PARSEINT Function

Evaluates a String input against the Integer datatype. If the input matches, the function outputs an Integer value. Input can be a literal, a column of values, or a function returning String values.

After you have converted your strings values to integers, if a sufficient percentage of input strings from a column are successfully converted to the other date type, the column may be retyped.

Wrangle vs. SQL: This function is part of Wrangle , a proprietary data transformation language. Wrangle is not SQL. For more information, see *Wrangle Language*.

Basic Usage

```
parseint(strInput)
```

Output: Returns the Integer data type value for `strInput` String values.

Syntax and Arguments

```
parseint(str_input)
```

Argument	Required?	Data Type	Description
str_input	Y	String	Literal, name of a column, or a function returning String values to match

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

str_input

Literal, column name, or function returning String values that are to be evaluated for conversion to Integer values.

- Missing values for this function in the source data result in null values in the output.
- Multiple columns and wildcards are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	String	`'5'`

Examples

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

Example - type parsing functions

This example shows how to use the following parsing functions for evaluating input against the function-specific data type:

- PARSEBOOL - If the input String value is a valid Boolean value, the value is returned as a Boolean data type value. See *PARSEBOOL Function*.
- PARSEDATE - If the input String value is valid against the specified or default Datetime formats, the value is returned as a Datetime value. See *PARSEDATE Function*.
- PARSEFLOAT - If the input String value is a valid Float (Decimal) value, the value is returned as a Decimal data type value. See *PARSEFLOAT Function*.
- PARSEINT - If the input String value is a valid Integer value, the value is returned as an Integer data type value. See *PARSEINT Function*.

Source:

The following table contains data on a series of races.

raceId	disqualified	date	racerId	time_sc
1	FALSE	2/1/20	1	24.22
2	f	2/8/20	1	25
3	no	2/8/20	1	24.11
4	n	1-Feb-20	2	26.1
5	TRUE	8-Feb-20	2.2	-25.22
6	t	2/8/2020 10:16:00 AM	2	25.44
7	yes	2/1/20	3	24
8	y	2/8/20	33	29.22
9	0	2/8/20	3	24.78
10	1	1-Feb-20	4	26.2.1
11	FALSE	8-Feb-20		28.22 sec
12	FALSE	2/8/2020 10:16:00 AM	4	27.11

As you can see, this dataset has variation in values (FALSE, f, no, n) and problems with the data.

Transformation:

When the data is first imported, it may be properly typed for each column. To use the parsing functions, these columns should be converted to String data type:

Transformation Name	Change column data type
Parameter: Columns	disqualified,date,racerId,time_sc
Parameter: New type	String

Now, you can parse individual columns.

disqualified column:

Transformation Name	Edit column with formula

Parameter: Columns	`disqualified`
Parameter: Formula	`PARSEBOOL($col)`

racerId column:

Transformation Name	`Edit column with formula`
Parameter: Columns	`racerId`
Parameter: Formula	`PARSEINT($col)`

time_sc column:

Transformation Name	`Edit column with formula`
Parameter: Columns	`time_sc`
Parameter: Formula	`PARSEFLOAT($col)`

date column:

For the date column, the PARSEDATE function supports a default set of Datetime formats. Since some of the listed formats are different from these defaults, you must specify all of the formats. These formats are specified as an array of string values as the second argument of the function:

> **Tip:** For the PARSEDATE function, it's useful to use the Preview to verify that all of the dates in the column are represented in the array of output formats. You can see the available output formats through the data type menu at the top of a column. See *Choose Datetime Format Dialog*.

Transformation Name	`Edit column with formula`
Parameter: Columns	`date`
Parameter: Formula	`PARSEDATE($col, ['yyyy-MM-dd','yyyy\/MM\/dd','M\/d\/yyy hh:mm','MMMM d, yyyy','MMM d, yyyy'])`

After all of the date values have been standardized to the output format of the PARSEDATE function, you may choose to remove the time element of the values:

Transformation Name	`Replace text or pattern`
Parameter: Column	`date`
Parameter: Find	`` ` {digit}{2}:{digit}{2}:{digit}{2}{end}` ``
Parameter: Replace with	`''`

Results:

After executing the above steps, the data appears as follows. Notes on each column's output are below the table.

raceId	disqualified	date	racerId	time_sc
1	false	2020-02-01	1	24.22

2	false	2020-02-08	1	25
3	false	2020-02-08	1	24.11
4	false	2020-02-01	2	26.1
5	true	2020-02-08	*null*	-25.22
6	true	2020-02-08	2	25.44
7	true	2020-02-01	3	24
8	true	2020-02-08	33	29.22
9	false	2020-02-08	3	24.78
10	true	2020-02-01	4	*null*
11	false	2020-02-08	*null*	*null*
12	false	2020-02-08	4	27.11

disqualified column:

- The PARSEBOOL function normalizes all valid Boolean values to either `false` or `true`.

racerId column:

- The PARSEINT function writes invalid values as null values.
- The function writes empty values as null values.
- The value 33 remains, since it is a valid Integer. This value should be fixed manually.

time_sc:

- The PARSEFLOAT function writes the source value `25.00` as `25` in output.
- The source value `-25.22` remains. However, since this is time-based data, it needs to be fixed.
- Invalid values are written as nulls.

date column:

- All values are written in the standardized format: `yyyy-MM-dd HH:mm:ss`. Time data has been stripped.