[**Apache Spark 2.x - Data processing - Getting Started - Introduction**](http://discuss.itversity.com/t/apache-spark-2-x-data-processing-getting-started-introduction/19482)

As part of this session we will understand what is Data Frames, how data frames can be created from (text) files, hive tables, relational databases using JDBC etc. We will also understand how data frame can be registered as in memory table/view and run SQL on top of it as well as some of the important functions that can be used to manipulate data as part of data frame operations.

* Data Frames – Overview
* Reading text data from files
* Reading data from hive
* Reading data from MySQL over JDBC
* Data Frame Operations – Overview
* Spark SQL – Overview
* Functions to manipulate data

Data Frames is nothing but RDD with structure.

* Data Frame can be created on any data set which have structure associated with it.
* Attributes/columns in a data frame can be referred using names.
* One can create data frame using data from files, hive tables, relational tables over JDBC.
* Common functions on Data Frames
  + printSchema – to print the column names and data types of data frame
  + show – to preview data (default 20 records)
  + describe – to understand characteristics of data
  + count – to get number of records
  + collect – to convert data frame into Array
* Once data frame is created, we can process data using 2 approaches.
  + Native Data Frame APIs
  + Register as temp table and run queries using spark.sql
* To work with Data Frames as well as Spark SQL, we need to create object of type SparkSession
* pyspark --master yarn --conf spark.ui.port=12576 --executor-memory 2G --num-executors 2

sc

<pyspark.context.SparkContext object at 0x7fdcae7f4e50>

pyspark

<module 'pyspark' from '/usr/hdp/current/spark-client/python/pyspark/\_\_init\_\_.py'>

orders = sc.textFile('/Users/itversity/Research/data/retail\_db/orders')

>>> type(orders)

<class 'pyspark.rdd.RDD'>

os.system("hdfs dfs -ls /user/saurabhbhanwala")

Found 3 items

drwxr-xr-x - saurabhbhanwala hdfs 0 2020-09-10 00:41 /user/saurabhbhanwala/.sparkStaging

drwx------ - saurabhbhanwala hdfs 0 2020-09-05 03:11 /user/saurabhbhanwala/.staging

drwxr-xr-x - saurabhbhanwala hdfs 0 2020-09-06 05:49 /user/saurabhbhanwala/retail\_db

orders = sc.textFile('file:///data/retail\_db/orders')

<https://stackoverflow.com/questions/35123245/load-local-file-not-hdfs-fails-at-spark>

orders = sc.textFile('/user/saurabhbhanwala/retail\_db/orders')

>>> for i in orders.take(10): print(i)

...

1,2013-07-25 00:00:00.0,11599,CLOSED

2,2013-07-25 00:00:00.0,256,PENDING\_PAYMENT

3,2013-07-25 00:00:00.0,12111,COMPLETE

4,2013-07-25 00:00:00.0,8827,CLOSED

5,2013-07-25 00:00:00.0,11318,COMPLETE

6,2013-07-25 00:00:00.0,7130,COMPLETE

7,2013-07-25 00:00:00.0,4530,COMPLETE

8,2013-07-25 00:00:00.0,2911,PROCESSING

9,2013-07-25 00:00:00.0,5657,PENDING\_PAYMENT

10,2013-07-25 00:00:00.0,5648,PENDING\_PAYMENT

orders.first()

u'1,2013-07-25 00:00:00.0,11599,CLOSED'

type(orders.first())

<type 'unicode'>

86

What about the shortcut CTRL+L?

It works for all shells e.g. Python, Bash, MySQL, MATLAB, etc.

export SPARK\_MAJOR\_VERSION=2

pyspark --master yarn --conf spark.ui.port=12576 --executor-memory 2G --num-executors 2

from pyspark.context import SparkContext

>>> from pyspark.sql.session import SparkSession

>>> spark = SparkSession.builder.master('yarn').appName('Create DataFrame from JDBC').getOrCreate()

ordersDF = spark.read.csv('/user/saurabhbhanwala/retail\_db/orders')

>>> type(ordersDF)

<class 'pyspark.sql.dataframe.DataFrame'>

ordersDF.first()

Row(\_c0=u'1', \_c1=u'2013-07-25 00:00:00.0', \_c2=u'11599', \_c3=u'CLOSED')

ordersDF.select('\_c0', '\_c1').show()

+---+--------------------+

|\_c0| \_c1|

+---+--------------------+

| 1|2013-07-25 00:00:...|

| 2|2013-07-25 00:00:...|

| 3|2013-07-25 00:00:...|

| 4|2013-07-25 00:00:...|

| 5|2013-07-25 00:00:...|

| 6|2013-07-25 00:00:...|

| 7|2013-07-25 00:00:...|

| 8|2013-07-25 00:00:...|

| 9|2013-07-25 00:00:...|

| 10|2013-07-25 00:00:...|

| 11|2013-07-25 00:00:...|

| 12|2013-07-25 00:00:...|

| 13|2013-07-25 00:00:...|

| 14|2013-07-25 00:00:...|

| 15|2013-07-25 00:00:...|

| 16|2013-07-25 00:00:...|

| 17|2013-07-25 00:00:...|

| 18|2013-07-25 00:00:...|

| 19|2013-07-25 00:00:...|

| 20|2013-07-25 00:00:...|

+---+--------------------+

<https://stackoverflow.com/questions/37741841/fetch-more-than-20-rows-and-display-full-value-of-column-in-spark-shell>

print schema

ordersDF.printSchema()

root

|-- \_c0: string (nullable = true)

|-- \_c1: string (nullable = true)

|-- \_c2: string (nullable = true)

|-- \_c3: string (nullable = true)

ordersDF.show()

+---+--------------------+-----+---------------+

|\_c0| \_c1| \_c2| \_c3|

+---+--------------------+-----+---------------+

| 1|2013-07-25 00:00:...|11599| CLOSED|

| 2|2013-07-25 00:00:...| 256|PENDING\_PAYMENT|

| 3|2013-07-25 00:00:...|12111| COMPLETE|

| 4|2013-07-25 00:00:...| 8827| CLOSED|

| 5|2013-07-25 00:00:...|11318| COMPLETE|

| 6|2013-07-25 00:00:...| 7130| COMPLETE|

| 7|2013-07-25 00:00:...| 4530| COMPLETE|

| 8|2013-07-25 00:00:...| 2911| PROCESSING|

| 9|2013-07-25 00:00:...| 5657|PENDING\_PAYMENT|

| 10|2013-07-25 00:00:...| 5648|PENDING\_PAYMENT|

| 11|2013-07-25 00:00:...| 918| PAYMENT\_REVIEW|

| 12|2013-07-25 00:00:...| 1837| CLOSED|

| 13|2013-07-25 00:00:...| 9149|PENDING\_PAYMENT|

| 14|2013-07-25 00:00:...| 9842| PROCESSING|

| 15|2013-07-25 00:00:...| 2568| COMPLETE|

| 16|2013-07-25 00:00:...| 7276|PENDING\_PAYMENT|

| 17|2013-07-25 00:00:...| 2667| COMPLETE|

| 18|2013-07-25 00:00:...| 1205| CLOSED|

| 19|2013-07-25 00:00:...| 9488|PENDING\_PAYMENT|

| 20|2013-07-25 00:00:...| 9198| PROCESSING|

+---+--------------------+-----+---------------+

ordersDF.show(30,False)

+---+---------------------+-----+---------------+

|\_c0|\_c1 |\_c2 |\_c3 |

+---+---------------------+-----+---------------+

|1 |2013-07-25 00:00:00.0|11599|CLOSED |

|2 |2013-07-25 00:00:00.0|256 |PENDING\_PAYMENT|

|3 |2013-07-25 00:00:00.0|12111|COMPLETE |

|4 |2013-07-25 00:00:00.0|8827 |CLOSED |

|5 |2013-07-25 00:00:00.0|11318|COMPLETE |

|6 |2013-07-25 00:00:00.0|7130 |COMPLETE |

|7 |2013-07-25 00:00:00.0|4530 |COMPLETE |

|8 |2013-07-25 00:00:00.0|2911 |PROCESSING |

|9 |2013-07-25 00:00:00.0|5657 |PENDING\_PAYMENT|

|10 |2013-07-25 00:00:00.0|5648 |PENDING\_PAYMENT|

|11 |2013-07-25 00:00:00.0|918 |PAYMENT\_REVIEW |

|12 |2013-07-25 00:00:00.0|1837 |CLOSED |

|13 |2013-07-25 00:00:00.0|9149 |PENDING\_PAYMENT|

|14 |2013-07-25 00:00:00.0|9842 |PROCESSING |

|15 |2013-07-25 00:00:00.0|2568 |COMPLETE |

|16 |2013-07-25 00:00:00.0|7276 |PENDING\_PAYMENT|

|17 |2013-07-25 00:00:00.0|2667 |COMPLETE |

|18 |2013-07-25 00:00:00.0|1205 |CLOSED |

|19 |2013-07-25 00:00:00.0|9488 |PENDING\_PAYMENT|

|20 |2013-07-25 00:00:00.0|9198 |PROCESSING |

|21 |2013-07-25 00:00:00.0|2711 |PENDING |

|22 |2013-07-25 00:00:00.0|333 |COMPLETE |

|23 |2013-07-25 00:00:00.0|4367 |PENDING\_PAYMENT|

|24 |2013-07-25 00:00:00.0|11441|CLOSED |

|25 |2013-07-25 00:00:00.0|9503 |CLOSED |

|26 |2013-07-25 00:00:00.0|7562 |COMPLETE |

|27 |2013-07-25 00:00:00.0|3241 |PENDING\_PAYMENT|

|28 |2013-07-25 00:00:00.0|656 |COMPLETE |

|29 |2013-07-25 00:00:00.0|196 |PROCESSING |

|30 |2013-07-25 00:00:00.0|10039|PENDING\_PAYMENT|

+---+---------------------+-----+---------------+

ordersDF.describe()

DataFrame[summary: string, \_c0: string, \_c1: string, \_c2: string, \_c3: string]

ordersDF.describe().show()

+-------+------------------+--------------------+-----------------+---------------+

|summary| \_c0| \_c1| \_c2| \_c3|

+-------+------------------+--------------------+-----------------+---------------+

| count| 68883| 68883| 68883| 68883|

| mean| 34442.0| null|6216.571098819738| null|

| stddev|19884.953633337947| null|3586.205241263963| null|

| min| 1|2013-07-25 00:00:...| 1| CANCELED|

| max| 9999|2014-07-24 00:00:...| 9999|SUSPECTED\_FRAUD|

+-------+------------------+--------------------+-----------------+---------------+

ordersDF.count()

68883

hdfs dfs -put /data/retail\_db\_json/ /user/saurabhbhanwala/

[saurabhbhanwala@gw03 ~]$ hdfs dfs -ls /user/saurabhbhanwala/

Found 4 items

drwxr-xr-x - saurabhbhanwala hdfs 0 2020-09-10 01:47 /user/saurabhbhanwala/.sparkStaging

drwx------ - saurabhbhanwala hdfs 0 2020-09-05 03:11 /user/saurabhbhanwala/.staging

drwxr-xr-x - saurabhbhanwala hdfs 0 2020-09-06 05:49 /user/saurabhbhanwala/retail\_db

drwxr-xr-x - saurabhbhanwala hdfs 0 2020-09-10 01:49 /user/saurabhbhanwala/retail\_db\_json

ordersDF = spark.read.json('/user/saurabhbhanwala/retail\_db\_json/orders')

>>> ordersDF.printSchema()

root

|-- order\_customer\_id: long (nullable = true)

|-- order\_date: string (nullable = true)

|-- order\_id: long (nullable = true)

|-- order\_status: string (nullable = true)

ordersDF.select('order\_id', 'order\_status').show()

+--------+---------------+

|order\_id| order\_status|

+--------+---------------+

| 1| CLOSED|

| 2|PENDING\_PAYMENT|

| 3| COMPLETE|

| 4| CLOSED|

| 5| COMPLETE|

| 6| COMPLETE|

| 7| COMPLETE|

| 8| PROCESSING|

| 9|PENDING\_PAYMENT|

| 10|PENDING\_PAYMENT|

| 11| PAYMENT\_REVIEW|

| 12| CLOSED|

| 13|PENDING\_PAYMENT|

| 14| PROCESSING|

| 15| COMPLETE|

| 16|PENDING\_PAYMENT|

| 17| COMPLETE|

| 18| CLOSED|

| 19|PENDING\_PAYMENT|

| 20| PROCESSING|

+--------+---------------+

ordersDF.createTempView('orders')

>>> spark.sql('select \* from orders limit 10').show()

+-----------------+--------------------+--------+---------------+

|order\_customer\_id| order\_date|order\_id| order\_status|

+-----------------+--------------------+--------+---------------+

| 11599|2013-07-25 00:00:...| 1| CLOSED|

| 256|2013-07-25 00:00:...| 2|PENDING\_PAYMENT|

| 12111|2013-07-25 00:00:...| 3| COMPLETE|

| 8827|2013-07-25 00:00:...| 4| CLOSED|

| 11318|2013-07-25 00:00:...| 5| COMPLETE|

| 7130|2013-07-25 00:00:...| 6| COMPLETE|

| 4530|2013-07-25 00:00:...| 7| COMPLETE|

| 2911|2013-07-25 00:00:...| 8| PROCESSING|

| 5657|2013-07-25 00:00:...| 9|PENDING\_PAYMENT|

| 5648|2013-07-25 00:00:...| 10|PENDING\_PAYMENT|

+-----------------+--------------------+--------+---------------+

Python3 -m pip install pyspark

# [Apache Spark 2.x - Data Frames and Pre-Defined Functions - Create Data Frames from Text Files](http://discuss.itversity.com/t/apache-spark-2-x-data-frames-and-pre-defined-functions-create-data-frames-from-text-files/19485)

Let us see how we can read text data from files into data frame. spark.read also have APIs for other types of file formats, but we will get into those details later.

* We can use spark.read.csv or spark.read.text to read text data.
* spark.read.csv can be used for comma separated data. Default field names will be in the form of \_c0, \_c1 etc
* spark.read.text can be used to read fixed length data where there is no delimiter. Default field name is value.
* We can define attribute names using toDF function
* In either of the case data will be represented as strings
* We can covert data types by using cast function – df.select(df.field.cast(IntegerType()))
* We will see all other functions soon, but let us perform the task of reading the data into data frame and represent it in their original format.

spark = SparkSession.builder.appName('Reading data from text files').master('yarn').getOrCreate()

orders = spark.read.format('csv').schema('order\_id int, order\_date string, order\_customer\_id int, order\_status string').load('/user/saurabhbhanwala/retail\_db/orders')

>>> orders.printSchema()

root

|-- order\_id: integer (nullable = true)

|-- order\_date: string (nullable = true)

|-- order\_customer\_id: integer (nullable = true)

|-- order\_status: string (nullable = true)

orders.show()

+--------+--------------------+-----------------+---------------+

|order\_id| order\_date|order\_customer\_id| order\_status|

+--------+--------------------+-----------------+---------------+

| 1|2013-07-25 00:00:...| 11599| CLOSED|

| 2|2013-07-25 00:00:...| 256|PENDING\_PAYMENT|

| 3|2013-07-25 00:00:...| 12111| COMPLETE|

| 4|2013-07-25 00:00:...| 8827| CLOSED|

| 5|2013-07-25 00:00:...| 11318| COMPLETE|

| 6|2013-07-25 00:00:...| 7130| COMPLETE|

| 7|2013-07-25 00:00:...| 4530| COMPLETE|

| 8|2013-07-25 00:00:...| 2911| PROCESSING|

| 9|2013-07-25 00:00:...| 5657|PENDING\_PAYMENT|

| 10|2013-07-25 00:00:...| 5648|PENDING\_PAYMENT|

| 11|2013-07-25 00:00:...| 918| PAYMENT\_REVIEW|

| 12|2013-07-25 00:00:...| 1837| CLOSED|

| 13|2013-07-25 00:00:...| 9149|PENDING\_PAYMENT|

| 14|2013-07-25 00:00:...| 9842| PROCESSING|

| 15|2013-07-25 00:00:...| 2568| COMPLETE|

| 16|2013-07-25 00:00:...| 7276|PENDING\_PAYMENT|

| 17|2013-07-25 00:00:...| 2667| COMPLETE|

| 18|2013-07-25 00:00:...| 1205| CLOSED|

| 19|2013-07-25 00:00:...| 9488|PENDING\_PAYMENT|

| 20|2013-07-25 00:00:...| 9198| PROCESSING|

+--------+--------------------+-----------------+---------------+

ordersItems = spark.read.format('csv').schema('order\_itemd\_id int, order\_item\_order\_id int, order\_item\_product\_id int, order\_item\_quantity int, order\_item\_subtotal float, order\_item\_product\_price float').load('/user/saurabhbhanwala/retail\_db/order\_items')

>>> orderItems.printSchema()

ordersItems.printSchema()

root

|-- order\_itemd\_id: integer (nullable = true)

|-- order\_item\_order\_id: integer (nullable = true)

|-- order\_item\_product\_id: integer (nullable = true)

|-- order\_item\_quantity: integer (nullable = true)

|-- order\_item\_subtotal: float (nullable = true)

|-- order\_item\_product\_price: float (nullable = true)

>>> ordersItems.show()

+--------------+-------------------+---------------------+-------------------+-------------------+------------------------+

|order\_itemd\_id|order\_item\_order\_id|order\_item\_product\_id|order\_item\_quantity|order\_item\_subtotal|order\_item\_product\_price|

+--------------+-------------------+---------------------+-------------------+-------------------+------------------------+

| 1| 1| 957| 1| 299.98| 299.98|

| 2| 2| 1073| 1| 199.99| 199.99|

| 3| 2| 502| 5| 250.0| 50.0|

| 4| 2| 403| 1| 129.99| 129.99|

| 5| 4| 897| 2| 49.98| 24.99|

| 6| 4| 365| 5| 299.95| 59.99|

| 7| 4| 502| 3| 150.0| 50.0|

| 8| 4| 1014| 4| 199.92| 49.98|

| 9| 5| 957| 1| 299.98| 299.98|

| 10| 5| 365| 5| 299.95| 59.99|

| 11| 5| 1014| 2| 99.96| 49.98|

| 12| 5| 957| 1| 299.98| 299.98|

| 13| 5| 403| 1| 129.99| 129.99|

| 14| 7| 1073| 1| 199.99| 199.99|

| 15| 7| 957| 1| 299.98| 299.98|

| 16| 7| 926| 5| 79.95| 15.99|

| 17| 8| 365| 3| 179.97| 59.99|

| 18| 8| 365| 5| 299.95| 59.99|

| 19| 8| 1014| 4| 199.92| 49.98|

| 20| 8| 502| 1| 50.0| 50.0|

+--------------+-------------------+---------------------+-------------------+-------------------+------------------------+

only showing top 20 rows

>>> type(ordersItems)

<class 'pyspark.sql.dataframe.DataFrame'>

>>> type(orders)

<class 'pyspark.sql.dataframe.DataFrame'>

>>> orders = spark.read.csv('/user/saurabhbhanwala/retail\_db/orders', sep=',', schema='order\_id int, order\_date string, order\_customer\_id int, order\_status string')

>>> type(orders)

<class 'pyspark.sql.dataframe.DataFrame'>

orderItems = spark.read.csv('/user/saurabhbhanwala/retail\_db/order\_items', sep=',', schema='order\_itemd\_id int, order\_item\_order\_id int, order\_item\_product\_id int, order\_item\_quantity int, order\_item\_subtotal float, order\_item\_product\_price float')

>>> type(orderItems)

<class 'pyspark.sql.dataframe.DataFrame'>

>>> orders= spark.read.csv('/user/saurabhbhanwala/retail\_db/orders', sep=',').toDF('order\_id', 'order\_date', 'order\_customer\_id', 'order\_status')

>>> orders.printSchema()

root

|-- order\_id: string (nullable = true)

|-- order\_date: string (nullable = true)

|-- order\_customer\_id: string (nullable = true)

|-- order\_status: string (nullable = true)

>>> orders.show()

+--------+--------------------+-----------------+---------------+

|order\_id| order\_date|order\_customer\_id| order\_status|

+--------+--------------------+-----------------+---------------+

| 1|2013-07-25 00:00:...| 11599| CLOSED|

| 2|2013-07-25 00:00:...| 256|PENDING\_PAYMENT|

| 3|2013-07-25 00:00:...| 12111| COMPLETE|

| 4|2013-07-25 00:00:...| 8827| CLOSED|

| 5|2013-07-25 00:00:...| 11318| COMPLETE|

| 6|2013-07-25 00:00:...| 7130| COMPLETE|

| 7|2013-07-25 00:00:...| 4530| COMPLETE|

| 8|2013-07-25 00:00:...| 2911| PROCESSING|

| 9|2013-07-25 00:00:...| 5657|PENDING\_PAYMENT|

| 10|2013-07-25 00:00:...| 5648|PENDING\_PAYMENT|

| 11|2013-07-25 00:00:...| 918| PAYMENT\_REVIEW|

| 12|2013-07-25 00:00:...| 1837| CLOSED|

| 13|2013-07-25 00:00:...| 9149|PENDING\_PAYMENT|

| 14|2013-07-25 00:00:...| 9842| PROCESSING|

| 15|2013-07-25 00:00:...| 2568| COMPLETE|

| 16|2013-07-25 00:00:...| 7276|PENDING\_PAYMENT|

| 17|2013-07-25 00:00:...| 2667| COMPLETE|

| 18|2013-07-25 00:00:...| 1205| CLOSED|

| 19|2013-07-25 00:00:...| 9488|PENDING\_PAYMENT|

| 20|2013-07-25 00:00:...| 9198| PROCESSING|

+--------+--------------------+-----------------+---------------+

from pyspark.sql.types import IntegerType

orders.select(orders.order\_id.cast("int"), orders.order\_date, orders.order\_customer\_id.cast(IntegerType()), orders.order\_status)

> orders.select(orders.order\_id.cast("int"), orders.order\_date, orders.order\_customer\_id.cast(IntegerType()), orders.order\_status)

DataFrame[order\_id: int, order\_date: string, order\_customer\_id: int, order\_status: string]

>>> orders.withColumn('order\_id', orders.order\_id.cast("int")).withColumn('order\_customer\_id', orders.order\_customer\_id.cast(IntegerType()))

DataFrame[order\_id: int, order\_date: string, order\_customer\_id: int, order\_status: string]

orders = spark.read.text('/user/saurabhbhanwala/retail\_db/orders')

>>> type(orders)

<class 'pyspark.sql.dataframe.DataFrame'>

>>>

orders.printSchema()

root

|-- value: string (nullable = true)

orders.show(truncate=False)

# [Apache Spark 2.x - Data Frames and Pre-Defined Functions - Create Data Frames from Hive Tables](http://discuss.itversity.com/t/apache-spark-2-x-data-frames-and-pre-defined-functions-create-data-frames-from-hive-tables/19486)

If Hive and Spark are integrated, we can create data frames from data in Hive tables or run Spark SQL queries against it.

* We can use spark.read.table to read data from Hive tables into Data Frame
* We can prefix database name to table name while reading Hive tables into Data Frame
* We can also run Hive queries directly using spark.sql
* Both spark.read.table and spark.sql returns Data Frame

>>> orders = spark.read.table('saurabhbhanwala\_retail\_db\_text.orders')

>>> orders.printSchema()

root

|-- order\_id: integer (nullable = true)

|-- order\_date: string (nullable = true)

|-- order\_customer\_id: integer (nullable = true)

|-- order\_status: string (nullable = true)

spark.sql('select \* from saurabhbhanwala\_retail\_db\_text.orders limit 10').show()

+--------+--------------------+-----------------+---------------+

|order\_id| order\_date|order\_customer\_id| order\_status|

+--------+--------------------+-----------------+---------------+

| 1|2013-07-25 00:00:...| 11599| CLOSED|

| 2|2013-07-25 00:00:...| 256|PENDING\_PAYMENT|

| 3|2013-07-25 00:00:...| 12111| COMPLETE|

| 4|2013-07-25 00:00:...| 8827| CLOSED|

| 5|2013-07-25 00:00:...| 11318| COMPLETE|

| 6|2013-07-25 00:00:...| 7130| COMPLETE|

| 7|2013-07-25 00:00:...| 4530| COMPLETE|

| 8|2013-07-25 00:00:...| 2911| PROCESSING|

| 9|2013-07-25 00:00:...| 5657|PENDING\_PAYMENT|

| 10|2013-07-25 00:00:...| 5648|PENDING\_PAYMENT|

+--------+--------------------+-----------------+---------------+

# [Apache Spark 2.x - Data Frames and Pre-Defined Functions - Create Data Frames using JDBC](http://discuss.itversity.com/t/apache-spark-2-x-data-frames-and-pre-defined-functions-create-data-frames-using-jdbc/19487)

pyspark2 --master yarn --conf spark.ui.port=12007 --jars /usr/share/java/mysql-connector-java.jar --driver-class-path /usr/share/java/mysql-connector-java.jar

**jdbc**(self, url, table, column=None, lowerBound=None, upperBound=None, numPartitions=None, predicates=None, properties=None) method of pyspark.sql.readwriter.DataFrameReader instance

Construct a :class:`DataFrame` representing the database table named ``table``

accessible via JDBC URL ``url`` and connection ``properties``.

Partitions of the table will be retrieved in parallel if either ``column`` or

``predicates`` is specified. ``lowerBound`, ``upperBound`` and ``numPartitions``

is needed when ``column`` is specified.

If both ``column`` and ``predicates`` are specified, ``column`` will be used.

.. note:: Don't create too many partitions in parallel on a large cluster; otherwise Spark might crash your external database systems.

:param url: a JDBC URL of the form ``jdbc:subprotocol:subname``

:param table: the name of the table

:param column: the name of an integer column that will be used for partitioning;

if this parameter is specified, then ``numPartitions``, ``lowerBound``

(inclusive), and ``upperBound`` (exclusive) will form partition strides

for generated WHERE clause expressions used to split the column

``column`` evenly

:param lowerBound: the minimum value of ``column`` used to decide partition stride

:param upperBound: the maximum value of ``column`` used to decide partition stride

:param numPartitions: the number of partitions

:param predicates: a list of expressions suitable for inclusion in WHERE clauses;

each one defines one partition of the :class:`DataFrame`

:param properties: a dictionary of JDBC database connection arguments. Normally at

least properties "user" and "password" with their corresponding values.

For example { 'user' : 'SYSTEM', 'password' : 'mypassword' }

:return: a DataFrame

orders = spark.read.format('jdbc').option('url', 'jdbc:mysql://ms.itversity.com').option('dbtable', 'retail\_db.orders').option('user', 'retail\_user').option('password', 'itversity').load()

type(orders)

<class 'pyspark.sql.dataframe.DataFrame'>

>>> type(orders)

<class 'pyspark.sql.dataframe.DataFrame'>

>>> orders.show()

+--------+-------------------+-----------------+---------------+

|order\_id| order\_date|order\_customer\_id| order\_status|

+--------+-------------------+-----------------+---------------+

| 1|2013-07-25 00:00:00| 11599| CLOSED|

| 2|2013-07-25 00:00:00| 256|PENDING\_PAYMENT|

| 3|2013-07-25 00:00:00| 12111| COMPLETE|

| 4|2013-07-25 00:00:00| 8827| CLOSED|

| 5|2013-07-25 00:00:00| 11318| COMPLETE|

| 6|2013-07-25 00:00:00| 7130| COMPLETE|

| 7|2013-07-25 00:00:00| 4530| COMPLETE|

| 8|2013-07-25 00:00:00| 2911| PROCESSING|

| 9|2013-07-25 00:00:00| 5657|PENDING\_PAYMENT|

| 10|2013-07-25 00:00:00| 5648|PENDING\_PAYMENT|

| 11|2013-07-25 00:00:00| 918| PAYMENT\_REVIEW|

| 12|2013-07-25 00:00:00| 1837| CLOSED|

| 13|2013-07-25 00:00:00| 9149|PENDING\_PAYMENT|

| 14|2013-07-25 00:00:00| 9842| PROCESSING|

| 15|2013-07-25 00:00:00| 2568| COMPLETE|

| 16|2013-07-25 00:00:00| 7276|PENDING\_PAYMENT|

| 17|2013-07-25 00:00:00| 2667| COMPLETE|

| 18|2013-07-25 00:00:00| 1205| CLOSED|

| 19|2013-07-25 00:00:00| 9488|PENDING\_PAYMENT|

| 20|2013-07-25 00:00:00| 9198| PROCESSING|

+--------+-------------------+-----------------+---------------+

only showing top 20 rows

>>> orders.printSchema()

root

|-- order\_id: integer (nullable = true)

|-- order\_date: timestamp (nullable = true)

|-- order\_customer\_id: integer (nullable = true)

|-- order\_status: string (nullable = true)

>>> orderItems = spark.read.jdbc('jdbc:mysql://ms.itversity.com:3306', 'retail\_db.order\_items', properties={ 'user': 'retail\_user', 'password': 'itversity'} )

>>> orderItems.printSchema()

root

|-- order\_item\_id: integer (nullable = true)

|-- order\_item\_order\_id: integer (nullable = true)

|-- order\_item\_product\_id: integer (nullable = true)

|-- order\_item\_quantity: integer (nullable = true)

|-- order\_item\_subtotal: double (nullable = true)

|-- order\_item\_product\_price: double (nullable = true)

>>> orderItems.show()

+-------------+-------------------+---------------------+-------------------+-------------------+------------------------+

|order\_item\_id|order\_item\_order\_id|order\_item\_product\_id|order\_item\_quantity|order\_item\_subtotal|order\_item\_product\_price|

+-------------+-------------------+---------------------+-------------------+-------------------+------------------------+

| 1| 1| 957| 1| 299.98| 299.98|

| 2| 2| 1073| 1| 199.99| 199.99|

| 3| 2| 502| 5| 250.0| 50.0|

| 4| 2| 403| 1| 129.99| 129.99|

| 5| 4| 897| 2| 49.98| 24.99|

| 6| 4| 365| 5| 299.95| 59.99|

| 7| 4| 502| 3| 150.0| 50.0|

| 8| 4| 1014| 4| 199.92| 49.98|

| 9| 5| 957| 1| 299.98| 299.98|

| 10| 5| 365| 5| 299.95| 59.99|

| 11| 5| 1014| 2| 99.96| 49.98|

| 12| 5| 957| 1| 299.98| 299.98|

| 13| 5| 403| 1| 129.99| 129.99|

| 14| 7| 1073| 1| 199.99| 199.99|

| 15| 7| 957| 1| 299.98| 299.98|

| 16| 7| 926| 5| 79.95| 15.99|

| 17| 8| 365| 3| 179.97| 59.99|

| 18| 8| 365| 5| 299.95| 59.99|

| 19| 8| 1014| 4| 199.92| 49.98|

| 20| 8| 502| 1| 50.0| 50.0|

+-------------+-------------------+---------------------+-------------------+-------------------+------------------------+

orderItems = spark.read.jdbc('jdbc:mysql://ms.itversity.com:3306', 'retail\_db.order\_items', properties={ 'user': 'retail\_user', 'password': 'itversity', 'numPartitions': '4', 'PartitionColumn': 'order\_item\_order\_id', 'lowerBound': '10000', 'upperBound': '20000'} )

>>> spark.write.csv('/user/saurabhbhanwala/orderItemscsvPartition')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'SparkSession' object has no attribute 'write'

>>> orderItems.write.csv('/user/saurabhbhanwala/orderItemscsvPartition')

[Stage 1:> (0 + 0) / 4]

>>>

>>>

>>> import os

>>> os.system('hadoop fs -ls /user/saurbahbhanwala/orderItemscsvPartition')

ls: `/user/saurbahbhanwala/orderItemscsvPartition': No such file or directory

256

>>> os.system('hadoop fs -ls /user/saurabhbhanwala/orderItemscsvPartition')

Found 5 items

-rw-r--r-- 2 saurabhbhanwala hdfs 0 2020-09-12 03:53 /user/saurabhbhanwala/orderItemscsvPartition/\_SUCCESS

-rw-r--r-- 2 saurabhbhanwala hdfs 936524 2020-09-12 03:53 /user/saurabhbhanwala/orderItemscsvPartition/part-00000-aac909ce-1286-4410-87fd-eee19a323a34-c000.csv

-rw-r--r-- 2 saurabhbhanwala hdfs 196316 2020-09-12 03:53 /user/saurabhbhanwala/orderItemscsvPartition/part-00001-aac909ce-1286-4410-87fd-eee19a323a34-c000.csv

-rw-r--r-- 2 saurabhbhanwala hdfs 192627 2020-09-12 03:53 /user/saurabhbhanwala/orderItemscsvPartition/part-00002-aac909ce-1286-4410-87fd-eee19a323a34-c000.csv

-rw-r--r-- 2 saurabhbhanwala hdfs 4083413 2020-09-12 03:53 /user/saurabhbhanwala/orderItemscsvPartition/part-00003-aac909ce-1286-4410-87fd-eee19a323a34-c000.csv

0

# [Apache Spark 2.x - Data Frames and Pre-Defined Functions - Data Frame Operations - Overview](http://discuss.itversity.com/t/apache-spark-2-x-data-frames-and-pre-defined-functions-data-frame-operations-overview/19489)

Let us see overview about Data Frame Operations. It is one of the 2 ways we can process Data Frames.

* Selection or Projection – select
* Filtering data – filter or where
* Joins – join (supports outer join as well)
* Aggregations – groupBy and agg with support of functions such as sum, avg, min, max etc
* Sorting – sort or orderBy
* Analytics Functions – aggregations, ranking and windowing functions

# [Apache Spark 2.x - Data Frames and Pre-Defined Functions - Spark SQL – Overview](http://discuss.itversity.com/t/apache-spark-2-x-data-frames-and-pre-defined-functions-spark-sql-overview/19490)

We can also use Spark SQL to process data in data frames.

* We can get list of tables by using spark.sql('show tables')
* We can register data frame as temporary view df.createTempView("view\_name")
* Output of show tables show the temporary tables as well
* Once temp view is created, we can use SQL style syntax and run queries against the tables/views
* Most of the hive queries will work out of the box

# [Apache Spark 2.x - Data Frames and Pre-Defined Functions - Functions to manipulate data - Overview of Functions to manipulate data in Data Frame fields or columns](http://discuss.itversity.com/t/apache-spark-2-x-data-frames-and-pre-defined-functions-functions-to-manipulate-data-overview-of-functions-to-manipulate-data-in-data-frame-fields-or-columns/19491)

Let us quickly look into some of the functions available in Data Frames.

* Main package for functions pyspark.sql.functions
* We can import by saying from pyspark.sql import functions as sf
* You will see many functions which are similar to the functions in traditional databases.
* These can be categorized into
  + String manipulation
  + Date manipulation
  + Type casting
  + Expressions such as case when
* We will see some of the functions in action
  + substring
  + lower, upper
  + trim
  + date\_format
  + trunc
  + Type Casting
  + case when