Java-Success.com

Prepare to fast-track, choose & go places with 800+ Java & Big Data Q&As with lots of code & diagrams.

10: Spark on Zeppelin – union, udf and explode

Posted on September 13, 2018

Pre-requisite: Docker is installed on your machine for Mac OS X (E.g. $ brew cask install docker) or Windows 10. Docker interview Q&As. This extends setting up Apache Zeppelin Notebook.

Step 1: Pull this from the docker hub, and build the image with the following command.

```
$ docker pull apache/zeppelin:0.7.3

```

300+ Java Interview FAQs

300+ Java FAQs 🔥

16+ Java Key Areas Q&As

150+ Java Architect FAQs

80+ Java Code Quality Q&As

150+ Java Coding Q&As

300+ Big Data Interview FAQs

300+ Big Data FAQs 🔥

Tutorials - Big Data

TUT - ⏯ Starting Big Data

TUT - Starting Spark & Scala

You can verify the image with the “docker images” command.

Step 2: Run the container with the above image.

```
$ docker run --rm -it -p 8080:8080 apache/zeppelin

```

Step 3: Open Zeppelin notebook via a web browser “http:localhost:8080”. Create a note book with “spark” as a default interpreter.

The following example adds new Rows with bonus (i.e. 10% of the salary) if the salary is <= 50K.

Using filter, withColumn and unionAll

filter and withColumn are used to create a new Dataframe with bonuses. Then it is combined with the employees Dataframe using the unionAll function.

```
%spark


import org.apache.spark.sql.types._

val schema = StructType(
  List(
    StructField("id", IntegerType, true),
    StructField("name", StringType, true),
    StructField("location", StringType, true),
    StructField("salary", DoubleType, true)
  )
)

val employees = Seq(
    Row(1, "John", "USA", 50000.0),
    Row(2, "Peter", "AU",60000.0),
    Row(3, "Sam", "AU", 60000.0),
    Row(4, "Susan", "USA", 50000.0),
    Row(5, "David", "USA", 70000.0),
    Row(6, "Elliot", "AU", 50000.0)
```

```
)

val employeeDf = spark.createDataFrame(
  spark.sparkContext.parallelize(employees),
  schema
)

val bonusDf = employeeDf.filter(x => x.getDouble(3
                        .withColumn("salary", $"sa


//union both Dataframes
val salaryWithBonusDf = employeeDf.unionAll(bonusD

salaryWithBonusDf.show()

```

Output:

```
import org.apache.spark.sql.types._
schema: org.apache.spark.sql.types.StructType = St
employees: Seq[org.apache.spark.sql.Row] = List([1
employeeDf: org.apache.spark.sql.DataFrame = [id: 
bonusDf: org.apache.spark.sql.DataFrame = [id: int
warning: there was one deprecation warning; re-run
salaryWithBonusDf: org.apache.spark.sql.Dataset[or
+---+------+--------+-------+
| id|  name|location| salary|
+---+------+--------+-------+
|  1|  John|     USA|50000.0|
|  2| Peter|      AU|60000.0|
|  3|   Sam|      AU|60000.0|
|  4| Susan|     USA|50000.0|
|  5| David|     USA|70000.0|
|  6|Elliot|      AU|50000.0|
|  1|  John|     USA| 5000.0|
|  4| Susan|     USA| 5000.0|
|  6|Elliot|      AU| 5000.0|
+---+------+--------+-------+

```

Using udf & explode functions

udf means "User Defined Function".

explode function creates **a new row for each element** in the given array or map column (in a DataFrame).

```
%spark


import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._  //for ud

val schema = StructType(
  List(
    StructField("id", IntegerType, true),
    StructField("name", StringType, true),
    StructField("location", StringType, true),
    StructField("salary", DoubleType, true)
  )
)

val employees = Seq(
    Row(1, "John", "USA", 50000.0),
    Row(2, "Peter", "AU",60000.0),
    Row(3, "Sam", "AU", 60000.0),
    Row(4, "Susan", "USA", 50000.0),
    Row(5, "David", "USA", 70000.0),
    Row(6, "Elliot", "AU", 50000.0)
)

val employeeDf = spark.createDataFrame(
  spark.sparkContext.parallelize(employees),
  schema
)

val calcBonus: (Double) => Seq[Double] =  {  (sal
    if (salary <= 50000) {
        Seq(salary) ++ Seq(salary * 0.10)
    } else {
        Seq(salary)
    }
}

val bonusUdf = udf(calcBonus)

val salaryWithBonusDf = employeeDf.withColumn("sa

salaryWithBonusDf.show()

```

Output:

```
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
schema: org.apache.spark.sql.types.StructType = S
employees: Seq[org.apache.spark.sql.Row] = List([
employeeDf: org.apache.spark.sql.DataFrame = [id:
calcBonus: Double => Seq[Double] = <function1>
bonusUdf: org.apache.spark.sql.expressions.UserDe
salaryWithBonusDf: org.apache.spark.sql.DataFrame
+---+------+--------+-------+
| id|  name|location| salary|
+---+------+--------+-------+
|  1|  John|     USA|50000.0|
|  1|  John|     USA| 5000.0|
|  2| Peter|      AU|60000.0|
|  3|   Sam|      AU|60000.0|
|  4| Susan|     USA|50000.0|
|  4| Susan|     USA| 5000.0|
|  5| David|     USA|70000.0|
|  6|Elliot|      AU|50000.0|
|  6|Elliot|      AU| 5000.0|
+---+------+--------+-------+

```

‹ 09: Spark on Zeppelin – convert DataFrames to RDD and RDD to DataFrame

11: Spark on Zeppelin – Dataframe groupBy, collect_list, explode & window ›