Processamento de Streams
Análise a Corridas de Táxis

André Lopes - 45617
Nelson Coquenim - 45694
Departamento de Informática
Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa
Almada, Portugal

I. Introdução

II. *Frequent Routes*

O objectivo desta primeira *query* é achar o *top* 10 das rotas mais frequentes durante um período de 30 minutos. Um rota é representada por uma *cell* inicial e uma *cell* final.

Na Figura 1 pode se observar a estrutura desta query. Primeiramente efetua-se uma janela deslizante de 30 minutos sobre o input. Finalmente, seleciona-se os 10 resultados com a frequência mais alta.

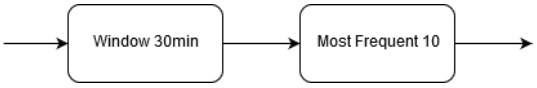

Fig. 1. Diagrama da query *Frequent Routes*.

O código siddhi que implementa esta *query* é o seguinte:

```
from TaxiSecStr#window.time(30 minutes)
select pickup_gridID,
  dropoff_gridID,
  count(*) as frequency
group by pickup_gridID,
  dropoff_gridID
order by frequency DESC
insert into RouteFrequencyStr;

from RouteFrequencyStr#window.length(10)
select *
insert into TopFreqRoutesStr;
```

III. *Profitables Areas*

Nesta query pretende-se identificar, de forma contínua, as áreas que são mais lucrativas para os taxistas. Para tal, o lucro de uma área é definido pelas receitas geradas nessa área a dividir pelo número de táxis vazios nessa mesma área.

A receita gerada numa área é a média das *fare* + *tip* de todas as corridas que originaram nessa área e que acabaram nos 15 minutos seguintes.

O número de táxis vazios num dada área consiste na soma dos táxis que efetuaram uma *dropoff* nessa área mas que após 30 minutos ainda não efetuaram uma *pickup*.

Na Figura 2 pode-se observar um diagrama que demonstra o fluxo desta *query*.

O código siddhi que implementa esta *query* é o seguinte:

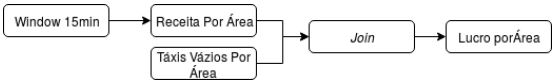

Fig. 2. Diagrama da query *Profitables Areas*.

```
from TaxiSecStr#window.time(15 min)
select pickup_gridID as areaID
   avg(FareTrip) as revenue
group by pickup_gridID
insert into RevenuePerAreaStr;

from e1 = TaxiSecStr ->
  TaxiSecStr[medallion==e1.medallion
  and pickup_datetime
  - e1.dropoff_datetime>30 mins]
select dropoff_gridID as areaID
insert into EmptyTaxisAreasStr;

partition with
   (areaID of RevenuePerAreaStr)
begin
  from EmptyTaxisAreasStr
  select areaID, count(*) as emptyTaxis
  insert into #EmptyTaxisPerAreaStr;

  from RevenuePerAreaStr as A
            join
  #EmptyTaxisPerAreaStr as B
    on A.areaID == B.areaID
  select A.areaID,
    revenue/emptyTaxis as profit
  insert into ProfitPerAreaStr;
end;
```

IV. *Idle Taxis*

Neste *use case* espera-se que seja emitido um alerta quando o número de táxis disponíveis torna-se superior ao pretendido. Para tal, deverá ser publicado um aviso quando o tempo de paragem médio (*idle time*) de todos os táxis é superior a 10 minutos. Define-se como tempo de paragem, o período de tempo entre uma *dropoff* e uma *pickup*. Finalmente, assume-se que um táxi encontra-se disponível se tiver realizado pelo menos uma viagem na última hora.

O diagrama na Figura 3 demonstra a lógica da implementação desta *query*.

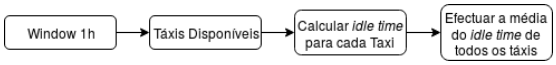

Fig. 3. Diagrama da query *Idle Taxis*.

Em seguida, apresenta-se o excerto de código da *query* em questão:

```
from TaxiSecStr#window.time(1 hour)
select *
```

V. *Congested Areas*

Nesta secção ir-se-à implementar uma *query* que emita as localizações onde possivelmente poderá haver congestionamentos no trânsito. Para tal, dever-se-á detetar picos nas durações das viagens dos táxis que são seguidos por pelo menos 3 viagens todas estas com durações crescentes.

A Figura 4 expõe o racional na construção desta *query*.

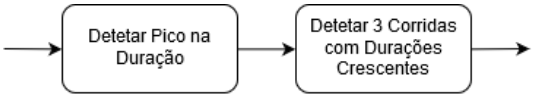

Fig. 4. Diagrama da query *Congested Areas*.

Finalmente, apresenta-se o código siddhi da *query congested areas*:

```
from every e1 = TaxiSecStr ->
  e2 = TaxiSecStr[medallion==e1.medallion
  and ride_duration > e1.ride_duration] ->
  e3 = TaxiSecStr[medallion==e2.medallion
  and ride_duration < e2.ride_duration] ->
  e4 = TaxiSecStr[medallion==e3.medallion
  and ride_duration > e3.ride_duration] ->
  e5 = TaxiSecStr[medallion==e4.medallion
  and ride_duration > e4.ride_duration] ->
  e6 = TaxiSecStr[medallion==e5.medallion
  and ride_duration > e5.ride_duration]
select e2.pickup_grid_x as grid_x,
  e2.pickup_grid_y as grid_y
insert into CongestedAreasStr;

insert into AvailableTaxisStr;

from e1 = AvailableTaxisStr ->
  e2 = AvailableTaxisStr[medallion
    == e1.medallion]
select e1.medallion as taxi,
  (e2.p_time-e1.d_time) as idle_time
insert into IdleTimeTaxisStr;

from IdleTimeTaxisStr
select avg(idle_time) as avg_idle_time
having avg_idle_time > 10 * 60
insert into IdleTaxisStr;
```

VI. *Most Pleasant Taxi Drivers*

Para recompensar os condutores de táxis mais simpáticos é necessário que seja emitido, uma vez por dia, o taxista que recebeu mais gorjetas nesse dia.

O fluxo da Figura 5 demonstra a construção desta *query*.

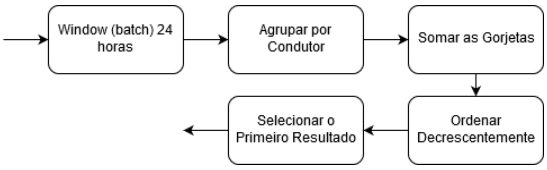

Fig. 5. Diagrama da query *Most Pleasant Taxi Driver*.

Por último, no segmento de código seguinte apresenta-se a solução referente a este *use case*:

```
from TaxiSecStr#window.timeBatch(24 hour)
select driver,
  sum(tip_amount) as tips_total
group by driver
order by tips_total DESC
insert into TodayDriversTips;

from TodayDriversTips#window.length(1)
select *
insert into PleasantDriverStr;
```

VII. Conclusão