# Spark DSL

Spark DSL is Domain Specific Language

## Transformation:

### select(columns):

val df = spark.read

.option("inferSchema","true")

.format("csv")

.load("file:D/BigData/spark-essentials-master/src/main/resources/data/movies.csv")

*val df = spark.read.option("inferSchema","true").option("header","true").format("csv").load("file:///D:/BigData/spark-essentials-master/src/main/resources/data/movies.csv")*

df.select('actor, 'year).show(5)

df.select('actor, ('year - ('year %10)).as('decade)).show(5)

### selectExpr(columns):

Select \* ==> df.selectExpr("\*").show(5)

df.selectExpr("title as movie\_name").show(5)

df.selectExpr("title as movie\_name").show(5)

### filler(condition)

### where(condition)

It is used to filter out the rows that don’t meet the given condition

filter ==> returns only the rows that meet the specified condition

df.filter('year < 2000)

df.filter('year === 2000).show()

df.filter('year =!= 2000).show

### distinct, dropDuplicates

dropDuplicates allows you to control which columns should be used in deduplication logic.

If none is specified, the deduplication logic will use all the columns in the DataFrame

df.select("title").distinct.selectExpr("count(title) as movies").show

df.dropDuplicates("title").selectExpr("count(title) as movies").show

### withColumn(colName, column)

add a new column to a DataFrame. It requires two input parameters: a column name and a value in the form of a column expression

df.printSchema()

|-- actor: string (nullable = true)

|-- title: string (nullable = true)

|-- year: integer (nullable = true)

df.withColumn("decade", ('year - 'year % 10)).show(5)

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

| actor| title|year|decade|

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

|McClure, Marc (I)|Freaky Friday|2003| 2000|

|McClure, Marc (I)| Coach Carter|2005| 2000|

|McClure, Marc (I)| Superman II|1980| 1980|

|McClure, Marc (I)| Apollo 13|1995| 1990|

|McClure, Marc (I)| Superman|1978| 1970|

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

scala> val df1 = df.withColumn("decade", ('year - 'year % 10))

df1: org.apache.spark.sql.DataFrame = [actor: string, title: string ... 2 more fields]

scala> df.printSchema

root

|-- actor: string (nullable = true)

|-- title: string (nullable = true)

|-- year: integer (nullable = true)

scala> df1.printSchema

root

|-- actor: string (nullable = true)

|-- title: string (nullable = true)

|-- year: integer (nullable = true)

|-- decade: integer (nullable = true)

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

### withColumnRenamed(existingColName, newColName)

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

This transformation is strictly about renaming an existing column name in a DataFrame

To rename a cryptic column name to a more human-friendly nam

Before joining two DataFrames that happen to have one or more same column name

nOte:Notice that if the provided existingColName doesn’t exist in the schema, Spark doesn’t throw an error, and it will silently do nothing

df1.withColumnRenamed("actor","hero")

// renaming a column

val carsWithColumnRenamed = carsDF.withColumnRenamed("Weight\_in\_lbs", "Weight in pounds")

### drop(columnName1, columnName2)

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

drops the specified columns from the DataFrames

Dropping Two Columns: One Exists and the Other One Doesn’t

scala> df3.printSchema

root

|-- actor: string (nullable = true)

|-- title: string (nullable = true)

|-- year: integer (nullable = true)

scala> val df4 = df3.drop("actor","popular")

df4: org.apache.spark.sql.DataFrame = [title: string, year: int]

scala> df4.printSchema

root

|-- title: string (nullable = true)

|-- year: integer (nullable = true)

## Working with Missing Data or Bad Data

There are three common ways of dealing with missing or bad data

* Drop the rows that have missing values in a one or more columns
* Fill those missing values with user-provided values
* Replace the bad data with something that you know how to deal with

Dropping Rows with Missing Data:

Creating a dataframe with null values

scala> val badMovies = Seq(Row(null, null, null), Row(null, null, 2018L), Row("John Doe", "Awesome Movie", null), Row(null, "Awesome Movie", 2018L), Row("Mary Jane", null, 2018L))

badMovies: Seq[org.apache.spark.sql.Row] = List([null,null,null], [null,null,2018], [John Doe,Awesome Movie,null], [null,Awesome Movie,2018], [Mary Jane,null,2018])

Converting to RDD:

scala> val badMoviesRDD = spark.sparkContext.parallelize(badMovies)

Creating a Schema

scala> val movies = StructType(Array(StructField("actor",StringType,true),StructField("movie",StringType,true),StructField("year",LongType,true)))

Creating a DataFrame:

val badMoviesDF = spark.createDataFrame(badMoviesRDD, movies)

badMoviesDF.show

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

| actor| movie|year|

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

| null| null|null|

| null| null|2018|

| John Doe|Awesome Movie|null|

| null|Awesome Movie|2018|

|Mary Jane| null|2018|

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

// dropping rows that have missing data in any columnRemove:

// both of the lines below will achieve the same purpose

scala> badMoviesDF.na.drop().show

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

|actor|movie|year|

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

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

scala> badMoviesDF.na.drop("any").show

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

|actor|movie|year|

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

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

// drop rows that have missing data in every single column 🡪 Drop row only if all the columns are null

scala> badMoviesDF.na.drop("all").show

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

| actor| movie|year|

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

| null| null|2018|

| John Doe|Awesome Movie|null|

| null|Awesome Movie|2018|

|Mary Jane| null|2018|

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

//Check null values by specifying the column name

scala> badMoviesDF.na.drop(Array("actor")).show

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

| actor| movie|year|

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

| John Doe|Awesome Movie|null|

|Mary Jane| null|2018|

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