Spark is a engine which will work on large data sets in a distributed systems. It uses in-memory (RAM) for processing the data and will get o/p fast. It has built-in modules Spark SQL, Streaming, Py-Spark, MLib, Graph X.... For Data Scientists It will be helpful for Exploratory Data Analysis (Visualizing), Feature Extraction (Consolidating Large Data Sets to Minimum Useful Information Data Sets) and Machine Learning.

Creating the connection is as simple as creating an instance of the SparkContext class. The class constructor takes a few optional arguments that allow you to specify the attributes of the cluster you're connecting to. An object holding all these attributes can be created with the SparkConf() constructor.

**Using DataFrames**

Spark's core data structure is the Resilient Distributed Dataset (RDD). This is a low level object that lets Spark work its magic by splitting data across multiple nodes in the cluster. However, RDDs are hard to work with directly, so in this course you'll be using the Spark DataFrame abstraction built on top of RDDs.

The Spark DataFrame was designed to behave a lot like a SQL table (a table with variables in the columns and observations in the rows). Not only are they easier to understand, DataFrames are also more optimized for complicated operations than RDDs.

When you start modifying and combining columns and rows of data, there are many ways to arrive at the same result, but some often take much longer than others. When using RDDs, it's up to the data scientist to figure out the right way to optimize the query, but the DataFrame implementation has much of this optimization built in!

To start working with Spark DataFrames, you first have to create a SparkSession object from your SparkContext. You can think of the SparkContext as your connection to the cluster and the SparkSession as your interface with that connection.

Remember, for the rest of this course you'll have a SparkSession called spark available in your workspace!

Q)Which of the following is an advantage of Spark DataFrames over RDDs?

A)Operations using DataFrames are automatically optimized.

Your SparkSession has an attribute called catalog which lists all the data inside the cluster. Thisattributeas a few methods for extracting different pieces of information.One of the most useful is the .listTables() method, which returns the names of all the tables in your cluster as a list.

spark.catalog.listTables()

* Use the .sql() method to get the first 10 rows of the flights table and save the result to flights10. The variable query contains the appropriate SQL query.
* Use the DataFrame method .show() to print flights10.

# Don't change this query

query ="FROM flights SELECT \* LIMIT 10"

# Get the first 10 rows of flights

flights10 = spark.sql(query)

# Show the results

flights10.show()

Sometimes it makes sense to then take that table and work with it locally using a tool like pandas. Spark DataFrames make that easy with the .toPandas() method. Calling this method on a Spark DataFrame returns the corresponding pandas DataFrame. It's as simple as that!

This time the query counts the number of flights to each airport from SEA and PDX.

Remember, there's already a SparkSession called spark in your workspace!

# Don't change this query

query = "SELECT origin, dest, COUNT(\*) as N FROM flights GROUP BY origin, dest"

# Run the query

flight\_counts = spark.sql(query)

# Convert the results to a pandas DataFrame

pd\_counts = flight\_counts.toPandas()

# Print the head of pd\_counts

print(pd\_counts.head())

In the last exercise, you saw how to move data from Spark to pandas. However, maybe you want to go the other direction, and put a pandas DataFrame into a Spark cluster! The SparkSession class has a method for this as well.

The .createDataFrame() method takes a pandas DataFrame and returns a Spark DataFrame.

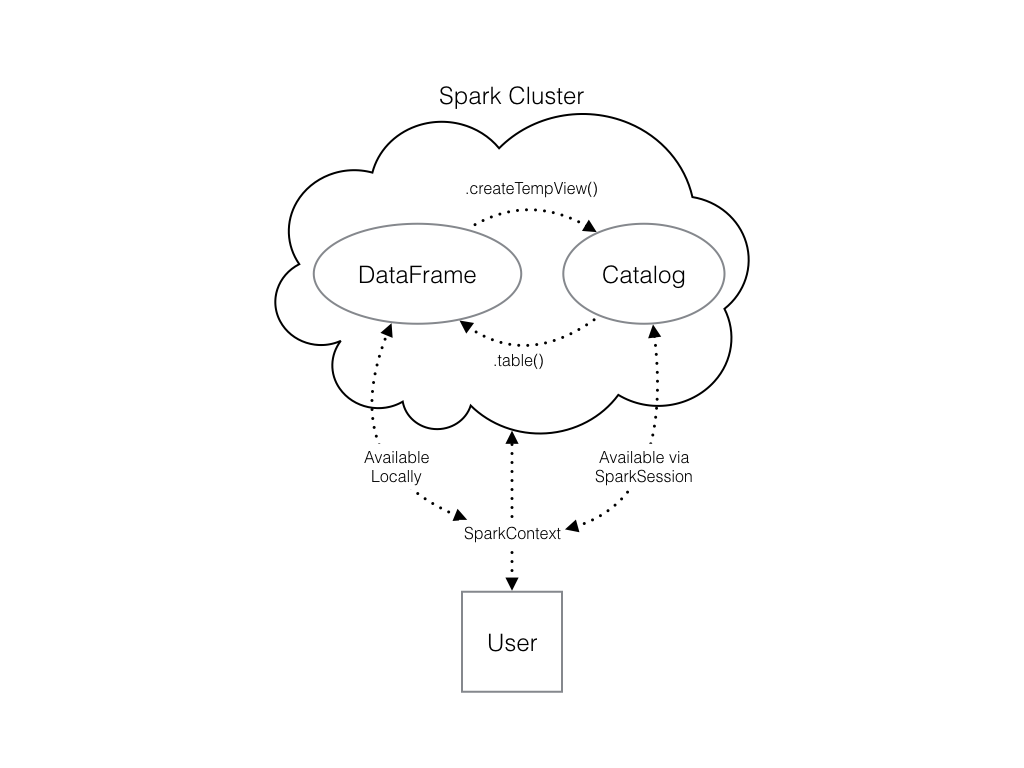
The output of this method is stored locally, not in the SparkSession catalog. This means that you can use all the Spark DataFrame methods on it, but you can't access the data in other contexts.

For example, a SQL query (using the .sql() method) that references your DataFrame will throw an error. To access the data in this way, you have to save it as a temporary table.

You can do this using the .createTempView() Spark DataFrame method, which takes as its only argument the name of the temporary table you'd like to register. This method registers the DataFrame as a table in the catalog, but as this table is temporary, it can only be accessed from the specific SparkSession used to create the Spark DataFrame.

There is also the method .createOrReplaceTempView(). This safely creates a new temporary table if nothing was there before, or updates an existing table if one was already defined. You'll use this method to avoid running into problems with duplicate tables.

Check out the diagram to see all the different ways your Spark data structures interact with each other.



There's already a SparkSession called spark in your workspace, numpy has been imported as np, and pandas as pd.

# Create pd\_temp

pd\_temp = pd.DataFrame(np.random.random(10))

# Create spark\_temp from pd\_temp

spark\_temp = spark.createDataFrame(pd\_temp)

# Examine the tables in the catalog

print(spark.catalog.listTables())

# Add spark\_temp to the catalog

spark\_temp.createOrReplaceTempView('temp')

# Examine the tables in the catalog again

print(spark.catalog.listTables())

Now you know how to put data into Spark via pandas, but you're probably wondering why deal with pandas at all? Wouldn't it be easier to just read a text file straight into Spark? Of course it would!

Luckily, your SparkSession has a .read attribute which has several methods for reading different data sources into Spark DataFrames. Using these you can create a DataFrame from a .csv file just like with regular pandas DataFrames!

The variable file\_path is a string with the path to the file airports.csv. This file contains information about different airports all over the world.

A SparkSession named spark is available in your workspace.

# Don't change this file path

file\_path = "/usr/local/share/datasets/airports.csv"

# Read in the airports data

airports = spark.read.csv(file\_path,header=True)

# Show the data

airports.show()

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

Yarn will allocate you resources like ram for execution of the program. So when you submit the program driver will send the program to all blocks which contain data and executers(Resource) in the nodes will execute program on the data and all the o/p will comes to the driver, In the driver we can write the code where we want to store the data. **Driver – App Master**