We will use the setup given here <https://github.com/ververica/sql-training/wiki/Setting-up-the-Training-Environment>.

This uses the New York city taxi dataset as a stream that is hosted on Kafka.

We have shown you in a previous session how to install Docker containers for this setup.

You can learn more about the configuration of Flink SQL client: <https://ci.apache.org/projects/flink/flink-docs-stable/dev/table/sqlClient.html>.

Schemas for the data

We have three tables: Rides, Fares and DriverChanges.

The schemas for these tables and the meaning of their data can be found in <https://github.com/ververica/sql-training/wiki/Introduction-to-SQL-on-Flink>. Also, the link contains description about the built-in and user-defined functions already available and that you might need to use to implement the queries in this project.

You have to implement the following 3 queries using Flink SQL. The implementation of each query might require several select statements and maybe you need to define views in order to breakdown the complexity of the query and address the requirements.

Queries

Query 1: Average idle time per hour per area

We need to find the average idle time, in minutes, taxis usually spend within a time window of one hour within a specific area. The final output for this query should be (areaId, avg(idleTime)) of the area within 1 hour.

HINT:

To answer this query, we need to break it down on several steps. First you need to find successive records of isStart=false (drop off) followed by isStart=true (pickup) for the same taxiID and produce the as output (taxiID, diff (dropoff, pickup) as idleTime, areaId, matchTime). idleTime is the difference between the timestamp of the new ride and the previous end ride) and matchTime is the MATCH\_ROWTIME() built-in in FlinkSQL that assigns the timestamp of the match. The areaId is the area ID or (cell) in which pickup took place. The areaId is obtained from the lon/lat of the start new ride (pickup) record.

You might want to enforce an upper bound on the matching time window using the WITHIN clause of Match\_Recognize.

The result obtained above should be tracked as a stream (you can create a view). Based on that stream, you need to compute the average idle time by windowing(1 hour) over the matchTime attribute.

Query 2: Frequent Routes

The goal of the query is to find the top 10 most frequent routes during the last 30 minutes. A route is represented by a starting areaID and an ending areaID. All routes completed within the last 30 minutes are considered for the query. A route is completed when we observe a ride start followed by a ride end for the same taxi.

The output should be the starting area Id, end area Id, and the number of completed trips for the top 10 most frequent routes.

Hints: you should use the toAreaId function to map the location to a cell id. Also, you can use built-in support for ranking in Flink https://ci.apache.org/projects/flink/flink-docs-master/dev/table/sql/queries.htmlLinks to an external site.

You should follow the same practice of breaking down the query into one or more sub-queries and maybe you use views as well.

Query 3: Profitable areas

The goal of this query is to identify the profitability per area for taxi drivers. The profitability of an area is determined by dividing the area profit by the number of empty taxis in that area within the last 15 minutes. The profit that originates from an area is computed by calculating the average fare + tip for trips that started in the area and ended within the last 15 minutes. The number of empty taxis in an area is the sum of taxis that had a drop-off location in that area less than 30 minutes ago and had no following pickup yet.

HINTS:

You need to decompose this query into at least two views. One view computes the profit per area and another one that needs to compute the number of empty taxis.

To compute the profit, you will need to join the Rides stream with the Fares stream. Consider that the fare is paid within 10 minutes before the rid ends.

You should use match recognize to identify both completed and incomplete rides for the two queries.

All previous practices about decomposition are helpful too.

Grading

Each query is worth 5 marks. The breakdown of the grading of each is whether you achieve the target and use Flink SQL features suggested to write concise and readable queries. Each part of each query should be documented to help follow up your logic.

Deliverables

For each query, you should deliver a separate .sql file that contains all the DDL/DML statements in Flink SQL syntax required for running the query. The header of the file should start with your general explanation of how you approached the implementation of the query using SQL comments (--). Then the different DDL/DML statements should be put in order. We will run a Flink SQL terminal from our side, following the same guidelines communicated in the lecture, and will feed each statement to the terminal and check the final output.