I am going to be creating a temporary table named “Fake\_data\_temp” so I can practice with the Spark SQL API and use SQL Syntax. The idea behind this was to use a couple of things from last homework and to also practice some SQL syntax. It is important to mentioned that many a few syntax/notes were taken from “Applied Database Technologies” class with Professor Olga Scrivner.

Query\_1 shows the implementation of spark.sql and how do we select and count all records in Birth\_Country, after that we assign the alias Total\_Country to it. Then we proceed to group be Birth\_Country and order it by Total\_Country in descending order (Limit 1 is used to select the first row).

This code is straight forward, we select Birth\_Country and Income from the DataFrame and assigned an alias to the MAX(Income) names Maximum\_Income. We group by Birth\_Country and order by Maximum\_Income in descending order. We store the new DataFrame in a variable called maximum\_income\_df and show results from highest to lowest.

Query\_3 Follows kind of the same approach as query\_1, but in much simpler way. We select the count of the records satisfied by the “where” clause and assign it to alias People. The “and” clause is the expand the task and find the Income greater than 100000 and Loan\_Approved equals to True to get the result. Just like query\_1, first() function is implemented to get the first value of the column People.

Ticks on the name (incluing Last\_Name), Income, and Job from the temporary table Fake\_data\_temp to get the top 10 people with highest income in the USA. To achieve this we need to use the “where” clause to set the Birth\_Country to ‘United States of America’ and order by income in descending order. Limit to 10 because we just want the top 10 people.

How many numbers of distinct jobs are there? Again, `query5' follows the basic SQL Syntax to calculate the number of distinct jobs out there. We select and count the distinct Job and assaying an alias to it named "Distnct\_Job". Then we call the first value of this select column to get the result.

This code it’s a more elaborated than the others, however it still uses the basics of SQL Syntax to achieve results. `query\_7` code calculates the total count of persons in each country who satisfied the filtering results. We assign these results to the column Highest\_Person\_Count using alias. We also use the WHERE, OR, AND to filter results. Finally, we group by Birth\_Country and we order in descending order the Highest\_Person\_Count. We limit to 5 since we only want the top 5 countries results.

Continues the basic SQL Syntax usage, we select SSN and calculate the total count of persons who contain the same SSN. We group by SSN and use HAVING COUNT( \* ) > 1) to filter the results than contain SSN duplicates.

It is important to understand that fault refers to failure, thus fault tolerance in Apache Spark is the capability to operate and to recover loss after a failure occurs. In Apache Spark terms, fault tolerance is achieved through the combination of lineage data and data replication.

Spark operates on data in fault-tolerance file systems like S3 or HDFS. This means that the data can be stored in a fault tolerant way (all the RDDs generated from fault-tolerant data is fault tolerant). The basic fault tolerant semantic of Apache Spark are:

* Since each Spark RDD is immutable, it remembers the lineage of deterministic operation that was used on fault-tolerant input dataset to create it.
* Spark can also use the lineage to recover and compute the loss (lost data) in case a worker node fails and RDD is lost.
* The data in the final transformed RDD will always be the same irrespective of failures in the Spark cluster (assuming that all the RDD transformations are deterministic).

To achieve fault tolerance for all the generated RDDs, the achieved data is replicated among multiple Spark executors in worder nodes in the cluster. This means that if something happens due to failure, the replicated data in clusters can be used to cover the loss.