# Homework 3

## Part 1

Purpose of HW is to learn how to work with external hive tables, how to handle late data, how to deal with duplicates. Use code from Homework 2.

1. Instead of using managed hive table, transition to using external hive table (be wary of all the key differences between managed and external tables when implementing other parts of this task)
2. Modify data generator so it can generate past and current dates for ‘SubmittedAt’ field. Add new integer input argument which defines how far back in past ‘SubmittedAt’ field could go. E.g. if you generate data on December 20 and a pass 3 as new input argument, ‘SubmittedAt’ field could have next values:
   * 2020-12-20 HH-mm-ss
   * 2020-12-19 HH-mm-ss
   * 2020-12-18 HH-mm-ss
   * 2020-12-17 HH-mm-ss

Alternatively, you can pass range\list of dates which data generator could use for ‘SubmittedAt’ field.

1. Modify data generator so it can create duplicated entries, meaning:
   1. Two or more entries(rows) have the same ‘Id’ and all other columns, **excluding column ‘**SubmittedAt’
   2. ‘SubmittedAt’ is different between two or more duplicated rows, but only the first one should be taken and inserted into the final table.
   3. ‘First’ row by ‘SubmittedAt’ is decided by ranking all duplicates and sorting by value of ‘SubmittedAt’ in ascending order and taking only the first result of the sort. Example: if we have three duplicated entries with “SubmittedAt” as **2020-10-11 12:11:35**, **2020-10-11 12:11:36** and **2020-10-11 10:10:00,** ONLY the row with value **2020-10-11 10:10:00** should be inserted into the final table, other 2 - completely disregarded.
   4. Duplicated rows should **not** be found in different files
2. Modify data generator so it can create files of types ORC, AVRO, PARQUET. Pass this option as configuration property. This step can be achieved in 2 ways:
   1. Re-writing generator app to Spark one and using converters provided by Spark (most common approach in real world situations)
   2. Writing types converters by yourselves, using whatever external library. (Cumbersome and will take much more time than the approach “a”. Choose this option only if you want the extra challenge and are sure that you can handle it in adequate timeframe. It is advised to do some research before proceeding with implementing this option).
3. Modify Spark program (the one that does the ETL thing) in a way so it can read those new file formats made by generator.
4. Basic level: Spark job should be able to add data to previous partitions (when data generator spawn data for already existing partitions by generating ‘SubmittedAt’ for past days).   
   Advanced level: you need to find a way to “merge” old data in Hive and new data that arrived. As result you should have old data in that partitions plus new portion of data from new files **BUT**, to avoid small files which may cause performance issues, you should not append to old partition (we simulate situation when main portion of data arrived on time and another small portions could arrive later).

As result, after your Spark job finishes processing data you should see new records in current and previous partitions.

## Part 2

Implement unit test for transformation(Spark SQL and Dataset/Dataframe API) that you have if your job. For dataframe/dataset comparison you can search library on github, e.g. <https://github.com/holdenk/spark-testing-base>. There is no standard way\library for it right now.