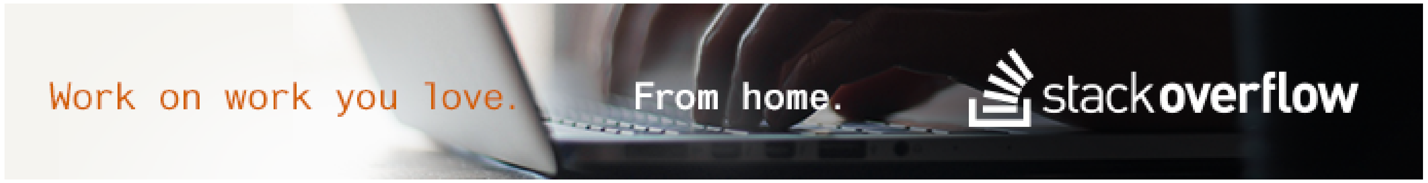
How to change column types in Spark SQL's DataFrame?



Suppose I'm doing something like:

```
val df = sqlContext.load("com.databricks.spark.csv", Map("path" -> "cars.csv", "header" ->
"true"))
df.printSchema()

root
 |-- year: string (nullable = true)
 |-- make: string (nullable = true)
 |-- model: string (nullable = true)
 |-- comment: string (nullable = true)
 |-- blank: string (nullable = true)

df.show()
year make  model comment              blank
2012 Tesla S     No comment
1997 Ford  E350  Go get one now th...
```

but I really wanted the `year` as `Int` (and perhaps transform some other columns).

The best I could come up with is

```
df.withColumn("year2", 'year.cast("Int")).select('year2 as 'year, 'make, 'model, 'comment,
'blank)
org.apache.spark.sql.DataFrame = [year: int, make: string, model: string, comment: string,
blank: string]
```

which is a bit convoluted.

I'm coming from R, and I'm used to being able to write, e.g.

```
df2 <- df %>%
   mutate(year = year %>% as.integer,
          make = make %>% toupper)
```

I'm likely missing something, since there should be a better way to do this in spark/scala...

scala apache-spark apache-spark-sql

edited Sep 12 '15 at 9:46
Jacek Laskowski
15.7k 4 51 118

asked Apr 1 '15 at 4:55
organic agave
837 1 5 18

8 Answers

[EDIT: March 2016: thanks for the votes! Though really, this is not the best answer, I think the solutions based on `withColumn` , `withColumnRenamed` and `cast` put forward by msemelman, Martin Senne and others are simpler and cleaner].

I think your approach is ok, recall that a Spark `DataFrame` is an (immutable) RDD of Rows, so we're never really *replacing* a column, just creating new `DataFrame` each time with a new schema.

Assuming you have an original df with the following schema:

```
scala> df.printSchema
root
 |-- Year: string (nullable = true)
 |-- Month: string (nullable = true)
 |-- DayofMonth: string (nullable = true)
 |-- DayOfWeek: string (nullable = true)
 |-- DepDelay: string (nullable = true)
 |-- Distance: string (nullable = true)
 |-- CRSDepTime: string (nullable = true)
```

And some UDF's defined on one or several columns:

```
import org.apache.spark.sql.functions._

val toInt    = udf[Int, String]( _.toInt)
val toDouble = udf[Double, String]( _.toDouble)
val toHour   = udf((t: String) => "%04d".format(t.toInt).take(2).toInt )
val days_since_nearest_holidays = udf(
  (year:String, month:String, dayOfMonth:String) => year.toInt + 27 + month.toInt-12
 )
```

Changing column types or even building a new DataFrame from another can be written like this:

```
val featureDf = df
.withColumn("departureDelay", toDouble(df("DepDelay")))
.withColumn("departureHour",  toHour(df("CRSDepTime")))
.withColumn("dayOfWeek",      toInt(df("DayOfWeek")))
.withColumn("dayOfMonth",     toInt(df("DayofMonth")))
.withColumn("month",          toInt(df("Month")))
.withColumn("distance",       toDouble(df("Distance")))
.withColumn("nearestHoliday", days_since_nearest_holidays(
              df("Year"), df("Month"), df("DayofMonth"))
            )
.select("departureDelay", "departureHour", "dayOfWeek", "dayOfMonth",
        "month", "distance", "nearestHoliday")
```

which yields:

```
scala> df.printSchema
root
 |-- departureDelay: double (nullable = true)
 |-- departureHour: integer (nullable = true)
 |-- dayOfWeek: integer (nullable = true)
 |-- dayOfMonth: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- distance: double (nullable = true)
 |-- nearestHoliday: integer (nullable = true)
```

This is pretty close to your own solution. Simply, keeping the type changes and other transformations as separate `udf val` s make the code more readable and re-usable.

edited Mar 30 at 15:10

answered Apr 10 '15 at 9:39
Svend
1,952 6 19

This is neither safe nor efficient. **Not safe** because a single `NULL` or malformed entry will crash a whole job. **Not efficient** because UDFs are not transparent to Catalyst. Using UDFs for complex operations is just fine, but there is no reason to use these for basic type casting. This why we have `cast` method (see an answer by Martin Senne). Making things transparent to Catalyst requires more work but basic safety is just a matter of putting `Try` and `Option` to work. – zero323 Mar 1 at 0:39

I didn't see anything related to converting string to date for example "05-APR-2015" – dbspace Apr 29 at 21:33

Is there a way to reduce your `withColumn()` section to a generic one that iterates through all columns? – Boern May 17 at 14:55

Since version 1.4 you can do this to replace the column:

```
val df2 = df.withColumn("yearTmp", df.year.cast(IntegerType))
    .drop("year")
    .withColumnRenamed("yearTmp", "year")
```

If you are using sql expressions you can also do:

```
val df2 = df.selectExpr("cast(year as int) year",
                        "make",
                        "model",
                        "comment",
                        "blank")
```

For more info check the docs:
http://spark.apache.org/docs/latest/api/scala/#org.apache.spark.sql.DataFrame

edited Apr 28 at 20:23

answered Oct 29 '15 at 20:27
msemelman
1,042 9 19

As the `cast` operation is available for Spark `Column` s (and as I personally do not favour `udf` s as proposed by Svend at this point), how about

```
df.select( df("year").cast(IntegerType).as("year"), ... )
```

to cast to the requested type? As a neat side effect, values not castable / "convertable" in that sense, will become `null` .

In case you need this as **a helper method**, use

```
object DFHelper
  def castColumnTo( df: DataFrame, cn: String, tpe: DataType ) : DataFrame = {
    df.withColumn( cn, df(cn).cast(tpe) )
  }
}
```

which is used like

```
import DFHelper._
val df2 = castColumnTo( df, "year", IntegerType )
```

edited Sep 17 '15 at 16:10

answered Sep 17 '15 at 15:54
Martin Senne
1,872 1 10 27

Can you advice me on how to proceed, if I need to cast and rename a whole bunch of columns (I have 50 columns, and fairly new to scala, not sure what is the best way to approach it without creating a massive duplication)? Some columns should stay String, some should be cast to Float. – Dmitry Smirnov Apr 14 at 13:03

how to convert a String to a Date for example "25-APR-2016" in the column and "20160302" – dbspace Apr 29 at 21:30

To convert the year from string to int, you can add the following option to the csv reader: "inferSchema" -> "true", see DataBricks documentation

answered Aug 16 '15 at 2:49

Peter Rose
64 1 2

2 This works nicely but the catch is that the reader must do a second pass of your file – beefyhalo Sep 1 '15 at 17:46

the Scala code most similar to R that I can achieve :

```
val df2 = df.select(
   df.columns.map {
     case year @ "year" => df(year).cast(IntegerType).as(year)
     case make @ "make" => functions.upper(df(make)).as(make)
     case other         => df(other)
   }: _*
)
```

Though the length is a little longer than R's. Note that the `mutate` is a function for R data frame , so Scala is very good enough in expressive power given without using a special function .

(`df.columns` is surprisingly a Array[String] instead of Array[Column], maybe they want it look like Python pandas's dataframe.)

edited Aug 21 '15 at 23:44

answered Aug 21 '15 at 13:12

WeiChing Lin
1,027 9 22

1 Could you please give the equivalent for pyspark? – iota Oct 3 '15 at 18:06

You can use `selectExpr` to make it a little cleaner:

```
df.selectExpr("cast(year as int) as year", "upper(make) as make",
    "model", "comment", "blank")
```

answered Aug 14 '15 at 19:24

dnlbrky
2,308 17 33

So this only really works if your having issues saving to a jdbc driver like sqlserver, but it's really helpful for errors you will run into with syntax and types.

```
import org.apache.spark.sql.jdbc.{JdbcDialects, JdbcType, JdbcDialect}
import org.apache.spark.sql.jdbc.JdbcType
val SQLServerDialect = new JdbcDialect {
  override def canHandle(url: String): Boolean = url.startsWith("jdbc:jtds:sqlserver") ||
url.contains("sqlserver")

  override def getJDBCType(dt: DataType): Option[JdbcType] = dt match {
    case StringType => Some(JdbcType("VARCHAR(5000)", java.sql.Types.VARCHAR))
    case BooleanType => Some(JdbcType("BIT(1)", java.sql.Types.BIT))
    case IntegerType => Some(JdbcType("INTEGER", java.sql.Types.INTEGER))
    case LongType => Some(JdbcType("BIGINT", java.sql.Types.BIGINT))
    case DoubleType => Some(JdbcType("DOUBLE PRECISION", java.sql.Types.DOUBLE))
    case FloatType => Some(JdbcType("REAL", java.sql.Types.REAL))
    case ShortType => Some(JdbcType("INTEGER", java.sql.Types.INTEGER))
    case ByteType => Some(JdbcType("INTEGER", java.sql.Types.INTEGER))
    case BinaryType => Some(JdbcType("BINARY", java.sql.Types.BINARY))
    case TimestampType => Some(JdbcType("DATE", java.sql.Types.DATE))
    case DateType => Some(JdbcType("DATE", java.sql.Types.DATE))
    //      case DecimalType.Fixed(precision, scale) => Some(JdbcType("NUMBER(" +
precision + "," + scale + ")", java.sql.Types.NUMERIC))
    case t: DecimalType => Some(JdbcType(s"DECIMAL(${t.precision},${t.scale})",
java.sql.Types.DECIMAL))
    case _ => throw new IllegalArgumentException(s"Don't know how to save ${dt.json} to
JDBC")
  }
}

JdbcDialects.registerDialect(SQLServerDialect)
```

answered Apr 19 at 7:32

ben jarman
218 2 5

Java code for modifying the datatype of the DataFrame from String to Integer

```
df.withColumn("col_name", df.col("col_name").cast(DataTypes.IntegerType))
```

It will simply cast the existing(String datatype) to Integer.

edited Jun 2 at 21:28

answered May 19 at 22:42

manishbelsare
1 1

1 There's no `DataTypes` in `sql.types` ! it's `DataType` . Moreover, one can simply import `IntegerType` and cast. – Ehsan M. Kermani Jul 13 at 17:31