

# **Inclusion Dependency Rules**

## **Homework II**

**Team Dally 04-06-21**

# Setup

- IntelliJ
- Execute build.sbt
- Include Scala-Plugin through IDEA
- Download Spark and start:

```
/usr/local/Cellar/apache-spark/3.1.2/libexec/sbin  
./start-all.sh
```

# Submit

➔ **spark-tutorial git:(master)** X spark-submit \  
 --class de.hpi.spark\_tutorial.SimpleSpark \  
 --master "local[8]" \  
 /path/SparkTutorialSBT-assembly-0.1.jar \  
 100

Or

➔ **spark-tutorial git:(master)** X spark-submit \  
 --class de.hpi.spark\_tutorial.SimpleSpark \  
 --master spark://Alisons-MacBook-Pro.local:7077\  
 /path/SparkTutorialSBT-assembly-0.1.jar \  
 100

# Submit IDEA Setup

- add `case PathList("META-INF","services",xs @ _*) => MergeStrategy.filterDistinctLines` to `build.sbt`
- *Final hint:* <https://stackoverflow.com/questions/67054414/running-fat-jar-and-getting-path-not-found-exception-in-sbt>

```
java -jar SparkTutorialSBT-assembly-0.1.jar --path ~/TPCH --cores 4
```

# Pipeline

## 1 - Read in Tables

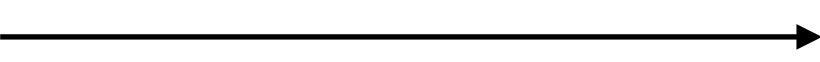
```
import spark.implicits._
```

importing encoders

```
val tableSetList = inputs.map(f => spark.read.
```

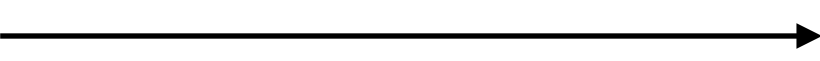
reading files

```
  .option("inferSchema","true")
```



Keeping the schema of the cols

```
  .option("header", "true")
```

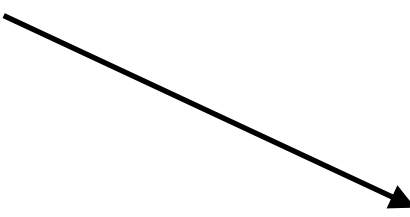


Output DF col names as header record and  
delimiter by passed value

```
  .option("delimiter", ";")
```

```
  .csv(f)
```

```
  .flatMap(row => row.schema.fields.zipWithIndex.map(tuple => (row.get(tuple._2).toString,tuple._1.name))))
```



1. Flattening (remove inner grouping) of the tableSetList

2. zipWithIndex to avoid creating auxiliary list for indexes  
(See: <https://www.baeldung.com/scala/iteration-index-value>)

_1	_2
Val...	Key...

```
tabletSetList :List[Dataset[String,String]]
```

# Pipeline

## 2 - Aggregate and GroupBy

```
val unionTables = tableSetList.reduce { (table1, table2) => table1.union(table2) }  
    .dropDuplicates
```

.show()

_1	_2
Supplier#000000000...	S_NAME
7627.85.	S_ACCTBAL
Supplier#000000001...	S_NAME
Id requests across...	S_COMMENT

1. Reduce List in order to union tables 2.
2. dropDuplicates (otherwise GC-Error caused)

```
val groupedValues = unionTables.groupByKey(_._1) —————> Extract values for further work
```

# Pipeline

## 3 - Inclusion

Create Dataset of keys

```
val keysToSet = groupedValues.mapGroups { case (_, rows) => rows.map(_._2).toSet }
    .dropDuplicates

+-----+ .show()
|      value      |
+-----+
|[O_CUSTKEY, C_CUS....|
|[P_PARTKEY, O_CUS....|
|[P_PARTKEY, O_ORD....|
```

Create the inclusion of the keys

```
val inclusion = keysToSet.flatMap(set => set.map(key => (key, set - key)))

+-----+-----+ .show()
|      _1      |      _2      |
+-----+-----+
| O_CUSTKEY|[C_CUSTKEY, L_PAR...|
| C_CUSTKEY|[O_CUSTKEY, L_PAR...|
| L_PARTKEY|[O_CUSTKEY, C_CUS...|
```

# Pipeline

## 4 - Intersection

Create the intersection

1. Reduce and intersect elements, adapt format
2. Filter Empty Values of String of Sets
3. Sort col for output

```
val intersect = inclusion.groupByKey(_. _1)
    .reduceGroups((a, b) => (a._1, a._2.intersect(b._2))).map {case (a,(_ ,b)) => (a,b) }
    .filter(_. _2.nonEmpty)
    .sort("_1")
```



# Pipeline

## 5 - Output data

```
intersect  
.collect()  
.foreach { case (dependentKey, referencedKey) => println(dependentKey + " < " +  
referencedKey.toList.sorted.reduce(_ + ", " + _)) }
```

—> OUT.txt