project operator

Article • 01/26/2023

Select the columns to include, rename or drop, and insert new computed columns.

The order of the columns in the result is specified by the order of the arguments. Only the columns specified in the arguments are included in the result. Any other columns in the input are dropped.

Syntax

T `| project` [*ColumnName* | `(`*ColumnName*[`,`]`)` `=`] *Expression* [`,` ...]

or

T `| project` *ColumnName* [`=` *Expression*] [`,` ...]

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input for which to project certain columns.
ColumnName	string		A column name or comma-separated list of column names to appear in the output.
Expression	string		The scalar expression to perform over the input.

- Either *ColumnName* or *Expression* must be specified.
- If there's no *Expression*, then a column of *ColumnName* must appear in the input.
- If *ColumnName* is omitted, the output column name of *Expression* will be automatically generated.
- If *Expression* returns more than one column, a list of column names can be specified in parentheses. If a list of the column names isn't specified, all *Expression*'s output columns with generated names will be added to the output.

> ⓘ Note
>
> It's not recommended to return a new calculated column with the same name as an existing column in the input.

Returns

A table with columns that were named as arguments. Contains same number of rows as the input table.

Examples

Only show specific columns

Only show the `EventId`, `State`, `EventType` of the `StormEvents` table.

Run the query

Kusto

```
StormEvents
| project EventId, State, EventType
```

The results table shows only the top 10 results.

EventId	State	EventType
61032	ATLANTIC SOUTH	Waterspout
60904	FLORIDA	Heavy Rain
60913	FLORIDA	Tornado
64588	GEORGIA	Thunderstorm Wind
68796	MISSISSIPPI	Thunderstorm Wind
68814	MISSISSIPPI	Tornado
68834	MISSISSIPPI	Thunderstorm Wind
68846	MISSISSIPPI	Hail
73241	AMERICAN SAMOA	Flash Flood
64725	KENTUCKY	Flood
...	...	...

Potential manipulations using project

The following query renames the `BeginLocation` column and creates a new column called `TotalInjuries` from a calculation over two existing columns.

Run the query

Kusto

```
StormEvents
| project StartLocation = BeginLocation, TotalInjuries = InjuriesDirect +
InjuriesIndirect
| where TotalInjuries > 5
```

The following table shows only the first 10 results.

StartLocation	TotalInjuries
LYDIA	15
ROYAL	15
GOTHENBURG	9
PLAINS	8
KNOXVILLE	9
CAROL STREAM	11
HOLLY	9
RUFFIN	9
ENTERPRISE MUNI ARPT	50
COLLIERVILLE	6
...	...

See also

- extend
- series_stats

Feedback

project-away operator

Article • 01/26/2023

Select what columns from the input table to exclude from the output table.

Syntax

T `| project-away` *ColumnNameOrPattern* [`,` ...]

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input from which to remove columns.
ColumnNameOrPattern	string	✓	One or more column names or column wildcard-patterns to be removed from the output.

Returns

A table with columns that weren't named as arguments. Contains same number of rows as the input table.

> Tip
>
> You can `project-away` any columns that are present in the original table or that were computed as part of the query.

> Note
>
> The order of the columns in the result is determined by their original order in the table. Only the columns that were specified as arguments are dropped. The other columns are included in the result.

Examples

The input table `PopulationData` has 2 columns: `State` and `Population`. Project-away the `Population` column and you're left with a list of state names.

Run the query

Kusto

```
PopulationData
| project-away Population
```

The following table shows only the first 10 results.

State
ALABAMA
ALASKA
ARIZONA
ARKANSAS
CALIFORNIA
COLORADO
CONNECTICUT
DELAWARE
DISTRICT OF COLUMBIA

State
FLORIDA
...

Project-away using a column name pattern

The following query removes columns starting with the word "session".

Run the query

Kusto

```
ConferenceSessions
| project-away session*
```

The following table shows only the first 10 results.

conference	owner	participants	URL	level
PASS Summit 2019	Avner Aharoni		https://www.eventbrite.com/e/near-real-time-interact-analytics-on-big-data-using-azure-data-explorer-fg-tickets-77532775619	
PASS Summit	Rohan Kumar	Ariel Pisetzky	https://www.pass.org/summit/2018/Learn/Keynotes.aspx	
Intelligent Cloud 2019	Rohan Kumar	Henning Rauch		
Ignite 2019	Jie Feng		`https://myignite.techcommunity.microsoft.com/sessions/83940`	100
Ignite 2019	Bernhard Rode	Le Hai Dang, Ricardo Niepel	`https://myignite.techcommunity.microsoft.com/sessions/81596`	200
Ignite 2019	Tzvia Gitlin	Troyna	`https://myignite.techcommunity.microsoft.com/sessions/83933`	400
Ignite 2019	Jie Feng	`https://myignite.techcommunity.microsoft.com/sessions/81057`	300	2019-11 06T20:3
Ignite 2019	Manoj Raheja		`https://myignite.techcommunity.microsoft.com/sessions/83939`	300
Ignite 2019	Uri Barash		`https://myignite.techcommunity.microsoft.com/sessions/81060`	300
Ignite 2018	Manoj Raheja		https://azure.microsoft.com/resources/videos/ignite-2018-azure-data-explorer-%E2%80%93-query-billions-of-records-in-seconds/	200
...	...	...	...	...

See also

- To choose what columns from the input to keep in the output, use project-keep.
- To rename columns, use project-rename.
- To reorder columns, use project-reorder.

Feedback

project-keep operator

Article • 01/26/2023

Select what columns from the input to keep in the output. Only the columns that are specified as arguments will be shown in the result. The other columns are excluded.

Syntax

T `| project-keep` *ColumnNameOrPattern* [`,` ...]

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input from which to keep columns.
ColumnNameOrPattern	string	✓	One or more column names or column wildcard-patterns to be kept in the output.

Returns

A table with columns that were named as arguments. Contains same number of rows as the input table.

> **Tip**
>
> You can `project-keep` any columns that are present in the original table or that were computed as part of the query.

> **Note**
>
> The order of the columns in the result is determined by their original order in the table. Only the columns that were specified as arguments are kept. The other columns are excluded from the result.

Example

The following query returns columns from the `ConferenceSessions` table that contain the word "session".

Run the query

Kusto

```
ConferenceSessions
| project-keep session*
```

The following table shows only the first 10 results.

sessionid	session_title	session_type	session_location
COM64	Focus Group: Azure Data Explorer	Focus Group	Online
COM65	Focus Group: Azure Data Explorer	Focus Group	Online
COM08	Ask the Team: Azure Data Explorer	Ask the Team	Online
COM137	Focus Group: Built-In Dashboard and Smart Auto Scaling Capabilities in Azure Data Explorer	Focus Group	Online
CON-PRT157	Roundtable: Monitoring and managing your Azure Data Explorer deployments	Roundtable	Online
CON-PRT103	Roundtable: Advanced Kusto query language topics	Roundtable	Online
CON-PRT157	Roundtable: Monitoring and managing your Azure Data Explorer deployments	Roundtable	Online
CON-PRT103	Roundtable: Advanced Kusto query language topics	Roundtable	Online
CON-PRT130	Roundtable: Data exploration and visualization with Azure Data Explorer	Roundtable	Online
CON-PRT130	Roundtable: Data exploration and visualization with Azure Data Explorer	Roundtable	Online
...	...	...	...

See also

- To choose what columns from the input to exclude from the output, use project-away.

- To rename columns, use project-rename.
- To reorder columns, use project-reorder.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

project-rename operator

Article • 01/31/2023

Renames columns in the output table.

Syntax

T `| project-rename` *NewColumnName* = *ExistingColumnName* [`,` ...]

Parameters

Name	Type	Required	Description
T	string	✓	The input tabular data.
NewColumnName	string	✓	The new column name.
ExistingColumnName	string	✓	The name of the existing column to rename.

Returns

A table that has the columns in the same order as in an existing table, with columns renamed.

Examples

Run the query

Kusto

```
print a='a', b='b', c='c'
|  project-rename new_b=b, new_a=a
```

Output

new_a	new_b	c
a	b	c

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

project-reorder operator

Article • 01/31/2023

Reorders columns in the output table.

Syntax

T `| project-reorder` *ColumnNameOrPattern* [`asc` | `desc` | `granny-asc` | `granny-desc`] [`,` ...]

Parameters

Name	Type	Required	Description
T	string	✓	The input tabular data.
ColumnNameOrPattern	string	✓	The name of the column or column wildcard pattern by which to order the columns.
`asc`, `desc`, `granny-asc`, `granny-desc`	string		Indicates how to order the columns when a wildcard pattern is used. `asc` or `desc` orders columns by column name in ascending or descending manner, respectively. `granny-asc` or `granny-desc` orders by ascending or descending, respectively, while secondarily sorting by the next numeric value. For example, `a100` comes before `a20` when `granny-asc` is specified.

> ⓘ Note
>
> - If no explicit ordering is specified, the order is determined by the matching columns as they appear in the source table.
> - In ambiguous *ColumnNameOrPattern* matching, the column appears in the first position matching the pattern.
> - Specifying columns for the `project-reorder` is optional. Columns that aren't specified explicitly appear as the last columns of the output table.
> - To remove columns, use **project-away**.
> - To choose which columns to keep, use **project-keep**.
> - To rename columns, use **project-rename**.

Returns

A table that contains columns in the order specified by the operator arguments. `project-reorder` doesn't rename or remove columns from the table, therefore, all columns that existed in the source table, appear in the result table.

Examples

Reorder a table with three columns (a, b, c) so the second column (b) will appear first.

Run the query

Kusto

```
print a='a', b='b', c='c'
|  project-reorder b
```

Output

b	a	c
b	a	c

Reorder columns of a table so that columns starting with `a` will appear before other columns.

Run the query

Kusto

```
print b = 'b', a2='a2', a3='a3', a1='a1'
|  project-reorder a* asc
```

Output

a1	a2	a3	b
a1	a2	a3	b

Feedback

parse operator

Article • 05/23/2023

Evaluates a string expression and parses its value into one or more calculated columns. The calculated columns will have nulls, for unsuccessfully parsed strings. If there's no need to use rows where parsing doesn't succeed, prefer using the parse-where operator.

Syntax

T `| parse` [`kind=`*kind* [`flags=`*regexFlags*]] *expression* `with` [`*`] *stringConstant* *columnName* [`:` *columnType*] [`*`] `,` ...

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input to parse.
kind	string	✓	One of the supported kind values. The default value is `simple`.
regexFlags	string		If *kind* is `regex`, then you can specify regex flags to be used like `U` for ungreedy, `m` for multi-line mode, `s` for match new line `\n`, and `i` for case-insensitive. More flags can be found in RE2 flags.
expression	string	✓	An expression that evaluates to a string.
stringConstant	string	✓	A string constant for which to search and parse.
columnName	string	✓	The name of a column to assign a value to, extracted from the string expression.
columnType	string		The scalar value that indicates the type to convert the value to. The default is `string`.

> ⓘ Note
>
> - The parse pattern may start with *ColumnName* and not only with *StringConstant*.
> - Use `*` in the pattern to skip junk values. The `*` can't be used after a `string` type column.

- If the parsed *expression* isn't of type `string`, it will be converted to type `string`.
- Use **project** if you also want to drop or rename some columns.

Supported kind values

Text	Description
`simple`	This is the default value. *stringConstant* is a regular string value and the match is strict. All string delimiters should appear in the parsed string, and all extended columns must match the required types.
`regex`	*stringConstant* may be a regular expression and the match is strict. All string delimiters, which can be a regex for this mode, should appear in the parsed string, and all extended columns must match the required types.
`relaxed`	*stringConstant* is a regular string value and the match is relaxed. All string delimiters should appear in the parsed string, but extended columns may partially match the required types. Extended columns that didn't match the required types will get the value `null`.

Regex mode

In regex mode, parse will translate the pattern to a regex. Use RE2 syntax to do the matching, and use numbered captured groups that are handled internally. For example:

Kusto

```
parse kind=regex Col with * <regex1> var1:string <regex2> var2:long
```

In the parse statement, the regex that will be internally generated by the parse is `.*?<regex1>(.*?)<regex2>(\-\d+)`.

- `*` was translated to `.*?`.
- `string` was translated to `.*?`.
- `long` was translated to `\-\d+`.

Returns

The input table, extended according to the list of columns that are provided to the operator.

Examples

The `parse` operator provides a streamlined way to `extend` a table by using multiple `extract` applications on the same `string` expression. This result is useful, when the table has a `string` column that contains several values that you want to break into individual columns. For example, a column that was produced by a developer trace ("`printf`"/"`Console.WriteLine`") statement.

In the example below, assume that the column `EventText` of table `Traces` contains strings of the form `Event: NotifySliceRelease (resourceName={0}, totalSlices={1}, sliceNumber={2}, lockTime={3}, releaseTime={4}, previousLockTime={5})`. The operation will extend the table with six columns: `resourceName`, `totalSlices`, `sliceNumber`, `lockTime`, `releaseTime`, and `previousLockTime`.

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=23, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00,
releaseTime=02/17/2016 08:40:00, previousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=20, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=16, lockTime=02/17/2016 08:41:00,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse EventText with * "resourceName=" resourceName ", totalSlices="
totalSlices: long * "sliceNumber=" sliceNumber: long * "lockTime=" lockTime
", releaseTime=" releaseTime: date "," * "previousLockTime="
previousLockTime: date ")" *
| project resourceName, totalSlices, sliceNumber, lockTime, releaseTime,
previousLockTime
```

Output

resourceName	totalSlices	sliceNumber	lockTime	releaseTime	previousLockTime
PipelineScheduler	27	15	02/17/2016 08:40:00	2016-02-17 08:40:00.0000000	2016-02-17 08:39:00.0000000
PipelineScheduler	27	23	02/17/2016 08:40:01	2016-02-17 08:40:01.0000000	2016-02-17 08:39:01.0000000
PipelineScheduler	27	20	02/17/2016 08:40:01	2016-02-17 08:40:01.0000000	2016-02-17 08:39:01.0000000
PipelineScheduler	27	16	02/17/2016 08:41:00	2016-02-17 08:41:00.0000000	2016-02-17 08:40:00.0000000
PipelineScheduler	27	22	02/17/2016 08:41:01	2016-02-17 08:41:00.0000000	2016-02-17 08:40:01.0000000

Regex mode

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=23, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00,
releaseTime=02/17/2016 08:40:00, previousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=20, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=16, lockTime=02/17/2016 08:41:00,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse kind = regex EventText with "(.*?)[a-zA-Z]*=" resourceName @",
totalSlices=\s*\d+\s*.*?sliceNumber=" sliceNumber: long  ".*?(previous)?
lockTime=" lockTime ".*?releaseTime=" releaseTime ".*?previousLockTime="
previousLockTime: date "\\)"
| project resourceName, sliceNumber, lockTime, releaseTime, previousLockTime
```

Output

resourceName	sliceNumber	lockTime	releaseTime	previousLockTime
PipelineScheduler	15	02/17/2016 08:40:00,	02/17/2016 08:40:00,	2016-02-17 08:39:00.0000000
PipelineScheduler	23	02/17/2016 08:40:01,	02/17/2016 08:40:01,	2016-02-17 08:39:01.0000000
PipelineScheduler	20	02/17/2016 08:40:01,	02/17/2016 08:40:01,	2016-02-17 08:39:01.0000000
PipelineScheduler	16	02/17/2016 08:41:00,	02/17/2016 08:41:00,	2016-02-17 08:40:00.0000000
PipelineScheduler	22	02/17/2016 08:41:01,	02/17/2016 08:41:00,	2016-02-17 08:40:01.0000000

Regex mode with regex flags

If you're interested in getting the resourceName only, use this query:

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=23, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00,
releaseTime=02/17/2016 08:40:00, previousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=20, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=16, lockTime=02/17/2016 08:41:00,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse kind = regex EventText with * "resourceName=" resourceName ',' *
| project resourceName
```

Output

resourceName
PipelineScheduler, totalSlices=27, sliceNumber=23, lockTime=02/17/2016 08:40:01, releaseTime=02/17/2016 08:40:01
PipelineScheduler, totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00, releaseTime=02/17/2016 08:40:00
PipelineScheduler, totalSlices=27, sliceNumber=20, lockTime=02/17/2016 08:40:01, releaseTime=02/17/2016 08:40:01
PipelineScheduler, totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01, releaseTime=02/17/2016 08:41:00
PipelineScheduler, totalSlices=27, sliceNumber=16, lockTime=02/17/2016 08:41:00, releaseTime=02/17/2016 08:41:00

You won't get the expected results, since the default mode is greedy. If you have a few records where the `resourceName` sometimes appears as lower-case and sometimes as upper-case, you may get nulls for some values.

To get the wanted result, run the query with the non-greedy `U`, and disable case-sensitive `i` regex flags.

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=23, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00,
releaseTime=02/17/2016 08:40:00, previousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=20, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=16, lockTime=02/17/2016 08:41:00,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse kind = regex flags = Ui EventText with * "RESOURCENAME="
```

```
resourceName ',' *
| project resourceName
```

Output

resourceName
PipelineScheduler
PipelineScheduler
PipelineScheduler
PipelineScheduler
PipelineScheduler

If the parsed string has newlines, use the flag `s`, to parse the text.

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease
(resourceName=PipelineScheduler\ntotalSlices=27\nsliceNumber=23\nlockTime=02
/17/2016 08:40:01\nreleaseTime=02/17/2016
08:40:01\npreviousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease
(resourceName=PipelineScheduler\ntotalSlices=27\nsliceNumber=15\nlockTime=02
/17/2016 08:40:00\nreleaseTime=02/17/2016
08:40:00\npreviousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease
(resourceName=PipelineScheduler\ntotalSlices=27\nsliceNumber=20\nlockTime=02
/17/2016 08:40:01\nreleaseTime=02/17/2016
08:40:01\npreviousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease
(resourceName=PipelineScheduler\ntotalSlices=27\nsliceNumber=22\nlockTime=02
/17/2016 08:41:01\nreleaseTime=02/17/2016
08:41:00\npreviousLockTime=02/17/2016 08:40:01)",
    "Event: NotifySliceRelease
(resourceName=PipelineScheduler\ntotalSlices=27\nsliceNumber=16\nlockTime=02
/17/2016 08:41:00\nreleaseTime=02/17/2016
08:41:00\npreviousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse kind=regex flags=s EventText with * "resourceName=" resourceName:
string "(.*?)totalSlices=" totalSlices: long "(.*?)lockTime=" lockTime:
datetime "(.*?)releaseTime=" releaseTime: datetime "(.*?)previousLockTime="
```

```
previousLockTime: datetime "\\)"
| project-away EventText
```

Output

resourceName	totalSlices	lockTime	releaseTime	previousLockTime
PipelineScheduler	27	2016-02-17 08:40:00.0000000	2016-02-17 08:40:00.0000000	2016-02-17 08:39:00.0000000
PipelineScheduler	27	2016-02-17 08:40:01.0000000	2016-02-17 08:40:01.0000000	2016-02-17 08:39:01.0000000
PipelineScheduler	27	2016-02-17 08:40:01.0000000	2016-02-17 08:40:01.0000000	2016-02-17 08:39:01.0000000
PipelineScheduler	27	2016-02-17 08:41:00.0000000	2016-02-17 08:41:00.0000000	2016-02-17 08:40:00.0000000
PipelineScheduler	27	2016-02-17 08:41:01.0000000	2016-02-17 08:41:00.0000000	2016-02-17 08:40:01.0000000

Relaxed mode

In this example for relaxed mode, `totalSlices` extended column must be of type `long`. However, in the parsed string, it has the value `nonValidLongValue`. In `releaseTime` extended column, the value `nonValidDateTime` can't be parsed as `datetime`. These two extended columns will get the value `null` while the other ones, such as `sliceNumber`, still get the correct values.

If you use option `kind = simple` for the same query below, you'll get `null` for all extended columns. This option is strict on extended columns, and is the difference between relaxed and simple mode.

> ⓘ Note
>
> In relaxed mode, extended columns can be partially matched.

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
```

```
totalSlices=27, sliceNumber=23, lockTime=02/17/2016 08:40:01,
releaseTime=nonValidDateTime 08:40:01, previousLockTime=02/17/2016
08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00,
releaseTime=nonValidDateTime, previousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=nonValidLongValue, sliceNumber=20, lockTime=02/17/2016 08:40:01,
releaseTime=nonValidDateTime 08:40:01, previousLockTime=02/17/2016
08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=nonValidLongValue, sliceNumber=16, lockTime=02/17/2016 08:41:00,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse kind=relaxed EventText with * "resourceName=" resourceName ",
totalSlices=" totalSlices: long * "sliceNumber=" sliceNumber: long *
"lockTime=" lockTime ", releaseTime=" releaseTime: date "," *
"previousLockTime=" previousLockTime: date ")" *
| project-away EventText
```

Output

resourceName	totalSlices	sliceNumber	lockTime	releaseTime	previousLockTime
PipelineScheduler	27	15	02/17/2016 08:40:00		2016-02-17 08:39:00.0000000
PipelineScheduler	27	23	02/17/2016 08:40:01		2016-02-17 08:39:01.0000000
PipelineScheduler		20	02/17/2016 08:40:01		2016-02-17 08:39:01.0000000
PipelineScheduler		16	02/17/2016 08:41:00	2016-02-17 08:41:00.0000000	2016-02-17 08:40:00.0000000
PipelineScheduler	27	22	02/17/2016 08:41:01	2016-02-17 08:41:00.0000000	2016-02-17 08:40:01.0000000

Feedback

parse-where operator

Article • 05/23/2023

Evaluates a string expression, and parses its value into one or more calculated columns. The result is only the successfully parsed strings.

`parse-where` parses the strings in the same way as parse, and filters out strings that were not parsed successfully.

See parse operator, which produces nulls for unsuccessfully parsed strings.

Syntax

T `| parse-where` [`kind=`*kind* [`flags=` *regexFlags*]] *expression* `with` `*` (*stringConstant* *columnName* [`:` *columnType*]) `*`...

Parameters

Name	Type	Required	Description
T	string	✓	The tabular input to parse.
kind	string	✓	One of the supported kind values. The default value is `simple`.
regexFlags	string		If *kind* is `regex`, then you can specify regex flags to be used like `U` for ungreedy, `m` for multi-line mode, `s` for match new line `\n`, and `i` for case-insensitive. More flags can be found in RE2 flags.
expression	string	✓	An expression that evaluates to a string.
stringConstant	string	✓	A string constant for which to search and parse.
columnName	string	✓	The name of a column to assign a value to, extracted from the string expression.
columnType	string		The scalar value that indicates the type to convert the value to. The default is the `string`.

> ⓘ Note
>
> - Use **project** if you also want to drop or rename some columns.

- Use `*` in the pattern to skip junk values. This value can't be used after `string` column.
- The parse pattern may start with *ColumnName*, in addition to *StringConstant*.
- If the parsed *expression* isn't of type `string`, it will be converted to type `string`.

Supported kind values

Text	Description
`simple`	This is the default value. *stringConstant* is a regular string value and the match is strict. All string delimiters should appear in the parsed string, and all extended columns must match the required types.
`regex`	*stringConstant* may be a regular expression and the match is strict. All string delimiters, which can be a regex for this mode, should appear in the parsed string, and all extended columns must match the required types.

Regex mode

In regex mode, parse will translate the pattern to a regex and use RE2 syntax in order to do the matching using numbered captured groups that are handled internally. For example:

Kusto

```
parse-where kind=regex Col with * <regex1> var1:string <regex2> var2:long
```

The regex that will be generated by the parse internally is `.*?<regex1>(.*?)<regex2>(\-\d+)`.

- `*` was translated to `.*?`.
- `string` was translated to `.*?`.
- `long` was translated to `\-\d+`.

Returns

The input table, which is extended according to the list of columns that are provided to the operator.

> ⓘ Note
>
> Only successfully parsed strings will be in the output. Strings that don't match the pattern will be filtered out.

Examples

The `parse-where` operator provides a streamlined way to `extend` a table by using multiple `extract` applications on the same `string` expression. This is most useful when the table has a `string` column that contains several values that you want to break into individual columns. For example, you can break up a column that was produced by a developer trace ("`printf`"/"`Console.WriteLine`") statement.

Using `parse`

In the example below, the column `EventText` of table `Traces` contains strings of the form `Event: NotifySliceRelease (resourceName={0}, totalSlices= {1}, sliceNumber={2}, lockTime={3}, releaseTime={4}, previousLockTime={5})`. The operation below will extend the table with six columns: `resourceName` , `totalSlices`, `sliceNumber`, `lockTime`, `releaseTime`, `previousLockTime`, `Month`, and `Day`.

A few of the strings don't have a full match.

Using `parse`, the calculated columns will have nulls.

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=invalid_number, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00,
releaseTime=invalid_datetime, previousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=20, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:01)",
```

```
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=invalid_number, sliceNumber=16, lockTime=02/17/2016 08:41:00,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse EventText with * "resourceName=" resourceName ", totalSlices="
totalSlices: long * "sliceNumber=" sliceNumber: long * "lockTime=" lockTime
", releaseTime=" releaseTime: date "," * "previousLockTime="
previouLockTime: date ")" *
| project
    resourceName,
    totalSlices,
    sliceNumber,
    lockTime,
    releaseTime,
    previouLockTime
```

Output

resourceName	totalSlices	sliceNumber	lockTime	releaseTime	previousLockTime
PipelineScheduler	27	20	02/17/2016 08:40:01	2016-02-17 08:40:01.0000000	2016-02-17 08:39:01.0000000
PipelineScheduler	27	22	02/17/2016 08:41:01	2016-02-17 08:41:00.0000000	2016-02-17 08:40:01.0000000

Using `parse-where`

Using 'parse-where' will filter-out unsuccessfully parsed strings from the result.

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=invalid_number, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00,
releaseTime=invalid_datetime, previousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=20, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
```

```
totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=invalid_number, sliceNumber=16, lockTime=02/17/2016 08:41:00,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse-where EventText with * "resourceName=" resourceName ", totalSlices="
totalSlices: long * "sliceNumber=" sliceNumber: long * "lockTime=" lockTime
", releaseTime=" releaseTime: date "," * "previousLockTime="
previousLockTime: date ")" *
| project
    resourceName,
    totalSlices,
    sliceNumber,
    lockTime,
    releaseTime,
    previousLockTime
```

Output

resourceName	totalSlices	sliceNumber	lockTime	releaseTime	previousLockTime
PipelineScheduler	27	20	02/17/2016 08:40:01	2016-02-17 08:40:01.0000000	2016-02-17 08:39:01.0000000
PipelineScheduler	27	22	02/17/2016 08:41:01	2016-02-17 08:41:00.0000000	2016-02-17 08:40:01.0000000

Regex mode using regex flags

To get the resourceName and totalSlices, use the following query:

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=non_valid_integer, sliceNumber=11, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00,
releaseTime=02/17/2016 08:40:00, previousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=non_valid_integer, sliceNumber=44, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01,
```

```
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=16, lockTime=02/17/2016 08:41:00,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse-where kind = regex EventText with * "RESOURCENAME=" resourceName ","
* "totalSlices=" totalSlices: long "," *
| project resourceName, totalSlices
```

Output

resourceName	totalSlices

`parse-where` with case-insensitive regex flag

In the above query, the default mode was case-sensitive, so the strings were parsed successfully. No result was obtained.

To get the required result, run `parse-where` with a case-insensitive (`i`) regex flag.

Only three strings will be parsed successfully, so the result is three records (some totalSlices hold invalid integers).

Run the query

Kusto

```
let Traces = datatable(EventText: string)
    [
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=non_valid_integer, sliceNumber=11, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=15, lockTime=02/17/2016 08:40:00,
releaseTime=02/17/2016 08:40:00, previousLockTime=02/17/2016 08:39:00)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=non_valid_integer, sliceNumber=44, lockTime=02/17/2016 08:40:01,
releaseTime=02/17/2016 08:40:01, previousLockTime=02/17/2016 08:39:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=22, lockTime=02/17/2016 08:41:01,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:01)",
    "Event: NotifySliceRelease (resourceName=PipelineScheduler,
totalSlices=27, sliceNumber=16, lockTime=02/17/2016 08:41:00,
releaseTime=02/17/2016 08:41:00, previousLockTime=02/17/2016 08:40:00)"
];
Traces
| parse-where kind = regex flags=i EventText with * "RESOURCENAME="
```

```
resourceName "," * "totalSlices=" totalSlices: long "," *
| project resourceName, totalSlices
```

Output

resourceName	totalSlices
PipelineScheduler	27
PipelineScheduler	27
PipelineScheduler	27

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

parse-kv operator

Article • 03/12/2023

Extracts structured information from a string expression and represents the information in a key/value form.

The following extraction modes are supported:

- **Specified delimeter**: Extraction based on specified delimiters that dictate how keys/values and pairs are separated from each other.
- **Non-specified delimeter**: Extraction with no need to specify delimiters. Any non-alphanumeric character is considered a delimiter.
- **Regex**: Extraction based on RE2 regular expression.

Syntax

Specified delimeter

T `|` `parse-kv` *Expression* `as` `(` *KeysList* `)` `with` `(` `pair_delimiter` `=` *PairDelimiter* `,` `kv_delimiter` `=` *KvDelimiter* [`,` `quote` `=` *QuoteChars* ... [`,` `escape` `=` *EscapeChar* ...]] [`,` `greedy` `=` `true`] `)`

Non-specified delimiter

T `|` `parse-kv` *Expression* `as` `(` *KeysList* `)` `with` `(` [`quote` `=` *QuoteChars* ... [`,` `escape` `=` *EscapeChar* ...]] `)`

Regex

T `|` `parse-kv` *Expression* `as` `(` *KeysList* `)` `with` `(` `regex` `=` *RegexPattern*`)` `)`

Parameters

Name	Type	Required	Description
Expression	string	✓	The expression from which to extract key values.

Name	Type	Required	Description
KeysList	string	✓	A comma-separated list of key names and their value data types. The order of the keys doesn't have to match the order in which they appear in the text.
PairDelimiter	string		A delimiter that separates key value pairs from each other.
KvDelimiter	string		A delimiter that separates keys from values.
QuoteChars	string		A one- or two-character string literal representing opening and closing quotes that key name or the extracted value may be wrapped with. The parameter can be repeated to specify a separate set of opening/closing quotes.
EscapeChar	string		A one-character string literal describing a character that may be used for escaping special characters in a quoted value. The parameter can be repeated if multiple escape characters are used.
RegexPattern	string		A RE2 regular expression containing two capturing groups exactly. The first group represents the key name, and the second group represents the key value.

Returns

The original input tabular expression *T*, extended with columns per specified keys to extract.

> ⓘ Note
>
> - If a key doesn't appear in a record, the corresponding column value will either be `null` or an empty string, depending on the column type.
> - Only keys that are listed in the operator are extracted.
> - The first appearance of a key is extracted, and subsequent values are ignored.
> - When extracting keys and values, leading and trailing white spaces are ignored.

Examples

Extraction with well-defined delimiters

In the following example, keys and values are separated by well defined delimiters. These delimeters are comma and colon characters.

Run the query

Kusto

```
print str="ThreadId:458745723, Machine:Node001, Text: The service is up,
Level: Info"
| parse-kv str as (Text: string, ThreadId:long, Machine: string) with
(pair_delimiter=',', kv_delimiter=':')
| project-away str
```

Output

Text	ThreadId	Machine
The service is up	458745723	Node001

Extraction with value quoting

Sometimes key names or values are wrapped in quotes, which allows the values themselves to contain delimiter characters. The following examples show how a `quote` argument is used for extracting such values.

Run the query

Kusto

```
print str='src=10.1.1.123 dst=10.1.1.124 bytes=125 failure="connection
aborted" "event time"=2021-01-01T10:00:54'
| parse-kv str as (['event time']:datetime, src:string, dst:string,
bytes:long, failure:string) with (pair_delimiter=' ', kv_delimiter='=',
quote='"')
| project-away str
```

Output

event time	src	dst	bytes	failure
2021-01-01 10:00:54.0000000	10.1.1.123	10.1.1.124	125	connection aborted

The following example uses different opening and closing quotes:

Run the query

Kusto

```
print str='src=10.1.1.123 dst=10.1.1.124 bytes=125 failure=(connection
aborted) (event time)=(2021-01-01 10:00:54)'
| parse-kv str as (['event time']:datetime, src:string, dst:string,
bytes:long, failure:string) with (pair_delimiter=' ', kv_delimiter='=',
quote='()')
| project-away str
```

Output

event time	src	dst	bytes	failure
2021-01-01 10:00:54.0000000	10.1.1.123	10.1.1.124	125	connection aborted

The values themselves may contain properly escaped quote characters, as the following example shows:

Run the query

Kusto

```
print str='src=10.1.1.123 dst=10.1.1.124 bytes=125 failure="the remote host
sent \\"bye!\\"" time=2021-01-01T10:00:54'
| parse-kv str as (['time']:datetime, src:string, dst:string, bytes:long,
failure:string) with (pair_delimiter=' ', kv_delimiter='=', quote='"',
escape='\\')
| project-away str
```

Output

time	src	dst	bytes	failure
2021-01-01 10:00:54.0000000	10.1.1.123	10.1.1.124	125	the remote host sent "bye!"

Extraction in greedy mode

There are cases when unquoted values may contain pair delimiters. In this case, use the `greedy` mode to indicate to the operator to scan until the next key appearance (or end of string) when looking for the value ending.

The following examples compare how the operator works with and without the `greedy` mode specified:

Run the query

Kusto

```
print str='name=John Doe phone=555 5555 city=New York'
| parse-kv str as (name:string, phone:string, city:string) with
(pair_delimiter=' ', kv_delimiter='=')
| project-away str
```

Output

name	phone	city
John	555	New

Run the query

Kusto

```
print str='name=John Doe phone=555 5555 city=New York'
| parse-kv str as (name:string, phone:string, city:string) with
(pair_delimiter=' ', kv_delimiter='=', greedy=true)
| project-away str
```

Output

name	phone	city
John Doe	555 5555	New York

Extraction with no well-defined delimiters

In the following example, any non-alphanumeric character is considered a valid delimiter:

Run the query

Kusto

```
print str="2021-01-01T10:00:34 [INFO] ThreadId:458745723, Machine:Node001,
Text: Started"
| parse-kv str as (Text: string, ThreadId:long, Machine: string)
| project-away str
```

Output

Text	ThreadId	Machine
Started	458745723	Node001

Values quoting and escaping is allowed in this mode as shown in the following example:

Run the query

Kusto

```
print str="2021-01-01T10:00:34 [INFO] ThreadId:458745723, Machine:Node001,
Text: 'The service \\' is up'"
| parse-kv str as (Text: string, ThreadId:long, Machine: string) with
(quote="'", escape='\\')
| project-away str
```

Output

Text	ThreadId	Machine
The service ' is up	458745723	Node001

Extraction using regex

When no delimiters define text structure well enough, regular expression-based extraction can be useful.

Run the query

Kusto

```
print str=@'["referer url: https://hostname.com/redirect?dest=/?h=1234",
"request url: https://hostname.com/?h=1234", "advertiser id: 24fefbca-cf27-
4d62-a623-249c2ad30c73"]'
| parse-kv str as (['referer url']:string, ['request url']:string,
['advertiser id']: guid) with (regex=@'"([\w ]+)\s*:\s*([^"]*)"')
| project-away str
```

Output

referer url	request url	advertiser id

referer url	request url	advertiser id
`https://hostname.com/redirect?dest=/?h=1234`	`https://hostname.com/?h=1234`	24fefbca-cf27-4d62-a623-249c2ad30c73

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Partition operator

Article • 02/15/2023

The partition operator partitions the records of its input table into multiple subtables according to values in a key column. The operator runs a subquery on each subtable, and produces a single output table that is the union of the results of all subqueries. This operator is useful when you need to perform a subquery only on a subset of rows that belongs to the same partition key, and not query the whole dataset. These subqueries could include aggregate functions, window functions, top N and others.

The partition operator supports several strategies of subquery operation:

- Native - use with an implicit data source with thousands of key partition values.
- Shuffle - use with an implicit source with millions of key partition values.
- Legacy - use with an implicit or explicit source for 64 or less key partition values.

Native strategy

This subquery is a tabular transformation that doesn't specify a tabular source. The source is implicit and is assigned according to the subtable partitions. It should be applied when the number of distinct values of the partition key isn't large, roughly in the thousand. Use `hint.strategy=native` for this strategy. There's no restriction on the number of partitions.

Shuffle strategy

This subquery is a tabular transformation that doesn't specify a tabular source. The source is implicit and will be assigned according to the subtable partitions. The strategy applies when the number of distinct values of the partition key is large, in the millions. Use `hint.strategy=shuffle` for this strategy. There's no restriction on the number of partitions. For more information about shuffle strategy and performance, see shuffle.

Native and shuffle strategy operators

The difference between `hint.strategy=native` and `hint.strategy=shuffle` is mainly to allow the caller to indicate the cardinality and execution strategy of the subquery, and can affect the execution time. There's no other semantic difference between the two.

For `native` and `shuffle` strategy, the source of the subquery is implicit, and can't be referenced by the subquery. This strategy supports a limited set of operators: `project`, `sort`, `summarize`, `take`, `top`, `order`, `mv-expand`, `mv-apply`, `make-series`, `limit`, `extend`, `distinct`, `count`, `project-away`, `project-keep`, `project-rename`, `project-reorder`, `parse`, `parse-where`, `reduce`, `sample`, `sample-distinct`, `scan`, `search`, `serialize`, `top-nested`, `top-hitters` and `where`.

Operators like `join`, `union`, `external_data`, `plugins`, or any other operator that involves table source that isn't the subtable partitions, aren't allowed.

Legacy strategy

Legacy subqueries can use the following sources:

- Implicit - The source is a tabular transformation that doesn't specify a tabular source. The source is implicit and will be assigned according to the subtable partitions. This scenario applies when there are 64 or less key values.
- Explicit - The subquery must include a tabular source explicitly. Only the key column of the input table is available in the subquery, and referenced by using its name in the `toscalar()` function.

For both implicit and explicit sources, the subquery type is used for legacy purposes only, and indicated by the use of `hint.strategy=legacy`, or by not including any strategy indication.

Any other reference to the source is taken to mean the entire input table, for example, by using the as operator and calling up the value again.

> ⓘ **Note**
>
> It is recommended to use the native or shuffle strategies rather than the legacy strategy, since the legacy strategy is limited to 64 partitions and is less efficient. The legacy partition operator is currently limited by the number of partitions. The operator will yield an error if the partition column (*Column*) has more than 64 distinct values.

All strategies

For native, shuffle and legacy subqueries, the result must be a single tabular result. Multiple tabular results and the use of the `fork` operator aren't supported. A subquery

can't include other statements, for example, it can't have a `let` statement.

Syntax

T `|` `partition` [`hint.strategy=` *strategy*] [*PartitionParameters*] `by` *Column* `(` *TransformationSubQuery* `)`

T `|` `partition` [*PartitionParameters*] `by` *Column* `{` *ContextFreeSubQuery* `}`

Parameters

Name	Type	Required	Description
T	string	✓	The tabular source whose data is to be processed by the operator.
strategy			The partition strategy, `native`, `shuffle` or `legacy`. `native` strategy is used with an implicit source with thousands of key partition values. `shuffle` strategy is used with an implicit source with millions of key partition values. `legacy` strategy is used with an explicit or implicit source with 64 or less key partition values.
Column		✓	The name of a column in *T* whose values determine how the input table is to be partitioned.
TransformationSubQuery		✓	A tabular transformation expression, whose source is implicitly the subtables produced by partitioning the records of *T*, each subtable being homogenous on the value of *Column*.
ContextFreeSubQuery		✓	A tabular expression that includes its own tabular source, such as a table reference. The expression can reference a single column from *T*, being the key column *Column* using the syntax `toscalar(`*Column*`)`.
PartitionParameters			Zero or more space-separated parameters in the form of: *HintName* `=` *Value* that control the behavior of the operator. See the supported hints.

Supported hints

HintName	Type	Description	Native/Shuffle/Legacy strategy
`hint.strategy`	string	The value `legacy`, `shuffle`, or `native`. This hint defines the execution strategy of the partition operator.	Native, Shuffle, Legacy
`hint.shufflekey`	string	The partition key. Runs the partition operator in shuffle strategy where the shuffle key is the specified partition key.	Shuffle
`hint.materialized`	bool	If set to `true`, will materialize the source of the `partition` operator. The default value is `false`.	Legacy
`hint.concurrency`	int	Hints the system how many partitions to run in parallel. The default value is 16.	Legacy
`hint.spread`	int	Hints the system how to distribute the partitions among cluster nodes. For example, if there are N partitions and the spread hint is set to P, then the N partitions will be processed by P different cluster nodes equally in parallel/sequentially depending on the concurrency hint. The default value is 1.	Legacy

Returns

The operator returns a union of the results of the individual subqueries.

Examples

Native strategy examples

Use `hint.strategy=native` for this strategy. See the following examples:

This query returns foreach InjuriesDirect, the count of events and total injuries in each State that starts with 'W'.

Run the query

```
Kusto
```

```
StormEvents
| where State startswith 'W'
| partition hint.strategy=native by InjuriesDirect (summarize
Events=count(), Injuries=sum(InjuriesDirect) by State);
```

Output

State	Events	Injuries
WISCONSIN	4	4
WYOMING	5	5
WEST VIRGINIA	1	1
WASHINGTON	2	2
WEST VIRGINIA	756	0
WYOMING	390	0
WASHINGTON	256	0
WISCONSIN	1845	0
WYOMING	1	4
WASHINGTON	1	5
WISCONSIN	1	2
WASHINGTON	1	2
WASHINGTON	1	10

This query returns the top 2 EventType by total injuries for each State that starts with 'W':

Run the query

Kusto

```
StormEvents
| where State startswith 'W'
| partition hint.strategy = native by State
    (
    summarize TotalInjueries = sum(InjuriesDirect) by EventType
    | top 2 by TotalInjueries
    )
```

Output

EventType	TotalInjueries
Tornado	4
Hail	1
Thunderstorm Wind	1
Excessive Heat	0
High Wind	13
Lightning	5
High Wind	5
Avalanche	3

Shuffle strategy example

Use `hint.strategy=shuffle` for this strategy. See the following example:

This query will return the top 3 DamagedProperty foreach EpisodeId, it returns also the columns EpisodeId and State.

Run the query

Kusto

```
StormEvents
| partition hint.strategy=shuffle by EpisodeId
    (
    top 3 by DamageProperty
    | project EpisodeId, State, DamageProperty
    )
| count
```

Output

Count
22345

Legacy strategy with explicit source

This strategy is for legacy purposes only, and indicated by the use of `hint.strategy=legacy` or by not including a strategy indication at all. See the following example:

This query will run two subqueries:

- When x == 1, the query will return all rows from StormEvents that have InjuriesIndirect == 1.
- When x == 2, the query will return all rows from StormEvents that have InjuriesIndirect == 2.

the final result is the union of these 2 subqueries.

Run the query

Kusto

```
range x from 1 to 2 step 1
| partition hint.strategy=legacy by x {StormEvents | where x ==
InjuriesIndirect}
| count
```

Output

Count
113

Partition operator

In some cases, it's more performant and easier to write a query using the `partition` operator than using the top-nested operator. The following example runs a subquery calculating `summarize` and `top` for each of States starting with `W`: (WYOMING, WASHINGTON, WEST VIRGINIA, WISCONSIN)

Run the query

Kusto

```
StormEvents
| where State startswith 'W'
| partition hint.strategy=native by State
    (
    summarize Events=count(), Injuries=sum(InjuriesDirect) by EventType,
State
```

```
    | top 3 by Events
    )
```

Output

EventType	State	Events	Injuries
Hail	WYOMING	108	0
High Wind	WYOMING	81	5
Winter Storm	WYOMING	72	0
Heavy Snow	WASHINGTON	82	0
High Wind	WASHINGTON	58	13
Wildfire	WASHINGTON	29	0
Thunderstorm Wind	WEST VIRGINIA	180	1
Hail	WEST VIRGINIA	103	0
Winter Weather	WEST VIRGINIA	88	0
Thunderstorm Wind	WISCONSIN	416	1
Winter Storm	WISCONSIN	310	0
Hail	WISCONSIN	303	1

Partition reference

The following example shows how to use the as operator to give a "name" to each data partition and then reuse that name within the subquery. This approach is only relevant to the legacy strategy.

Kusto

```
T
| partition by Dim
(
    as Partition
    | extend MetricPct = Metric * 100.0 / toscalar(Partition | summarize
sum(Metric))
)
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

print operator

Article • 01/26/2023

Outputs a single row with one or more scalar expression results as columns.

Syntax

`print` [*ColumnName* `=`] *ScalarExpression* [',' ...]

Parameters

Name	Type	Required	Description
ColumnName	string		The name to assign to the output column.
ScalarExpression	string	✓	The expression to evaluate.

Returns

A table with one or more columns and a single row. Each column returns the corresponding value of the evaluated *ScalarExpression*.

Examples

Run the query

Kusto

```
print 0 + 1 + 2 + 3 + 4 + 5, x = "Wow!"
```

Output

print_0	x
15	Wow!

Run the query

Kusto

```
print banner=strcat("Hello", ", ", "World!")
```

Output

banner
Hello, World!

Feedback

range operator

Article • 03/12/2023

Generates a single-column table of values.

> ⓘ Note
>
> This operator doesn't take a tabular input.

Syntax

`range` *columnName* `from` *start* `to` *stop* `step` *step*

Parameters

Name	Type	Required	Description
columnName	string	✓	The name of the single column in the output table.
start	int, long, real, datetime, or timespan	✓	The smallest value in the output.
stop	int, long, real, datetime, or timespan	✓	The highest value being generated in the output or a bound on the highest value if *step* steps over this value.
step	int, long, real, datetime, or timespan	✓	The difference between two consecutive values.

> ⓘ Note
>
> The values can't reference the columns of any table. If you want to compute the range based on an input table, use the **range** function potentially with the **mv-expand** operator.

Returns

A table with a single column called *columnName*, whose values are *start*, *start* `+` *step*, ... up to and until *stop*.

Examples

The following example creates a table with entries for the current time stamp extended over the past seven days, once a day.

Run the query

```
Kusto
```

```
range LastWeek from ago(7d) to now() step 1d
```

Output

LastWeek
2015-12-05 09:10:04.627
2015-12-06 09:10:04.627
...
2015-12-12 09:10:04.627

The following example shows how to use the `range` operator with parameters, which are then extended and consumed as a table.

Run the query

```
Kusto
```

```
let toUnixTime = (dt:datetime)
{
    (dt - datetime(1970-01-01)) / 1s
};
let MyMonthStart = startofmonth(now()); //Start of month
let StepBy = 4.534h; //Supported timespans
let nn = 64000; // Row Count parametrized
let MyTimeline = range MyMonthHour from MyMonthStart to now() step StepBy
| extend MyMonthHourinUnixTime = toUnixTime(MyMonthHour), DateOnly =
bin(MyMonthHour,1d), TimeOnly = MyMonthHour - bin(MyMonthHour,1d)
; MyTimeline | order by MyMonthHour asc | take nn
```

Output

MyMonthHour	MyMonthHourinUnixTime	DateOnly	TimeOnly
2023-02-01	00:00:00.0000000	1675209600	2023-02-01 00:00:00.0000000
2023-02-01	04:32:02.4000000	1675225922.4	2023-02-01 00:00:00.0000000
2023-02-01	09:04:04.8000000	1675242244.8	2023-02-01 00:00:00.0000000
2023-02-01	13:36:07.2000000	1675258567.2	2023-02-01 00:00:00.0000000
...	...	...	...

The following example creates a table with a single column called `Steps` whose type is `long` and whose values are `1`, `4`, and `7`.

Run the query

Kusto

```
range Steps from 1 to 8 step 3
```

The following example shows how the `range` operator can be used to create a small, ad-hoc, dimension table that is then used to introduce zeros where the source data has no values.

Kusto

```
range TIMESTAMP from ago(4h) to now() step 1m
| join kind=fullouter
  (Traces
      | where TIMESTAMP > ago(4h)
      | summarize Count=count() by bin(TIMESTAMP, 1m)
  ) on TIMESTAMP
| project Count=iff(isnull(Count), 0, Count), TIMESTAMP
| render timechart
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

reduce operator

Article • 03/22/2023

Groups a set of strings together based on value similarity.

For each such group, the operator returns a `pattern`, `count`, and `representative`. The `pattern` best describes the group, in which the `*` character represents a wildcard. The `count` is the number of values in the group, and the `representative` is one of the original values in the group.

Syntax

T `|` `reduce` [`kind` `=` *ReduceKind*] `by` *Expr* [`with` [`threshold` `=` *Threshold*] [`,` `characters` `=` *Characters*]]

Parameters

Name	Type	Required	Description
Expr	string	✓	The value by which to reduce.
Threshold	real		A value between 0 and 1 that determines the minimum fraction of rows required to match the grouping criteria in order to trigger a reduction operation. The default value is 0.1. We recommend setting a small threshold value for large inputs. With a smaller threshold value, more similar values are grouped together, resulting in fewer but more similar groups. A larger threshold value requires less similarity, resulting in more groups that are less similar. See Examples.
Characters	string		A list of characters that separate between terms. The default is every non-ascii numeric character. For examples, see Behavior of Characters parameter.
ReduceKind	string		The only valid value is `source`. If `source` is specified, the operator appends the `Pattern` column to the existing rows in the table instead of aggregating by `Pattern`.

Returns

A table with as many rows as there are groups and columns titled `pattern`, `count`, and `representative`. The `pattern` best describes the group, in which the `*` character represents a wildcard, or placeholder for an arbitrary insertion string. The `count` is the number of values in the group, and the `representative` is one of the original values in the group.

For example, the result of `reduce by city` might include:

Pattern	Count	Representative
San *	5182	San Bernard
Saint *	2846	Saint Lucy
Moscow	3726	Moscow
* -on- *	2730	One -on- One
Paris	2716	Paris

Examples

Small threshold value

Run the query

Kusto

```
range x from 1 to 1000 step 1
| project MyText = strcat("MachineLearningX", tostring(toint(rand(10))))
| reduce by MyText  with threshold=0.001 , characters = "X"
```

Output

Pattern	Count	Representative
MachineLearning*	1000	MachineLearningX4

Large threshold value

Run the query

Kusto

```
range x from 1 to 1000 step 1
| project MyText = strcat("MachineLearningX", tostring(toint(rand(10))))
| reduce by MyText  with threshold=0.9 , characters = "X"
```

Output

Pattern	Count	Representative
MachineLearning*	177	MachineLearningX9
MachineLearning*	102	MachineLearningX0
MachineLearning*	106	MachineLearningX1
MachineLearning*	96	MachineLearningX6
MachineLearning*	110	MachineLearningX4
MachineLearning*	100	MachineLearningX3
MachineLearning*	99	MachineLearningX8
MachineLearning*	104	MachineLearningX7
MachineLearning*	106	MachineLearningX2

Behavior of Characters parameter

If the *Characters* parameter is unspecified, then every non-ascii numeric character will be a term separator.

Run the query

Kusto

```
range x from 1 to 10 step 1 | project str = strcat("foo", "Z", tostring(x))
| reduce by str
```

Output

Pattern	Count	Representative
others	10	

However, if you specify that "Z" is a separator, then it's as if each value in `str` is 2 terms: `foo` and `tostring(x)`:

Run the query

Kusto

```
range x from 1 to 10 step 1 | project str = strcat("foo", "Z", tostring(x))
| reduce by str with characters="Z"
```

Output

Pattern	Count	Representative
foo*	10	fooZ1

Apply `reduce` to sanitized input

The following example shows how one might apply the `reduce` operator to a "sanitized" input, in which GUIDs in the column being reduced are replaced prior to reducing

Kusto

```
// Start with a few records from the Trace table.
Trace | take 10000
// We will reduce the Text column which includes random GUIDs.
// As random GUIDs interfere with the reduce operation, replace them all
// by the string "GUID".
| extend Text=replace_regex(Text, @"[[:xdigit:]]{8}-[[:xdigit:]]{4}-
[[:xdigit:]]{4}-[[:xdigit:]]{4}-[[:xdigit:]]{12}", @"GUID")
// Now perform the reduce. In case there are other "quasi-random"
identifiers with embedded '-'
// or '_' characters in them, treat these as non-term-breakers.
| reduce by Text with characters="-_"
```

See also

autocluster

> **Note**
>
> The implementation of `reduce` operator is largely based on the paper **A Data Clustering Algorithm for Mining Patterns From Event Logs**, by Risto Vaarandi.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

render operator

Article • 03/08/2023

Instructs the user agent to render a visualization of the query results.

The render operator must be the last operator in the query, and can only be used with queries that produce a single tabular data stream result. The render operator does not modify data. It injects an annotation ("Visualization") into the result's extended properties. The annotation contains the information provided by the operator in the query. The interpretation of the visualization information is done by the user agent. Different agents, such as Kusto.Explorer or Azure Data Explorer web UI, may support different visualizations.

The data model of the render operator looks at the tabular data as if it has three kinds of columns:

- The x axis column (indicated by the `xcolumn` property).
- The series columns (any number of columns indicated by the `series` property.) For each record, the combined values of these columns defines a single series, and the chart has as many series as there are distinct combined values.
- The y axis columns (any number of columns indicated by the `ycolumns` property). For each record, the series has as many measurements ("points" in the chart) as there are y-axis columns.

> **Tip**
>
> - Use `where`, `summarize` and `top` to limit the volume that you display.
> - Sort the data to define the order of the x-axis.
> - User agents are free to "guess" the value of properties that are not specified by the query. In particular, having "uninteresting" columns in the schema of the result might translate into them guessing wrong. Try projecting-away such columns when that happens.

Syntax

T `|` `render` *visualization* [`with` `(` *propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
visualization	string	✓	Indicates the kind of visualization to use. Must be one of the supported values in the following list.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Visualization

visualization	**Description**
anomalychart	Similar to timechart, but highlights anomalies using series_decompose_anomalies function.
areachart	Area graph.
barchart	displayed as horizontal strips.
card	First result record is treated as set of scalar values and shows as a card.
columnchart	Like `barchart` with vertical strips instead of horizontal strips.
ladderchart	Last two columns are the x-axis, other columns are y-axis.
linechart	Line graph.
piechart	First column is color-axis, second column is numeric.
pivotchart	Displays a pivot table and chart. User can interactively select data, columns, rows and various chart types.
scatterchart	Points graph.
stackedareachart	Stacked area graph.
table	Default - results are shown as a table.
timechart	Line graph. First column is x-axis, and must be datetime. Other (numeric) columns are y-axes.
timepivot	Interactive navigation over the events time-line (pivoting on time axis)
treemap	Displays hierarchical data as a set of nested rectangles.

ⓘ Note

The ladderchart, pivotchart, timepivot, and treemap visualizations can be used in Kusto.Explorer but are not available in the Azure Data Explorer web UI.

Supported properties

PropertyName/*PropertyValue* indicate additional information to use when rendering. All properties are optional. The supported properties are:

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`kind`	Further elaboration of the visualization kind. For more information, see kind property.
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ysplit`	How to split multiple the visualization. For more information, see y-split property.
`ytitle`	The title of the y-axis (of type `string`).
`anomalycolumns`	Property relevant only for `anomalychart`. Comma-delimited list of columns, which will be considered as anomaly series and displayed as points on the chart

`kind` property

This visualization can be further elaborated by providing the `kind` property. The supported values of this property are:

Visualization	`kind`	Description
`areachart`	`default`	Each "area" stands on its own.
	`unstacked`	Same as `default`.
	`stacked`	Stack "areas" to the right.
	`stacked100`	Stack "areas" to the right and stretch each one to the same width as the others.
`barchart`	`default`	Each "bar" stands on its own.
	`unstacked`	Same as `default`.
	`stacked`	Stack "bars".
	`stacked100`	Stack "bars" and stretch each one to the same width as the others.
`columnchart`	`default`	Each "column" stands on its own.
	`unstacked`	Same as `default`.
	`stacked`	Stack "columns" one atop the other.
	`stacked100`	Stack "columns" and stretch each one to the same height as the others.
`scatterchart`	`map`	Expected columns are [Longitude, Latitude] or GeoJSON point. Series column is optional. For more information, see Geospatial visualizations.
`piechart`	`map`	Expected columns are [Longitude, Latitude] or GeoJSON point, color-axis and numeric. Supported in Kusto Explorer desktop. For more information, see Geospatial visualizations.

`ysplit` property

Some visualizations support splitting into multiple y-axis values:

`ysplit`	Description
`none`	A single y-axis is displayed for all series data. (Default)
`axes`	A single chart is displayed with multiple y-axes (one per series).

`ysplit`	**Description**
`panels`	One chart is rendered for each `ycolumn` value (up to some limit).

How to render continuous data

Several visualizations are used for rendering sequences of values, for example, `linechart`, `timechart`, and `areachart`. These visualizations have the following conceptual model:

- One column in the table represents the x-axis of the data. This column can be explicitly defined using the `xcolumn` property. If not defined, the user agent will pick the first column that is appropriate for the visualization.
  - For example: in the `timechart` visualization, the user agent will use the first `datetime` column.
  - If this column is of type `dynamic` and it holds an array, the individual values in the array will be treated as the values of the x-axis.
- One or more columns in the table represent one or more measures that vary by the x-axis. These columns can be explicitly defined using the `ycolumns` property. If not defined, the user agent will pick all columns that are appropriate for the visualization.
  - For example: in the `timechart` visualization, the user agent will use all columns with a numeric value that haven't been specified otherwise.
  - If the x-axis is an array, the values of each y-axis should also be an array of a similar length, with each y-axis occurring in a single column.
- Zero or more columns in the table represent a unique set of dimensions that group together the measures. These columns can be specified by the `series` property, or the user agent will pick them automatically from the columns that are otherwise unspecified.

See also

- Rendering examples in the tutorial
- Anomaly detection

Feedback

Anomaly chart

Article • 02/28/2023

The anomaly chart visualization is similar to a timechart, but highlights anomalies using the series_decompose_anomalies function.

> ⓘ Note
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `anomalychart` [`with` `(` *propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).

PropertyName	*PropertyValue*
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ysplit`	How to split multiple the visualization. For more information, see Multiple y-axes.
`ytitle`	The title of the y-axis (of type `string`).
`anomalycolumns`	Comma-delimited list of columns, which will be considered as anomaly series and displayed as points on the chart

`ysplit` property

This visualization supports splitting into multiple y-axis values. The supported values of this property are:

`ysplit`	**Description**
`none`	A single y-axis is displayed for all series data. (Default)
`axes`	A single chart is displayed with multiple y-axes (one per series).
`panels`	One chart is rendered for each `ycolumn` value (up to some limit).

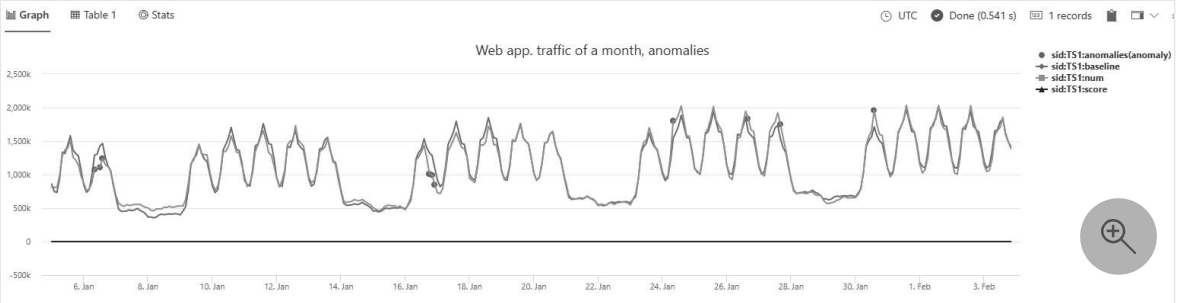
Example

Run the query

Kusto

```
let min_t = datetime(2017-01-05);
let max_t = datetime(2017-02-03 22:00);
let dt = 2h;
demo_make_series2
| make-series num=avg(num) on TimeStamp from min_t to max_t step dt by sid
| where sid == 'TS1'   //  select a single time series for a cleaner
visualization
```

```
| extend (anomalies, score, baseline) = series_decompose_anomalies(num, 1.5, -1, 'linefit')
| render anomalychart with(anomalycolumns=anomalies, title='Web app. traffic of a month, anomalies') //use "| render anomalychart with anomalycolumns=anomalies" to render the anomalies as bold points on the series charts.
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Area chart

Article • 02/28/2023

The area chart visual shows a time-series relationship. The first column of the query should be numeric and is used as the x-axis. Other numeric columns are the y-axes. Unlike line charts, area charts also visually represent volume. Area charts are ideal for indicating the change among different data sets.

> ⓘ Note
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `areachart` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`kind`	Further elaboration of the visualization kind. For more information, see kind property.
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.

PropertyName	*PropertyValue*
`ymin`	The minimum value to be displayed on Y-axis.
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

`kind` property

This visualization can be further elaborated by providing the `kind` property. The supported values of this property are:

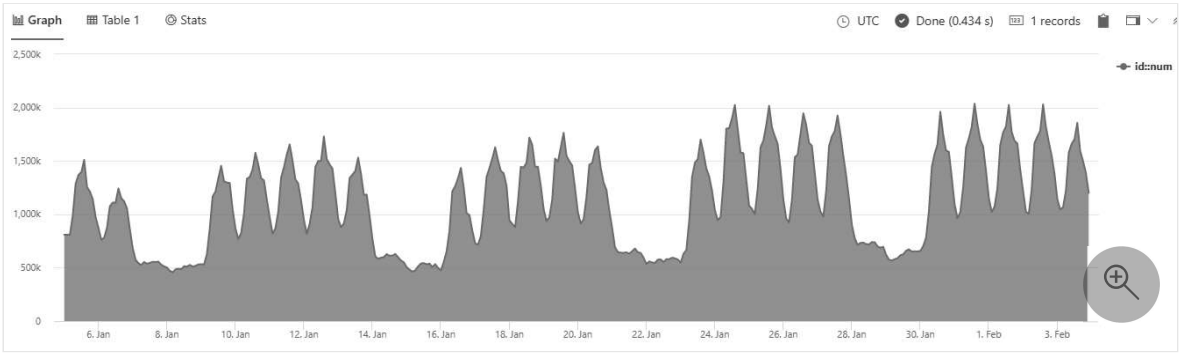
`kind` value	Description
`default`	Each "area" stands on its own.
`unstacked`	Same as `default`.
`stacked`	Stack "areas" to the right.
`stacked100`	Stack "areas" to the right and stretch each one to the same width as the others.

Example

Run the query

Kusto

```
demo_series3
| render areachart
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Bar chart

Article • 02/28/2023

The bar chart visual needs a minimum of two columns in the query result. By default, the first column is used as the y-axis. This column can contain text, datetime, or numeric data types. The other columns are used as the x-axis and contain numeric data types to be displayed as horizontal lines. Bar charts are used mainly for comparing numeric and nominal discrete values, where the length of each line represents its value.

> ⓘ Note
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `barchart` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`kind`	Further elaboration of the visualization kind. For more information, see kind property.
`legend`	Whether to display a legend or not (`visible` or `hidden`).

PropertyName	*PropertyValue*
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

`kind` property

This visualization can be further elaborated by providing the `kind` property. The supported values of this property are:

`kind` value	Description
`default`	Each "bar" stands on its own.
`unstacked`	Same as `default`.
`stacked`	Stack "bars".
`stacked100`	Stack "bars" and stretch each one to the same width as the others.

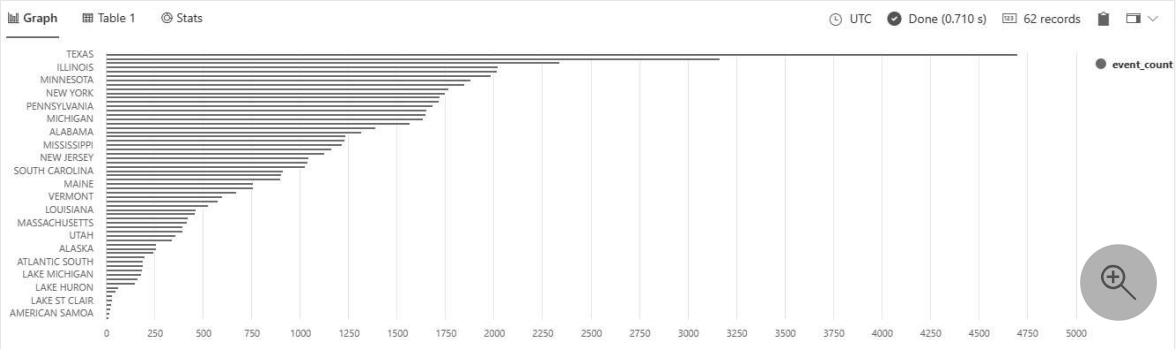
Example

Run the query

Kusto

```
StormEvents
| summarize event_count=count() by State
```

```
| where event_count > 10
| project State, event_count
| render barchart
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Card

Article • 02/28/2023

The card visual only shows one element. If there are multiple columns and rows in the output, the first result record is treated as set of scalar values and shows as a card.

> **Note**
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `card` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`title`	The title of the visualization (of type `string`).

Example

Run the query

Kusto

```
StormEvents
| where State=="VIRGINIA" and EventType=="Flood"
```

```
| count
| render card with (title="Floods in Virginia")
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Column chart

Article • 02/28/2023

The column chart visual needs a minimum of two columns in the query result. By default, the first column is used as the x-axis. This column can contain text, datetime, or numeric data types. The other columns are used as the y-axis and contain numeric data types to be displayed as vertical lines. Column charts are used for comparing specific sub category items in a main category range, where the length of each line represents its value.

> ⓘ **Note**
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `columnchart` [`with` (*propertyName* `=` *propertyValue* [`,` ...])]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`kind`	Further elaboration of the visualization kind. For more information, see kind property.
`legend`	Whether to display a legend or not (`visible` or `hidden`).

PropertyName	*PropertyValue*
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

`kind` property

This visualization can be further elaborated by providing the `kind` property. The supported values of this property are:

`kind` value	Definition
`default`	Each "column" stands on its own.
`unstacked`	Same as `default`.
`stacked`	Stack "columns" one atop the other.
`stacked100`	Stack "columns" and stretch each one to the same height as the others.

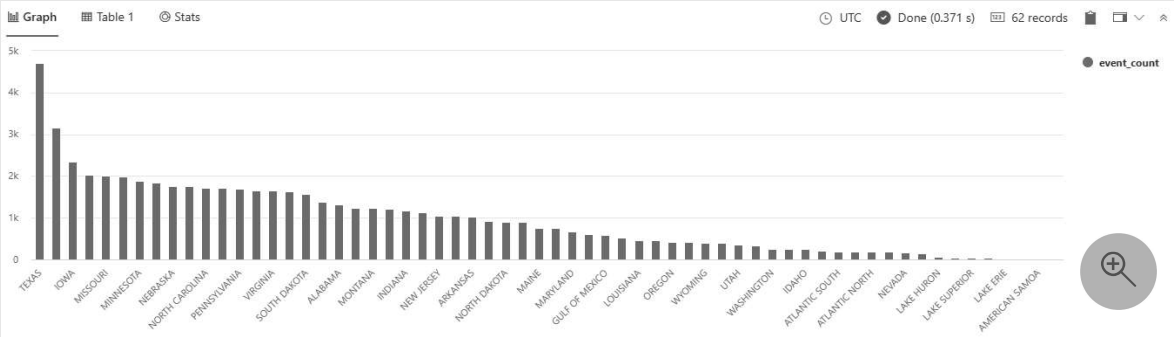
Example

Run the query

Kusto

```
StormEvents
| summarize event_count=count() by State
```

```
| where event_count > 10
| project State, event_count
| render columnchart
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Ladder chart

Article • 02/28/2023

Last two columns are the x-axis, other columns are y-axis

> ⓘ Note
>
> - This visualization can only be used in the context of the **render operator**.
> - This visualization can be used in Kusto.Explorer but is not available in the Azure Data Explorer web UI.

Syntax

T `|` `render` `ladderchart` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.
`ymax`	The maximum value to be displayed on Y-axis.

PropertyName	*PropertyValue*
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

Feedback

Line chart

Article • 02/28/2023

The line chart visual is the most basic type of chart. The first column of the query should be numeric and is used as the x-axis. Other numeric columns are the y-axes. Line charts track changes over short and long periods of time. When smaller changes exist, line graphs are more useful than bar graphs.

> ⓘ Note
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `linechart` [`with` `(` *propertyName* `=` *propertyValue* [`,` ...] `)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.

PropertyName	*PropertyValue*
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ysplit`	How to split multiple the visualization. For more information, see Multiple y-axes.
`ytitle`	The title of the y-axis (of type `string`).

`ysplit` property

This visualization supports splitting into multiple y-axis values:

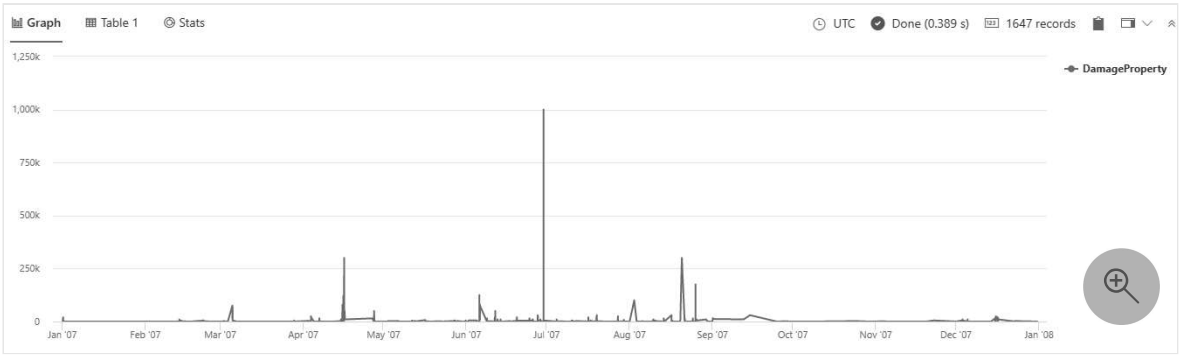
`ysplit`	**Description**
`none`	A single y-axis is displayed for all series data. (Default)
`axes`	A single chart is displayed with multiple y-axes (one per series).
`panels`	One chart is rendered for each `ycolumn` value (up to some limit).

Example

Run the query

Kusto

```
StormEvents
| where State=="VIRGINIA"
| project StartTime, DamageProperty
| render linechart
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Pie chart

Article • 02/28/2023

The pie chart visual needs a minimum of two columns in the query result. By default, the first column is used as the color axis. This column can contain text, datetime, or numeric data types. Other columns will be used to determine the size of each slice and contain numeric data types. Pie charts are used for presenting a composition of categories and their proportions out of a total.

The pie chart visual can also be used in the context of Geospatial visualizations.

> ⓘ **Note**
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `piechart` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`kind`	Further elaboration of the visualization kind. For more information, see kind property.
`legend`	Whether to display a legend or not (`visible` or `hidden`).

PropertyName	*PropertyValue*
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

`kind` property

This visualization can be further elaborated by providing the `kind` property. The supported values of this property are:

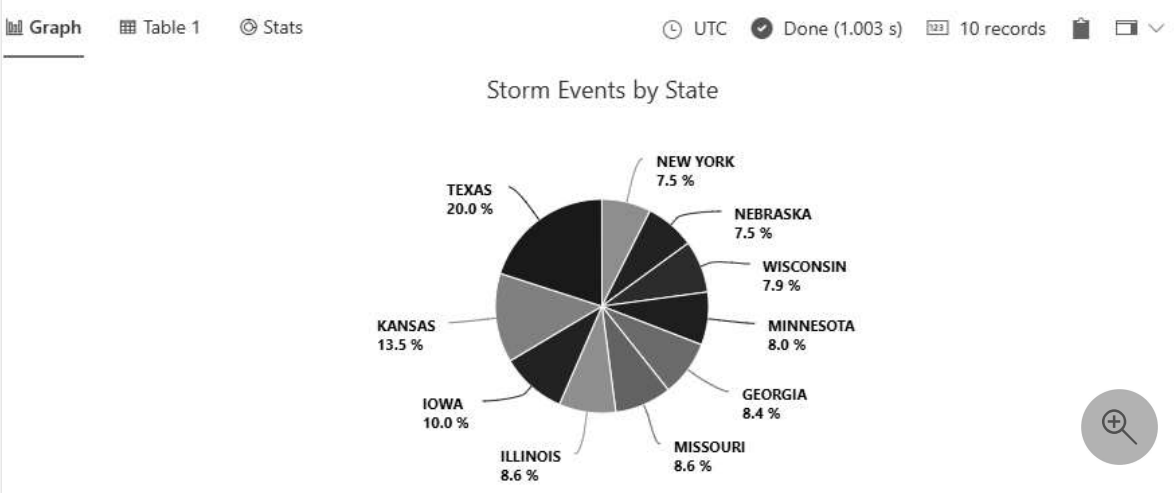
`kind` value	Description
`map`	Expected columns are [Longitude, Latitude] or GeoJSON point, color-axis and numeric. Supported in Kusto Explorer desktop. For more information, see Geospatial visualizations

Example

Run the query

Kusto

```
StormEvents
| summarize statecount=count() by State
| sort by statecount
| limit 10
| render piechart with(title="Storm Events by State")
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Pivot chart

Article • 05/24/2023

Displays a pivot table and chart. You can interactively select data, columns, rows and various chart types.

> ⓘ Note
>
> - This visualization can only be used in the context of the **render operator**.
> - This visualization can be used in Kusto.Explorer but is not available in the Azure Data Explorer web UI.

Syntax

T `|` `render` `pivotchart` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.

PropertyName	*PropertyValue*
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

Feedback

Scatter chart

Article • 02/28/2023

In a scatter chart visual, the first column is the x-axis and should be a numeric column. Other numeric columns are y-axes. Scatter plots are used to observe relationships between variables. The scatter chart visual can also be used in the context of Geospatial visualizations.

> ⓘ Note
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `scatterchart` [`with` (*propertyName* `=` *propertyValue* [`,` ...])]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`kind`	Further elaboration of the visualization kind. For more information, see kind property.
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.

PropertyName	*PropertyValue*
`ymin`	The minimum value to be displayed on Y-axis.
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

`kind` property

This visualization can be further elaborated by providing the `kind` property. The supported values of this property are:

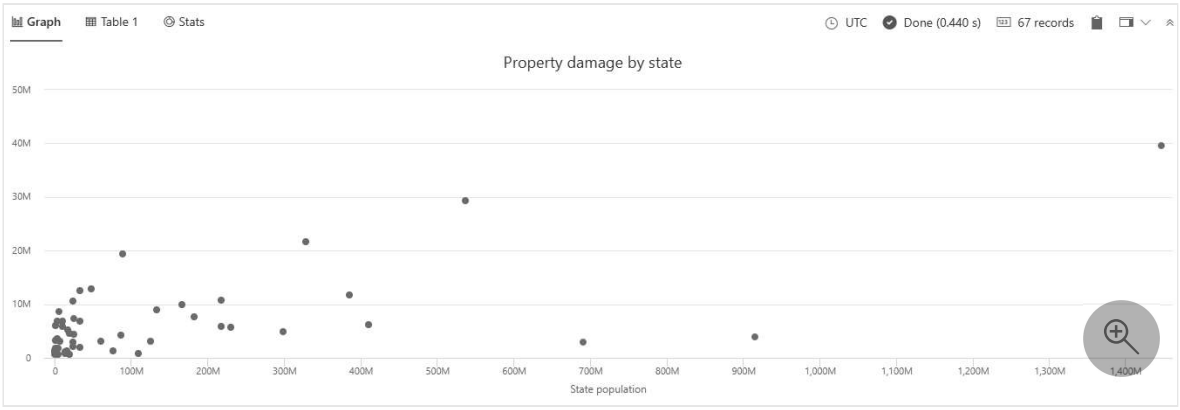
`kind` value	Description
`map`	Expected columns are [Longitude, Latitude] or GeoJSON point. Series column is optional. For more information, see Geospatial visualizations.

Example

Run the query

Kusto

```
StormEvents
| summarize sum(DamageProperty)by State
| lookup PopulationData on State
| project-away State
| render scatterchart with (xtitle="State population", title="Property
damage by state", legend=hidden)
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Stacked area chart

Article • 02/28/2023

The stacked area chart visual shows a continuous relationship. This visual is similar to the Area chart, but shows the area under each element of a series. The first column of the query should be numeric and is used as the x-axis. Other numeric columns are the y-axes. Unlike line charts, area charts also visually represent volume. Area charts are ideal for indicating the change among different data sets.

> ⓘ Note
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `stackedareachart` [`with` (*propertyName* `=` *propertyValue* [`,` ...])]

Supported parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.

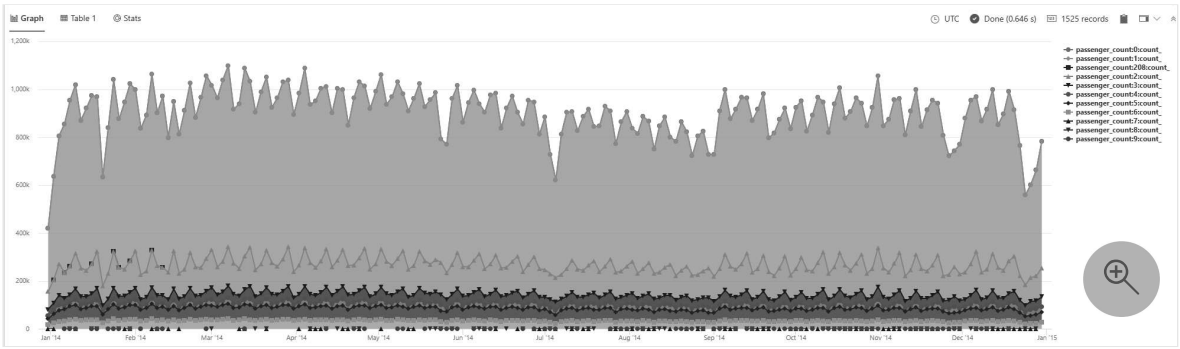
PropertyName	*PropertyValue*
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

Example

Run the query

Kusto

```
nyc_taxi
| summarize count() by passenger_count, bin(pickup_datetime, 2d)
| render stackedareachart with (xcolumn=pickup_datetime,
series=passenger_count)
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Table

Article • 02/28/2023

Default - results are shown as a table.

> ⓘ Note
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `table` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).

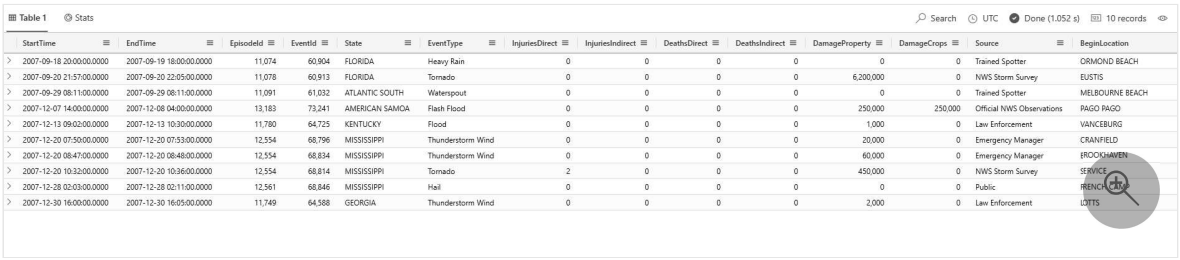
PropertyName	*PropertyValue*
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

Example

Run the query

Kusto

```
StormEvents
| take 10
| render table
```



StartTime	EndTime	EpisodeId	EventId	State	EventType	InjuriesDirect	InjuriesIndirect	DeathsDirect	DeathsIndirect	DamageProperty	DamageCrops	Source	BeginLocation
2007-09-18 20:00:00.0000	2007-09-19 18:00:00.0000	11,074	60,904	FLORIDA	Heavy Rain	0	0	0	0	0	0	Trained Spotter	ORMOND BEACH
2007-09-20 21:57:00.0000	2007-09-20 22:05:00.0000	11,078	60,913	FLORIDA	Tornado	0	0	0	0	6,200,000	0	NWS Storm Survey	EUSTIS
2007-09-29 08:11:00.0000	2007-09-29 08:11:00.0000	11,091	61,032	ATLANTIC SOUTH	Waterspout	0	0	0	0	0	0	Trained Spotter	MELBOURNE BEACH
2007-12-07 14:00:00.0000	2007-12-08 04:00:00.0000	13,183	73,241	AMERICAN SAMOA	Flash Flood	0	0	0	0	250,000	250,000	Official NWS Observations	PAGO PAGO
2007-12-13 09:02:00.0000	2007-12-13 10:30:00.0000	11,780	64,725	KENTUCKY	Flood	0	0	0	0	1,000	0	Law Enforcement	VANCEBURG
2007-12-20 07:50:00.0000	2007-12-20 07:53:00.0000	12,554	68,796	MISSISSIPPI	Thunderstorm Wind	0	0	0	0	20,000	0	Emergency Manager	CRANFIELD
2007-12-20 08:47:00.0000	2007-12-20 08:48:00.0000	12,554	68,834	MISSISSIPPI	Thunderstorm Wind	0	0	0	0	60,000	0	Emergency Manager	BROOKHAVEN
2007-12-20 10:32:00.0000	2007-12-20 10:36:00.0000	12,554	68,814	MISSISSIPPI	Tornado	2	0	0	0	450,000	0	NWS Storm Survey	SERVICE
2007-12-28 02:03:00.0000	2007-12-28 02:11:00.0000	12,561	68,846	MISSISSIPPI	Hail	0	0	0	0	0	0	Public	FRENCH CAMP
2007-12-30 16:00:00.0000	2007-12-30 16:05:00.0000	11,749	64,588	GEORGIA	Thunderstorm Wind	0	0	0	0	2,000	0	Law Enforcement	LOTTS

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Time chart

Article • 02/28/2023

A time chart visual is a type of line graph. The first column of the query is the x-axis, and should be a datetime. Other numeric columns are y-axes. One string column values are used to group the numeric columns and create different lines in the chart. Other string columns are ignored. The time chart visual is similar to a line chart except the x-axis is always time.

> ⓘ Note
>
> This visualization can only be used in the context of the **render operator**.

Syntax

T `|` `render` `timechart` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.

PropertyName	*PropertyValue*
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ysplit`	How to split multiple the visualization. For more information, see Multiple y-axes.
`ytitle`	The title of the y-axis (of type `string`).

`ysplit` property

This visualization supports splitting into multiple y-axis values:

`ysplit`	**Description**
`none`	A single y-axis is displayed for all series data. (Default)
`axes`	A single chart is displayed with multiple y-axes (one per series).
`panels`	One chart is rendered for each `ycolumn` value (up to some limit).

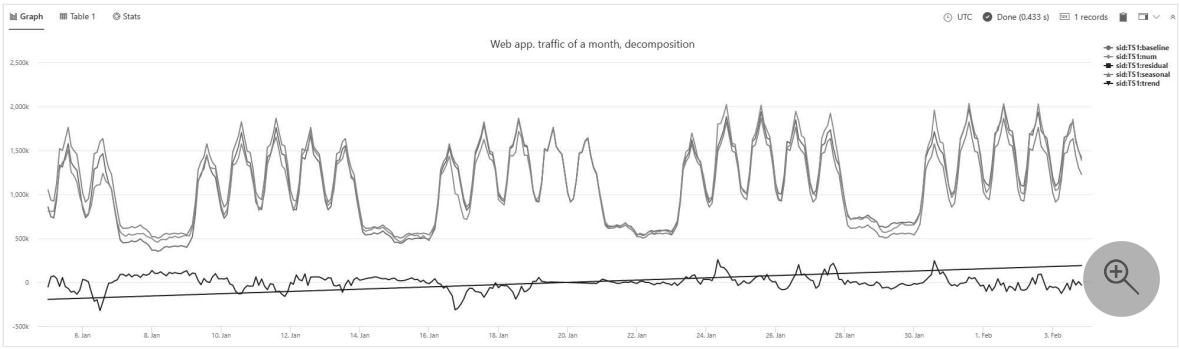
Example

Run the query

Kusto

```
let min_t = datetime(2017-01-05);
let max_t = datetime(2017-02-03 22:00);
let dt = 2h;
demo_make_series2
| make-series num=avg(num) on TimeStamp from min_t to max_t step dt by sid
| where sid == 'TS1'   //  select a single time series for a cleaner
visualization
| extend (baseline, seasonal, trend, residual) = series_decompose(num, -1,
```

```
'linefit')  //  decomposition of a set of time series to seasonal, trend,
residual, and baseline (seasonal+trend)
| render timechart with(title='Web app. traffic of a month, decomposition')
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Time pivot

Article • 02/28/2023

The time pivot visualization is an interactive navigation over the events time-line pivoting on time axis.

> ⓘ Note
>
> - This visualization can only be used in the context of the **render operator**.
> - This visualization can be used in Kusto.Explorer but is not available in the Azure Data Explorer web UI.

Syntax

T `|` `render` `timepivot` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

All properties are optional.

PropertyName	***PropertyValue***
`accumulate`	Whether the value of each measure gets added to all its predecessors. (`true` or `false`)
`legend`	Whether to display a legend or not (`visible` or `hidden`).
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.
`ymin`	The minimum value to be displayed on Y-axis.

PropertyName	*PropertyValue*
`ymax`	The maximum value to be displayed on Y-axis.
`title`	The title of the visualization (of type `string`).
`xaxis`	How to scale the x-axis (`linear` or `log`).
`xcolumn`	Which column in the result is used for the x-axis.
`xtitle`	The title of the x-axis (of type `string`).
`yaxis`	How to scale the y-axis (`linear` or `log`).
`ycolumns`	Comma-delimited list of columns that consist of the values provided per value of the x column.
`ytitle`	The title of the y-axis (of type `string`).

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Treemap

Article • 05/24/2023

Treemaps display hierarchical data as a set of nested rectangles. Each level of the hierarchy is represented by a colored rectangle (branch) containing smaller rectangles (leaves).

> ⓘ Note
>
> - This visualization can only be used in the context of the **render operator**.
> - This visualization can be used in Kusto.Explorer but is not available in the Azure Data Explorer web UI.

Syntax

T `|` `render` `treemap` [`with` `(`*propertyName* `=` *propertyValue* [`,` ...]`)`]

Parameters

Name	Type	Required	Description
T	string	✓	Input table name.
propertyName, *propertyValue*	string		A comma-separated list of key-value property pairs. See supported properties.

Supported properties

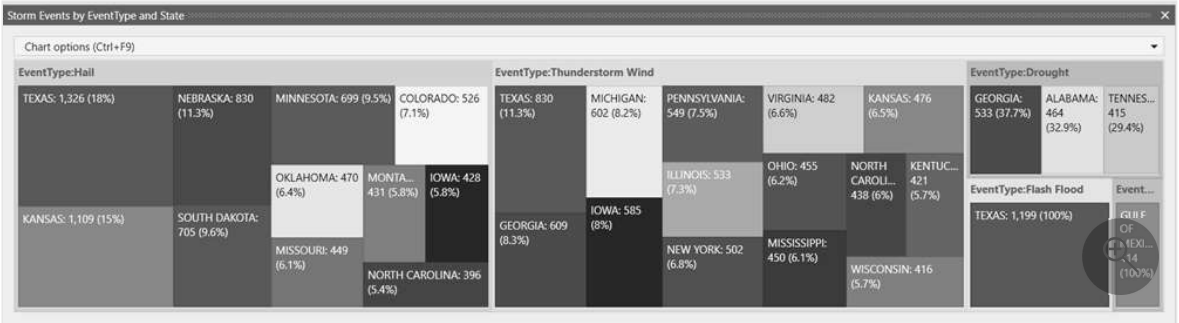
All properties are optional.

PropertyName	*PropertyValue*
`series`	Comma-delimited list of columns whose combined per-record values define the series that record belongs to.

Example

```
Kusto
```

```
StormEvents
| summarize StormEvents=count() by EventType, State
| sort by StormEvents
| limit 30
| render treemap with(title="Storm Events by EventType and State")
```



Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

sample operator

Article • 01/31/2023

Returns up to the specified number of random rows from the input table.

> ⓘ Note
>
> - `sample` is geared for speed rather than even distribution of values. Specifically, it means that it will not produce 'fair' results if used after operators that union 2 data sets of different sizes (such as a `union` or `join` operators). It's recommended to use `sample` right after the table reference and filters.
> - `sample` is a non-deterministic operator, and will return different result set each time it is evaluated during the query. For example, the following query yields two different rows (even if one would expect to return the same row twice).

Syntax

T `| sample` *NumberOfRows*

Parameters

Name	Type	Required	Description
T	string	✓	The input tabular expression.
NumberOfRows	int, long, or real	✓	The number of rows to return. You can specify any numeric expression.

Examples

Run the query

Kusto

```
let _data = range x from 1 to 100 step 1;
let _sample = _data | sample 1;
union (_sample), (_sample)
```

Output

x
83
3

To ensure that in example above `_sample` is calculated once, one can use materialize() function:

Run the query

Kusto

```
let _data = range x from 1 to 100 step 1;
let _sample = materialize(_data | sample 1);
union (_sample), (_sample)
```

Output

x
34
34

To sample a certain percentage of your data (rather than a specified number of rows), you can use

Run the query

Kusto

```
StormEvents | where rand() < 0.1
```

To sample keys rather than rows (for example - sample 10 Ids and get all rows for these Ids) you can use sample-distinct in combination with the `in` operator.

Run the query

Kusto

```
let sampleEpisodes = StormEvents | sample-distinct 10 of EpisodeId;
StormEvents
```

```
| where EpisodeId in (sampleEpisodes)
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

sample-distinct operator

Article • 01/31/2023

Returns a single column that contains up to the specified number of distinct values of the requested column.

The operator tries to return an answer as quickly as possible rather than trying to make a fair sample.

Syntax

T `| sample-distinct` *NumberOfValues* `of` *ColumnName*

Parameters

Name	Type	Required	Description
T	string	✓	The input tabular expression.
NumberOfValues	int, long, or real	✓	The number distinct values of *T* to return. You can specify any numeric expression.
ColumnName	string	✓	The name of the column from which to sample.

> **Tip**
>
> - Use the **top-hitters** operator to get the top values.
> - Refer to the **sample operator** to sample data rows.

Examples

Get 10 distinct values from a population

Run the query

Kusto

```
StormEvents | sample-distinct 10 of EpisodeId
```

Sample a population and do further computation without exceeding the query limits in the summarize

Run the query

Kusto

```
let sampleEpisodes = StormEvents | sample-distinct 10 of EpisodeId;
StormEvents
| where EpisodeId in (sampleEpisodes)
| summarize totalInjuries=sum(InjuriesDirect) by EpisodeId
```

Feedback