CSC 369: Distributed Computing
Spring 2020
Lab 8-2
Due: Wednesday, June 9, end of the day

This is an individual assignment.

Dataset

For this assignment you are working with the CSC 365 KAZTENJAMMER dataset. The dataset follows the recording and performing career of an all-female Norwegian band **Katzenjammer**. The four members of the band, ***Sloveig, Marianne, Anne-Marit*** and ***Turid*** turn their live shows into a cascade of instrument changes. The dataset documents the albums the band released, the songs on the albums, and the typical ways in which the songs are performed live: the instruments each band member plays, the vocals, and their position on the stage. Examples of the band's performances can be found in abundance on Youtube.

The dataset consists of seven CSV files:

File Name	Columns	Explanation
`Band.csv`	Id,Firstname,Lastname	List of members of the band
`Albums.csv`	AId,Title,Year,Label,Type	List of albums the band released
`Songs.csv`	SongId, Title	List of all songs recorded/performed live
`Instruments.csv`	SongId,BandMate,Instrument	Instruments played by each bandmate in live performances of each song
`Performance.csv`	SongId,BandMate,StagePosition	Stage position occupied by each bandmate in live performances of each song
`Vocals.csv`	SongId,BandMate,Type	Type of vocals (lead, shared, chorus) contributed by the bandmates to each live performance of each song.
`Tracklists.csv`	AlbumId,Position,SongId	List of songs on each album

Notes. The README file for the dataset with a fuller explanation is provided on the static course web page. All "connections" (foreign keys) between tables are straightforward: e.g., **`SongId`** columns from **`Performance.csv`**, **`Vocals.csv`**, and **`Instruments.csv`** files take values from the **`SongId`** column in the **`Songs.csv`** file.

Setting Up Data Frames.

I am sharing some starter code with you that takes care of turning the seven CSV files in the dataset into PySpark Data Frames. The code is available in the **`lab08.py`** file found on the Lab 8 resources page on the static web site. The setup portion of the code looks as follows:

```
# KATZENJAMMER dataset files

# filenames
filenames = {"albums":"/data/katzenjammer/Albums.csv",
             "band": "/data/katzenjammer/Band.csv",
             "instruments": "/data/katzenjammer/Instruments.csv",
             "performance": "/data/katzenjammer/Performance.csv",
             "songs": "/data/katzenjammer/Songs.csv",
             "tracklists": "/data/katzenjammer/Tracklists.csv",
             "vocals": "/data/katzenjammer/Vocals.csv"}

# attribute lists for each data filke
attributeLists = {"albums": [tp.StructField("AId",tp.IntegerType()),
                             tp.StructField("Title", tp.StringType()),
                             tp.StructField("Year", tp.IntegerType()),
                             tp.StructField("Label",tp.StringType()),
                             tp.StructField("Type", tp.StringType())],
                  "band": [tp.StructField("Id", tp.IntegerType()),
                           tp.StructField("Firstname", tp.StringType()),
                           tp.StructField("Lastname", tp.StringType())],
                  "instruments": [tp.StructField("SongId", tp.IntegerType()),
                                  tp.StructField("BandmateId", tp.IntegerType()),
                                  tp.StructField("Instrument", tp.StringType())],
                   "performance": [tp.StructField("SongId", tp.IntegerType()),
                                   tp.StructField("BandMate", tp.IntegerType()),
                                   tp.StructField("StagePosition", tp.StringType())],
                    "songs": [tp.StructField("SongId", tp.IntegerType()),
                              tp.StructField("Title", tp.StringType())],
                     "tracklists": [tp.StructField("AlbumId", tp.IntegerType()),
                                    tp.StructField("Position", tp.IntegerType()),
                                    tp.StructField("SongId", tp.IntegerType())],
                     "vocals": [tp.StructField("SongId", tp.IntegerType()),
                                tp.StructField("Bandmate", tp.IntegerType()),
                                tp.StructField("Type", tp.StringType())]
                      }
# schemas
schemas = {key: tp.StructType(alist) for key,alist in zip(attributeLists,
                attributeLists.values())}
```

The data frame definitions are shown below (please note, the code is formatted for readability, Python will not properly parse it - use the code provided in **`lab08.py`** file instead for running)

```
albumsDF = spark.read.format("csv").schema(schemas["albums"])
                     .option("header",True).option("quote", "'")
                     .load(filenames["albums"])

bandDF = spark.read.format("csv").schema(schemas["band"])
                                 .option("header",True).option("quote","'")
                                  .load(filenames["band"])

instrumentsDF = spark.read.format("csv").schema(schemas["instruments"])
                          .option("header",True).option("quote","'")
                          .load(filenames["instruments"])

performanceDF = spark.read.format("csv").schema(schemas["performance"])
                          .option("header",True).option("quote","'")
                          .load(filenames["performance"])

songsDF = spark.read.format("csv").schema(schemas["songs"])
                    .option("header",True).option("quote","'")
                    .load(filenames["songs"])

tracklistsDF = spark.read.format("csv").schema(schemas["tracklists"])
                         .option("header",True).option("quote",".")
                         .load(filenames["tracklists"])

vocalsDF = spark.read.format("csv").schema(schemas["vocals"])
                     .option("header",True).option("quote","'")
                     .load(filenames["vocals"])
```

This code should yield correctly parsed data frames for each of the seven data files. While our assignment includes active use of a subset of the data frames, it is recommended that you keep the code in its entirety, in case you need information from any of the data frames. Here is how the `instrumentsDF` data frame looks like:

```
>>> instrumentsDF.show(10)
+------+----------+--------------+
|SongId|BandmateId|    Instrument|
+------+----------+--------------+
|     1|         1|       trumpet|
|     1|         2|      keyboard|
|     1|         3|     accordion|
|     1|         4|bass balalaika|
|     2|         1|       trumpet|
|     2|         2|         drums|
|     2|         3|        guitar|
|     2|         4|bass balalaika|
|     3|         1|         drums|
|     3|         1|       ukalele|
+------+----------+--------------+
only showing top 10 rows
```

Assignment

Part 1

The KATZENJAMMER dataset is what we call *normalized:* the data is broken into separate collections (tables, in case of relational databases), and each collection/data file contains only information about a single aspect of the dataset. Such normalization reduces duplication and makes the dataset more maintainable in case new information needs to be added, but provides for poor visualization of information. For example, this is how the **`tracklistsDF`** data frame looks:

```
>>> tracklistsDF.show(10)
+-------+--------+------+
|AlbumId|Position|SongId|
+-------+--------+------+
|      1|       1|     1|
|      1|       2|     2|
|      1|       3|     3|
|      1|       4|     4|
|      1|       5|     5|
|      1|       6|     6|
|      1|       7|     7|
|      1|       8|     8|
|      1|       9|     9|
|      1|      10|    10|
+-------+--------+------+
only showing top 10 rows
```

We want to concentrate our work with the KATZENJAMMER dataset on creating data frames that provide immediate visual insight into our data. Specifically, we are interested in creating a data frame that has one row for each song, and lists what instruments each member of Katzenjammer plays on this song, and who sings lead vocals. This task would be fairly straightforward, if it wasn't for the fact that some performers play multiple instruments on certain songs, and some performers share lead vocals on some songs.

As such, we will accomplish our ultimate task in a set of steps.

Step 1. Our first goal is to condense the **`instrumentsDF`** data frame, so that each performer-song combination shows in exactly one row, and if a performer plays multiple instruments on a song, they are presented in a list. (In fact, we will create a list of instruments for each performer-song combination, some lists will consist of just one entry though). As we are doing this, we also want to replace SongID and Band member Id in this data frame with the title of the song and the ***first name*** for the performer (Solveig, Marianne, Anne-Marit, or Turid).

Write a series of pySpark statements that produce a data frame that consists of the following columns:

- **Song:** the title of the song
- **Name:** the first name of one of the performers
- **Instruments:** the list of instrument the performer plays on the song.

The output should look like this (notice multiple instruments Solveig plays on "Demon Kitty Rag"; also, this list includes just the first 20 records from the data frame):

```
+--------------------+----------+----------------+
|                Song|      Name|     Instruments|
+--------------------+----------+----------------+
|            Overture|   Solveig|       [trumpet]|
|            Overture|  Marianne|      [keyboard]|
|            Overture|Anne-Marit|     [accordion]|
|            Overture|     Turid|[bass balalaika]|
|  A Bar In Amsterdam|   Solveig|       [trumpet]|
|  A Bar In Amsterdam|  Marianne|         [drums]|
|  A Bar In Amsterdam|Anne-Marit|        [guitar]|
|  A Bar In Amsterdam|     Turid|[bass balalaika]|
|     Demon Kitty Rag|   Solveig|[drums, ukalele]|
|     Demon Kitty Rag|  Marianne|         [banjo]|
|     Demon Kitty Rag|Anne-Marit|[bass balalaika]|
|     Demon Kitty Rag|     Turid|     [keyboards]|
|   Tea With Cinnamon|   Solveig|         [drums]|
|   Tea With Cinnamon|  Marianne|       [ukalele]|
|   Tea With Cinnamon|Anne-Marit|     [accordion]|
|   Tea With Cinnamon|     Turid|[bass balalaika]|
|'Hey Ho on the De...|   Solveig|         [drums]|
|'Hey Ho on the De...|  Marianne|     [keyboards]|
|'Hey Ho on the De...|Anne-Marit|        [guitar]|
|'Hey Ho on the De...|     Turid|[bass balalaika]|
+--------------------+----------+----------------+
```

Hint: In the **`instrumentDF`** data frame, each row contains a single instrument for a performer in a given song. In the output, these instruments, whenever there are multiple ones, need to be combined in a single row. The operation that achieves effects like that is ***grouping.*** However, if you carefully look at the Data Frame API, you will notice that it has very SQL-like grouping and aggregation operations, that allow for computing of aggregate values like sum, average or count of elements, but **do not** easily lend itself to list construction. This is where you can revert to RDDs. Think for the following set of steps:

- use the RDD component of the data frame
- transform the RDD, if needed, from a collection of **`pyspark.sql.Row`** elements, to a collection of tuples/lists.
- Perform appropriate grouping/aggregation operations, that combine instruments for each song-performer combination into an list.
- Convert the RDD you obtained back to a data frame.

To get the last step correctly, you need to define the schema of the resulting data frame. Use the column names from the illustration above.

Step 2. Now, let us refactor the data frame you obtained at the end of **Step 1.** We want to produce a new data frame that in each row list the title of a song, and the instruments each performer is playing - one column per person, with columns named after that person. The output should look as follows:

```
+--------------------+--------------------+--------------------+--------------+--------------------+
|                Song|             Solveig|            Marianne|     AnneMarit|               Turid|
+--------------------+--------------------+--------------------+--------------+--------------------+
|    Curvaceous Needs|             [drums]|            [guitar]|       [banjo]|    [bass balalaika]|
|          Der Kapitan|    [trumpet, drums]|        [keyboards]|   [accordion]|    [bass balalaika]|
|            Oh My God|           [ukalele]|             [drums]|   [accordion]|              [bass]|
|          Let it Snow|         [washboard]|   [ukalele, kazoo]|       [banjo]|    [bass balalaika]|
|    Soviet Trumpeter|[trumpet, keyboards]|        [keyboards]|       [banjo]|[xylophone, mando...|
|   Tea With Cinnamon|             [drums]|          [ukalele]|   [accordion]|    [bass balalaika]|
|                Ouch|             [drums]|           [guitar]|       [banjo]|    [bass balalaika]|
|           Cherry Pie|         [washboard]|          [ukalele]|       [banjo]|    [bass balalaika]|
|  A Bar In Amsterdam|           [trumpet]|             [drums]|      [guitar]|    [bass balalaika]|
|          My Own Tune|             [drums]|[guitar, toy piano]|     [ukalele]|[bass balalaika, ...|
|              My Dear|             [drums]|              [bass]|      [guitar]|            [guitar]|
|         Lady Marlene|         [keyboards]|             [banjo]|   [keyboards]|          [xylophone]|
|             Rockland|            [guitar]|              [bass]|[small guitar]|          [accordion]|
|         Old De Spain|           [ukalele]|            [guitar]|       [banjo]|    [bass balalaika]|
|            Lady Gray|           [ukalele]|            [guitar]|    [mandolin]|              [bass]|
|Cocktails and Rub...|         [keyboards]|             [drums]|      [guitar]|    [bass balalaika]|
|             Overture|           [trumpet]|          [keyboard]|   [accordion]|    [bass balalaika]|
|'Hey Ho on the De...|             [drums]|        [keyboards]|      [guitar]|    [bass balalaika]|
|Listening to the ...|             [drums]|            [guitar]|       [banjo]|    [bass balalaika]|
|     Mother Superior|    [bass balalaika]|        [accordion]|   [keyboards]|           [mandolin]|
+--------------------+--------------------+--------------------+--------------+--------------------+
only showing top 20 rows
```

Hint. One straightforward way to solve this problem is to create four data frames featuring only the instrument play of each of the four performers. Here is, for example, what Solveig's data frame would look like:

```
>>> solveigDF.show(10)
+--------------------+-------+--------------------+-------------+
|                Song|   Name|             Solveig|SolveigVocals|
+--------------------+-------+--------------------+-------------+
|             Overture|Solveig|           [trumpet]|           No|
|  A Bar In Amsterdam|Solveig|           [trumpet]|         lead|
|      Demon Kitty Rag|Solveig|    [drums, ukalele]|           No|
|   Tea With Cinnamon|Solveig|             [drums]|       chorus|
|'Hey Ho on the De...|Solveig|             [drums]|       chorus|
|    Wading in Deeper|Solveig|         [xylophone]|         lead|
|               Le Pop|Solveig|             [drums]|           No|
|          Der Kapitan|Solveig|    [trumpet, drums]|       chorus|
|      Virginia Clemm|Solveig|         [xylophone]|         lead|
|Play My Darling, ...|Solveig|[guitar, harmonica]|       chorus|
+--------------------+-------+--------------------+-------------+
only showing top 10 rows
```

Once you create these data frames, assembling the data frame above is a matter of multiple careful join operations.

Note on syntax. When joining two data frames, the result of the join operation often may contain multiple columns with the same name. PySpark is not very graceful in handling these, and in followup operations (joins, filters and so on) the most intuitive ways of referring to these columns might yield syntax errors. There are multiple ways to manage this. One way is to proactively use **`withColumnRenamed()`** to disambiguate all column names in the resulting data frame. Here is a short illustration using two toy dataframes.

```
>>> testSchema01 = tp.StructType([tp.StructField("Id",tp.IntegerType()),
tp.StructField("Name", tp.StringType())])
>>> testSchema02 =  tp.StructType([tp.StructField("Id",tp.IntegerType()),
tp.StructField("Place", tp.StringType())])
>>>
>>> df1 = spark.createDataFrame([[1,"bob"],[2, "diana"], [3, "kyle"], [4, "mary"]],
testSchema01)
>>> df2 = spark.createDataFrame([[1,"chicago"], [2, "new york"], [3, "new york"], [4, "san
jose"]], testSchema02)

>>>
>>> dj = df1.join(df2, df1.Id == df2.Id, "inner")
>>> dj.show()

dj.filter(dj.Id == 1)

df1 = df1.withColumnRenamed("Id","Person")

dj1 = df1.join(df2, df1.Person == df2.Id, "inner")

dj1.filter(dj1.Id==1).show()+---+-----+---+--------+
| Id| Name| Id|   Place|
+---+-----+---+--------+
|  1|  bob|  1| chicago|
|  3| kyle|  3|new york|
|  4| mary|  4|san jose|
|  2|diana|  2|new york|
+---+-----+---+--------+

>>>
>>> dj.filter(dj.Id == 1)
Traceback (most recent call last):
  File "/usr/hdp/3.1.0.0-78/spark2/python/pyspark/sql/utils.py", line 63, in deco
    return f(*a, **kw)
 …. … … <<PySpark/Yarn Error Messages>>

pyspark.sql.utils.AnalysisException: "Reference 'Id' is ambiguous, could be: Id, Id.;"
>>>
>>> df1 = df1.withColumnRenamed("Id","Person")
>>>
>>> dj1 = df1.join(df2, df1.Person == df2.Id, "inner")
>>>
>>> dj1.filter(dj1.Id==1).show()
+------+----+---+-------+
|Person|Name| Id|  Place|
+------+----+---+-------+
|     1| bob|  1|chicago|
+------+----+---+-------+
```

The second solution leverages the fact that in PySpark data frames columns retain their provenance. in the original data frame **`dj1`** in the example above, the two Id columns internally are distinguished by the names of data frames they came from. So, as counter-intuitive it is for those of us who are used to scoping rules in SQL to write this, the following actually works:

```
>>> dj = df1.join(df2, df1.Id == df2.Id, "inner")
>>> dj.show()
+---+-----+---+--------+
| Id| Name| Id|   Place|
+---+-----+---+--------+
|  1|  bob|  1| chicago|
|  3| kyle|  3|new york|
|  4| mary|  4|san jose|
|  2|diana|  2|new york|
+---+-----+---+--------+

>>> dj.filter(df1.Id==1).show()
+---+----+---+-------+
| Id|Name| Id|  Place|
+---+----+---+-------+
|  1| bob|  1|chicago|
+---+----+---+-------+
```

Same thing applies when the operation you are trying to perform is join.

Step 3. At the end of **Step 2** you should have a data frame formatted as follows:

```
+--------------------+--------------------+--------------------+--------------+--------------------+
|                Song|             Solveig|           Marianne|     AnneMarit|               Turid|
+--------------------+--------------------+--------------------+--------------+--------------------+
```

Here, the contents of each of the columns named **`Solveig, Marianne, AnneMarit`** and **`Turid`** are lists of instruments.

Now, let us add one last piece of information and create our final data frame. This time, we want to add the lead vocalist to the data frame.

To do this, first and foremost, you can filter the **`vocalsDF`** data frame to keep only lead vocals information. Next, you need to recognize the fact that a single song can have multiple lead vocalists. We can address this the same way we addressed on person playing multiple instruments in **Step 2.** (I won't say more here, but do remember to use an appropriate schema when converting back to data frames).

You should get something like this, at this stage (notice multiple lead vocalists on several songs):

```
+--------------------+-----------------+
|                Song|       LeadVocals|
+--------------------+-----------------+
|  A Bar In Amsterdam|        [Solveig]|
|     Demon Kitty Rag|       [Marianne]|
|   Tea With Cinnamon|       [Marianne]|
|'Hey Ho on the De...|       [Marianne]|
|    Wading in Deeper| [Turid, Solveig]|
|              Le Pop|[Turid, Marianne]|
|      Virginia Clemm|        [Solveig]|
|Play My Darling, ...|     [Anne-Marit]|
|          To the Sea|     [Anne-Marit]|
|     Mother Superior|          [Turid]|
+--------------------+-----------------+
only showing top 10 rows
```

Your final step is to combine this data frame with the data frame you obtained on **Step 2.**

The final comment is this: not all songs have a lead vocalist, but we DO WANT your final data frame to contain entrees for ALL songs. In addition, you should replace ***null*** values (if you wind up having them) with something nicer, e.g., **`"[No one]"`** (notice, this still must be a list). **Note:** this may require going a couple of steps back and changing things earlier…

Step 4. Our last task is separate from the **Step 1 -- Step 3** sequence, but I want you all to observe the convenience of contingency tables.

We are interested in looking at the incidences of one musician playing one specific instrument on a song, and another one - playing a different instrument.

To keep our output under control, we concentrate on Solveig and Marianne (at the end you should be able to notice a certain pattern), and we restrict the instruments to the following set:

- drums
- keyboards
- bass balalaika
- strings

As "strings" we count the following instruments: "guitar", "ukalele", "banjo", "mandolin", and "small guitar"

Write a sequence of PySpark statements that creates a data frame that contains the contingency table for Solveig and Marianne playing the four instrument types above. Your contingency table should look as follows:

```
+----------------+--------------+-----+---------+-------+
|Solveig_Marianne|bass balalaika|drums|keyboards|strings|
+----------------+--------------+-----+---------+-------+
|  bass balalaika|             0|    0|        1|      0|
|           drums|             1|    0|        7|      9|
|         strings|             3|    3|        0|      4|
|       keyboards|             0|    1|        1|      1|
+----------------+--------------+-----+---------+-------+
```

(it is possible that your table will be transposed - both variants are considered correct).

Hint. You can approach this problem by creating one data frame for Solveig's instruments and one - for Marianne's. You can then combine the two data frames in the same way you combined data in **Step 2**. Your data frame is essentially similar to that from **Step 2** with main difference being that your rows have atomic values in them, not lists (and, of course, you only need data for Solveig and Marianne).

Finishing touches.

Put all your code in **`lab08-2.py`** file.

In the file, make sure your name appears in the header comment. Additionally, make sure that the code executing each task is clearly labelled. You can keep all code (except for functions that may be necessary for RDD operations) in a single "main" sequence - but again - PLEASE label each chunk of code.

Each step should yield one data frame that we call final for that step. Figure out the size of each final data frame (you can just run **`.count()`** in your code, or off-line). Then use the

```
df.show(<size>)
```

statement at the end of each step where `df` should be replaced by the name of your final data frame and **`<size>`** is the size of that data frame.

Your code shall run without error when processed using **`spark-submit.`**

Submission

Handin from unixN.csc.calpoly.edu:

`$ handin dekhtyar lab08-2 <file>`