

|  |
| --- |
|  |
| Apache spark |
| an introduction to Apache Spark and Pyspark |
| Abdul wasay |

Industries are using Hadoop extensively to analyze their data sets. The reason is that the Hadoop framework is based on a simple programming model (MapReduce) and it enables a computing solution that is scalable, flexible, fault-tolerant, and cost-effective. Here, the main concern is to maintain speed in processing large datasets in terms of waiting time between queries and waiting time to run the program.

Spark was introduced by Apache Software Foundation for speeding up the Hadoop computational computing software process.

As against a common belief, **Spark is not a modified version of Hadoop** and is not, really, dependent on Hadoop because it has its own cluster management. Hadoop is just one of the ways to implement Spark.

Spark uses Hadoop in two ways – one is **storage** and the second is **processing**. Since Spark has its own cluster management computation, it uses Hadoop for storage purposes only

Contents

[an introduction to apache spark and Pyspark 1](#_Toc148347770)

[**Introduction to Apache Spark:** 4](#_Toc148347771)

[**Evolution of Apache Spark** 4](#_Toc148347772)

[**Features of Apache Spark** 5](#_Toc148347773)

[**Components of Spark** 5](#_Toc148347774)

[ Apache Spark Core 5](#_Toc148347775)

[ Spark SQL 5](#_Toc148347776)

[ Spark Streaming 5](#_Toc148347777)

[ MLlib (Machine Learning Library) 5](#_Toc148347778)

[ GraphX 5](#_Toc148347779)

[**Apache Spark – RDD** 6](#_Toc148347780)

[**Resilient Distributed Datasets** 6](#_Toc148347781)

[**Apache Spark - Installation** 7](#_Toc148347782)

[**Step 1: Verifying Java Installation** 7](#_Toc148347783)

[**Step 2: Verifying Scala installation** 7](#_Toc148347784)

[**Step 3: Downloading Apache Spark** 7](#_Toc148347785)

[**Step 4: Verifying the Spark Installation** 7](#_Toc148347786)

[**Spark Modes** 9](#_Toc148347787)

[**Spark Cluster Mode Configuration Ubuntu** 10](#_Toc148347788)

[**Introduction to PySpark** 12](#_Toc148347789)

[**Features of PySpark** 12](#_Toc148347790)

[**advantages of PySpark** 13](#_Toc148347791)

[**PySpark Architecture** 14](#_Toc148347792)

[**Spark Context VS Spark Session** 14](#_Toc148347793)

[**Cluster Manager Types** 15](#_Toc148347794)

[**Pyspark Installation** 17](#_Toc148347795)

[**Spark Web UI** 18](#_Toc148347796)

[**PySpark RDD** 19](#_Toc148347797)

[**RDD Operations** 19](#_Toc148347798)

[**PySpark DataFrame** 21](#_Toc148347799)

[**Supported file formats** 24](#_Toc148347800)

# **Introduction to Apache Spark:**

Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing. The main feature of Spark is its **in-memory cluster computing** which increases the processing speed of an application.

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries, and streaming. Apart from supporting all these workloads in a respective system, it reduces the management burden of maintaining separate tools.

Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python, and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including [Spark SQL](https://spark.apache.org/docs/latest/sql-programming-guide.html) for SQL and structured data processing, [pandas API on Spark](https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_ps.html) for pandas workloads, [MLlib](https://spark.apache.org/docs/latest/ml-guide.html) for machine learning, [GraphX](https://spark.apache.org/docs/latest/graphx-programming-guide.html) for graph processing, and [Structured Streaming](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html) for incremental computation and stream processing.

# **Evolution of Apache Spark**

Spark is one of Hadoop’s sub-projects developed in 2009 in UC Berkeley’s AMPLab by Matei Zaharia. It was Open Sourced in 2010 under a BSD license. It was donated to the Apache software foundation in 2013, and now Apache Spark has become a top-level Apache project since Feb-2014.

# **Features of Apache Spark**

Apache Spark has following features.

* **Speed** − Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
* **Supports multiple languages** − Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
* **Advanced Analytics** − Spark not only supports ‘Map’ and ‘reduce’. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

# **Components of Spark**

### Apache Spark Core

Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

### Spark SQL

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

### Spark Streaming

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

### MLlib (Machine Learning Library)

MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times as fast as the Hadoop disk-based version of **Apache Mahout** (before Mahout gained a Spark interface).

### GraphX

GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

# **Apache Spark – RDD**

## **Resilient Distributed Datasets**

Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

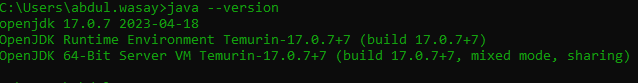
There are two ways to create RDDs − **parallelizing** an existing collection in your driver program, or **referencing a dataset** in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

# **Apache Spark - Installation**

## **Step 1: Verifying Java Installation**

For this make sure you have installed java on your system , I am currently using windows for installation so to check java is installed or not first check java version go to command prompt and type java –version



Here it shows that I have installed jdk 17.

## **Step 2: Verifying Scala installation**

Once you have done the installation and verified Java the 2nd step is to verify Scala installation.

For this go to the command prompt and type Scala –version



It shows that I have installed Scala version 3.2.2

In case you don’t have installed Scala on your system go to their website and download and install Scala from there. And also add environmental variables for Scala and a path to Scala as well.

## **Step 3: Downloading Apache Spark**

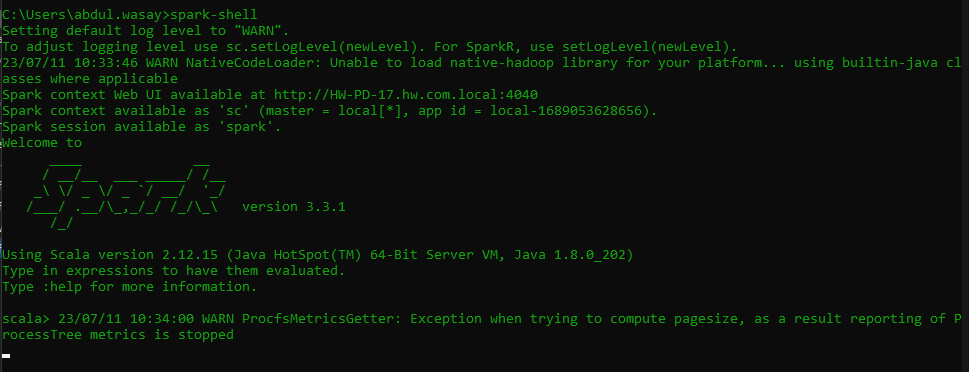
After successful installation and verification of Scala, the next step is to download Apache Spark for this go to their website and download the zip file from there extract the zip file and make sure to add the path to Spark.

%SPARK\_HOME%\bin

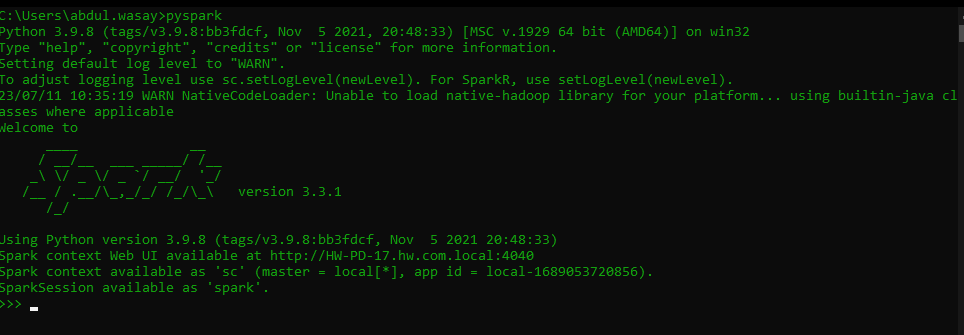
## **Step 4: Verifying the Spark Installation**

Once you correctly installed Spark on your system the next step is to verify installation if you going to use Pyspark make sure to install Python in administrator mode.

Now run the command spark-shell in command prompt.



Now you can see the spark is loaded and it is with Scala to use it with python that is pyspark run command pyspark.



Now it is with python. Since we are going to use PySpark so all the example will be on python.

# **Spark Modes**

* **Local Mode:**

In this mode, Spark runs on a single machine, using all available CPU cores. It's primarily used for development, debugging, and testing on a local workstation.

It's not suitable for distributed processing, as it doesn't take advantage of a cluster's resources.

* **Standalone Mode:**

The standalone mode is Spark's built-in cluster manager. It allows you to run Spark applications on a cluster of machines without the need for external cluster managers like Apache Hadoop YARN or Apache Mesos.

It's suitable for small to medium-sized clusters and is often used for learning purposes or in environments with limited resources.

* **YARN Mode:**

Apache Hadoop YARN (Yet Another Resource Negotiator) is a resource management and job scheduling framework that can be used to run Spark applications in a distributed cluster.

In YARN mode, Spark jobs are submitted to YARN, which manages cluster resources and schedules tasks.

* **Mesos Mode:**

Apache Mesos is another cluster manager that can be used to deploy Spark applications. Mesos allows for efficient resource sharing and isolation between different applications.

You can run Spark on Mesos using the Mesos cluster manager.

* **Cluster Mode:**

In this mode, Spark runs on a cluster using a cluster manager like YARN, Mesos, or Kubernetes. It's the mode typically used in production environments.

Spark driver runs on one of the cluster nodes, and Spark applications are submitted to the cluster manager, which allocates resources and manages task execution.

* **Kubernetes Mode:**

Apache Spark can also run on Kubernetes, a popular container orchestration platform. In this mode, Spark runs as a set of containers managed by Kubernetes.

Kubernetes provides dynamic resource allocation and can be a good fit for cloud-native deployments.

## **Spark Cluster Mode Configuration Ubuntu**

* + 1. Make sure you have installed Java.
    2. Make sure you have installed scala.
    3. Make sure you have installed spark and you have set environmental variable.
    4. Install open-ssh
    5. sudo vim /etc/hosts
    6. Now add entries of master and slaves in host’s file.
       - <MASTER-IP> master
       - <SLAVE01-IP> slave01
       - <SLAVE02-IP> slave02
    7. sudo apt-get install openssh-server openssh-client
    8. Generate key pairs
       - ssh-keygen -t rsa -P ""
    9. Copy the content of .ssh/id\_rsa.pub (of master) to .ssh/authorized\_keys (of all the slaves as well as master).
    10. Check by SSH to all the slaves
        - $ ssh slave01
        - $ ssh slave02
    11. **Spark Master Configuration**
        - Edit spark-env.sh
          * $ cd /usr/local/spark/conf
          * $ cp spark-env.sh.template spark-env.sh
        - Now edit the configuration file spark-env.sh.
          * $ sudo vim spark-env.sh
          * And set the following parameters.
          * export SPARK\_MASTER\_HOST='<MASTER-IP>'
          * export JAVA\_HOME=<Path\_of\_JAVA\_installation>
    12. **Add Workers**
        - Edit the configuration file slaves in (/usr/local/spark/conf).
        - $ sudo vim slaves
        - And add the following entries.
          * master
          * slave01
          * slave02
    13. **Start Spark Cluster**
        - To start the spark cluster, run the following command on master.
          * $ cd /usr/local/spark
          * $ ./sbin/start-all.sh
        - To stop the spark cluster, run the following command on master.
          * $ cd /usr/local/spark
          * $ ./sbin/stop-all.sh
    14. Check whether services have been started
        - $ jps
    15. Spark Web UI
        - Spark Master UI
          * http://<MASTER-IP>:8080/
        - Spark Application UI
          * http://<MASTER\_IP>:4040/

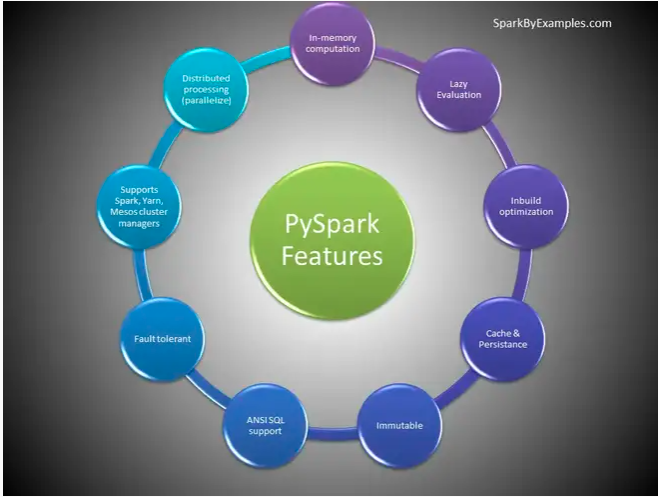
# **Introduction to PySpark**

PySpark is a Spark library written in Python to run Python applications using Apache Spark capabilities, using PySpark we can run applications parallelly on the distributed cluster (multiple nodes).

In other words, PySpark is a Python API for Apache Spark. Apache Spark is an analytical processing engine for large scale powerful distributed data processing and machine-learning applications



## **Features of PySpark**



|  |  |
| --- | --- |
| * In-memory computation * Distributed processing using parallelize * Can be used with many cluster managers (Spark, Yarn, Mesos e.t.c) * Fault-tolerant * Immutable * Lazy evaluation * Cache & persistence * Inbuild-optimization when using DataFrames * Supports ANSI SQL  **advantages of PySpark**  * PySpark is a general-purpose, in-memory, distributed processing engine that allows you to process data efficiently in a distributed fashion. * Applications running on PySpark are 100x faster than traditional systems. * You will get great benefits using PySpark for data ingestion pipelines. * Using PySpark we can process data from Hadoop HDFS, AWS S3, and many file systems. * PySpark also is used to process real-time data using Streaming and Kafka. * Using PySpark streaming you can also stream files from the file system and also stream from the socket. * PySpark natively has machine learning and graph libraries.ss |  |
|  |

# **PySpark Architecture**

|  |  |
| --- | --- |
| Apache Spark works in a master-slave architecture where the master is called “Driver” and slaves are called “Workers”. When you run a Spark application, Spark Driver creates a context that is an entry point to your application, and all operations (transformations and actions) are executed on worker nodes, and the resources are managed by Cluster Manager. **Spark Context VS Spark Session** Spark 1. x comes with three entry points: SparkContext, SQLContext, and HiveContext. And with the introduction of Spark 2. x, a new entry point named SparkSession was added. As a result, this single-entry point effectively combines all of the functionality available in the three aforementioned contexts. Let’s do a comparison between SparkSession vs SparkContext.  ***Spark Context:***  SparkContext is the primary point of entry for Spark capabilities. A SparkContext represents a Spark cluster’s connection that is useful in building RDDs, accumulators, and broadcast variables on the cluster. It enables your Spark Application to connect to the Spark Cluster using Resource Manager. Also, before the creation of SparkContext, SparkConf must be created.  ***Spark session:***  Apache Spark 2.0 is the company’s next significant release. This is a significant shift in the degree of abstraction for the Spark API and libraries. Previously, as RDD was the major API, SparkContext was the entry point for Spark. It was constructed and modified with the help of context APIs. At that time, we have to use a distinct context for each API. We required StreamingContext for Streaming, SQLContext for SQL, and HiveContext for Hive. However, because the DataSet and DataFrassme APIs are becoming new independent APIs, we require an entry-point construct for them. As a result, in Spark 2.0, we have a new entry point built for DataSet and DataFrame APIs called SparkSession. |  |
|  |
|  |

## **Cluster Manager Types**

|  |  |  |
| --- | --- | --- |
| * **Standalone –**   A simple cluster manager included with Spark that makes it easy to set up a cluster.   * **Apache Mesos –**   Mesons is a Cluster manager that can also run Hadoop MapReduce and PySpark applications.   * **Hadoop YARN –**   the resource manager in Hadoop 2. This is mostly used, as a cluster manager.   * **Kubernetes –**   an open-source system for automating deployment, scaling, and management of containerized applications. |  |  |
| local – which is not really a cluster manager but still I wanted to mention as we use “local” for master() in order to run Spark on your laptop/computer.  **PySpark Modules & Packages**   * PySpark RDD (pyspark.RDD) * PySpark DataFrame and SQL (pyspark.sql) * PySpark Streaming (pyspark.streaming) * PySpark MLib (pyspark.ml, pyspark.mllib) * PySpark GraphFrames (GraphFrames) * PySpark Resource (pyspark.resource) |  |  |
|  |

# **Pyspark Installation**

|  |
| --- |
|  |

* Step1: Verify Java installation
* Step 2: Verify Python Installation
* Step 3: Verify Spark Installation
* Step 4: Add winutils Path

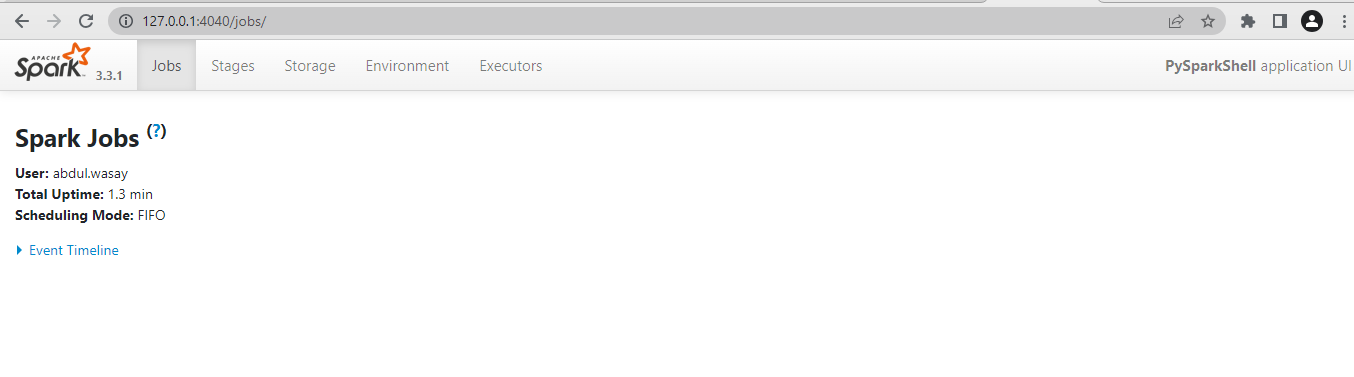
And then run pyspark in cmd.

# **Spark Web UI**



Now go to the local host address

And see Spark web UI.



Spark UI is separated into below tabs

* Spark Jobs (It tell us the scheduling mode, number of spark jobs, number of stages and description.)
* Stages
* Tasks
* Storage
* Environment
* Executors
* SQL

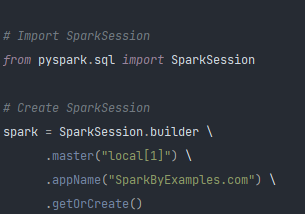
# **PySpark RDD**

PySpark RDD (Resilient Dissstributed Dataset) is a fundamental data structure of PySpark that is fault-tolerant, immutable distributed collections of objects, which means once you create an RDD you cannot change it. Each dataset in RDD is divided into logical partitions, which can be computed on different nodes of the cluster.

In order to create an RDD, first, you need to create a SparkSession which is an entry point to the PySpark application. SparkSession can be created using a builder() or newSession() method of the SparkSession.

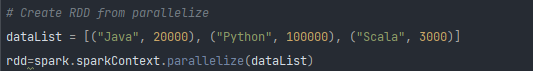
There are two methods to create RDD 1) Parallelize 2) External source

Here is the code



Using parallelize:

SparkContext has several functions to use with RDDs. For example, its parallelize() method is used to create an RDD from a list.



Using text file (external source):

RDD can also be created from a text file using the textFile() function of the SparkContext.



## **RDD Operations**

**RDD transformations –**

Transformations are lazy operations. When you run a transformation (for example update), instead of updating a current RDD, these operations return another RDD. Transformations on Spark RDD return another RDD and transformations are lazy meaning they don’t execute until you call an action on RDD. Some transformations on RDDs are flatMap(), map(), reduceByKey(), filter(), sortByKey(), and return new RDD instead of updating the current.s

**RDD actions –**

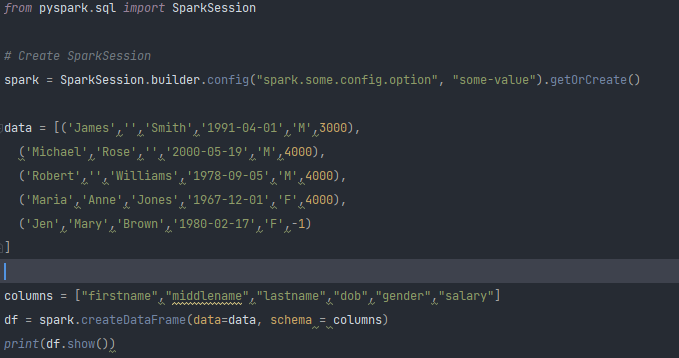
Operations that trigger computation and return RDD values to the driver. RDD Action operation returns the values from an RDD to a driver node. In other words, any RDD function that returns non-RDD [T] is considered an action.ss

Some actions on RDDs are count(), collect(), first(), max(), reduce() and more.

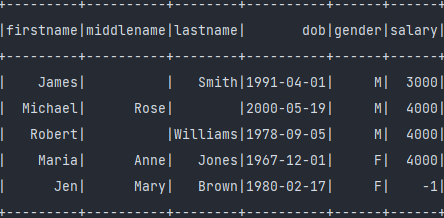
# **PySpark DataFrame**

DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as structured data files, tables in Hive, external databases, or existing RDDs.

Creating Dataframe:



Output:



*print*(df.schema)

it will print schema



Now we are going to create data frame from csv and then we will write that data frame into the database without even creating a database table

First read csv

*from* pyspark.sql *import* SparkSession  
*from* pyspark.sql.types *import* \*  
*from* pyspark *import* SparkConf  
  
  
spark = SparkSession.builder.appName('WriteDB').getOrCreate()

conf = SparkConf().setAll(pairs=[("spark.jars","C:/Spark/jars/postgresql-42.5.2"), ("spark.jars.packages","org.postgresql:postgresql-42.5.2")])

# the above jars for writing data into database.  
df = spark.read.csv('employees.csv', header=*True*, inferSchema=*True*)

*print*(df.show()) *# print dataframe*

*print*(df.columns) *# column name*

*print*(df.schema) *# show datatypes*

*print*(df.select('EMPLOYEE\_ID', 'FIRST\_NAME').groupby('EMPLOYEE\_ID', 'FIRST\_NAME').count().show())

#same as sql select and group by statement.  
*print*(df.filter(df.EMPLOYEE\_ID > 121).show()) # filter the column employee\_id

*print*(df.filter(df['EMPLOYEE\_ID']>200).limit(5).show()) # first filter, then limit records to 5 and then it will print the result.

*print*(df.na.drop().show()) *#drop null values*

df2 = df.dropDuplicates() # it will drop duplicate values

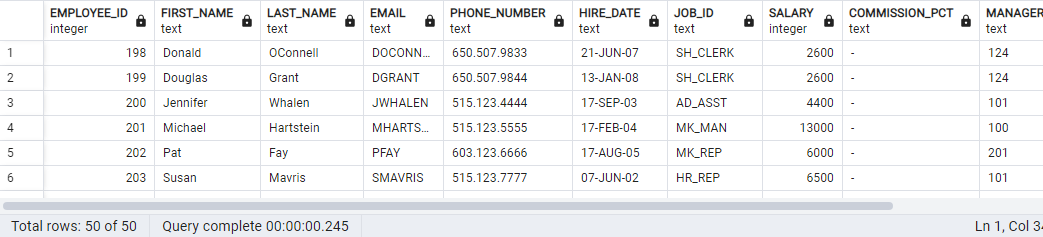
*print*(df2.show()) #it will print the result

the below code will create a SQL template

df.createOrReplaceTempView('emp') #create a SQL template  
sqlquery=spark.sql('select EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME from emp where EMPLOYEE\_ID > 200 group by EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME order by EMPLOYEE\_ID limit 5')  
*print*(sqlquery.show()) # print result

next we will write data into the database

*# writing to database*url = "jdbc:postgresql://192.168.2.49:5432/HMS\_source"  
table="employeeData"  
driver="org.postgresql.Driver"  
user="your-user"  
password="your-passsword"  
df.write.format('jdbc').option("driver", driver).option("url",url).option("dbtable",table).option("user",user).option("password",password).save()

You can see that data has been loaded into database.

In order to append data into existing table we use an option and mode parameter

# .mode("append")

df.write.format('jdbc').option("driver", driver).option("url",url).mode("append") option("dbtable",table).option("user",user).option("password",password).save()

So the

There are many other functions available in the Spark data frame, you can visit Spark’s official documentation.

Read data from the table

In order to read data from the table we use dataframe.read instead of dataframe.write

Here is the code

url = "jdbc:postgresql://192.168.2.49:5432/HMS\_source"  
table='"EDW".booking\_fact'  
driver="org.postgresql.Driver"  
user="your-user"  
password="your-password"  
query='select count(1), nbook\_id\_bk from "EDW".booking\_fact group by nbook\_id\_bk' # the query will query the result  
  
  
df = spark.read.format('jdbc').option("driver", driver).option("url",url).option("dbtable",table).option("user",user).option("password",password).load()  
*print*(df.show()) #print the result

there are many things you can do with data frame functions.

* You can rename a column.
* You can add a new column, or remove a column.
* You can add auto-incremented id **monotonically\_increasing\_id**
* You can define your own schema.
* You can also convert pandas data frames to spark data frames.
* **And many more.**

# **Supported file formats**

DataFrame has a rich set of APIS that supports reading and writing several file formats

• csv

• text

• Avro

• Parquet

• tsv

• XML and many more