Flatten Transform

Contents:

> **NOTE:** Transforms are a part of the underlying language, which is not directly accessible to users. This content is maintained for reference purposes only. For more information on the user-accessible equivalent to transforms, see *Transformation Reference*.

Unpacks array data into separate rows for each value. This transform operates on a single column.

This transform does not reference keys in the array. If your array data contains keys, use the `unnest` transform. See *Unnest Transform*.

Basic Usage

```
flatten col: myArray
```

Output: Generates a separate row for each value in the array. Values of other columns in generated rows are copied from the source.

Syntax and Parameters

```
flatten: col: column_ref
```

Token	Required?	Data Type	Description
flatten	Y	transform	Name of the transform
col	Y	string	Source column name

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

col

Identifies the column to which to apply the transform. You can specify only one column.

Usage Notes:

Required?	Data Type
Yes	String (column name)

Examples

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

Example - Flatten an array

In this example, the source data includes an array of scores that need to broken out into separate rows.

Source:

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

Transformation:

When the data is imported, you might have to re-type the `Scores` column as an array:

Transformation Name	`Change column data type`
Parameter: Columns	`Scores`
Parameter: New type	`Array`

You can now flatten the `Scores` column data into separate rows:

Transformation Name	`Expand Array into rows`
Parameter: Column	`Scores`

Results:

LastName	FirstName	Scores
Adams	Allen	81
Adams	Allen	87
Adams	Allen	83
Adams	Allen	79
Burns	Bonnie	98
Burns	Bonnie	94
Burns	Bonnie	92
Burns	Bonnie	85
Cannon	Chris	88
Cannon	Chris	81
Cannon	Chris	85
Cannon	Chris	78

This example is extended below.

Example - Flatten and unnest together

While the above example nicely flattens out your data, there are two potential problems with the results:

- There is no identifier for each test. For example, Allen Adams' score of 87 cannot be associated with the specific test on which he recorded the score.
- There is no unique identifier for each row.

The following example addresses both of these issues. It also demonstrates differences between the `unnest` and the `flatten` transform, including how you use `unnest` to flatten array data based on specified keys.

- For more information, see *Unnest Transform*.

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]

Transformation:

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

Transformation Name	`Rename column with row(s)`
Parameter: Option	`Use row(s) as column names`
Parameter: Type	`Use a single row to name columns`
Parameter: Row number	`1`

Transformation Name	`Replace text or pattern`
Parameter: Column	`colScores`
Parameter: Find	`'\"'`
Parameter: Replace with	`''`
Parameter: Match all occurrences	`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:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`(4 - arraylen(Scores))`
Parameter: New column name	`'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`:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`range(0,arraylen(Scores))`
Parameter: New column name	`'Tests'`

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

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`sourcerownumber()`
Parameter: New column name	`'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` fun ction:

Transformation Name	`New formula`

Parameter: Formula type	`Single row formula`
Parameter: Formula	`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]]

Use the following to unpack the nested array:

Transformation Name	`Expand arrays to rows`
Parameter: Column	`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 the following:

Transformation Name	`Unnest Objects into columns`
Parameter: Column	`column1`
Parameter: Paths to elements	`'[0]','[1]'`

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

Transformation Name	`Rename columns`
Parameter: Option	`Manual rename`
Parameter: Column	`column_0`
Parameter: New column name	`'TestNum'`

Transformation Name	`Rename columns`
Parameter: Option	`Manual rename`
Parameter: Column	`column_1`
Parameter: New column name	`'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:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`(orderIndex * 10) + TestNum`

Parameter: New column name	`'TestId'`

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

Transformation Name	`Merge columns`
Parameter: Columns	`'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:

Transformation Name	`Merge columns`
Parameter: Columns	`'LastName','FirstName'`
Parameter: Separator	`'-'`
Parameter: New column name	`'studentId'`

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

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`average(TestScore)`
Parameter: Group rows by	`studentId`
Parameter: New column name	`'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