Pre-processing Streaming Data

- Introduction
- Tutorial steps

Introduction

In the previous tutorials, we learned how to create a simple Siddhi application and capture the events coming from external sources for analysis. This tutorial shows how to pre-process data before analyzing them.

Let's consider a scenario where the Sweet Factory foreman needs to identify events generated by Sweet Bots that match the following criteria:

- The sweet category produced is either eclair or gingerbread.
- The amount produced is greater than 10.

The results generated must be presented with the name of the sweet, the total value of the batch and the currency unit. Each eclair and gingerbread costs £0.4 to produce. In addition, the production overhead per batch is £2. The currency unit is GBP.

This tutorial covers the following concepts:

- Siddhi filters
- Transformations and defaults

Before you begin:

This tutorial reuses the source configuration that was first created and explained in Consuming Events. Therefore, it is recommended to try Tutorial 2 before following this tutorial.

Tutorial steps

Let's get started!

1. Start the WSO2 SP in the editor mode and login to the Stream Processor Studio. Then open a new Siddhi file.

2. In this tutorial, you can use the same input stream that you have been using in all the previous tutorials where the events receive include the sweet category name and the quantity produced. Let's add it as follows.

   ```
   define stream SweetProductionStream (name string, amount long);
   ```

3. Let's define the output stream based on what information you require as the output. As mentioned in the introduction, this scenario requires the output to include the name of the sweet, the value of the batch and the currency unit. Therefore, let's add an output stream named FilteredSweetStream with an attribute for each detail mentioned.

   ```
   define stream FilteredSweetStream (name string, value double, currency string);
   ```

4. To process the input data and insert the results into the FilteredSweetStream stream, let's add a query as follows.
In this query, the amount produced of sweet category is multiplied by the production cost per unit, and 2 (representing the production overhead per batch of £2) is added to the result to derive the total production cost per batch. This is a Siddhi transformation where the value of an input attribute is changed while processing.

Also note that the GBP static value is specified as the currency. This is how static values are added to data streams. Each event that goes through this preprocessing step contains this static value for the intended attribute.

5. The query added in the previous step outputs all the events after calculating the batch value. However, the output must consist of only events where the sweet category is either eclair or gingerbread, and the amount produced is greater than 10. To achieve this, let's add two filters as follows.
   a. Add a filter with the [ ] notation to specify the conditions based on which the events are filtered.

   ```sql
   [name == "eclair"], [name == "gingerbread"], [amount > 10]
   ```

   b. In the above step, you have specified the conditions to be met, but not how those conditions work together. You can use and, or, and not operators to specify how the conditions are related.

   ```sql
   [ (name == "eclair" or name == "gingerbread") and amount > 10 ]
   ```

   In this scenario, the sweet category can be eclair or gingerbread. Therefore, the or operator is used to denote the connection between the conditions [name == "eclair"] and [name == "gingerbread"] where it indicates that either condition can apply. The [amount > 10] must apply together with either of the first two conditions. This is indicated via the and operator.

   The query with the completed filter looks as follows.

   ```sql
   from SweetProductionStream select name, ((amount * 0.4) + 2) as value, "GBP" as currency insert into FilteredSweetStream;
   ```

   The completed Siddhi application (which you can name as SweetProductionFilteringApp) looks as follows.
@App:name('SweetProductionFilteringApp')

@source(type='http',
receiver.url='http://localhost:5005/SweetProductionEP', @map(type = 'json', @attributes(name = '$.sweet', amount = '$.batch.count')))
define stream SweetProductionStream (name string, amount long);

@sink(type='log', prefix='Conditionally filtered Sweets:')
define stream FilteredSweetStream (name string, value double, currency string);

from SweetProductionStream [(name == "eclair" or name == "gingerbread") and amount > 10]
select name, ((amount * 0.4) + 2) as value, "GBP" as currency
insert into FilteredSweetStream;

6. To see how the output is generated by this Siddhi application, let's start the application in the editor and simulate five events by issuing the following cURL commands. Only the last two events sent match the filter conditions. Therefore, only those two events are returned with the pre-processed values.

curl -X POST \
   http://localhost:5005/SweetProductionEP \
   -H 'content-type: application/json' \
   -d '
   "sweet": "Jaffa Cake",
   "batch": {
     "batch id": "batch1",
     "count": 10
   }
   ' 

curl -X POST \
   http://localhost:5005/SweetProductionEP \
   -H 'content-type: application/json' \
   -d '
   "sweet": "Jaffa Cake",
   "batch": {
     "batch id": "batch1",
     "count": 10
   }
   ' 

curl -X POST \
http://localhost:5005/SweetProductionEP \
-H 'content-type: application/json' \
-d '{
  "sweet": "gingerbread",
  "batch": {
    "batch id": "batch1",
    "count": 10
  }
}'

curl -X POST \
http://localhost:5005/SweetProductionEP \
-H 'content-type: application/json' \
-d '{
  "sweet": "eclair",
  "batch": {
    "batch id": "batch1",
    "count": 11
  }
}'

curl -X POST \
http://localhost:5005/SweetProductionEP \
-H 'content-type: application/json' \
-d '{
  "sweet": "gingerbread",
  "batch": {
    "batch id": "batch1",
    "count": 12
  }
}'

The output is logged as follows:

```
[2017-12-19 13:56:55,394] INFO {org.wso2.siddhi.core.stream.output.sink.LogSink} - Conditionally filtered Sweets: Event(timestamp=1513672082599, data=[gingerbread, 6.80000000000001, GBP], isExpired=false)
```

Only last two events are returned with the pre-processed values.