

Inclusion Dependency Discovery in Spark

**Tobias Altenbrand
Robin Schumacher**

Transformation Pipelines

Iterate over all inputTables
Iterate over the rows, for every row iterate over the columns

```
inputs.map(input => readData(input, spark))
  .map(inputTable => {
    val columns = inputTable.columns
    inputTable.flatMap(row => {
      for (i <- columns.indices) yield {
        (columns(i), row.getString(i))
      }
    })
  })
```

map

N_NA	0
N_NAME	ALGERIA
N_RE	0
N_NA	1
N_NAME	ARGENTINA
N_RE	1
N_NA	2
N_NAME	BRAZIL
N_RE	1

Union tables, group by value
create a set with all values

```
.reduce((firstDataset, secondDataSet) => firstDataset union secondDataSet)
.groupByKey(tuple => tuple._2)
.mapGroups { case (key, iter) =>
  val listBuffer1 = new ListBuffer[String]
  for (i <- iter) {
    listBuffer1 += i._1
  }
  (listBuffer1.toSet)
}
```

mapGroups

{N_NA, N_RE}
{N_NAME}
{N_RE}
{N_NA, N_RE}
{N_NAME}
{N_NA}
{N_NAME}

Transformation Pipelines

Create combinations
Group / intersect by key

```
.flatMap(attributeSet => attributeSet
  .map(currentAttribute =>
    (currentAttribute, attributeSet.filter(attribute => attribute != currentAttribute))))
.groupByKey(row => row._1)
.mapGroups((key, iter) =>
  (key, iter.map(row => row._2).reduce((firstSet, secondSet) => firstSet.intersect(secondSet))))
```

map / flatMap

N_NA	{N_RE}
N_NAME	{}
N_RE	{}
N_NA	{N_RE}
N_NAME	{}
N_NA	{}
N_NAME	{}

N_NA	{N_RE}
N_NAME	{}
N_RE	{}
N_NA	{}

Transformation Pipelines

Filter empty entries
Sort and print

```
.filter(row => row._2.nonEmpty)
.collect()
.sortBy(ind => ind._1)
.foreach(ind => println(ind._1 + " < " + ind._2.mkString(", ")))
```

mapGroups

N_NA	{N_RE}
N_NAME	{}
N_RE	{}
N_NA	{}

filter

N_NA	{N_RE}
------	--------

Transformation Pipelines

