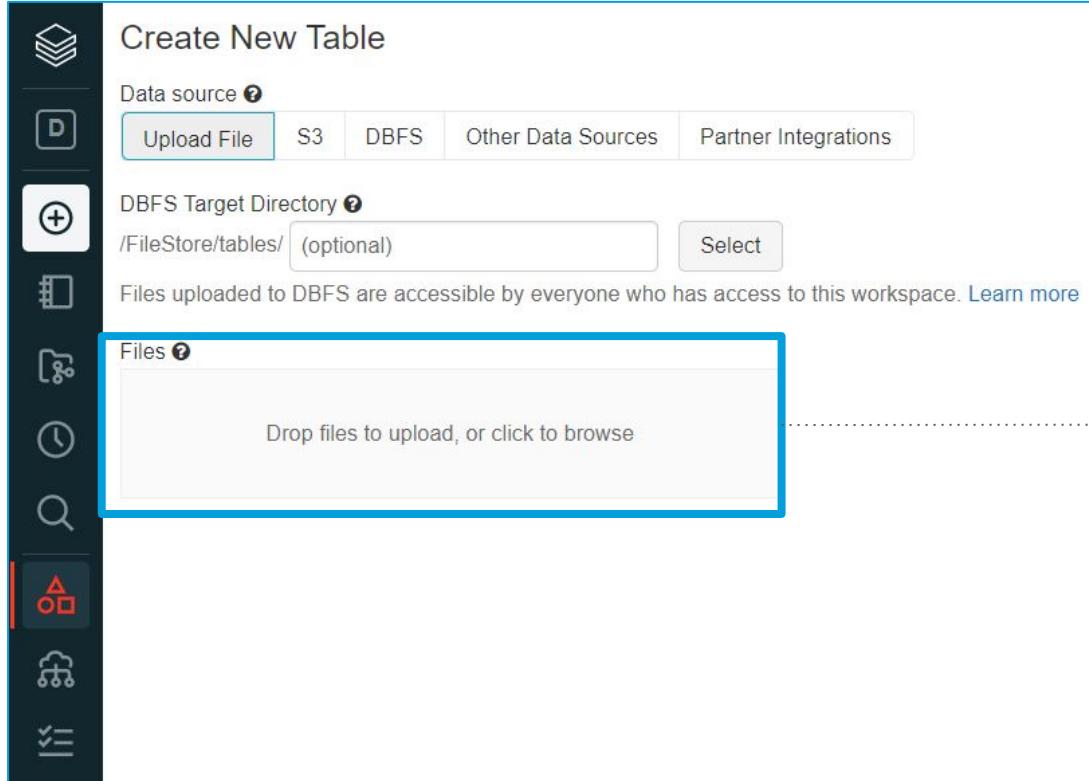
DBFS Target Directory ?
/FileStore/tables/ (optional) Select

Files uploaded to DBFS are accessible by everyone who has access to this workspace. [Learn more](#)

Files ?

Drop files to upload, or click to browse

Upload your files



This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Loading file(s) to DBFS

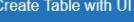
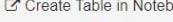
Create New Table Connecting... 

Data source      

Select a file from DBFS 

FileStore	tables	
		 SalesTraining.csv

/FileStore/tables/SalesTraining.csv

File Loaded
Successfully

Note : File loaded to "/FileStore/tables/SalesTraining.csv" (DBFS)

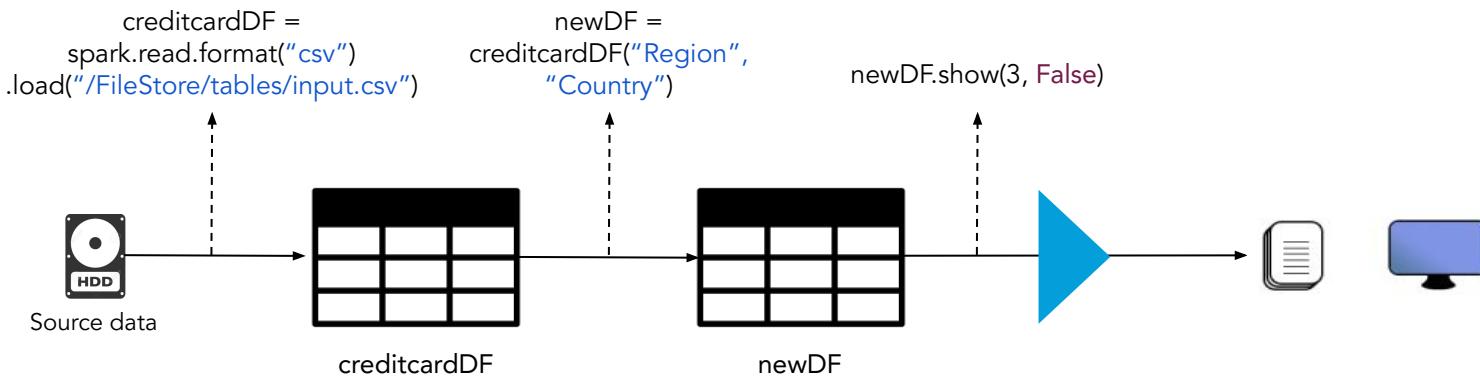
This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Working in Databricks

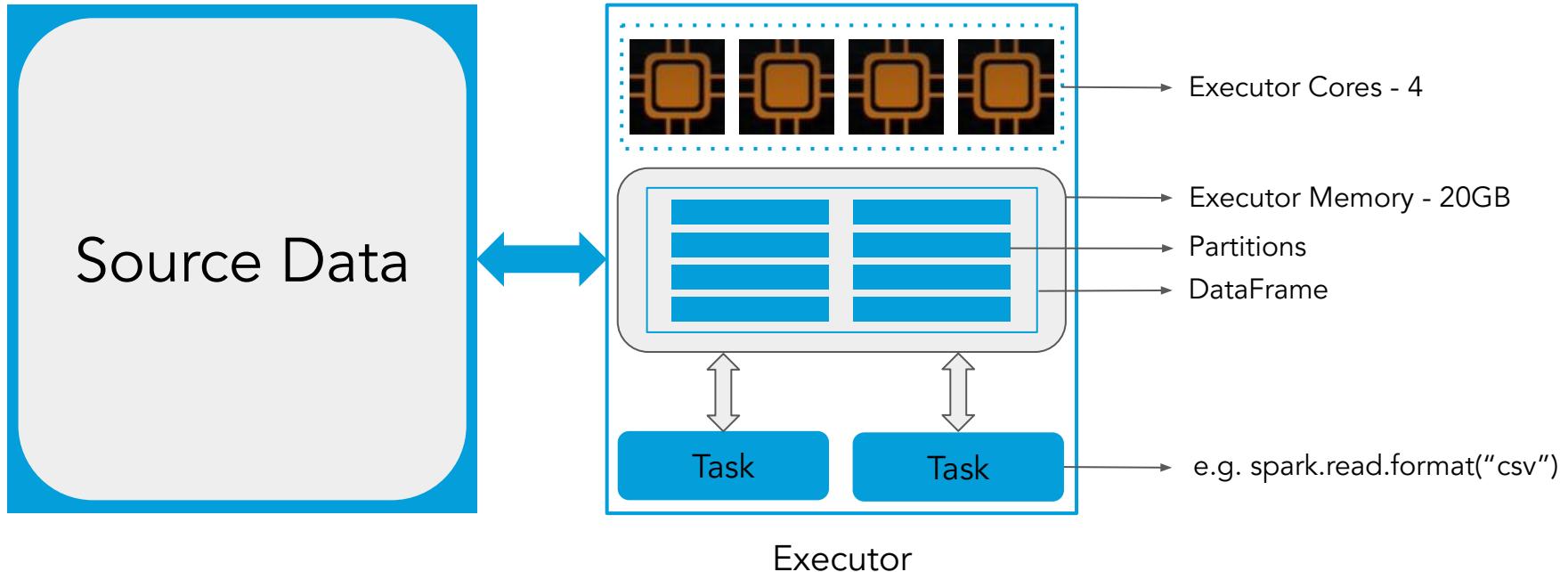
```

1 %python
2
3
4 creditcardDF = spark.read.format("csv").option("header", "true").load("/FileStore/tables/SalesTraining.csv")
5
6 newDF = creditcardDF.select("Region", "Country")
7
8 print("Number of records :", creditcardDF.count())
9
10 newDF.show(3, False)
  
```



This file is meant for personal use by n.g.ravugupta91@gmail.com only.
 Sharing or publishing the contents in part or full is liable for legal action.

What happens in the background?



Class Exercise

- Create a CSV file with customer information (custID, custName, city, state, custType, balance) having a few records. Example:

```
custID,custName,city,state,custType,balance
101,John Bastin,New York,New York,savings,1000
```

- Load the file in to DBFS ("FileStore/tables/customer.csv")
- Create the RDD by reading the file using sparkContext.textFile method
- Apply transformations like map, filter to create a new RDD having records with balance > 1000. Don't include the header record in the new RD

Spark DataFrames

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Table of Content

- What is DataFrame in Spark?
- Creating DataFrames
 - From RDDs
 - From various sources / datasets
- DataFrameReader
- Working with a CSV file
 - Defining your own Schema
 - Selecting DataFrame columns
 - Renaming columns
 - Adding and Dropping DataFrame columns
 - selectExpr
 - Handling null Values
 - DataFrame Aggregations
 - Coalesce vs Repartition
 - Cache vs persist
- DataFrameWriter This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Spark DataFrames

- Immutable, distributed collection of data that is organized into rows and columns
- Each column has a name and associated type
- Similar to a table in RDBMS
- Like RDD APIs, the DataFrame APIs are classified into 2 buckets : transformations, actions
- Transformations are lazily evaluated, and actions are eagerly evaluated
- DataFrame APIs are available in Scala, Python, Java and R
- DataFrames can be created from existing RDDs, and data sources
- Spark supports 6 built-in data formats - csv, parquet, json, jdbc, text and orc
- DataFrames can be accessed by Domain-Specific Language (DSL)

Creating DataFrames

- Creating DataFrames from RDDs
- Creating DataFrame from Data Sources/External Datasets

Creating DataFrames from RDDs

```
1 creditcardRDD = sc.textFile("/FileStore/tables/SalesTraining.csv")      # Reading the file from DBFS location
2
3 # display record count - before removing the header record
4 print("Total records including header : ", creditcardRDD.count())
5
6 header = creditcardRDD.first() # extract the header record from the RDD
7 dataRDD = creditcardRDD.filter(lambda row: row != header)    # use 'filter' (transformation) to remove the header
8
9 # display record count - after removing the header record
10 print("Total records including header : ", dataRDD.count())
11
12 # split the record by ',' and select only the first 3 columns - Region, Country and Item Type
13 tupleRDD = dataRDD.map(lambda each_row: each_row.split(',')).map(lambda each_row: (each_row[0], each_row[1], each_row[2]))
14
15 # convert the RDD to a DataFrame, display first 5 rows with no truncation of column data
16 columns = ["Region", "Country", "ItemType"]
17
18 # convert the RDD to DataFrame. Show() is an action, which shows 1st 5 records with record truncate=False
19 tupleRDD.toDF(columns).show(5, False)
```

► (5) Spark Jobs

```
Total records including header :  950
Total records including header :  949
+-----+-----+-----+
|Region      |Country     |ItemType   |
+-----+-----+-----+
|Sub-Saharan Africa|Burundi|Vegetables|
|Europe        |Ukraine    |Cosmetics  |
|Europe        |Croatia    |Beverages  |
|Sub-Saharan Africa|Madagascar|Fruits    |
|Asia          |Malaysia   |Snacks    |
+-----+-----+-----+
only showing top 5 rows
```

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

DataFrame Schema and Columns

```
1 # convert the tuple RDD to DataFrame with header record
2 creditcardDF = tupleRDD.toDF(columns)
3
4 # display the schema
5 creditcardDF.printSchema()
```

```
▶ (1) Spark Jobs
▶ └─ creditcardDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 1 more fields]
root
|-- Region: string (nullable = true)
|-- Country: string (nullable = true)
|-- ItemType: string (nullable = true)
```

```
1 # display the columns
2 creditcardDF.columns
3
4 # display the RDD data - top 3 records
5 creditcardDF.take(3)|
```

```
▶ (1) Spark Jobs
Out[11]: [Row(Region='Sub-Saharan Africa', Country='Burundi', ItemType='Vegetables'),
Row(Region='Europe', Country='Ukraine', ItemType='Cosmetics'),
Row(Region='Europe', Country='Croatia', ItemType='Beverages')]
```

Creating DataFrames from Data Sources



- DataFrames can directly be created from various data sources or external datasets
- Spark supports 6 built-in data sources, where each data source is mapped to a data format
- Common formats for text data are CSV, XML and JSON
- Common formats for binary data are Avro, Parquet, and ORC
- The two main classes in Spark for reading and writing data are DataFrameReader and DataFrameWriter
- DataFrameReader and DataFrameWriter are available with SparkSession

DataFrameReader

- `spark.read.format(...).option("key", value").schema(...).load()`
 - format - json, parquet, jdbc, orc, csv, text
 - option - DataFrameReader has a set of default options for each data source format, which can be overridden
 - schema - Some data sources have the schema embedded inside the data files, i.e., Parquet and ORC. For other formats, provide a schema

```
1 # create a DF from a CSV file
2 creditcardDF = spark.read.format("csv") \
3             .option("header", "true") \
4             .option("inferSchema", "true") \
5             .load("/FileStore/tables/SalesTraining.csv")
6
7 # display the schema
8 creditcardDF.printSchema()
```

Working with CSV files

1



- One of the commonly used file formats
- Databricks can connect and read data from number of sources like Cloud Storage Services, Databricks File System, HADOOP
- Read the data using the DataFrameReader by passing all the necessary details
- Apply DataFrame APIs to transform the data, push the data on to respective sources by executing the actions

Working with CSV files

2



```
1 # create a DF from a CSV file
2 creditcardDF = spark.read.format("csv") \
3     .option("header", "true") \
4     .option("inferSchema", "true") \
5     .load("/FileStore/tables/SalesTraining.csv")
6
7 # display the schema
8 creditcardDF.printSchema()
```

```
▶ (2) Spark Jobs
▶ creditcardDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 12 more fields]
root
|-- Region: string (nullable = true)
|-- Country: string (nullable = true)
|-- Item Type: string (nullable = true)
|-- Sales Channel: string (nullable = true)
|-- Order Priority: string (nullable = true)
|-- Order Date: string (nullable = true)
|-- Order ID: integer (nullable = true)
|-- Ship Date: string (nullable = true)
|-- Units Sold: integer (nullable = true)
|-- Unit Price: double (nullable = true)
|-- Unit Cost: double (nullable = true)
|-- Total Revenue: double (nullable = true)
|-- Total Cost: double (nullable = true)
|-- Total Profit: double (nullable = true)
```

- `spark.read.format(..)` returns the DataFrame object
- Option “`inferSchema`” infers the schema for each column
- Option “`header`” treats the first record as header record

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Working with CSV files

3



```
1 # display first 5 records with no truncation
2 creditcardDF.show(5, False)
```

▶ (1) Spark Jobs

Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID	Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
Sub-Saharan Africa	Burundi	Vegetables	Online	M	11/17/2010	951380240	12/20/2010	3410	154.06	90.93	525344.6	310071.3	215273.3
Europe	Ukraine	Cosmetics	Online	M	11/13/2014	270001733	1/1/2015	8368	437.2	263.33	13658489.6	2203545.44	1454944.16
Europe	Croatia	Beverages	Online	C	6/16/2016	681941401	7/28/2016	470	47.45	31.79	22301.5	14941.3	7360.2
Sub-Saharan Africa	Madagascar	Fruits	Online	L	5/31/2016	566935575	6/7/2016	7690	9.33	6.92	71747.7	53214.8	18532.9
Asia	Malaysia	Snacks	Offline	M	10/6/2012	175033080	11/5/2012	5033	152.58	97.44	1767935.14	490415.52	277519.62

only showing top 5 rows

- `df.show(..)` is an action, which brings the transformed data to the Driver node
- Spark executes all transformations present in the Lineage (DAG), when an action is triggered

Defining Own Schema

1



```
1 # defining your own schema
2 from pyspark.sql.types import StructType, StructField, StringType, DoubleType, FloatType, LongType, IntegerType, DateType
3
4 # define the structure
5 schema = StructType([
6     StructField("Region", StringType()),
7     StructField("Country", StringType()),
8     StructField("Item", StringType()),
9     StructField("SalesChannel", StringType()),
10    StructField("OrderPriority", StringType()),
11    StructField("OrderDate", StringType()),
12    StructField("OrderID", LongType()),
13    StructField("ShipDate", StringType()),
14    StructField("UnitsSold", IntegerType()),
15    StructField("UnitPrice", FloatType()),
16    StructField("UnitCost", FloatType()),
17    StructField("TotalRevenue", DoubleType()),
18    StructField("TotalCost", DoubleType()),
19    StructField("TotalProfit", DoubleType())
20])
21
22 # read the file by using the defined schema
23 creditcardDF1 = spark.read.format("csv").option("header", "true").schema(schema).load("/FileStore/tables/SalesTraining.csv")
24
25 # display the schema
26 creditcardDF1.printSchema()
```

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Defining Own Schema

2



```
23 creditcardDF1.printSchema()
```

```
▶ creditcardDF1: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 12 more fields]
root
|-- Region: string (nullable = true)
|-- Country: string (nullable = true)
|-- Item: string (nullable = true)
|-- SalesChannel: string (nullable = true)
|-- OrderPriority: string (nullable = true)
|-- OrderDate: string (nullable = true)
|-- OrderID: long (nullable = true)
|-- ShipDate: string (nullable = true)
|-- UnitsSold: integer (nullable = true)
|-- UnitPrice: float (nullable = true)
|-- UnitCost: float (nullable = true)
|-- TotalRevenue: double (nullable = true)
|-- TotalCost: double (nullable = true)
|-- TotalProfit: double (nullable = true)
```

Defining Own Schema

3



```
1 # display the records
2 creditcardDF1.show(3, False)
```

▶ (1) Spark Jobs

Region	Country	Item	SalesChannel	OrderPriority	OrderDate	OrderID	ShipDate	UnitsSold	UnitPrice	UnitCost	TotalRevenue	TotalCost	TotalProfit
Sub-Saharan Africa	Burundi	Vegetables	Online	M	11/17/2010	951380240	12/20/2010	3410	154.06	90.93	525344.6	310071.3	215273.3
Europe	Ukraine	Cosmetics	Online	M	11/13/2014	270001733	1/1/2015	8368	1437.2	263.33	3658489.6	2203545.44	1454944.16
Europe	Croatia	Beverages	Online	C	6/16/2016	681941401	7/28/2016	470	47.45	31.79	22301.5	14941.3	7360.2

only showing top 3 rows

Selecting DataFrame Columns

```
1 # selecting all the columns
2 creditcardDF.select("*").show(4, False)
```

▶ (1) Spark Jobs

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Region      |Country    |Item Type |Sales Channel|Order Priority|Order Date|Order ID |Ship Date |Units Sold|Unit Price|Unit Cost|Total Revenue|Total Cost|Total Profit|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Sub-Saharan Africa|Burundi |Vegetables|Online        |M          |11/17/2010|951380240|12/20/2010|3410     |154.06    |90.93    |525344.6   |310071.3  |215273.3  |
|Europe       |Ukraine    |Cosmetics  |Online        |M          |11/13/2014|270001733|1/1/2015   |8368     |437.2     |263.33    |3658489.6  |2203545.44|1454944.16 |
|Europe       |Croatia    |Beverages  |Online        |C          |6/16/2016  |681941401|7/28/2016  |470      |47.45     |31.79     |22301.5    |14941.3   |7360.2    |
|Sub-Saharan Africa|Madagascar|Fruits    |Online        |L          |5/31/2016  |566935575|6/7/2016   |7690     |9.33      |6.92      |71747.7   |53214.8   |18532.9  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 4 rows
```

```
1 # selecting only a few columns
2 from pyspark.sql.functions import col, column
3 import pyspark.sql.functions as F
4
5 # select a few columns
6 creditcardDF.select("Region", "Country", F.col("Item Type"), "Order Date", "Units Sold", "Total Revenue", F.lit('DefaultValue')).show(4, False)
```

▶ (1) Spark Jobs

```
+-----+-----+-----+-----+-----+
|Region      |Country    |Item Type |Order Date|Units Sold|Total Revenue|DefaultValue|
+-----+-----+-----+-----+-----+
|Sub-Saharan Africa|Burundi |Vegetables|11/17/2010|3410     |525344.6   |DefaultValue|
|Europe       |Ukraine    |Cosmetics  |11/13/2014|8368     |3658489.6  |DefaultValue|
|Europe       |Croatia    |Beverages  |6/16/2016  |470      |22301.5    |DefaultValue|
|Sub-Saharan Africa|Madagascar|Fruits    |5/31/2016  |7690     |71747.7   |DefaultValue|
+-----+-----+-----+-----+-----+
only showing top 4 rows
```

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Renaming Columns

1



```
1 # renaming a column
2 creditcardDF.columns
```

```
Out[41]: ['Region',
'Country',
'Item Type',
'Sales Channel',
'Order Priority',
'Order Date',
'Order ID',
'Ship Date',
'Units Sold',
'Unit Price',
'Unit Cost',
'Total Revenue',
'Total Cost',
'Total Profit']
```

Renaming Columns

2



```
1 # renaming the columns - "Sales Channel" to "SalesChannel" and "Item Type" to "Item"
2 creditcardDF2 = creditcardDF.withColumnRenamed("Sales Channel", "SalesChannel").withColumnRenamed('Item Type','Item')
```

```
creditcardDF2: pyspark.sql.dataframe.DataFrame
```

```
Region: string
Country: string
Item: string
SalesChannel: string
Order Priority: string
Order Date: string
Order ID: integer
Ship Date: string
Units Sold: integer
Unit Price: double
Unit Cost: double
Total Revenue: double
Total Cost: double
Total Profit: double
```

```
Command took 0.04 seconds -- by balamurugan.shss@gmail.com at 5/24/2021, 11:16:38 AM on training
```

```
md 10
```

```
1 # display records
2 creditcardDF2.show(3, False)
```

```
▶ (1) Spark Jobs
```

Region	Country	Item	SalesChannel	Order Priority	Order Date	Order ID	Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
Sub-Saharan Africa	Burundi	Vegetables	Online	M	11/17/2010 951380240 12/20/2010 3410		154.06	90.93	525344.6	310071.3	215273.3		
Europe	Ukraine	Cosmetics	Online	M	11/13/2014 270001733 1/1/2015	8368	437.2	263.33	3658489.6	2203545.44	1454944.16		
Europe	Croatia	Beverages	Online	C	6/16/2016 681941401 7/28/2016	470	47.45	31.79	22301.5	14941.3	7360.2		

only showing top 3 rows

Changing Columns Data Type

```
1 # change column data type
2 from pyspark.sql.functions import col
3
4 # change the data type from integer to long
5 df = creditcardDF.withColumn("Order ID", col("Order ID").cast("long"))
6 df.printSchema() # display the schema

▶ df: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 12 more fields]
root
|-- Region: string (nullable = true)
|-- Country: string (nullable = true)
|-- Item Type: string (nullable = true)
|-- Sales Channel: string (nullable = true)
|-- Order Priority: string (nullable = true)
|-- Order Date: string (nullable = true)
|-- Order ID: long (nullable = true) [highlighted]
|-- Ship Date: string (nullable = true)
|-- Units Sold: integer (nullable = true)
|-- Unit Price: double (nullable = true)
|-- Unit Cost: double (nullable = true)
|-- Total Revenue: double (nullable = true)
|-- Total Cost: double (nullable = true)
|-- Total Profit: double (nullable = true)
```

Adding Columns to a DataFrame

```
1 # adding columns to a dataframe
2 import pyspark.sql.functions as F
3
4 # add a new column "Register_Site" with default value "www.google.com"
5 dataDF = creditcardDF.withColumn("Register_Site", F.lit("www.google.com"))
6
7 # display only a few columns
8 dataDF.select("Region", "Country", "Item Type", "Register_Site").show(3, False)
```

- ▶ (1) Spark Jobs
- ▶  dataDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 13 more fields]

Region	Country	Item Type	Register_Site
Sub-Saharan Africa	Burundi	Vegetables	www.google.com
Europe	Ukraine	Cosmetics	www.google.com
Europe	Croatia	Beverages	www.google.com

only showing top 3 rows

Removing Columns from a DataFrame



```
1 # removing columns from a DataFrame
2
3 # number of columns in a dataframe - before removing columns
4 print("Number of columns : ", len(dataDF.columns))
5
6 # columns - before dropping
7 print(list(dataDF.columns))
8
9 # drop columns - "Country", "Item Type"
10 dataNewDF = dataDF.drop("Country", "Item Type")
11
12 # number of columns in a dataframe - after removing columns
13 print("Number of columns : ", len(dataNewDF.columns))
14
15 # columns - after dropping
16 print(list(dataNewDF.columns))

▶ └── dataNewDF: pyspark.sql.dataframe.DataFrame = [Region: string, Sales Channel: string ... 11 more fields]
Number of columns : 15
['Region', 'Country', 'Item Type', 'Sales Channel', 'Order Priority', 'Order Date', 'Order ID', 'Ship Date', 'Units Sold', 'Unit Price', 'Unit Cost', 'Total Revenue', 'Total Cost',
'Total Profit', 'Register_Site']
Number of columns : 13
['Region', 'Sales Channel', 'Order Priority', 'Order Date', 'Order ID', 'Ship Date', 'Units Sold', 'Unit Price', 'Unit Cost', 'Total Revenue', 'Total Cost', 'Total Profit', 'Register_Site']
```

Arithmetic with DataFrames

```
1 # arithmetic with dataframes
2 # number of columns in a dataframe - before adding a column
3 print("Number of columns : ", len(creditcardDF.columns))
4
5 # perform arithmetic operations on a dataframe column
6 creditnewDF = creditcardDF.withColumn("TotalSale", col("Units Sold") * col("Unit Price"))
7
8 # number of columns in a dataframe - after adding columns
9 print("Number of columns : ", len(creditnewDF.columns))
10
11 # display records
12 creditnewDF.show(3)
```

► (1) Spark Jobs

► creditnewDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 13 more fields]

Number of columns : 14

Number of columns : 15

Region Country Item Type Sales Channel Order Priority Order Date Order ID Ship Date Units Sold Unit Price Unit Cost Total Revenue Total Cost Total Profit TotalSale
Sub-Saharan Africa Burundi Vegetables Online M 11/17/2010 951380240 12/20/2010 3410 154.06 90.93 525344.6 310071.3 215273.3 525344.6
Europe Ukraine Cosmetics Online M 11/13/2014 270001733 1/1/2015 8368 437.2 263.33 3658489.6 2203545.44 1454944.16 3658489.6
Europe Croatia Beverages Online C 6/16/2016 681941401 7/28/2016 470 47.45 31.79 22301.5 14941.3 7360.2 22301.5

only showing top 3 rows

How to Filter a DataFrame

```

1 # filter a dataframe
2
3 creditcardDF.where(col("Region") == "Asia").show(5)

```

► (1) Spark Jobs

Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID	Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
Asia	Malaysia	Snacks	Offline	M	10/6/2012	175033080	11/5/2012	5033	152.58	97.44	767935.14	490415.52	277519.62
Asia	Uzbekistan	Office Supplies	Offline	L	3/10/2012	276595246	3/15/2012	9535	651.21	524.96	6209287.35	5005493.6	1203793.75
Asia	Nepal	Vegetables	Offline	C	6/2/2014	443121373	6/19/2014	8316	154.06	90.93	1281162.96	756173.88	524989.08
Asia	India	Fruits	Online	H	7/29/2010	658348691	8/22/2010	8862	9.33	6.92	82682.46	61325.04	21357.42
Asia	South Korea	Fruits	Offline	L	3/7/2010	769205892	3/17/2010	3972	9.33	6.92	37058.76	27486.24	9572.52

only showing top 5 rows

```

1 # filter a dataframe - multiple columns
2
3 creditcardDF.where((col("Region") == "Asia") & (col("Sales Channel") == "Online")).show(5)

```

► (1) Spark Jobs

Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID	Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
Asia	India	Fruits	Online	H	7/29/2010	658348691	8/22/2010	8862	9.33	6.92	82682.46	61325.04	21357.42
Asia	Philippines	Baby Food	Online	L	2/23/2014	160127294	3/23/2014	4079	255.28	159.42	1041287.12	650274.18	391012.94
Asia	Bangladesh	Baby Food	Online	H	8/17/2011	254927718	9/7/2011	7632	255.28	159.42	1948296.96	1216693.44	731603.52
Asia	Vietnam	Baby Food	Online	C	6/20/2011	881974112	7/11/2011	4594	255.28	159.42	1172756.32	732375.48	440380.84
Asia	Taiwan	Household	Online	C	1/23/2017	607261836	2/22/2017	1127	668.27	502.54	753140.29	566362.58	186777.71

only showing top 5 rows

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Dropping Duplicate Rows

1



```
1 # dropping rows
2 testDF = [[1, "January"], [2, "February"], [1, "January"], [3, "March"], [3, "March"], [3, "March"], [4, "April"], [4, "April"], [5, "May"], [5, "May"],
3     [4, "April"], [6, "June"], [5, "April"]]
4
5 # import the modules
6 from pyspark.sql.types import *
7
8 # define the schema
9 schema = StructType([StructField("ID", IntegerType()),StructField("Month", StringType())])
10
11 # create the dataframe by applying schema
12 df = spark.createDataFrame(testDF,schema=schema)
13
14 # display the records
15 df.show()
```

▶ (3) Spark Jobs

▶ df: pyspark.sql.dataframe.DataFrame = [ID: integer, Month: string]

```
+---+-----+
| ID|    Month|
+---+-----+
|  1| January|
|  2|February|
|  1| January|
|  3|   March|
|  3|   March|
|  3|   March|
|  4| April|
|  4| April|
|  5|    May|
|  5|    May|
|  4| April|
|  6|   June|
|  5| April|
+---+-----+
```

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Dropping Duplicate Rows

2



```
1 # display distinct rows
2 df.distinct().show()
```

▶ (2) Spark Jobs

```
+---+-----+
| ID| Month|
+---+-----+
| 1| January|
| 2| February|
| 3| March|
| 4| April|
| 5| May|
| 5| April|
| 6| June|
+---+-----+
```

Dropping Duplicate Rows

3



```
1 # drop duplicate records based a column value
2 df.dropDuplicates(['Month']).show()
3
4 # drop duplicate records based multiple column values
5 df.dropDuplicates(['Month', 'ID']).show()
```

▶ (4) Spark Jobs

```
+---+-----+
| ID| Month|
+---+-----+
| 1| January|
| 2| February|
| 3| March|
| 4| April|
| 5| May|
| 6| June|
+---+-----+
```

```
+---+-----+
| ID| Month|
+---+-----+
| 1| January|
| 2| February|
| 3| March|
| 4| April|
| 5| May|
| 6| June|
| 5| April|
+---+-----+
```

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

selectExpr

```
1 # rename existing columns
2 newDF = creditcardDF.withColumnRenamed("Unit Price", "UnitPrice").withColumnRenamed("Total Profit", "Total_Profit")
3
4 creditcardDF.show(3) # display records
5
6 from pyspark.sql.functions import expr # define the modules
7
8 # using select expression
9 newDF.select("Region", "Country",expr("CASE WHEN Total_Profit > 300000 THEN 'Good' ELSE 'Average' END AS value_desc")).show(3)
```

► (2) Spark Jobs

► newDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 12 more fields]

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Region|Country|Item Type|Sales Channel|Order Priority|Order Date| Ship Date|Units Sold|Unit Price|Unit Cost|Total Revenue|Total Cost|Total Profit|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Sub-Saharan Africa|Burundi|Vegetables| Online| M|11/17/2010|951380240|12/20/2010| 3410| 154.06| 90.93| 525344.6| 310071.3| 215273.3|
| Europe|Ukraine| Cosmetics| Online| M|11/13/2014|270001733| 1/1/2015| 8368| 437.2| 263.33| 3658489.6|2203545.44| 1454944.16|
| Europe|Croatia| Beverages| Online| C| 6/16/2016|681941401| 7/28/2016| 470| 47.45| 31.79| 22301.5| 14941.3| 7360.2|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

only showing top 3 rows

```
+-----+-----+
| Region|Country|value_desc|
+-----+-----+
|Sub-Saharan Africa|Burundi| Average|
| Europe|Ukraine| Good|
| Europe|Croatia| Average|
+-----+-----+
```

only showing top 3 rows

Handling null Values

1



```
1 from pyspark.sql.types import * # import the libraries
2
3 # define a list
4 list_data = [["Bill Gates",23],["Henry Ford", None], ["Tim Cook", None]]
5
6 # define the schema
7 schema = StructType([StructField("Name", StringType()),StructField("Experience", IntegerType())])
8
9 # create a dataframe
10 df = spark.createDataFrame(list_data,schema=schema)
11
12 df.show() # display the dataframe
```

▶ (3) Spark Jobs

▶ df: pyspark.sql.dataframe.DataFrame = [Name: string, Experience: integer]

```
+-----+-----+
|      Name|Experience|
+-----+-----+
|Bill Gates|      23|
|Henry Ford|     null|
| Tim Cook|     null|
+-----+-----+
```

Handling null Values

2



```
1 # drop null value rows
2 df.na.drop().show()
```

► (3) Spark Jobs

```
+-----+-----+
|      ID|Month|
+-----+-----+
|Bill Gates|    23|
+-----+-----+
```

```
1 # fill null value with a constant value
2 df.fillna(34).show()
```

► (3) Spark Jobs

```
+-----+-----+
|      Name|Experience|
+-----+-----+
|Bill Gates|    23|
|Henry Ford|    34|
| Tim Cook|    34|
+-----+-----+
```

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Replacing Values

```
1 # replace a single value
2 df.na.replace('Bill Gates', 'Satya Nadella').show()
```

► (3) Spark Jobs

```
+-----+-----+
|      Name|Experience|
+-----+-----+
|Satya Nadella|      23|
|  Henry Ford|      null|
|   Tim Cook|      null|
+-----+-----+
```

```
1 # replace multiple values and also fill 'null' with a constant value
2 df.na.replace(['Bill Gates', 'Tim Cook'], ['Satya N', 'Time'], 'Name').fillna(40).show()
```

► (3) Spark Jobs

```
+-----+-----+
|      Name|Experience|
+-----+-----+
|   Satya N|      23|
|Henry Ford|      40|
|     Time|      40|
+-----+-----+
```

DataFrame Aggregations

1



- A set of methods for aggregations on a DataFrame:
 - agg
 - avg
 - count
 - max
 - mean
 - min
 - pivot
 - sum

DataFrame Aggregations

2



```
1 # rename the existing columns - "Item Type" to "ItemType" and "Total Profit" to "Total_Profit"
2 newDF = creditcardDF.withColumnRenamed("Item Type", "ItemType").withColumnRenamed("Total Profit", "Total_Profit")
3
4 # find maximum total_profit for each region and alias the column to "Maximum"
5 newDF.groupBy("Region").max("Total_Profit").alias("Maximum").show(10, False)
```

► (2) Spark Jobs

```
▶ [ ] newDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 12 more fields]
```

Region	max(Total_Profit)
Middle East and North Africa	1682887.73
Australia and Oceania	1631943.31
Europe	1726181.36
Sub-Saharan Africa	1571089.32
Central America and the Caribbean	1631422.21
North America	1541620.46
Asia	1725485.88

DataFrame Aggregations

3



```
1 # count of items in each region
2 newDF.groupBy("Region").agg({'ItemType':'count'}).show(10, False)
```

► (2) Spark Jobs

Region	count(ItemType)
Middle East and North Africa	132
Australia and Oceania	76
Europe	255
Sub-Saharan Africa	247
Central America and the Caribbean	93
North America	16
Asia	130

DataFrame Aggregations

4



```
1 from pyspark.sql.functions import avg # include the library
2
3 # find average of column - "Total_Profit"
4 newDF.select(avg("Total_Profit").alias("Average Profit")).show()
```

▶ (2) Spark Jobs

```
+-----+
| Average Profit|
+-----+
|392951.5061011591|
+-----+
```

Ordering DataFrame Rows

```

1 # include the library
2 from pyspark.sql.functions import col
3
4 # order the records by region - ascending
5 creditcardDF.orderBy('Region', ascending=True).select("Region", "Country", "Item Type", "Order ID", "Total Profit").show(3)

```

▶ (1) Spark Jobs

Region	Country	Item Type	Order ID	Total Profit
Asia	Uzbekistan	Office Supplies	276595246	1203793.75
Asia	South Korea	Fruits	769205892	9572.52
Asia	Malaysia	Snacks	175033080	277519.62

only showing top 3 rows

```

1 # include the library
2 from pyspark.sql.functions import col
3
4 # order the records by region - descending
5 creditcardDF.orderBy('Region', ascending=False).select("Region", "Country", "Item Type", "Order ID", "Total Profit").show(3)

```

▶ (1) Spark Jobs

Region	Country	Item Type	Order ID	Total Profit
Sub-Saharan Africa	Botswana	Clothes	680517470	668083.68
Sub-Saharan Africa	Central African R...	Office Supplies	668599021	273078.75
Sub-Saharan Africa	Tanzania	Personal Care	400304734	198500.26

only showing top 3 rows

Cache and Persist

1



- Cache() - defaults storage level to MEMORY_ONLY
- Persist() - saves the intermediate results in 5 storage levels:
 - MEMORY_ONLY
 - MEMORY_AND_DISK
 - MEMORY_ONLY_SER
 - MEMORY_AND_DISK_SER
 - DISK_ONLY

Cache and Persist

2



```
1 # cache and persist
2 from pyspark import StorageLevel
3
4 # cache the dataframe in in-memory
5 cacheDF = creditcardDF.cache()
6
7 # read the records from cache
8 cacheDF.select("Region", "Country", "Item Type", "Sales Channel", "Order Priority", \
9                 "Order Date", "Order ID", "Ship Date").show(4, truncate=False)
```

▶ (1) Spark Jobs

▶ [cacheDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 12 more fields]]

Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID	Ship Date
Sub-Saharan Africa	Burundi	Vegetables	Online	M	11/17/2010	951380240	12/20/2010
Europe	Ukraine	Cosmetics	Online	M	11/13/2014	270001733	1/1/2015
Europe	Croatia	Beverages	Online	C	6/16/2016	681941401	7/28/2016
Sub-Saharan Africa	Madagascar	Fruits	Online	L	5/31/2016	566935575	6/7/2016

only showing top 4 rows

Cache and Persist

3



```
1 # cache and persist
2 from pyspark import StorageLevel
3
4 # persist the dataframe in both memo
5 persistDF = creditcardDF.persist(StorageLevel.MEMORY_AND_DISK)
6
7 # read the records from saved dataframe
8 persistDF.select("Region", "Country", "Item Type", "Sales Channel", "Order Priority", \
9                 "Order Date", "Order ID").show(4, truncate=False)
```

▶ (1) Spark Jobs

▶ pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 12 more fields]

Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID
Sub-Saharan Africa	Burundi	Vegetables	Online	M	11/17/2010	951380240
Europe	Ukraine	Cosmetics	Online	M	11/13/2014	270001733
Europe	Croatia	Beverages	Online	C	6/16/2016	681941401
Sub-Saharan Africa	Madagascar	Fruits	Online	L	5/31/2016	566935575

only showing top 4 rows

Coalesce and Repartition

1



SNo	Coalesce	Repartition
1	Reduces the number of partitions	Can be used to either increase or decrease the number of partitions
2	Uses existing partitions to minimize the amount of data shuffling	Creates new partitions and performs full shuffling
3	Results in partitions having different sizes	Results in partitions having same size of data

Coalesce and Repartition

2



```
1 # coalesce vs repartition
2 print("Number of partitions : ", creditcardDF.rdd.getNumPartitions())
3
4 # increase the number of partitions
5 cDF = creditcardDF.repartition(2)
6
7 # number of partitions after repartitioning
8 print("Number of partitions : ", cDF.rdd.getNumPartitions())
9
10 # reduce the number of partitions
11 cDF = cDF.coalesce(1)
12
13 # number of partitions after coalesce
14 print("Number of partitions : ", cDF.rdd.getNumPartitions())
```

- ▶ (2) Spark Jobs
- ▶ cDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 12 more fields]
Number of partitions : 1
Number of partitions : 2
Number of partitions : 1

DataFrameWriter

1



```
1 # aggregates the Item Type count by region, brings the data to a single partition
2 writeDF = newDF.groupBy("Region").agg({'ItemType':'count'}).coalesce(1)
3
4 # write to DBFS - mode: "overwrite" replaces the existing file and "append" adds the content
5 writeDF.write.option("header","true").option("sep",",").mode("overwrite").csv("/FileStore/tables/Aggregate/")
```

▶ (2) Spark Jobs

▶ writeDF: pyspark.sql.dataframe.DataFrame = [Region: string, count(ItemType): long]

```
1 %fs ls "/FileStore/tables/Aggregate/"
```

	path	name
1	dbfs:/FileStore/tables/Aggregate/_SUCCESS	_SUCCESS
2	dbfs:/FileStore/tables/Aggregate/_committed_2341650654956389289	_committed_2341650654956389289
3	dbfs:/FileStore/tables/Aggregate/_committed_3214256169176087794	_committed_3214256169176087794
4	dbfs:/FileStore/tables/Aggregate/_committed_vacuum8977280182979154844	_committed_vacuum8977280182979154844
5	dbfs:/FileStore/tables/Aggregate/_started_2341650654956389289	_started_2341650654956389289
6	dbfs:/FileStore/tables/Aggregate/part-00000-tid-2341650654956389289-3daeebed-c9cf-4cf0-98bb-7a2a9f21cae7-175-1-c000.csv	part-00000-tid-2341650654956389289-3daeebed-c9cf-4cf0-98bb-7a2a9f21cae7-175-1-c000.csv

Showing all 6 rows.

%fs executes the list command ("ls") and displays all the files in the mentioned folder

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

DataFrameWriter

2



```
1 # read the csv file
2 newDF = spark.read.format("csv").option("header", "true").option("inferSchema", "true") \
3     .load("/FileStore/tables/Aggregate/part-00000-tid-2341650654956389289-3daeebed-c9cf-4cf0-98bb-7a2a9f21cae7-175-1-c000.csv")
4
5 # display the records
6 newDF.show(10, False)
```

▶ (3) Spark Jobs

▶ newDF: pyspark.sql.dataframe.DataFrame = [Region: string, count(ItemType): integer]

Region	count(ItemType)
Middle East and North Africa	132
Australia and Oceania	76
Europe	255
Sub-Saharan Africa	247
Central America and the Caribbean	93
North America	16
Asia	130

Class Exercise

- Upload a CSV file in to DBFS ("FileStore/tables/credit.csv")
- Create a DataFrame by reading CSV file using spark.read.format("csv")
- Apply DSL methods to perform the below:
 - Select all columns, and select only a few columns
 - Rename and drop existing columns
 - Change the data type of a column from float to double
 - Replace null values
 - Apply agg functions
- Write the result DataFrame in to DBFS ("FileStore/tables/result/")

Spark SQL

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Table of Content

- What is Spark SQL
- Creating Temp View Tables from DataFrames
- Querying Temp Views
- Apply Aggregations using SQL Queries
- Ordering the Records using SQLs
- SQL Joins
- Writing the results in to DBFS
- Class Exercise

Spark SQL

- Spark SQL is a Spark module for structured data processing
- Provides powerful integration with the rest of the Spark ecosystem
- Spark SQL brings native support for SQL to Spark
- Provides a programming abstraction called DataFrames
- Enables working with structured data, either within Spark programs or through JDBC and ODBC connectors

Create Table from a DataFrame

1



```
1 # spark SQL
2 # create a DataFrame
3 from pyspark.sql.types import * # import the library
4 leader_data = [["Middle East and North Africa", "Mohammed Saif"], ["Australia and Oceania", "George Carlin"], \
5                 ["Europe", "Stuart Broad"], ["Sub-Saharan Africa", "Abdalla"], ["Central America and the Caribbean", "Chris Gayle"], \
6                 ["North America", "George Bush"], ["Asia", "Tatyaso Martin"]]
7
8 # define the schema
9 schema = StructType([StructField("Region", StringType()), StructField("SalesPerson", StringType())])
10
11 # create a dataframe and display the records
12 df = spark.createDataFrame(leader_data,schema=schema)
13 df.show(10, False)
```

▶ (3) Spark Jobs

▶ df: pyspark.sql.dataframe.DataFrame = [Region: string, SalesPerson: string]

Region	SalesPerson
Middle East and North Africa	Mohammed Saif
Australia and Oceania	George Carlin
Europe	Stuart Broad
Sub-Saharan Africa	Abdalla
Central America and the Caribbean	Chris Gayle
North America	George Bush
Asia	Tatyaso Martin

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Create Table from a DataFrame

2



```
1 df.createOrReplaceTempView("sales_table") # convert dataframe to view
2
3 # write sql queries using sql()
4 sql("select * from sales_table").show(10, False)
```

► (3) Spark Jobs

Region	SalesPerson
Middle East and North Africa	Mohammed Saif
Australia and Oceania	George Carlin
Europe	Stuart Broad
Sub-Saharan Africa	Abdalla
Central America and the Caribbean	Chris Gayle
North America	George Bush
Asia	Tatyaso Martin

Querying the Table

```
1 | sql("select * from sales_table where Region = 'Asia'").show(10, False)
```

▶ (3) Spark Jobs

```
+-----+-----+
|Region|SalesPerson    |
+-----+-----+
|Asia   |Tatyaso Martin|
+-----+-----+
```

```
1 | sql("select * from sales_table where SalesPerson like '%George%'").show(10, False)
```

▶ (3) Spark Jobs

```
+-----+-----+
|Region           |SalesPerson    |
+-----+-----+
|Australia and Oceania|George Carlin|
|North America      |George Bush   |
+-----+-----+
```

```
1 | sql("select count(*) from sales_table").show()
```

▶ (2) Spark Jobs

```
+-----+
|count(1)|
+-----+
|      7|
+-----+
```

Aggregations

```
1 # renaming a column using DSL
2 newDF = creditcardDF.withColumnRenamed("Total Revenue", "TotalRevenue")
3
4 # create a temp view
5
6 newDF.createOrReplaceTempView("creditcard")
7
8 # apply aggregations on the table data
9 sql("select Region, max(TotalRevenue) from creditcard group by Region").show(truncate=False)
```

▶ (2) Spark Jobs

▶  newDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 12 more fields]

Region	max(TotalRevenue)
Middle East and North Africa	6160781.13
Australia and Oceania	6580454.69
Europe	6617209.54
Sub-Saharan Africa	6263026.44
Central America and the Caribbean	6354579.43
North America	6216247.54
Asia	6557065.24

Ordering the Records (Asc, Desc)

```
1 | sql("select Region, max(TotalRevenue) from creditcard group by Region order by Region").show(truncate=False)
```

▶ (2) Spark Jobs

Region	max(TotalRevenue)
Asia	6557065.24
Australia and Oceania	6580454.69
Central America and the Caribbean	6354579.43
Europe	6617209.54
Middle East and North Africa	6160781.13
North America	6216247.54
Sub-Saharan Africa	6263026.44

```
1 | sql("select Region, max(TotalRevenue) from creditcard group by Region order by Region desc").show(truncate=False)
```

▶ (2) Spark Jobs

Region	max(TotalRevenue)
Sub-Saharan Africa	6263026.44
North America	6216247.54
Middle East and North Africa	6160781.13
Europe	6617209.54
Central America and the Caribbean	6354579.43
Australia and Oceania	6580454.69
Asia	6557065.24

This file is meant for personal use by rg.ravigupta91@gmail.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Joining the Tables

```
1 # join (inner) creditcard and sales_table, display the results
2 sql("""select a.Region, a.Country, b.SalesPerson
3     from creditcard a
4     join sales_table b
5     on trim(a.Region) = trim(b.Region""").show(5, False)
```

▶ (2) Spark Jobs

```
+-----+-----+-----+
|Region          |Country           |SalesPerson   |
+-----+-----+-----+
|Middle East and North Africa|United Arab Emirates|Mohammed Saif|
|Middle East and North Africa|Azerbaijan        |Mohammed Saif|
|Middle East and North Africa|Yemen              |Mohammed Saif|
|Middle East and North Africa|Bahrain            |Mohammed Saif|
|Middle East and North Africa|Jordan              |Mohammed Saif|
+-----+-----+-----+
only showing top 5 rows
```

Joining the Tables

```
1 # join (inner) creditcard and sales_table, apply a where condition, display the results
2 df = sql("""select a.Region, a.Country, b.SalesPerson
3     from creditcard a
4     join sales_table b
5     on trim(a.Region) = trim(b.Region)
6     where trim(a.Region) = "Asia"
7     """).show(5, False)
```

► (3) Spark Jobs

```
+-----+-----+
|Region|Country |SalesPerson   |
+-----+-----+
|Asia  |Nepal    |Tatyaso Martin|
|Asia  |Mongolia|Tatyaso Martin|
|Asia  |Brunei   |Tatyaso Martin|
|Asia  |Laos     |Tatyaso Martin|
|Asia  |Mongolia|Tatyaso Martin|
+-----+-----+
only showing top 5 rows
```

Saving the Results in to DBFS

1



```
1 # write the results in to DBFS
2 df = sql("""select a.Region, a.Country, b.SalesPerson
3     from creditcard a
4     join sales_table b
5     on trim(a.Region) = trim(b.Region)
6     where trim(a.Region) = "Asia"
7     """
8
9
10 df.coalesce(1).write.option("header","true").mode("overwrite").csv("/FileStore/tables/spark/")
```

- ▶ (1) Spark Jobs
- ▶ df: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 1 more fields]

Saving the Results in to DBFS

2



1	%fs ls "/FileStore/tables/spark/"
2	path
1	dbfs:/FileStore/tables/spark/_SUCCESS
2	dbfs:/FileStore/tables/spark/_committed_6906066050643623387
3	dbfs:/FileStore/tables/spark/_started_6906066050643623387
4	dbfs:/FileStore/tables/spark/part-00000-tid-6906066050643623387-cf95ee82-b0a3-45fe-8105-779871192a2c-479-1-c000.csv

Showing all 4 rows.

```
1 # read the csv file from stored location
2 newDF = spark.read.format("csv").option("header", "true").option("inferSchema", "true") \
3 .load("/FileStore/tables/spark/part-00000-tid-7896517250555459845-acebf487-b3db-448d-ab44-06942b9588b2-254-1-c000.csv")
4
5 newDF.show(10, False)
```

► (3) Spark Jobs

► newDF: pyspark.sql.dataframe.DataFrame = [Region: string, Country: string ... 1 more fields]

```
+-----+-----+-----+
|Region|Country|SalesPerson   |
+-----+-----+-----+
|Asia  |Nepal  |Tatyaso Martin|
|Asia  |Mongolia|Tatyaso Martin|
|Asia  |Brunei |Tatyaso Martin|
|Asia  |Laos   |Tatyaso Martin|
|Asia  |Mongolia|Tatyaso Martin|
|Asia  |Indonesia|Tatyaso Martin|
|Asia  |Bhutan |Tatyaso Martin|
|Asia  |Tajikistan|Tatyaso Martin|
|Asia  |India   |Tatyaso Martin|
|Asia  |Maldives|Tatyaso Martin|
+-----+-----+-----+
```

only showing top 10 rows

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Summary

- Apache Spark is a open Source from Apache Software Foundation
- It is 100 times faster than Hadoop MapReduce
- Resilient Distributed Dataset (RDD) and Direct Acyclic Graph (DAG) are the two important abstractions of Spark
- There are two types of RDD Operations: Transformation and Actions
- Spark SQL API provides structured data processing



Thank You

This file is meant for personal use by rg.ravigupta91@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.