between operator

Article • 03/12/2023

Filters a record set for data matching the values in an inclusive range.

`between` can operate on any numeric, datetime, or timespan expression.

Syntax

T `|` `where` *expr* `between` `(`*leftRange*`..`*rightRange*`)`

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input whose records are to be matched. For example, the table name.
expr	scalar	✓	The expression used to filter.
leftRange	int, long, real, or datetime	✓	The expression of the left range. The range is inclusive.
rightRange	int, long, real, datetime, or timespan	✓	The expression of the right range. The range is inclusive. This value can only be of type timespan if *expr* and *leftRange* are both of type `datetime`. See example.

Returns

Rows in *T* for which the predicate of (*expr* >= *leftRange* and *expr* <= *rightRange*) evaluates to `true`.

Examples

Filter numeric values

Run the query

```
```

Kusto

```
range x from 1 to 100 step 1
| where x between (50 .. 55)
```

Output

x
50
51
52
53
54
55

Filter datetime

Run the query

Kusto

```
StormEvents
| where StartTime between (datetime(2007-07-27) .. datetime(2007-07-30))
| count
```

Output

Count
476

Filter datetime using a timespan range

Run the query

Kusto

```
StormEvents
| where StartTime between (datetime(2007-07-27) .. 3d)
```

```
| count
```

Output

Count
476

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

!between operator

Article • 03/12/2023

Matches the input that is outside of the inclusive range.

`!between` can operate on any numeric, datetime, or timespan expression.

Syntax

T `|` `where` *expr* `!between` `(` *leftRange* `..` *rightRange* `)`

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input whose records are to be matched.
expr	scalar	✓	The expression to filter.
leftRange	int, long, real, or datetime	✓	The expression of the left range. The range is inclusive.
rightRange	int, long, real, datetime, or timespan	✓	The expression of the right range. The range is inclusive. This value can only be of type timespan if *expr* and *leftRange* are both of type `datetime`. See example.

Returns

Rows in *T* for which the predicate of (*expr* < *leftRange* or *expr* > *rightRange*) evaluates to `true`.

Examples

Filter numeric values

Run the query

Kusto

```
range x from 1 to 10 step 1
| where x !between (5 .. 9)
```

Output

x
1
2
3
4
10

Filter datetime

Run the query

Kusto

```
StormEvents
| where StartTime !between (datetime(2007-07-27) .. datetime(2007-07-30))
| count
```

Output

Count
58590

Filter datetime using a timespan range

Run the query

Kusto

```
StormEvents
| where StartTime !between (datetime(2007-07-27) .. 3d)
| count
```

Output

Count
58590

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Bitwise (binary) operators

Article • 04/11/2023

Kusto support several bitwise (binary) operators between integers:

- binary_and
- binary_not
- binary_or
- binary_shift_left
- binary_shift_right
- binary_xor

Feedback

Datetime / timespan arithmetic

Article • 03/23/2023

Kusto supports performing arithmetic operations on values of types `datetime` and `timespan`.

Supported operations

- One can subtract (but not add) two `datetime` values to get a `timespan` value expressing their difference. For example, `datetime(1997-06-25) - datetime(1910-06-11)` is how old was Jacques-Yves Cousteau when he died.
- One can add or subtract two `timespan` values to get a `timespan` value which is their sum or difference. For example, `1d + 2d` is three days.
- One can add or subtract a `timespan` value from a `datetime` value. For example, `datetime(1910-06-11) + 1d` is the date Cousteau turned one day old.
- One can divide two `timespan` values to get their quotient. For example, `1d / 5h` gives `4.8`. This gives one the ability to express any `timespan` value as a multiple of another `timespan` value. For example, to express an hour in seconds, simply divide `1h` by `1s`: `1h / 1s` (with the obvious result, `3600`).
- Conversely, one can multiple a numeric value (such as `double` and `long`) by a `timespan` value to get a `timespan` value. For example, one can express an hour and a half as `1.5 * 1h`.

Examples

Unix time, which is also known as POSIX time or UNIX Epoch time, is a system for describing a point in time as the number of seconds that have elapsed since 00:00:00 Thursday, 1 January 1970, Coordinated Universal Time (UTC), minus leap seconds.

If your data includes representation of Unix time as an integer, or you require converting to it, the following functions are available.

From Unix time

Run the query

Kusto

```
let fromUnixTime = (t: long) {
    datetime(1970-01-01) + t * 1sec
};
print result = fromUnixTime(1546897531)
```

Output

result
2019-01-07 21:45:31.0000000

To Unix time

Run the query

Kusto

```
let toUnixTime = (dt: datetime) {
    (dt - datetime(1970-01-01)) / 1s
};
print result = toUnixTime(datetime(2019-01-07 21:45:31.0000000))
```

Output

result
1546897531

See also

For unix-epoch time conversions, see the following functions:

- unixtime_seconds_todatetime()
- unixtime_milliseconds_todatetime()
- unixtime_microseconds_todatetime()
- unixtime_nanoseconds_todatetime()

Feedback

in operator

Article • 03/29/2023

Filters a record set for data with a case-sensitive string.

The following table provides a comparison of the `in` operators:

Operator	Description	Case-Sensitive	Example (yields `true`)
in	Equals to one of the elements	Yes	`"abc" in ("123", "345", "abc")`
!in	Not equals to any of the elements	Yes	`"bca" !in ("123", "345", "abc")`
in~	Equals to any of the elements	No	`"Abc" in~ ("123", "345", "abc")`
!in~	Not equals to any of the elements	No	`"bCa" !in~ ("123", "345", "ABC")`

> ⓘ Note
>
> Nested arrays are flattened into a single list of values. For example, `x in (dynamic([1,[2,3]]))` becomes `x in (1,2,3)`.

For further information about other operators and to determine which operator is most appropriate for your query, see datatype string operators.

Case-insensitive operators are currently supported only for ASCII-text. For non-ASCII comparison, use the tolower() function.

Performance tips

> ⓘ Note
>
> Performance depends on the type of search and the structure of the data. For best practices, see **Query best practices**.

Syntax

T `|` `where` *col* `in` `(`*expression*`,` ... `)`

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input to filter.
col	string	✓	The column by which to filter.
expression	scalar or tabular	✓	An expression that specifies the values for which to search. the values for which to search. Each expression can be a scalar value or a tabular expression that produces a set of values. If a tabular expression has multiple columns, the first column is used. The search will consider up to 1,000,000 distinct values.

> ⓘ Note
>
> An inline tabular expression must be enclosed with double parentheses. See example.

Returns

Rows in *T* for which the predicate is `true`.

Examples

List of scalars

The following query shows how to use `in` with a list of scalar values.

Run the query

Kusto

```
StormEvents
| where State in ("FLORIDA", "GEORGIA", "NEW YORK")
| count
```

Output

Count
4775

Dynamic array

The following query shows how to use `in` with a dynamic array.

Run the query

Kusto

```
let states = dynamic(['FLORIDA', 'ATLANTIC SOUTH', 'GEORGIA']);
StormEvents
| where State in (states)
| count
```

Output

Count
3218

Tabular expression

The following query shows how to use `in` with a tabular expression.

Run the query

Kusto

```
let Top_5_States =
    StormEvents
    | summarize count() by State
    | top 5 by count_;
StormEvents
| where State in (Top_5_States)
| count
```

The same query can be written with an inline tabular expression statement. Notice that an inline tabular expression must be enclosed with double parentheses.

Run the query

Kusto

```
StormEvents
| where State in ((
    StormEvents
    | summarize count() by State
    | top 5 by count_
    ))
| count
```

Output

Count
14242

Top with other example

Run the query

Kusto

```
let Lightning_By_State = materialize(StormEvents
    | summarize lightning_events = countif(EventType == 'Lightning') by
State);
let Top_5_States = Lightning_By_State | top 5 by lightning_events | project
State;
Lightning_By_State
| extend State = iff(State in (Top_5_States), State, "Other")
| summarize sum(lightning_events) by State
```

Output

State	sum_lightning_events
ALABAMA	29
WISCONSIN	31
TEXAS	55
FLORIDA	85
GEORGIA	106
Other	415

Use a static list returned by a function

Run the query

```
Kusto

StormEvents
| where State in (InterestingStates())
| count
```

Output

Count
4775

The function definition.

Run the query

```
Kusto

.show function InterestingStates
```

Output

Name	Parameters	Body	Folder	DocString
InterestingStates	()	{ dynamic(["WASHINGTON", "FLORIDA", "GEORGIA", "NEW YORK"]) }		

Feedback

in~ operator

Article • 04/03/2023

Filters a record set for data with a case-insensitive string.

The following table provides a comparison of the `in` operators:

Operator	Description	Case-Sensitive	Example (yields `true`)
in	Equals to one of the elements	Yes	`"abc" in ("123", "345", "abc")`
!in	Not equals to any of the elements	Yes	`"bca" !in ("123", "345", "abc")`
in~	Equals to any of the elements	No	`"Abc" in~ ("123", "345", "abc")`
!in~	Not equals to any of the elements	No	`"bCa" !in~ ("123", "345", "ABC")`

> ⓘ Note
>
> Nested arrays are flattened into a single list of values. For example, `x in (dynamic([1,[2,3]]))` becomes `x in (1,2,3)`.

For further information about other operators and to determine which operator is most appropriate for your query, see datatype string operators.

Case-insensitive operators are currently supported only for ASCII-text. For non-ASCII comparison, use the tolower() function.

Performance tips

> ⓘ Note
>
> Performance depends on the type of search and the structure of the data. For best practices, see **Query best practices**.

When possible, use the case-sensitive in.

Syntax

T `|` `where` *col* `in~` `(`*expression*`,` ... `)`

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input to filter.
col	string	✓	The column by which to filter.
expression	scalar or tabular	✓	An expression that specifies the values for which to search. Each expression can be a scalar value or a tabular expression that produces a set of values. If a tabular expression has multiple columns, the first column is used. The search will consider up to 1,000,000 distinct values.

> ⓘ **Note**
>
> An inline tabular expression must be enclosed with double parentheses. See **example**.

Returns

Rows in *T* for which the predicate is `true`.

Examples

List of scalars

The following query shows how to use `in~` with a comma-separated list of scalar values.

Run the query

Kusto

```
StormEvents
| where State in~ ("FLORIDA", "georgia", "NEW YORK")
| count
```

Output

Count
4775

Dynamic array

The following query shows how to use `in~` with a dynamic array.

Run the query

Kusto

```
StormEvents
| where State in~ (dynamic(["FLORIDA", "georgia", "NEW YORK"]))
| count
```

Output

Count
4775

The same query can also be written with a let statement.

Run the query

Kusto

```
let states = dynamic(["FLORIDA", "georgia", "NEW YORK"]);
StormEvents
| where State has_any (states)
| summarize count() by State
```

Output

Count
4775

Tabular expression

The following query shows how to use `in~` with an inline tabular expression. Notice that an inline tabular expression must be enclosed with double parentheses.

Run the query

Kusto

```
StormEvents
| where State in~ ((PopulationData | where Population > 5000000 | project
State))
| summarize count() by State
```

Output

State	count_
TEXAS	4701
ILLINOIS	2022
MISSOURI	2016
GEORGIA	1983
MINNESOTA	1881
...	...

The same query can also be written with a let statement. Notice that the double parentheses as provided in the last example aren't necessary in this case.

Run the query

Kusto

```
let large_states = PopulationData | where Population > 5000000 | project
State;
StormEvents
| where State in~ (large_states)
| summarize count() by State
```

Output

State	count_
TEXAS	4701

State	count_
ILLINOIS	2022
MISSOURI	2016
GEORGIA	1983
MINNESOTA	1881
...	...

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

!in operator

Article • 04/03/2023

Filters a record set for data without a case-sensitive string.

The following table provides a comparison of the `in` operators:

Operator	Description	Case-Sensitive	Example (yields `true`)
in	Equals to one of the elements	Yes	`"abc" in ("123", "345", "abc")`
!in	Not equals to any of the elements	Yes	`"bca" !in ("123", "345", "abc")`
in~	Equals to any of the elements	No	`"Abc" in~ ("123", "345", "abc")`
!in~	Not equals to any of the elements	No	`"bCa" !in~ ("123", "345", "ABC")`

> ⓘ Note
>
> Nested arrays are flattened into a single list of values. For example, `x in (dynamic([1,[2,3]]))` becomes `x in (1,2,3)`.

For further information about other operators and to determine which operator is most appropriate for your query, see datatype string operators.

Case-insensitive operators are currently supported only for ASCII-text. For non-ASCII comparison, use the tolower() function.

Performance tips

> ⓘ Note
>
> Performance depends on the type of search and the structure of the data. For best practices, see **Query best practices**.

Syntax

T `|` `where` *col* `!in` `(` *expression* `,` `...` `)`

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input to filter.
col	string	✓	The column by which to filter.
expression	scalar or tabular	✓	An expression that specifies the values for which to search. Each expression can be a scalar value or a tabular expression that produces a set of values. If a tabular expression has multiple columns, the first column is used. The search will consider up to 1,000,000 distinct values.

> ⓘ Note
>
> An inline tabular expression must be enclosed with double parentheses. See example.

Returns

Rows in *T* for which the predicate is `true`.

Example

List of scalars

The following query shows how to use `!in` with a comma-separated list of scalar values.

Run the query

Kusto

```
StormEvents
| where State !in ("FLORIDA", "GEORGIA", "NEW YORK")
| count
```

Output

Count
54291

Dynamic array

The following query shows how to use `!in` with a dynamic array.

Run the query

Kusto

```
StormEvents
| where State !in (dynamic(["FLORIDA", "GEORGIA", "NEW YORK"]))
| count
```

Output

Count
54291

The same query can also be written with a let statement.

Run the query

Kusto

```
let states = dynamic(["FLORIDA", "GEORGIA", "NEW YORK"]);
StormEvents
| where State !in (states)
| summarize count() by State
```

Output

Count
54291

Tabular expression

The following query shows how to use `!in` with an inline tabular expression. Notice that an inline tabular expression must be enclosed with double parentheses.

Run the query

```kusto
StormEvents
| where State !in ((PopulationData | where Population > 5000000 | project State))
| summarize count() by State
```

Output

State	Count
KANSAS	3166
IOWA	2337
NEBRASKA	1766
OKLAHOMA	1716
SOUTH DAKOTA	1567
...	...

The same query can also be written with a let statement. Notice that the double parentheses as provided in the last example aren't necessary in this case.

Run the query

```kusto
let large_states = PopulationData | where Population > 5000000 | project State;
StormEvents
| where State !in (large_states)
| summarize count() by State
```

Output

State	Count
KANSAS	3166
IOWA	2337
NEBRASKA	1766

State	Count
OKLAHOMA	1716
SOUTH DAKOTA	1567
...	...

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

!in~ operator

Article • 03/29/2023

Filters a record set for data without a case-insensitive string.

The following table provides a comparison of the `in` operators:

Operator	Description	Case-Sensitive	Example (yields `true`)
in	Equals to one of the elements	Yes	`"abc" in ("123", "345", "abc")`
!in	Not equals to any of the elements	Yes	`"bca" !in ("123", "345", "abc")`
in~	Equals to any of the elements	No	`"Abc" in~ ("123", "345", "abc")`
!in~	Not equals to any of the elements	No	`"bCa" !in~ ("123", "345", "ABC")`

> ⓘ Note
>
> Nested arrays are flattened into a single list of values. For example, `x in (dynamic([1,[2,3]]))` becomes `x in (1,2,3)`.

For further information about other operators and to determine which operator is most appropriate for your query, see datatype string operators.

Case-insensitive operators are currently supported only for ASCII-text. For non-ASCII comparison, use the tolower() function.

Performance tips

> ⓘ Note
>
> Performance depends on the type of search and the structure of the data. For best practices, see **Query best practices**.

When possible, use the case-sensitive !in~.

Syntax

T `|` `where` *col* `!in~` `(` *expression* `,` `...` `)`

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input to filter.
col	string	✓	The column by which to filter.
expression	scalar or tabular	✓	An expression that specifies the values for which to search. Each expression can be a scalar value or a tabular expression that produces a set of values. If a tabular expression has multiple columns, the first column is used. The search will consider up to 1,000,000 distinct values.

> ⓘ Note
>
> An inline tabular expression must be enclosed with double parentheses. See example.

Returns

Rows in *T* for which the predicate is `true`.

Example

List of scalars

The following query shows how to use `!in~` with a comma-separated list of scalar values.

Run the query

Kusto

```
StormEvents
| where State !in~ ("Florida", "Georgia", "New York")
| count
```

Output

Count
54,291

Dynamic array

The following query shows how to use `!in~` with a dynamic array.

Run the query

Kusto

```
StormEvents
| where State !in~ (dynamic(["Florida", "Georgia", "New York"]))
| count
```

Output

Count
54291

The same query can also be written with a let statement.

Run the query

Kusto

```
let states = dynamic(["Florida", "Georgia", "New York"]);
StormEvents
| where State !in~ (states)
| summarize count() by State
```

Output

Count
54291

Tabular expression

The following query shows how to use `!in~` with an inline tabular expression. Notice that an inline tabular expression must be enclosed with double parentheses.

Run the query

Kusto

```
StormEvents
| where State !in~ ((PopulationData | where Population > 5000000 | project
State))
| summarize count() by State
```

Output

State	count_
KANSAS	3166
IOWA	2337
NEBRASKA	1766
OKLAHOMA	1716
SOUTH DAKOTA	1567
...	...

The same query can also be written with a let statement. Notice that the double parentheses as provided in the last example aren't necessary in this case.

Run the query

Kusto

```
let large_states = PopulationData | where Population > 5000000 | project
State;
StormEvents
| where State !in~ (large_states)
| summarize count() by State
```

Output

State	count_
KANSAS	3166

State	count_
IOWA	2337
NEBRASKA	1766
OKLAHOMA	1716
SOUTH DAKOTA	1567
...	...

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Logical (binary) operators

Article • 01/10/2023

The following logical operators are supported between two values of the `bool` type:

> **Note**
>
> These logical operators are sometimes referred-to as Boolean operators, and sometimes as binary operators. The names are all synonyms.

Operator name	Syntax	Meaning
Equality	`==`	Yields `true` if both operands are non-null and equal to each other. Otherwise, `false`.
Inequality	`!=`	Yields `true` if any of the operands are null, or if the operands aren't equal to each other. Otherwise, `false`.
Logical and	`and`	Yields `true` if both operands are `true`.
Logical or	`or`	Yields `true` if one of the operands is `true`, regardless of the other operand.

> **Note**
>
> Due to the behavior of the Boolean null value `bool(null)`, two Boolean null values are neither equal nor non-equal (in other words, `bool(null) == bool(null)` and `bool(null) != bool(null)` both yield the value `false`).
>
> On the other hand, `and`/`or` treat the null value as equivalent to `false`, so `bool(null) or true` is `true`, and `bool(null) and true` is `false`.

Feedback

Numerical operators

Article • 05/23/2023

The types `int`, `long`, and `real` represent numerical types. The following operators can be used between pairs of these types:

Operator	Description	Example
`+`	Add	`3.14 + 3.14`, `ago(5m) + 5m`
`-`	Subtract	`0.23 - 0.22`,
`*`	Multiply	`1s * 5`, `2 * 2`
`/`	Divide	`10m / 1s`, `4 / 2`
`%`	Modulo	`4 % 2`
`<`	Less	`1 < 10`, `10sec < 1h`, `now() < datetime(2100-01-01)`
`>`	Greater	`0.23 > 0.22`, `10min > 1sec`, `now() > ago(1d)`
`==`	Equals	`1 == 1`
`!=`	Not equals	`1 != 0`
`<=`	Less or Equal	`4 <= 5`
`>=`	Greater or Equal	`5 >= 4`
`in`	Equals to one of the elements	see here
`!in`	Not equals to any of the elements	see here

> ⓘ Note
>
> To convert from one numerical type to another, use `to*()` functions. For example, see **tolong()** and **toint()**.

Type rules for arithmetic operations

The data type of the result of an arithmetic operation is determined by the data types of the operands. If one of the operands is of type `real`, the result will be of type `real`. If

both operands are of type `int`, the result will also be of type `int`.

Due to these rules, the result of division operations that only involve integers will be truncated to an integer, which may not always be what you want. To avoid truncation, convert at least one of the `int` values to `real` using the real() function before performing the operation.

The following examples illustrate how the operand types affect the result type in division operations.

Operation	Result	Description
`1.0 / 2`	`0.5`	One of the operands is of type `real`, so the result is `real`.
`1 / 2.0`	`0.5`	One of the operands is of type `real`, so the result is `real`.
`1 / 2`	`0`	Both of the operands are of type `int`, so the result is `int`. Integer division occurs and the decimal is truncated, resulting in `0` instead of `0.5`, as one might expect.
`real(1) / 2`	`0.5`	To avoid truncation due to integer division, one of the `int` operands was first converted to `real` using the `real()` function.

Comment about the modulo operator

The modulo of two numbers always returns in Kusto a "small non-negative number". Thus, the modulo of two numbers, *N* % *D*, is such that: 0 ≤ (*N* % *D*) < abs(*D*).

For example, the following query:

Kusto

```
print plusPlus = 14 % 12, minusPlus = -14 % 12, plusMinus = 14 % -12,
minusMinus = -14 % -12
```

Produces this result:

plusPlus	minusPlus	plusMinus	minusMinus
2	10	2	10

Feedback