SOURCEROWNUMBER Function

Contents:

NOTE: This function has been superseded by the `$sourcerownumber` reference. While this function is still usable in the product, it is likely to be deprecated in a future release. Please use `$sourcerownumber` instead. For more information, see *Source Metadata References*.

Returns the row number of the current row as it appeared in the original source dataset before any steps had been applied.

The following transforms might make original row information invalid or otherwise unavailable. In these cases, the function returns null values:

- `pivot`
- `flatten`
- `join`
- `lookup`
- `union`
- `unnest`
- `unpivot`

NOTE: This function does not apply to relational database inputs, such as Hive or JDBC sources.

NOTE: If the dataset is sourced from multiple files, a predictable original source row number cannot be guaranteed, and null values are returned.

Tip: If the source row information is still available, you can hover over the left side of a row in the data grid to see the source row number in the original source data.

Basic Usage

Derive Example:

```
derive type:single value:SOURCEROWNUMBER() as:'OriginalRowNums'
```

Output: Generates a new `OriginalRowNums` column containing the row numbers for each row as it appeared in the original data.

Sort Example:

```
sort order: SOURCEROWNUMBER()
```

Output: Rows in the dataset are re-sorted according to the original order in the dataset.

Delete Example:

```
delete row:SOURCEROWNUMBER() > 101
```

Output: Deletes the rows in the dataset that were after row #101 in the original source data.

Syntax and Arguments

There are no arguments for this function.

Examples

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

Example - Header from row that is not the first one

Source:

You have imported the following racer data on heat times from a CSV file. When loaded in the Transformer page, it looks like the following:

(rowId)	column2	column3	column4	column5
1	Racer	Heat 1	Heat 2	Heat 3
2	Racer X	37.22	38.22	37.61
3	Racer Y	41.33	DQ	38.04
4	Racer Z	39.27	39.04	38.85

In the above, the `(rowId)` column references the row numbers displayed in the data grid; it is not part of the dataset. This information is available when you hover over the black dot on the left side of the screen.

Transform:

You have examined the best performance in each heat according to the sample. You then notice that the data contains headers, but you forget how it was originally sorted. The data now looks like the following:

(rowId)	column2	column3	column4	column5
1	Racer Y	41.33	DQ	38.04
2	Racer	Heat 1	Heat 2	Heat 3
3	Racer X	37.22	38.22	37.61
4	Racer Z	39.27	39.04	38.85

You can use the following transformation to use the third row as your header for each column:

> **NOTE:** The following does not use the header transform.

```
rename type: header method: index sourcerownumber: 3
```

Results:

After you have applied the above transformation, your data should look like the following:

(rowId)	Racer	Heat_1	Heat_2	Heat_3
3	Racer Y	41.33	DQ	38.04
2	Racer X	37.22	38.22	37.61
4	Racer Z	39.27	39.04	38.85

You can sort by the `Racer` column in ascending order to return to the original sort order.

Example - Using sourcerownumber to create unique row identifiers

The following example demonstrates how to unpack nested data. As part of this example, the `SOURCEROWNUMBER` function is used as part of a method to create unique row identifiers.

Source:

You have the following data on student test scores. Scores on individual scores are stored in the `Scores` array, and you need to be able to track each test on a uniquely identifiable row. This example has two goals:

1. One row for each student test
2. Unique identifier for each student-score combination

LastName	FirstName	Scores
Adams	Allen	[81,87,83,79]
Burns	Bonnie	[98,94,92,85]
Cannon	Charles	[88,81,85,78]

Transform:

When the data is imported from CSV format, you must add a `header` transform and remove the quotes from the `Scores` column:

```
header
```

```
replace col:Scores with:'' on:`"` global:true
```

Validate test date: To begin, you might want to check to see if you have the proper number of test scores for each student. You can use the following transform to calculate the difference between the expected number of elements in the `Scores` array (4) and the actual number:

```
derive type:single value: (4 - ARRAYLEN(Scores)) as: 'numMissingTests'
```

When the transform is previewed, you can see in the sample dataset that all tests are included. You might or might not want to include this column in the final dataset, as you might identify missing tests when the recipe is run at scale.

Unique row identifier: The `Scores` array must be broken out into individual rows for each test. However, there is no unique identifier for the row to track individual tests. In theory, you could use the combination of `LastName-FirstName-Scores` values to do so, but if a student recorded the same score twice, your dataset has duplicate rows. In the following transform, you create a parallel array called `Tests`, which contains an index array for the number of values in the `Scores` column. Index values start at `0`:

```
derive type:single value:RANGE(0,ARRAYLEN(Scores)) as:'Tests'
```

Also, we will want to create an identifier for the source row using the `SOURCEROWNUMBER` function:

```
derive type:single value:SOURCEROWNUMBER() as:'orderIndex'
```

One row for each student test: Your data should look like the following:

LastName	FirstName	Scores	Tests	orderIndex
Adams	Allen	[81,87,83,79]	[0,1,2,3]	2
Burns	Bonnie	[98,94,92,85]	[0,1,2,3]	3
Cannon	Charles	[88,81,85,78]	[0,1,2,3]	4

Now, you want to bring together the `Tests` and `Scores` arrays into a single nested array using the `ARRAYZIP` function:

```
derive type:single value:ARRAYZIP([Tests,Scores])
```

Your dataset has been changed:

LastName	FirstName	Scores	Tests	orderIndex	column1
Adams	Allen	[81,87,83,79]	[0,1,2,3]	2	[[0,81],[1,87],[2,83],[3,79]]
Adams	Bonnie	[98,94,92,85]	[0,1,2,3]	3	[[0,98],[1,94],[2,92],[3,85]]
Cannon	Charles	[88,81,85,78]	[0,1,2,3]	4	[[0,88],[1,81],[2,85],[3,78]]

With the `flatten` transform, you can unpack the nested array:

```
flatten col: column1
```

Each test-score combination is now broken out into a separate row. The nested Test-Score combinations must be broken out into separate columns using `unnest`:

```
unnest col:column1 keys:'[0]','[1]'
```

After you delete `column1`, which is no longer needed you should rename the two generated columns:

```
rename mapping:[column_0,'TestNum']
```

```
rename mapping:[column_1,'TestScore']
```

Unique row identifier: You can do one more step to create unique test identifiers, which identify the specific test for each student. The following uses the original row identifier `OrderIndex` as an identifier for the student and the `TestNumber` value to create the `TestId` column value:

```
derive type:single value: (orderIndex * 10) + TestNum as: 'TestId'
```

The above are integer values. To make your identifiers look prettier, you might add the following:

```
merge col:'TestId00','TestId'
```

Extending: You might want to generate some summary statistical information on this dataset. For example, you might be interested in calculating each student's average test score. This step requires figuring out how to properly group the test values. In this case, you cannot group by the `LastName` value, and when executed at scale, there might be collisions between first names when this recipe is run at scale. So, you might need to create a kind of primary key using the following:

```
merge col:'LastName','FirstName' with:'-' as:'studentId'
```

You can now use this as a grouping parameter for your calculation:

```
derive type:single value:AVERAGE(TestScore) group:studentId as:'avg_TestScore'
```

Results:

After you delete unnecessary columns and move your columns around, the dataset should look like the following:

TestId	LastName	FirstName	TestNum	TestScore	studentId	avg_TestScore
TestId0021	Adams	Allen	0	81	Adams-Allen	82.5
TestId0022	Adams	Allen	1	87	Adams-Allen	82.5
TestId0023	Adams	Allen	2	83	Adams-Allen	82.5
TestId0024	Adams	Allen	3	79	Adams-Allen	82.5
TestId0031	Adams	Bonnie	0	98	Adams-Bonnie	92.25
TestId0032	Adams	Bonnie	1	94	Adams-Bonnie	92.25
TestId0033	Adams	Bonnie	2	92	Adams-Bonnie	92.25
TestId0034	Adams	Bonnie	3	85	Adams-Bonnie	92.25
TestId0041	Cannon	Chris	0	88	Cannon-Chris	83
TestId0042	Cannon	Chris	1	81	Cannon-Chris	83
TestId0043	Cannon	Chris	2	85	Cannon-Chris	83
TestId0044	Cannon	Chris	3	78	Cannon-Chris	83

Example - Delete rows based on source row numbers

Source:

Your dataset is the following set of orders.

CustId	FirstName	LastName	City	State	LastOrder
1001	Skip	Jones	San Francisco	CA	25
1002	Adam	Allen	Oakland	CA	1099
1003	David	Wiggins	Oakland	MI	125.25
1004	Amanda	Green	Detroit	MI	452.5

1005	Colonel	Mustard	Los Angeles	CA	950
1006	Pauline	Hall	Sagninaw	MI	432.22
1007	Sarah	Miller	Cheyenne	WY	724.22
1008	Teddy	Smith	Juneau	AK	852.11
1009	Joelle	Higgins	Sacramento	CA	100

Transform:

Initially, you want to review your list of orders by last name.

```
sort order:LastName
```

During your review, you notice that two customer orders are no longer valid and need to be removed. They are:

- LastName: Hall
- LastName: Jones

You might hover over the left side of the screen to reveal the row numbers. You select the row numbers for each of these rows, and a delete suggestion is provided for you. When you click **Modify**, you see the following transform:

```
delete row: IN(SOURCEROWNUMBER(), [2,7])
```

The above checks the results of the `SOURCEROWNUMBER` function, which returns the original row order for the selected rows. If a selected row matches values in the `[2,7]` array of row numbers, then the row is deleted.

Results:

When the preceding transform is added, your dataset looks like the following, and your sort order is maintained:

Source:

CustId	FirstName	LastName	City	State	LastOrder
1002	Adam	Allen	Oakland	CA	1099
1004	Amanda	Green	Detroit	MI	452.5
1009	Joelle	Higgins	Sacramento	CA	100
1007	Sarah	Miller	Cheyenne	WY	724.22
1005	Colonel	Mustard	Los Angeles	CA	950
1008	Teddy	Smith	Juneau	AK	852.11
1003	David	Wiggins	Oakland	MI	125.25