Flink Project

Yellow Taxi and Limousine Commission Trips

Description: New York Taxi and Limousine Commission stores all trips done by yellow and green taxis. This data is reported by each taxi and is sent to a data center that processes this information for different purposes.

Each taxi reports all the information regarding each trip with the following information: *VendorID*, *tpep_pickup_datetime*, *tpep_dropoff_datetime, passenger_count, trip_distance, RatecodeID, store_and_fwd_flag, PULocationID, DOLocationID, payment_type, fare_amount, extra, mta_tax, tip_amount, tolls_amount, improvement_surcharge, total_amount, congestion_surcharge, airport_fee*.

VendorID: A code indicating the TPEP provider that provided the record. 1 = Creative Mobile Technologies, LLC; 2 = VeriFone Inc…

tpep_pickup_datetime: The date and time when the meter was engaged.

tpep_dropoff_datetime: The date and time when the meter was disengaged.

passenger_count: The number of passengers in the vehicle. This is a driver-entered value.

trip_distance: The elapsed trip distance in miles reported by the taximeter.

RatecodeID: The final rate code in effect at the end of the trip. 1 = Standard rate; 2 = JFK; 3 = Newark; 4 = Nassau or Westchester; 5 = Negotiated fare; 6 = Group ride.

store_and_fwd_flag: This flag indicates whether the trip record was held in the vehicle memory before sending it to the vendor, aka “store and forward”, because the vehicle did not have connection to the server. Y= store and forward trip; N= not a store and forward trip.

PULocationID: TLC Taxi Zone in which the taximeter was engaged.

DOLocationID: TLC Taxi Zone in which the taximeter was disengaged.

payment_type: A numeric code signifying how the passenger paid for the trip. 1 = Credit card; 2 = Cash; 3 = No charge; 4 = Dispute; 5 = Unknown; 6 = Voided trip.

fare_amount: The time-and-distance fare calculated by the meter.

Extra: Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
mta_tax: $0.50 MTA tax that is automatically triggered based on the metered rate in use.

tip_amount: Tip amount – _This field is automatically populated for credit card tips. Cash tips are not included.

tolls_amount: Total amount of all tolls paid in trip.

improvement_surcharge: $0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.

total_amount: The total amount charged to passengers. It does not include cash tips.

congestion_surcharge: The surcharge applied when the trip goes through a congested area.

airport_fee: The fee applied to the trip when it starts or ends at an airport.

The goal of this project is to **develop two Java programs using *Flink* for implementing the following functionality:**

1. **Congested Area:** The commission wants to control the number of taxis accessing the most congested areas of the city, more concretely the commission is interested in knowing for each day, the number of trips accessing these areas and the average cost of the trips that go through these areas. **(Points:6)**

   Each line in the output has this format:
   Date, NumberOfTrips, CostAvg.

   This is an example of the expected output (pay attention to the output format):

   ```
   2022/03/01,1441,22.55
   2022/03/01,102996,18.98
   2022/03/02,108095,18.97
   2022/03/03,117646,18.97
   2022/03/04,119130,19.09
   2022/03/05,109554,18.42
   2022/03/06,76960,20.05
   ```

   Notes:
   1. **The program name must be *CongestedArea***.
   2. If the field congestion_surcharge has no value, replace it by 0.0.

2. **Saturated Vendor:** The commission wants to register when a *vendorID* is saturated. A vendor is saturated when a trip starts in less than 10 minutes after the previous trip finished.

If there are two or more consecutive trips from a vendor (a trip starts in less than 10 minutes after the previous trip finished), a tuple is generated with this information : *vendorID*, start of the first trip (*tpep_pickup_datetime*), finish time of the last trip, total number of consecutive trips. **(Points:4)**

An example of the expected output is (pay attention to the output format) :

```
1,2022-03-01 12:00:00,2022-03-01 12:09:02,2
1,2022-03-01 12:00:03,2022-03-01 12:18:11,2
1,2022-03-01 12:00:05,2022-03-01 12:15:02,2
2,2022-03-01 12:00:04,2022-03-01 12:02:47,2
1,2022-03-01 12:00:06,2022-03-01 12:10:58,2
2,2022-03-01 12:00:07,2022-03-01 12:07:59,2
2,2022-03-01 12:00:09,2022-03-01 12:29:19,2
1,2022-03-01 12:00:08,2022-03-01 12:02:13,2
2,2022-03-01 12:00:13,2022-03-01 12:14:45,2
1,2022-03-01 12:00:14,2022-03-01 12:15:13,2
```

Notes:

1. ***The program name must be SaturatedVendor***.
2. If the field trip_distance has no value, replace it by 0.0.

The Java programs will read the events from a CSV file with the information the taxis report.

```
1,2022-03-01 00:00:00,2022-03-01 00:28:47,1.0,16.0,1.0,N,132,255,1,45.5,1.75,0.5,9.6,0.0,0.3,57.65,0.0,1.25
1,2022-03-01 00:00:03,2022-03-01 00:09:02,2.0,3.0,1.0,N,263,170,1,10.5,3.0,0.5,0.0,0.0,0.3,14.3,2.5,0.0
2,2022-03-01 00:00:04,2022-03-01 00:04:33,1.0,0.69,1.0,N,162,100,1,5.0,0.5,0.5,1.76,0.0,0.3,10.56,2.5,0.0
1,2022-03-01 00:00:05,2022-03-01 00:18:11,1.0,6.1,1.0,N,132,134,2,21.0,1.75,0.5,0.0,0.0,0.3,23.55,0.0,1.25
1,2022-03-01 00:00:06,2022-03-01 00:15:02,2.0,5.9,1.0,Y,209,50,2,18.5,3.0,0.5,0.0,0.0,0.3,22.3,2.5,0.0
2,2022-03-01 00:00:07,2022-03-01 00:02:47,1.0,0.43,1.0,N,68,249,1,4.0,0.5,0.5,1.56,0.0,0.3,9.36,2.5,0.0
1,2022-03-01 00:00:08,2022-03-01 00:10:58,1.0,2.3,1.0,Y,170,4,1,10.0,3.0,0.5,2.0,0.0,0.3,15.8,2.5,0.0
2,2022-03-01 00:00:09,2022-03-01 00:07:59,1.0,1.66,1.0,N,113,68,1,8.0,0.5,0.5,2.36,0.0,0.3,14.16,2.5,0.0
2,2022-03-01 00:00:13,2022-03-01 00:29:19,1.0,17.18,2.0,N,132,140,1,52.0,0.0,0.5,8.0,0.0,0.3,64.55,2.5,1.25
1,2022-03-01 00:00:14,2022-03-01 00:02:13,2.0,0.5,1.0,N,142,50,1,3.5,3.0,0.5,2.15,0.0,0.3,9.45,2.5,0.0
2,2022-03-01 00:00:14,2022-03-01 00:14:45,2.0,7.86,1.0,N,138,75,1,23.0,0.5,0.5,6.17,6.55,0.3,38.27,0.0,1.25
1,2022-03-01 00:00:16,2022-03-01 00:15:13,1.0,8.1,1.0,Y,138,116,2,23.5,1.75,0.5,0.0,6.55,0.3,32.6,0.0,1.25
```

Information

1. Data file
   The file with taxi data is available at:
   https://dl.lsdupm.ovh/CLOUD/2122/July/yellow_tripdata_2022-03.csv

2. Software requirements

   The project must be implemented using Oracle Java 11 and Flink 1.14.0. **Please, check the most suitable type of window at https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/operators/windows/**

3. Parallelism

   The parallelism for the write operation to the output files must be 1.

4. Execution

   The project will be tested using the following procedure from the root folder of your project:

   - mvn clean package
   - flink run -c es.upm.fi.cloud.YellowTaxiTrip.*CLASS* YellowTaxiTrip-1.0-SNAPSHOT.jar --input $PATH_TO_INPUT_FILE --output $PATH_TO_OUTPUT_FILE

     Where:

     - *CLASS* is the java class of the exercise (CongestionArea, SaturatedVendor)
     - $PATH_TO_INPUT_FILE it the full path to the input file, i.e. /home/user/yellow_tripdata_2022-03.csv
     - $PATH_TO_OUTPUT_FILE it the full path to the output file, i.e. /home/user/outputFolder/congestionArea.csv or /home/user/outputFolder/saturatedVendor.csv

5. Submission

   - Groups of two students from the same master program
   - Only one submission per group
   - Submission name: **surnameStudent1-surnameStudent2.zip**
   - The zip file must have this structure:
     - YellowTaxiTrip/
       - src/
       - pom.xml
   - Exercises must be implemented as efficient as possible.
   - The zip file must be uploaded to **Moodle by 3th July 2022 at 23:55**

6. Evaluation

   Each exercise will be evaluated independently. The execution time and the memory usage will be taken into account in the grade.