How-To Library

Informatica

Stateful Computing on the Spark Engine for Big Data Management 10.2

Abstract

You can use window functions to perform stateful calculations on the Spark engine. Window functions operate on a partition or "window" of data, and return a value for every row in that window. This article describes the steps to configure a transformation for windowing and define window functions in an Expression transformation. This article assumes that you are familiar with mappings and know how to develop and run a mapping.

Supported Versions

- Big Data Management® 10.2

Table of Contents

Overview

Stateful computation uses the information about past or future rows to affect the processing of the current row. The data from these rows is used to modify the state of the current row, returning a value that has a new state. You can use window functions to perform stateful calculations. Window functions operate on a group of rows and calculate a return value for every input row. When you want to retrieve data from upstream or downstream rows and perform cumulative calculations, use windowing to access the data and operate on different groups of rows.

Use Case: Compare Current and Previous Records

You are a developer working for a commuter rail line. Multiple trains run on the rail line and stop at different stations throughout the day. The location of each train is recorded at 10-second intervals.

The train location and timestamp are recorded in the following format:

Train ID	Station	Date	Timestamp
1	AMS	17/30/09	10:00:00
1	AMS	17/30/09	10:00:10
1	AMS	17/30/09	10:00:20
1	HVS	17/30/09	10:15:20
1	HVS	17/30/09	10:15:30

Train ID	Station	Date	Timestamp
1	HVS	17/30/09	10:15:40
1	AMF	17/30/09	10:33:10

You need to determine the arrival and departure times of a train at each station. To calculate what time a train departs from a station, you must determine if the stations in the previous record and the current record differ. If the stations differ, the current record is considered a new stop. The previous timestamp is the departure time from the previous station, and the current timestamp is the arrival time at the next station. If the stations do not differ, the current record is not considered a new stop.

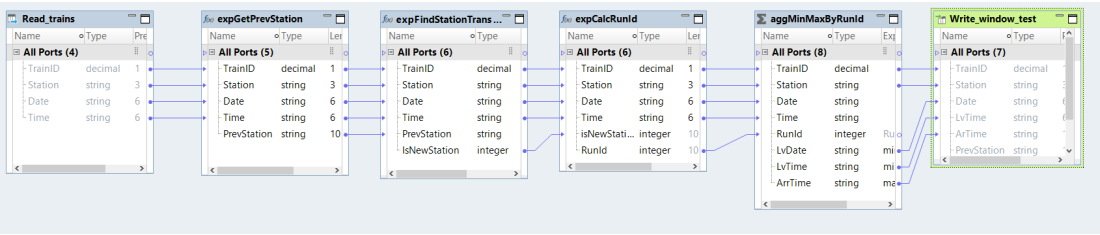
For example, the stations in the third and fourth record above are different. One station is AMS and the next station is HVS. Because these records differ, you determine that the train departs from AMS station at 10:00:20 and arrives at HVS station at 10:15:20.

Mapping

To create the mapping, perform the following high-level steps:

1. Define a window function to access the previous row for every record.
2. Configure an Expression transformation to identify when a train changes stations.
3. Define an aggregate window function to assign a unique value to the rows corresponding to each station.
4. Configure an Aggregator transformation to aggregate by the unique station value. Define aggregate expressions to calculate the arrival and departure times at each station.

The following image shows the mapping:



Step 1. Access the Previous Row for Every Record

The LAG window function returns the value that is an offset number of rows before the current row. You need to get the state of the previous row for each record so that you can use it to identify when a station changes. Configure an Expression transformation called GetPrevStation. Configure a port called PrevStation with a LAG function to access the previous station for every record.

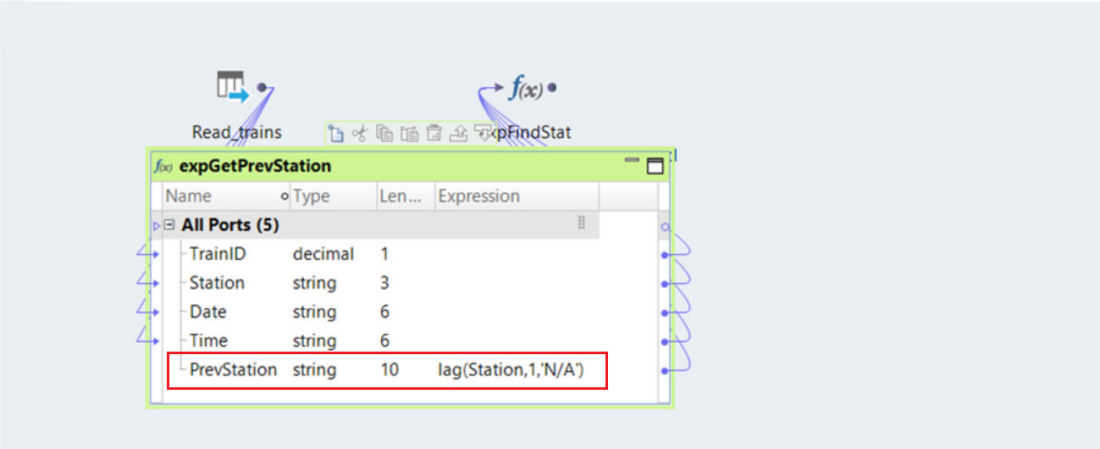
Define the following function on the Ports tab of the Expression transformation:

```
LAG ( Station, 1 )
```

Where:

- `Station` is the target column that the function operates on.
- `1` is the offset. This value accesses the row immediately before the current row.

The following image shows the window function:



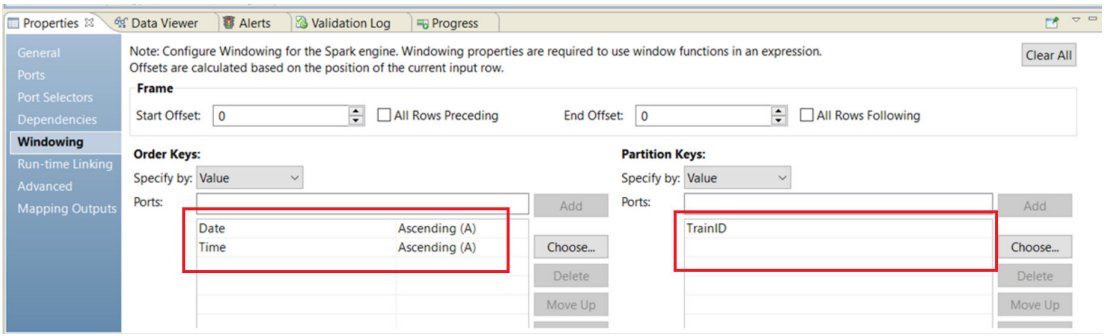
When you include a window function in an Expression transformation, you configure the windowing properties associated with the function. Windowing properties define the partitioning, ordering, and frame boundaries associated with a particular input row.

Create a partition to process the rows by train ID. Order the data by date and time.

Define the following windowing properties on the Windowing tab:

Property	Value
Order key	`Date, Time`. Displays the data chronologically.
Partition key	`TrainID`. Processes the rows according to train ID.

The following image shows the windowing configuration:



Step 2. Determine When a Station Changes

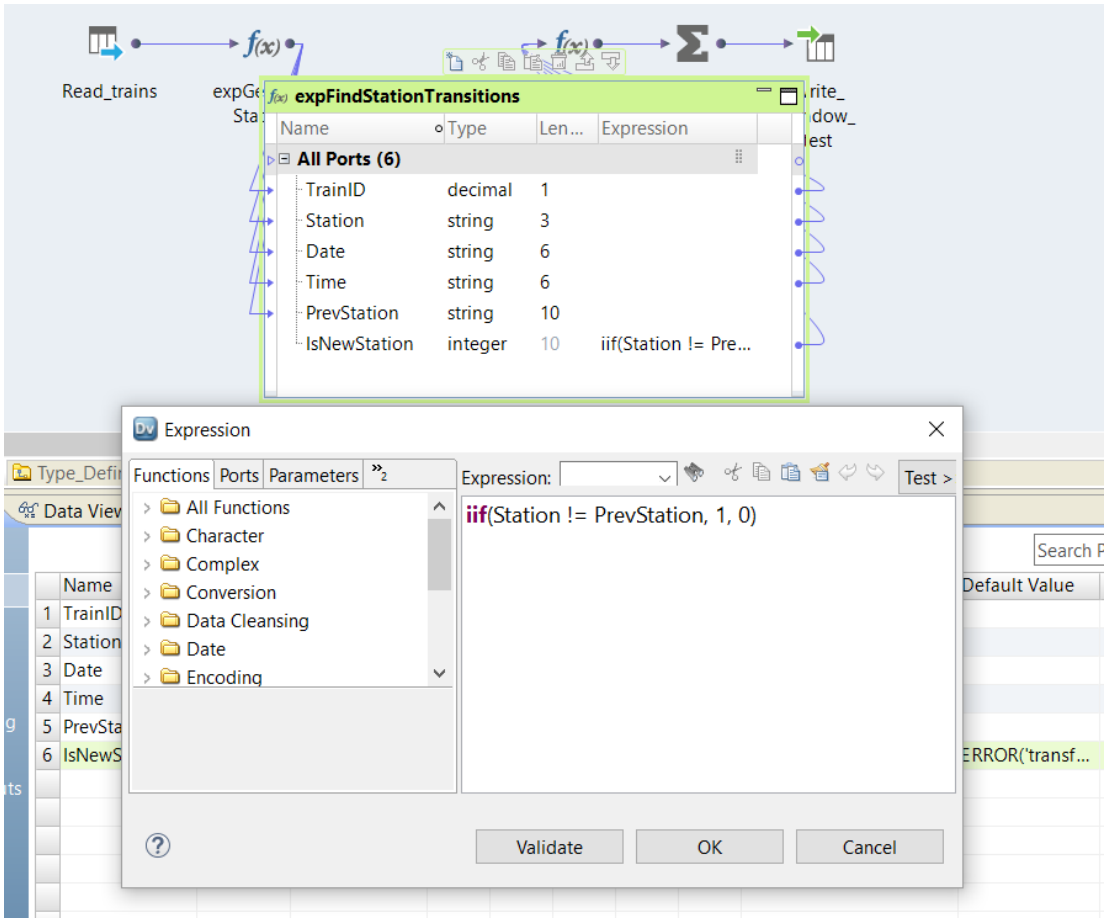
Configure an Expression transformation called FindStationTransitions that determines when a train travels from one station to the next. Define a port called IsNewStation that compares the station in the current record with the station in the previous record.

Define the following function to check if the current record identifies a different station:

```
IIF ( Station != PrevStation, 1, 0 )
```

The value of the IsNewStation port is 1 when the station changes. The value is 0 when the station does not change.

The following image shows the function in the Expression Editor:



Step 3. Assign a Unique Value to the Rows Pertaining to Each Station

You can use aggregate functions as window functions to perform cumulative calculations on a set of data. An aggregate function acts as a window function when you configure the transformation for windowing.

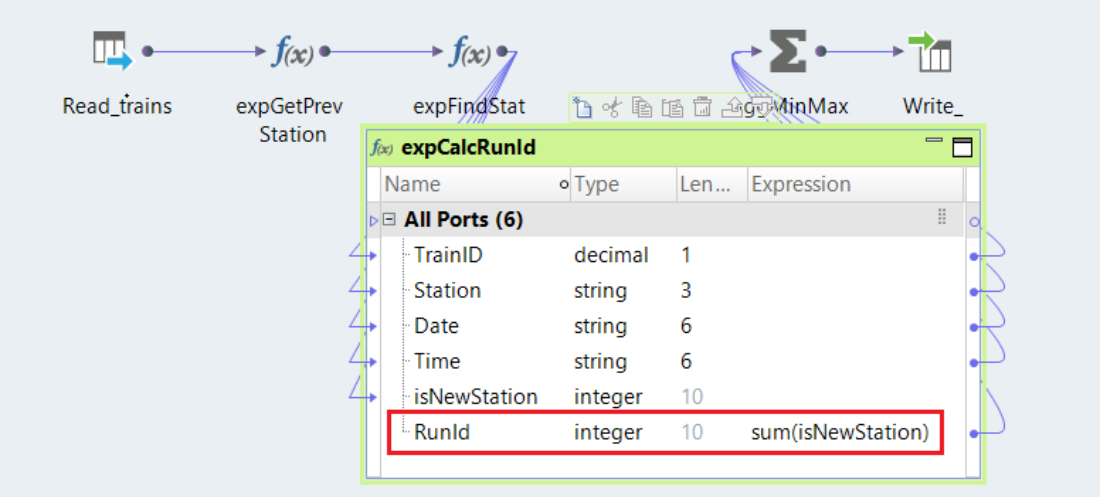
Configure an Expression transformation called CalcRunID. Configure a port called RunID with a SUM function to calculate the sum of the IsNewStation port. When you configure the transformation for windowing, the SUM function calculates a *cumulative* sum for every row.

IsNewStation is 1 every time a station changes. So as the cumulative sum processes each row chronologically, the RunID port increments at each station and stays the same until the station changes again. This assigns a unique value to the rows pertaining to each station.

Define the following aggregate function in the Expression Editor:

```
SUM ( IsNewStation )
```

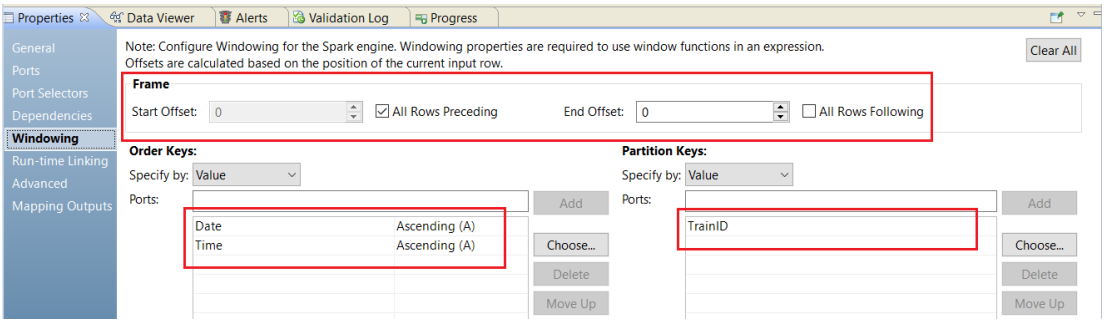
The following image shows the aggregate function:



Define the following windowing properties on the Windowing tab:

Property	Value
Start offset	`All Rows Preceding`. Includes all previous records within the partition in the calculation, producing a cumulative sum.
End offset	`0`
Order key	`Date, Time`. Displays the data chronologically.
Partition key	`TrainID`. Processes the data according to train ID.

The following image shows the windowing configuration:



Step 4. Aggregate the Data to Calculate Departure and Arrival Times

Configure an Aggregator transformation to aggregate by the unique station value assigned in Step 3. Define aggregate expressions to calculate the departure and arrival times at each station.

Define the following Group By ports in the Aggregator transformation:

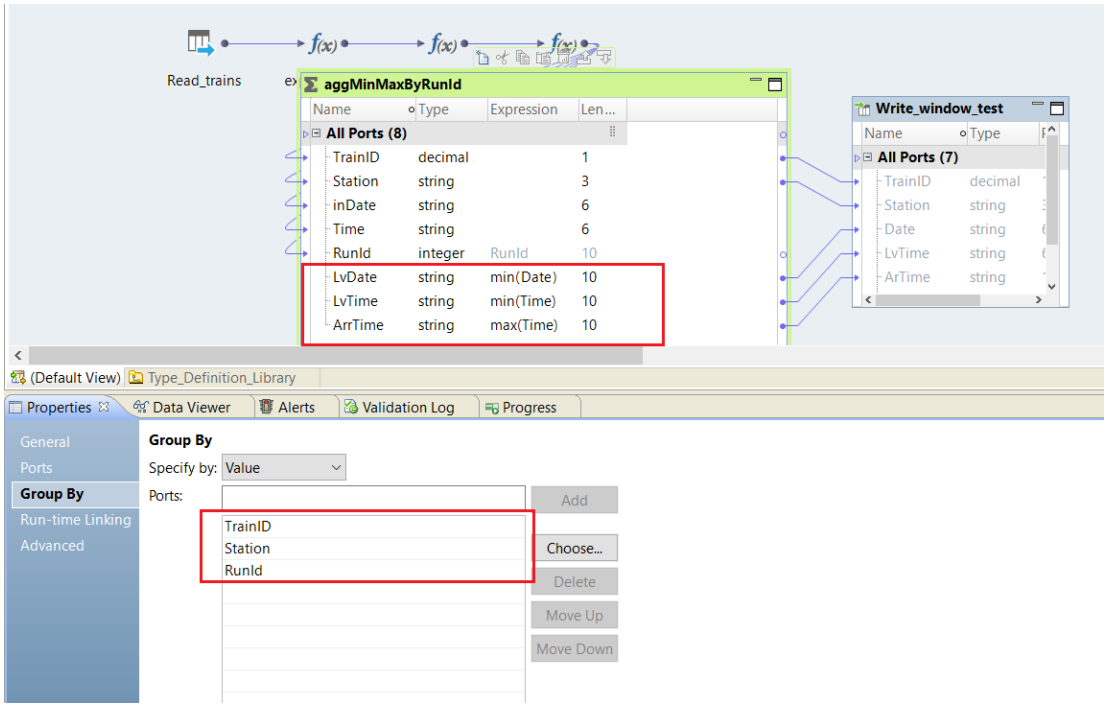
- `TrainID`
- `Station`

- `RunID`

Define the following aggregate expressions in the output ports of the Aggregator transformation:

- `MIN ( Time )`. Calculates the departure time.
- `MAX ( Time )`. Calculates the arrival time.
- `MIN ( Date )`. Calculates the departure date. If the departure date is different than the arrival date, the mapping uses the departure date in the final output.

The following image shows the Group By ports and aggregate expressions in the Aggregator transformation:



Mapping Output

The mapping produces data in the following format:

Train ID	Station	Date	Arrival Time	Departure Time
1	AMS	17/30/09	10:00:00	10:00:20
1	HVS	17/30/09	10:15:20	10:15:40
1	AMF	17/30/09	10:33:10	10:33:30
2	AMS	17/30/09	12:00:00	12:00:20
2	HVS	17/30/09	12:20:10	12:20:30
2	AMF	17/30/09	12:32:30	12:32:50

Author

Elizabeth Snyder