

# Java-Success.com

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

[Home](#) [Why? ▾](#) [300+ Java FAQs ▾](#) [300+ Big Data FAQs ▾](#) [Courses ▾](#)

[👤 Membership ▾](#) [Your Career ▾](#)

[Home](#) › [bigdata-success.com](#) › [Tutorials - Big Data](#) › [TUT - Spark Scala on Zeppelin](#) ›

10: Spark on Zeppelin – union, udf and explode

## 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.

```
1 $ docker pull apache/zeppelin:0.7.3
2
```

### 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.

```
1 $ docker run --rm -it -p 8080:8080 apache/zeppelin
2
```

**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.

```
1 %spark
2
3
4 import org.apache.spark.sql.types._
5
6 val schema = StructType(
7   List(
8     StructField("id", IntegerType, true),
9     StructField("name", StringType, true),
10    StructField("location", StringType, true),
11    StructField("salary", DoubleType, true)
12  )
13 )
14
15 val employees = Seq(
16   Row(1, "John", "USA", 50000.0),
17   Row(2, "Peter", "AU", 60000.0),
18   Row(3, "Sam", "AU", 60000.0),
19   Row(4, "Susan", "USA", 50000.0),
20   Row(5, "David", "USA", 70000.0),
21   Row(6, "Elliot", "AU", 50000.0)
```

TUT - Starting with Python

TUT - Kafka

TUT - Pig

TUT - Apache Storm

TUT - Spark Scala on Zeppelin

TUT - Cloudera

TUT - Cloudera on Docker

TUT - File Formats

TUT - Spark on Docker

TUT - Flume

TUT - Hadoop (HDFS)

TUT - HBase (NoSQL)

TUT - Hive (SQL)

TUT - Hadoop & Spark

TUT - MapReduce

TUT - Spark and Scala

TUT - Spark & Java

TUT - PySpark on Databricks

TUT - Zookeeper

## 800+ Java Interview Q&As

300+ Core Java Q&As



300+ Enterprise Java Q&As



150+ Java Frameworks Q&As



120+ Companion Tech Q&As



Tutorials - Enterprise Java



```

22 )
23
24 val employeeDf = spark.createDataFrame(
25     spark.sparkContext.parallelize(employees),
26     schema
27 )
28
29 val bonusDf = employeeDf.filter(x => x.getDouble(3) > 0)
30     .withColumn("salary", $"salary" * 10)
31
32
33 //union both Dataframes
34 val salaryWithBonusDf = employeeDf.unionAll(bonusDf)
35
36 salaryWithBonusDf.show()
37

```

## Output:

```

1 import org.apache.spark.sql.types._
2 schema: org.apache.spark.sql.types.StructType = StructType(StructField(id,IntegerType,true),StructField(name,StringType,true),StructField(location,StringType,true),StructField(salary,DoubleType,true))
3 employees: Seq[org.apache.spark.sql.Row] = List([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])
4 employeeDf: org.apache.spark.sql.DataFrame = [id: int, name: string, location: string, salary: double]
5 bonusDf: org.apache.spark.sql.DataFrame = [id: int, name: string, location: string, salary: double]
6 warning: there was one deprecation warning; re-run with -deprecation for details
7 salaryWithBonusDf: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [id: int, name: string, location: string, salary: double]
8 +-----+
9 | id | name | location | salary |
10 +-----+
11 | 1 | John | USA | 50000.0 |
12 | 2 | Peter | AU | 60000.0 |
13 | 3 | Sam | AU | 60000.0 |
14 | 4 | Susan | USA | 50000.0 |
15 | 5 | David | USA | 70000.0 |
16 | 6 | Elliot | AU | 50000.0 |
17 | 1 | John | USA | 5000.0 |
18 | 4 | Susan | USA | 5000.0 |
19 | 6 | Elliot | AU | 5000.0 |
20 +-----+
21

```

## 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).

```
1 %spark
2
3
4 import org.apache.spark.sql.types._
5 import org.apache.spark.sql.functions._ //for udf
6
7 val schema = StructType(
8   List(
9     StructField("id", IntegerType, true),
10    StructField("name", StringType, true),
11    StructField("location", StringType, true),
12    StructField("salary", DoubleType, true)
13  )
14 )
15
16 val employees = Seq(
17   Row(1, "John", "USA", 50000.0),
18   Row(2, "Peter", "AU", 60000.0),
19   Row(3, "Sam", "AU", 60000.0),
20   Row(4, "Susan", "USA", 50000.0),
21   Row(5, "David", "USA", 70000.0),
22   Row(6, "Elliot", "AU", 50000.0)
23 )
24
25 val employeeDf = spark.createDataFrame(
26   spark.sparkContext.parallelize(employees),
27   schema
28 )
29
30 val calcBonus: (Double) => Seq[Double] = { (salary) => {
31   if (salary <= 50000) {
32     Seq(salary) ++ Seq(salary * 0.10)
33   } else {
34     Seq(salary)
35   }
36 }
37
38 val bonusUdf = udf(calcBonus)
39
40 val salaryWithBonusDf = employeeDf.withColumn("salaryWithBonus", bonusUdf($"salary"))
41
42 salaryWithBonusDf.show()
43
```

## Output:

```
1 import org.apache.spark.sql.types._
2 import org.apache.spark.sql.functions._
3 schema: org.apache.spark.sql.types.StructType = StructType(StructField(id,IntegerType,true),StructField(name,StringType,true),StructField(location,StringType,true),StructField(salary,DoubleType,true))
4 employees: Seq[org.apache.spark.sql.Row] = List([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])
```

```

5 employeeDf: org.apache.spark.sql.DataFrame = [id:
6 calcBonus: Double => Seq[Double] = <function1>
7 bonusUdf: org.apache.spark.sql.expressions.UserDe
8 salaryWithBonusDf: org.apache.spark.sql.DataFrame
9 +---+-----+-----+-----+
10 | id|  name|location| salary|
11 +---+-----+-----+-----+
12 |  1|  John|    USA|50000.0|
13 |  1|  John|    USA| 5000.0|
14 |  2| Peter|    AU|60000.0|
15 |  3|  Sam|    AU|60000.0|
16 |  4| Susan|    USA|50000.0|
17 |  4| Susan|    USA| 5000.0|
18 |  5| David|    USA|70000.0|
19 |  6|Elliot|    AU|50000.0|
20 |  6|Elliot|    AU| 5000.0|
21 +---+-----+-----+-----+
22

```

◀ 09: Spark on Zeppelin – convert DataFrames to RDD and RDD to

DataFrame

11: Spark on Zeppelin – Dataframe groupBy, collect\_list, explode &  
window ▶

## Disclaimer

The contents in this Java-Success are copyrighted and from EmpoweringTech pty ltd. The EmpoweringTech pty ltd has the right to correct or enhance the current content without any prior notice. These are general advice only, and one needs to take his/her own circumstances into consideration. The EmpoweringTech pty ltd will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. Links to external sites do not imply endorsement of the linked-to sites. [Privacy Policy](#)