ROLLINGAVERAGE Function

Contents:

Computes the rolling average of values forward or backward of the current row within the specified column.

- If an input value is missing or null, it is not factored in the computation. For example, for the first row in the dataset, the rolling average of previous values is the value in the first row.
- The row from which to extract a value is determined by the order in which the rows are organized based on the `order` parameter.

  If you are working on a randomly generated sample of your dataset, the values that you see for this function might not correspond to the values that are generated on the full dataset during job execution.
- The function takes a column name and two optional integer parameters that determine the window backward and forward of the current row.
  - The default integer parameter values are `-1` and `0`, which computes the rolling average from the current row back to the first row of the dataset.
- This function works with the following transforms:
  - *Window Transform*
  - *Set Transform*
  - *Derive Transform*

For more information on a non-rolling version of this function, see *AVERAGE Function*.

Basic Usage

Column example:

```
derive type:single value:ROLLINGAVERAGE(myCol)
```

Output: Generates a new column containing the rolling average of all values in the `myCol` column from the first row of the dataset to the current one.

Rows before example:

```
window value:ROLLINGAVERAGE(myNumber, 3)
```

Output: Generates the new column, which contains the rolling average of the current row and the three previous row values in the `myNumber` column.

Rows before and after example:

```
window value:ROLLINGAVERAGE(myNumber, 3, 2)
```

Output: Generates the new column , which contains the rolling average of the three previous row values, the current row value, and the two rows after the current one in the `myNumber` column.

Syntax and Arguments

```
window value:ROLLINGAVERAGE(col_ref, rowsBefore_integer, rowsAfter_integer) order:
order_col [group: group_col]
```

Argument	Required?	Data Type	Description
col_ref	Y	string	Name of column whose values are applied to the function
rowsBefore_integer	N	integer	Number of rows before the current one to include in the computation
rowsAfter_integer	N	integer	Number of rows after the current one to include in the computation

For more information on the `order` and `group` parameters, see *Window Transform*.

For more information on syntax standards, see *Language Documentation Syntax Notes*.

col_ref

Name of the column whose values are used to compute the rolling average.

- Multiple columns and wildcards are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	String (column reference to Integer or Decimal values)	`myColumn`

rowsBefore_integer, rowsAfter_integer

Integers representing the number of rows before or after the current one from which to compute the rolling average, including the current row. For example, if the first value is `5`, the current row and the five rows before it are used in the computation. Negative values for `k` compute the rolling average from rows preceding the current one.

- `rowBefore=0` generates the current row value only.
- `rowBefore=-1` uses all rows preceding the current one.
- If `rowsAfter` is not specified, then the value `0` is applied.
- If a `group` parameter is applied, then these parameter values should be no more than the maximum number of rows in the groups.

Usage Notes:

Required?	Data Type	Example Value
No	Integer	4

Examples

Tip: For additional examples, see *Common Tasks*.

Example - Compute prior quarter values

The following dataset contains order information for the preceding 12 months. You want to compare the current month's average against the preceding quarter.

Source:

Date	Amount
12/31/15	118
11/30/15	6
10/31/15	443
9/30/15	785
8/31/15	77
7/31/15	606
6/30/15	421
5/31/15	763
4/30/15	305
3/31/15	824
2/28/15	135
1/31/15	523

Transform:

Using the `ROLLINGAVERAGE` function, you can generate a column containing the rolling average of the current month and the two previous months:

```
window value: ROLLINGAVERAGE(Amount, 3, 0) order: -Date
```

Note the sign of the second parameter and the `order` parameter. The sort is in the reverse order of the `Date` par ameter, which preserves the current sort order. As a result, the second parameter, which identifies the number of rows to use in the calculation, must be positive to capture the previous months.

Technically, this computation does not capture the prior quarter, since it includes the current quarter as part of the computation. You can use the following column to capture the rolling average of the preceding month, which then becomes the true rolling average for the prior quarter. The `window` column refers to the name of the column generated from the previous step:

```
window value: NEXT(window, 1) order: -Date
```

Note that the order parameter must be preserved. This new column, `window1`, contains your prior quarter rolling average:

```
rename col:window1 to:'Amount_PriorQtr'
```

You can reformat this numeric value:

```
set col:Amount_PriorQtr value:NUMFORMAT(Amount_PriorQtr, '###.00')
```

You can use the following transform to calculate the net change. This formula computes the change as a percentage of the prior quarter and then formats it as a two-digit percentage.

```
derive type:single value:NUMFORMAT(((Amount - Amount_PriorQtr) / Amount_PriorQtr) * 100,
'##.##') as:'NetChangePct_PriorQtr'
```

Results:

NOTE: You might notice that there are computed values for `Amount_PriorQtr` for February and March. These values do not factor in a full three months because the data is not present. The January value does not exist since there is no data preceding it.

Date	Amount	Amount_PriorQtr	NetChangePct_PriorQtr
12/31/15	118	411.33	-71.31
11/30/15	6	435.00	-98.62
10/31/15	443	489.33	-9.47
9/30/15	785	368.00	113.32
8/31/15	77	596.67	-87.1
7/31/15	606	496.33	22.1
6/30/15	421	630.67	-33.25
5/31/15	763	421.33	81.09
4/30/15	305	494.00	-38.26
3/31/15	824	329.00	150.46
2/28/15	135	523.00	-.74.19
1/31/15	523		

Example - Rolling window functions

This example describes how to use the rolling computational functions:

- `ROLLINGSUM` - computes a rolling sum from a window of rows before and after the current row. See *ROLLINGSUM Function*.
- `ROLLINGAVERAGE` - computes a rolling average from a window of rows before and after the current row. See *ROLLINGAVERAGE Function*.
- `ROWNUMBER` - computes the row number for each row, as determined by the ordering column. See *ROWNUMBER Function*.

The following dataset contains sales data over the final quarter of the year.

Source:

Date	Sales
10/2/16	200
10/9/16	500
10/16/16	350
10/23/16	400
10/30/16	190
11/6/16	550
11/13/16	610

11/20/16	480
11/27/16	660
12/4/16	690
12/11/16	810
12/18/16	950
12/25/16	1020
1/1/17	680

Transform:

First, you want to maintain the row information as a separate column. Since data is ordered already by the `Date` column, you can use the following:

```
window value:ROWNUMBER() order:Date
```

Rename this column to `rowId` for week of quarter.

Now, you want to extract month and week information from the `Date` values. Deriving the month value:

```
derive type:single value:MONTH(Date) as:'Month'
```

Deriving the quarter value:

```
derive type:single value:(1 + FLOOR(((month-1)/3))) as:'QTR'
```

Deriving the week-of-quarter value:

```
window value:ROWNUMBER() order:Date group:QTR
```

Rename this column `WOQ` (week of quarter).

Deriving the week-of-month value:

```
window value:ROWNUMBER() group:Month order:Date
```

Rename this column `WOM` (week of month).

Now, you perform your rolling computations. Compute the running total of sales using the following:

```
window value: ROLLINGSUM(Sales, -1, 0) order: Date group:QTR
```

The `-1` parameter is used in the above computation to gather the rolling sum of all rows of data from the current one to the first one. Note that the use of the `QTR` column for grouping, which moves the value for the `01/01/2017` into its own computational bucket. This may or may not be preferred.

Rename this column `QTD` (quarter to-date). Now, generate a similar column to compute the rolling average of weekly sales for the quarter:

```
window value: ROUND(ROLLINGAVERAGE(Sales, -1, 0)) order: Date group:QTR
```

Since the `ROLLINGAVERAGE` function can compute fractional values, it is wrapped in the `ROUND` function for neatness. Rename this column `avgWeekByQuarter`.

Results:

When the unnecessary columns are dropped and some reordering is applied, your dataset should look like the following:

Date	WOQ	Sales	QTD	avgWeekByQuarter
10/2/16	1	200	200	200
10/9/16	2	500	700	350
10/16/16	3	350	1050	350
10/23/16	4	400	1450	363
10/30/16	5	190	1640	328
11/6/16	6	550	2190	365
11/13/16	7	610	2800	400
11/20/16	8	480	3280	410
11/27/16	9	660	3940	438
12/4/16	10	690	4630	463
12/11/16	11	810	5440	495
12/18/16	12	950	6390	533
12/25/16	13	1020	7410	570
1/1/17	1	680	680	680

Example - Rolling computations for racing splits

This example describes how to use the rolling computational functions:

- `ROLLINGAVERAGE` - computes a rolling average from a window of rows before and after the current row. See *ROLLINGAVERAGE Function*.
- `ROLLINGMIN` - computes a rolling minimum from a window of rows. See *ROLLINGMIN Function*.
- `ROLLINGMAX` - computes a rolling maximum from a window of rows. See *ROLLINGMAX Function*.
- `ROLLINGSTDEV` - computes a rolling standard deviation from a window of rows. See *ROLLINGSTDEV Function*.
- `ROLLINGVAR` - computes a rolling variance from a window of rows. See *ROLLINGVAR Function*.

Source:

In this example, the following data comes from times recorded at regular intervals during a three-lap race around a track. The source data is in cumulative time in seconds (`time_sc`). You can use ROLLING and other windowing functions to break down the data into more meaningful metrics.

lap	quarter	time_sc
1	0	0.000
1	1	19.554
1	2	39.785
1	3	60.021
2	0	80.950

2	1	101.785
2	2	121.005
2	3	141.185
3	0	162.008
3	1	181.887
3	2	200.945
3	3	220.856

Transform:

Primary key: Since the quarter information repeats every lap, there is no unique identifier for each row. The following steps create this identifer:

```
settype col: lap,quarter type: 'String'
```

```
derive type:single value: MERGE(['l',lap,'q',quarter]) as: 'splitId'
```

Get split times: Use the following transform to break down the splits for each quarter of the race:

```
derive type:single value: ROUND(time_sc - PREV(time_sc, 1), 3) order: splitId as:
'split_time_sc'
```

Compute rolling computations: You can use the following types of computations to provide rolling metrics on the current and three previous splits:

```
derive type:single value: ROLLINGAVERAGE(split_time_sc, 3) order: splitId as: 'ravg'
```

```
derive type:single value: ROLLINGMAX(split_time_sc, 3) order: splitId as: 'rmax'
```

```
derive type:single value: ROLLINGMIN(split_time_sc, 3) order: splitId as: 'rmin'
```

```
derive type:single value: ROUND(ROLLINGSTDEV(split_time_sc, 3), 3) order: splitId as:
'rstdev'
```

```
derive type:single value: ROUND(ROLLINGVAR(split_time_sc, 3), 3) order: splitId as: 'rvar'
```

Results:

When the above transforms have been completed, the results look like the following:

lap	quarter	splitId	time_sc	split_time_sc	rvar	rstdev	rmin	rmax	ravg
1	0	l1q0	0						
1	1	l1q1	20.096	20.096	0	0	20.096	20.096	20.096
1	2	l1q2	40.53	20.434	0.029	0.169	20.096	20.434	20.265
1	3	l1q3	61.031	20.501	0.031	0.177	20.096	20.501	20.344
2	0	l2q0	81.087	20.056	0.039	0.198	20.056	20.501	20.272
2	1	l2q1	101.383	20.296	0.029	0.17	20.056	20.501	20.322
2	2	l2q2	122.092	20.709	0.059	0.242	20.056	20.709	20.39
2	3	l2q3	141.886	19.794	0.113	0.337	19.794	20.709	20.214

3	0	l3q0	162.581	20.695	0.139	0.373	19.794	20.709	20.373
3	1	l3q1	183.018	20.437	0.138	0.371	19.794	20.709	20.409
3	2	l3q2	203.493	20.475	0.113	0.336	19.794	20.695	20.35
3	3	l3q3	222.893	19.4	0.252	0.502	19.4	20.695	20.252

You can reduce the number of steps by applying a `window` transform such as the following:

```
window value: window1 = lap,rollingaverage(split_time_sc, 0, 3), rollingmax(split_time_sc,
0, 3),rollingmin(split_time_sc, 0, 3),round(rollingstdev(split_time_sc, 0, 3), 3),round
(rollingvar(split_time_sc, 0, 3), 3) group: lap order: lap
```

However, you must rename all of the generated `windowX` columns.