VALID Function

Contents:

Tests whether a set of values is valid for a specified data type and is not a null value.

- For a specified data type and set of values, this function returns `true` or `false`.
- Inputs can be literal values or column references.

You can use the `ISVALID` function keywords interchangeably.

- You can define a conditional test in a single step for valid values. See *IFVALID Function*.
- This function is similar to the `ISMISMATCHED` function, which tests for mismatches against a specified data type. However, the `ISMISMATCHED` function also matches against missing values, while the `ISVALID` function does not. See *ISMISMATCHED Function*.

Basic Usage

Column reference example:

```
keep row:(ISVALID(Qty, 'Integer') && (Qty > 0))
```

Output: Keeps any row in which the value in the `Qty` column contains a valid Integer and the value is greater than zero.

Numeric literal example:

```
derive type:single value: ISVALID('ZZ', 'State')
```

Output: Generates a new column containing `false`, since the value ZZ is not a valid U.S. State code.

Syntax and Arguments

```
derive type:single value:ISVALID(column_string,datatype_literal)
```

Argument	Required?	Data Type	Description
column_string	Y	string	Name of column or string literal to be applied to the function
datatype_literal	Y	string	String literal that identifies the data type against which to validate the source values

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

column_string

Name of the column or string literal to be evaluated for validity.

- Missing literals or column values generate missing string results.
    - Constants must be quoted (`'Hello, World'`).
- Multiple columns and wildcards are not supported.

Usage Notes:

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

datatype_literal

Literal value for data type to which to match the source column or string. For more information, see *Valid Data Type Strings*.

- Column references are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	String literal	`'Integer'`

Valid data type strings:

When referencing a data type within a transform, you can use the following strings to identify each type:

NOTE: In Wrangle transforms, these values are case-sensitive.

NOTE: When specifying a data type by name, you must use the String value listed below. The Data Type value is the display name for the type.

Data Type	String
String	`'String'`
Integer	`'Integer'`
Decimal	`'Float'`
Boolean	`'Bool'`
Social Security Number	`'SSN'`
Phone Number	`'Phone'`
Email Address	`'Emailaddress'`
Credit Card	`'Creditcard'`
Gender	`'Gender'`
Object	`'Map'`
Array	`'Array'`
IP Address	`'Ipaddress'`

URL	`'Url'`
HTTP Code	`'Httpcodes'`
Zip Code	`'Zipcode'`
State	`'State'`
Date / Time	`'Datetime'`

For custom types, you should reference the name of the type in the string value. For more information, see *Create Custom Data Types*.

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