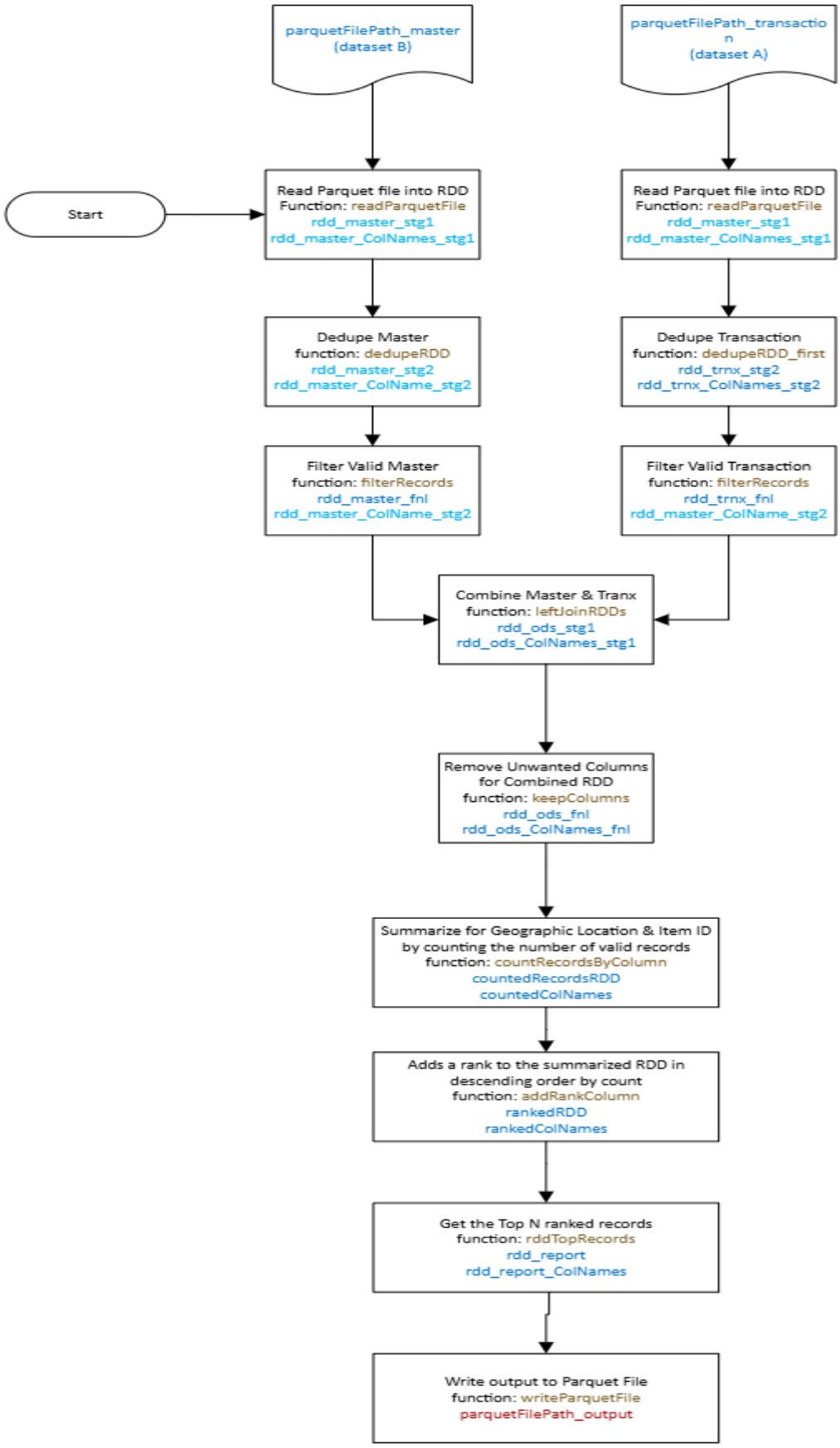
Data Flow Chart



Design considerations

1. The data flow was designed modular by function calls. Each function are parameterized to enable other type of datasets to be used no only pertaining to the given dataset A & dataset B.
2. The deduplication placed a valid flag where:
    a. N depict that the records are duplicated but the records are not similar
    b. X depict that the records are duplicated but the records are identical
    c. Y depict that record is valid for computation and reporting
3. Depending on the type of records, the first occurrence or last occurrence will be used as the valid record. Transaction might take the first or last dependent on the duplicated scenario. Master takes on the latest to ensure any master updates will be reflected.
4. The valid flag X & N can be used for investigation purposes if required on duplication scenarios.
5. The creation of a combined clean ODS is to allows other type of analysis if required
6. The ranking provides a descending or ascending allows the analysis of either the Top X or Bottom X records

The design is to be design such that it can be reuse as much as possible with minimal code change.

As this is my first time doing Scala programming and given time constraints, I would have put a configuration file to input all the parameters for each steps such that the same program can be reused with the change of the configuration file.

Note: For the output, we assume that the `geograhical_location` should varchar rather than bigint and should be referencing to the Dataset B `geograhical_location`

The parameter required to run the program will be as follows:

Usage: ParquetReader <parquetFilePath_master> <parquetFilePath_transaction> <parquetFilePath_output> <TopNRec>

Where parquetFilePath_master is DataSet B
parquetFilePath_transaction is DataSet A
parquetFilePath_output is where the output parquet directory
TopNRec is the number of Top N records to be output.

Testing Strategy

The typical testing strategy includes

Test Type	Description	How	Goal
Unit Test	To ensure each independent component developed return the expected result. This test every single function, every single output independently.	Test each function. Ensure the return result from the function is corrected, accurate and expected. Test to ensure each function can be executed successfully within expected performance.	Make sure every single module/functions/piece of codes works before stitching them into bigger piece which makes debugging more complicated.
System Integration Test	To ensure all component stitched together can be executed successfully and is according to the technical design.	Execute the test end-to-end according to the design. Ensure that the whole cycle can be executed successfully, result is correct, and the performance is within expectation.	Make sure the system is ready and in accordance with the design. System integration will mitigate any technical issues that will be raised by users subsequently during Users Acceptance Test and hence boosting users' confidence.
Users Acceptance Test	To ensure that the system is developed in accordance with the users' requirements. Use cases that align with users requirements and expectation.	Let users use their own dataset to test the system. Ensure the result return is according to the users' expectation	Make sure the system is accepted by users. The system meets the users requirements and expectation.

A typical template for documenting the test cases as follows:

Scenario	Test Case	Tested by	Test Date	Test Evidence	Status	Remarks

The following are some example of test scenarios and test case for unit test where it will validate and test individual scenarios for each smaller unit of the program developed.

Scenario	Test Case	Test Date	Evidence	Status
Ensure Master file (Dataset 2) is read in correctly from parquet file	Check Data Text are read in correctly		1,Aether Bay 2,Aura Bluff 3,Elysian Springs 4,Harmonious Haven 5,Harmony Meadows 6,Paradise Bay 7,Celestial Reach 8,Serenity Springs 9,Tranquil City 10,Valhalla City	Pass
	Check total records are correct		10000	Pass
Ensure Transaction file (Dataset 1) is read in correctly from parquet file	Check total records are correct		1000000	Pass
	Checksum other sumable columns (geographical_oid)			
	Checksum other sumable columns (video_camera_oid)			
	Checksum other sumable columns (detection_oid)			
Ensure dedupe for Master RDD is working correctly	Check total records are correct			
	Validate deduped RDD is the same as read in RDD from master file			
	Ensure valid_flag "X" is correctly populated correctly. Ingest a few records with the scenario and show that it is correctly flag			
	Ensure valid_flag "Y" is correctly populated correctly. Ingest a few records with the scenario and show that it is correctly flag			
	Ensure valid_flag "N" is correctly populated correctly. Ingest a few records with the scenario and show that it is correctly flag			

Integration Test is performed upon completion of the development. This is to test that the whole module can be executed successfully and according to the expected result. The following are some integration test scenario and case example:

Scenario	Test Case	Test Date	Evidence	Status
Program can be executed successfully	Ensure there are no errors during compilation			
	Ensure that error handling where no parameter is passed in			
System Parameter Check	Ensure that the output is correct according to the parameter given			
	Ensure that the number records in the output file is according to the number of Top N in the parameter given			
	Ensure that the data is correctly read in from the source file provided in the parameter			
	Ensure that the output file will be overwritten if exists and system will not have error			
Logic & Output	Check that the output file has the correct output according to the design specification			
	Check that the Top N is correctly ranked in the output file			
	Check that any temporary files are cleaned up and cache released			