ISMISSING Function

The `ISMISSING` function tests whether a column of values is missing or null. For input column references, this function returns `true` or `false`.

- You can define a conditional test in a single step for valid values. See *IFMISSING Function*.
- Missing values are different from null values. To test for the presence of null values exclusively, see *ISNULL Function*.

Basic Usage

```
delete row:ISMISSING(Qty)
```

Output: Deletes any row in which the value in the `Qty` column is missing.

Syntax and Arguments

```
delete value:ISMISSING(column_string)
```

Argument	Required?	Data Type	Description
column_string	Y	string	Name of column or string literal to be applied to the function

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

column_string

Name of the column or string literal to be tested for missing values.

- Missing literals or column values generate missing string results.

Multiple columns can be specified as an array, as in the following:

```
delete value:ISMISSING([col1,col3,col5])
```

- Column ranges are not supported.
- Wildcards are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	String literal or column reference	`myColumn`

Examples

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

Example - Type check functions

This example illustrates how various type checking functions can be applied to your data.

- `ISVALID` - Returns `true` if the input matches the specified data type. See *VALID Function*.
- `ISMISMATCHED` - Returns `true` if the input does not match the specified data type. See *ISMISMATCHED Function*.
- `ISMISSING` - Returns `true` if the input value is missing. See *ISMISSING Function*.
- `ISNULL` - Returns `true` if the input value is null. See *ISNULL Function*.
- `NULL` - Generates a null value. See *NULL Function*.

Source:

Some source values that should match the State and Integer data types:

State	Qty
CA	10
OR	-10
WA	2.5
ZZ	15
ID	
	4

Transform:

You can test for invalid values for State using the following:

```
derive type:single value: ISMISMATCHED (State, 'State')
```

You can test for valid matches for Qty using the following:

```
derive type:single value: (ISVALID (Qty, 'Integer') && (Qty > 0)) as:'valid_Qty'
```

The first transform flags rows 4 and 6 as mismatched.

NOTE: A missing value is not valid for a type, including String type.

The second transform flags as valid all rows where the `Qty` column is a valid integer that is greater than zero.

The following transform tests for the presence of missing values in either column:

```
derive type:single value: (ISMISSING(State) || ISMISSING(Qty)) as:'missing_State_Qty'
```

After re-organizing the columns using the `move` transform, the dataset should now look like the following:

State	Qty	mismatched_State	valid_Qty	missing_State_Qty
CA	10	false	true	false
OR	-10	false	false	false
WA	2.5	false	false	false
ZZ	15	true	true	false
ID		false	false	true
	4	false	true	true

Since the data does not contain null values, the following transform generates null values based on the preceding criteria:

```
derive type:single value: ((mismatched_State == 'true') || (valid_Qty == 'false') ||
(missing_State_Qty == 'true')) ? NULL() : 'ok' as:'status'
```

You can then use the `ISNULL` check to remove the rows that fail the above test:

```
delete row: ISNULL('status')
```

Results:

Based on the above tests, the output dataset contains one row:

State	Qty	mismatched_State	valid_Qty	missing_State_Qty	status
CA	10	false	true	false	ok