Scalar function types at a glance

Article • 03/16/2023

This article lists all available scalar functions grouped by type. For aggregation functions, see Aggregation function types.

Binary functions

Function Name	Description
binary_and()	Returns a result of the bitwise and operation between two values.
binary_not()	Returns a bitwise negation of the input value.
binary_or()	Returns a result of the bitwise or operation of the two values.
binary_shift_left()	Returns binary shift left operation on a pair of numbers: a << n.
binary_shift_right()	Returns binary shift right operation on a pair of numbers: a >> n.
binary_xor()	Returns a result of the bitwise xor operation of the two values.
bitset_count_ones()	Returns the number of set bits in the binary representation of a number.

Conversion functions

Function Name	Description
tobool()	Convert inputs to boolean (signed 8-bit) representation.
todatetime()	Converts input to datetime scalar.
todouble()	Converts the input to a value of type real.
tostring()	Converts input to a string representation.
totimespan()	Converts input to timespan scalar.

DateTime/timespan functions

Function Name	Description
ago()	Subtracts the given timespan from the current UTC clock time.

Function Name	Description
datetime_add()	Calculates a new datetime from a specified datepart multiplied by a specified amount, added to a specified datetime.
datetime_diff()	Returns the end of the year containing the date, shifted by an offset, if provided.
datetime_local_to_utc()	Converts local datetime to UTC datetime using a time-zone specification.
datetime_part()	Extracts the requested date part as an integer value.
datetime_utc_to_local()	Converts UTC datetimgoe to local datetime using a time-zone specification.
dayofmonth()	Returns the integer number representing the day number of the given month.
dayofweek()	Returns the integer number of days since the preceding Sunday, as a timespan.
dayofyear()	Returns the integer number represents the day number of the given year.
endofday()	Returns the end of the day containing the date, shifted by an offset, if provided.
endofmonth()	Returns the end of the month containing the date, shifted by an offset, if provided.
endofweek()	Returns the end of the week containing the date, shifted by an offset, if provided.
endofyear()	Returns the end of the year containing the date, shifted by an offset, if provided.
format_datetime()	Formats a datetime parameter based on the format pattern parameter.
format_timespan()	Formats a format-timespan parameter based on the format pattern parameter.
getyear()	Returns the year part of the datetime argument.
hourofday()	Returns the integer number representing the hour number of the given date.
make_datetime()	Creates a datetime scalar value from the specified date and time.

Function Name	Description
make_timespan()	Creates a timespan scalar value from the specified time period.
monthofyear()	Returns the integer number that represents the month number of the given year.
now()	Returns the current UTC clock time, optionally offset by a given timespan.
startofday()	Returns the start of the day containing the date, shifted by an offset, if provided.
startofmonth()	Returns the start of the month containing the date, shifted by an offset, if provided.
startofweek()	Returns the start of the week containing the date, shifted by an offset, if provided.
startofyear()	Returns the start of the year containing the date, shifted by an offset, if provided.
todatetime()	Converts input to datetime scalar.
totimespan()	Converts input to timespan scalar.
unixtime_microseconds_todatetime()	Converts unix-epoch microseconds to UTC datetime.
unixtime_milliseconds_todatetime()	Converts unix-epoch milliseconds to UTC datetime.
unixtime_nanoseconds_todatetime()	Converts unix-epoch nanoseconds to UTC datetime.
unixtime_seconds_todatetime()	Converts unix-epoch seconds to UTC datetime.
weekofyear()	Returns an integer representing the week number.

Dynamic/array functions

Function Name	Description
array_concat()	Concatenates a number of dynamic arrays to a single array.
array_iff()	Applies element-wise iif function on arrays.
array_index_of()	Searches the array for the specified item, and returns its position.
array_length()	Calculates the number of elements in a dynamic array.
array_reverse()	Reverses the order of the elements in a dynamic array.

Function Name	Description
array_rotate_left()	Rotates values inside a dynamic array to the left.
array_rotate_right()	Rotates values inside a dynamic array to the right.
array_shift_left()	Shifts values inside a dynamic array to the left.
array_shift_right()	Shifts values inside a dynamic array to the right.
array_slice()	Extracts a slice of a dynamic array.
array_sort_asc()	Sorts a collection of arrays in ascending order.
array_sort_desc()	Sorts a collection of arrays in descending order.
array_split()	Builds an array of arrays split from the input array.
array_sum()	Calculates the sum of a dynamic array.
bag_has_key()	Checks whether a dynamic bag column contains a given key.
bag_keys()	Enumerates all the root keys in a dynamic property-bag object.
bag_merge()	Merges dynamic property-bags into a dynamic property-bag with all properties merged.
bag_pack()	Creates a dynamic object (property bag) from a list of names and values.
bag_pack_columns()	Creates a dynamic object (property bag) from a list of columns.
bag_remove_keys()	Removes keys and associated values from a dynamic property-bag.
bag_set_key()	Sets a given key to a given value in a dynamic property-bag.
jaccard_index()	Computes the Jaccard index of two sets.
pack_all()	Creates a dynamic object (property bag) from all the columns of the tabular expression.
pack_array()	Packs all input values into a dynamic array.
repeat()	Generates a dynamic array holding a series of equal values.
set_difference()	Returns an array of the set of all distinct values that are in the first array but aren't in other arrays.
set_has_element()	Determines whether the specified array contains the specified element.
set_intersect()	Returns an array of the set of all distinct values that are in all arrays.
set_union()	Returns an array of the set of all distinct values that are in any of provided arrays.

Function Name	Description
treepath()	Enumerates all the path expressions that identify leaves in a dynamic object.
zip()	The zip function accepts any number of dynamic arrays. Returns an array whose elements are each an array with the elements of the input arrays of the same index.

Window scalar functions

Function Name	Description
next()	For the serialized row set, returns a value of a specified column from the later row according to the offset.
prev()	For the serialized row set, returns a value of a specified column from the earlier row according to the offset.
row_cumsum()	Calculates the cumulative sum of a column.
row_number()	Returns a row's number in the serialized row set - consecutive numbers starting from a given index or from 1 by default.
row_rank_dense()	Returns a row's dense rank in the serialized row set.
row_rank_min()	Returns a row's minimal rank in the serialized row set.

Flow control functions

Function Name	Description
toscalar()	Returns a scalar constant value of the evaluated expression.

Mathematical functions

Function Name	Description
abs()	Calculates the absolute value of the input.
acos()	Returns the angle whose cosine is the specified number (the inverse operation of cos()).

Function Name	Description
asin()	Returns the angle whose sine is the specified number (the inverse operation of sin()).
atan()	Returns the angle whose tangent is the specified number (the inverse operation of tan()).
atan2()	Calculates the angle, in radians, between the positive x-axis and the ray from the origin to the point (y, x).
beta_cdf()	Returns the standard cumulative beta distribution function.
beta_inv()	Returns the inverse of the beta cumulative probability beta density function.
beta_pdf()	Returns the probability density beta function.
cos()	Returns the cosine function.
cot()	Calculates the trigonometric cotangent of the specified angle, in radians.
degrees()	Converts angle value in radians into value in degrees, using formula degrees = (180 / PI) * angle-in-radians.
exp()	The base-e exponential function of x, which is e raised to the power x: e^x.
exp10()	The base-10 exponential function of x, which is 10 raised to the power x: 10^x.
exp2()	The base-2 exponential function of x, which is 2 raised to the power x: 2^x.
gamma()	Computes gamma function.
isfinite()	Returns whether input is a finite value (isn't infinite or NaN).
isinf()	Returns whether input is an infinite (positive or negative) value.
isnan()	Returns whether input is Not-a-Number (NaN) value.
log()	Returns the natural logarithm function.
log10()	Returns the common (base-10) logarithm function.
log2()	Returns the base-2 logarithm function.
loggamma()	Computes log of absolute value of the gamma function.
not()	Reverses the value of its bool argument.
pi()	Returns the constant value of Pi (π).
pow()	Returns a result of raising to power.

Function Name	Description
radians()	Converts angle value in degrees into value in radians, using formula radians = (PI / 180) * angle-in-degrees.
rand()	Returns a random number.
range()	Generates a dynamic array holding a series of equally spaced values.
round()	Returns the rounded source to the specified precision.
sign()	Sign of a numeric expression.
sin()	Returns the sine function.
sqrt()	Returns the square root function.
tan()	Returns the tangent function.
welch_test()	Computes the p-value of the Welch-test function.

Metadata functions

Function Name	Description
column_ifexists()	Takes a column name as a string and a default value. Returns a reference to the column if it exists, otherwise - returns the default value.
current_cluster_endpoint()	Returns the current cluster running the query.
current_database()	Returns the name of the database in scope.
current_principal()	Returns the current principal running this query.
current_principal_details()	Returns details of the principal running the query.
current_principal_is_member_of()	Checks group membership or principal identity of the current principal running the query.
cursor_after()	Used to access to the records that were ingested after the previous value of the cursor.
estimate_data_size()	Returns an estimated data size of the selected columns of the tabular expression.
extent_id()	Returns a unique identifier that identifies the data shard ("extent") that the current record resides in.

Function Name	Description
extent_tags()	Returns a dynamic array with the tags of the data shard ("extent") that the current record resides in.
ingestion_time()	Retrieves the record's $IngestionTime hidden datetime column, or null.

Rounding functions

Function Name	Description
bin()	Rounds values down to an integer multiple of a given bin size.
bin_at()	Rounds values down to a fixed-size "bin", with control over the bin's starting point. (See also bin function.)
ceiling()	Calculates the smallest integer greater than, or equal to, the specified numeric expression.

Conditional functions

Function Name	Description
case()	Evaluates a list of predicates and returns the first result expression whose predicate is satisfied.
coalesce()	Evaluates a list of expressions and returns the first non-null (or non-empty for string) expression.
iff()	Evaluate the first argument (the predicate), and returns the value of either the second or third arguments, depending on whether the predicate evaluated to true (second) or false (third).
max_of()	Returns the maximum value of several evaluated numeric expressions.
min_of()	Returns the minimum value of several evaluated numeric expressions.

Series element-wise functions

Function Name	Description
series_abs()	Calculates the element-wise absolute value of the numeric series input.

Function Name	Description
series_acos()	Calculates the element-wise arccosine function of the numeric series input.
series_add()	Calculates the element-wise addition of two numeric series inputs.
series_asin()	Calculates the element-wise arcsine function of the numeric series input.
series_atan()	Calculates the element-wise arctangent function of the numeric series input.
series_ceiling()	Calculates the element-wise ceiling function of the numeric series input.
series_cos()	Calculates the element-wise cosine function of the numeric series input.
series_divide()	Calculates the element-wise division of two numeric series inputs.
series_equals()	Calculates the element-wise equals (`==`) logic operation of two numeric series inputs.
series_exp()	Calculates the element-wise base-e exponential function (e^x) of the numeric series input.
series_floor()	Calculates the element-wise floor function of the numeric series input.
series_greater()	Calculates the element-wise greater (`>`) logic operation of two numeric series inputs.
series_greater_equals()	Calculates the element-wise greater or equals (`>=`) logic operation of two numeric series inputs.
series_less()	Calculates the element-wise less (`<`) logic operation of two numeric series inputs.
series_less_equals()	Calculates the element-wise less or equal (`<=`) logic operation of two numeric series inputs.
series_log()	Calculates the element-wise natural logarithm function (base-e) of the numeric series input.
series_multiply()	Calculates the element-wise multiplication of two numeric series inputs.
series_not_equals()	Calculates the element-wise not equals (`!=`) logic operation of two numeric series inputs.
series_pow()	Calculates the element-wise power of two numeric series inputs.
series_sign()	Calculates the element-wise sign of the numeric series input.
series_sin()	Calculates the element-wise sine function of the numeric series input.

Function Name	Description
series_subtract()	Calculates the element-wise subtraction of two numeric series inputs.
series_tan()	Calculates the element-wise tangent function of the numeric series input.

Series processing functions

Function Name	Description
series_decompose()	Does a decomposition of the series into components.
series_decompose_anomalies()	Finds anomalies in a series based on series decomposition.
series_decompose_forecast()	Forecast based on series decomposition.
series_fill_backward()	Performs backward fill interpolation of missing values in a series.
series_fill_const()	Replaces missing values in a series with a specified constant value.
series_fill_forward()	Performs forward fill interpolation of missing values in a series.
series_fill_linear()	Performs linear interpolation of missing values in a series.
series_fft()	Applies the Fast Fourier Transform (FFT) on a series.
series_fir()	Applies a Finite Impulse Response filter on a series.
series_fit_2lines()	Applies two segments linear regression on a series, returning multiple columns.
series_fit_2lines_dynamic()	Applies two segments linear regression on a series, returning dynamic object.
series_fit_line()	Applies linear regression on a series, returning multiple columns.
series_fit_line_dynamic()	Applies linear regression on a series, returning dynamic object.
series_fit_poly()	Applies polynomial regression on a series, returning multiple columns.
series_ifft()	Applies the Inverse Fast Fourier Transform (IFFT) on a series.
series_iir()	Applies an Infinite Impulse Response filter on a series.
series_outliers()	Scores anomaly points in a series.
series_pearson_correlation()	Calculates the Pearson correlation coefficient of two series.

Function Name	Description
series_periods_detect()	Finds the most significant periods that exist in a time series.
series_periods_validate()	Checks whether a time series contains periodic patterns of given lengths.
series_seasonal()	Finds the seasonal component of the series.
series_stats()	Returns statistics for a series in multiple columns.
series_stats_dynamic()	Returns statistics for a series in dynamic object.

String functions

Function Name	Description
base64_encode_tostring()	Encodes a string as base64 string.
base64_encode_fromguid()	Encodes a GUID as base64 string.
base64_decode_tostring()	Decodes a base64 string to a UTF-8 string.
base64_decode_toarray()	Decodes a base64 string to an array of long values.
base64_decode_toguid()	Decodes a base64 string to a GUID.
countof()	Counts occurrences of a substring in a string. Plain string matches may overlap; regex matches don't.
extract()	Get a match for a regular expression from a text string.
extract_all()	Get all matches for a regular expression from a text string.
extract_json()	Get a specified element out of a JSON text using a path expression.
has_any_index()	Searches the string for items specified in the array and returns the position of the first item found in the string.
indexof()	Function reports the zero-based index of the first occurrence of a specified string within input string.
isempty()	Returns true if the argument is an empty string or is null.
isnotempty()	Returns true if the argument isn't an empty string or a null.
isnotnull()	Returns true if the argument is not null.
isnull()	Evaluates its sole argument and returns a bool value indicating if the argument evaluates to a null value.

Function Name	Description
parse_command_line()	Parses a Unicode command line string and returns an array of the command line arguments.
parse_csv()	Splits a given string representing comma-separated values and returns a string array with these values.
parse_ipv4()	Converts input to long (signed 64-bit) number representation.
parse_ipv4_mask()	Converts input string and IP-prefix mask to long (signed 64-bit) number representation.
parse_ipv6()	Converts IPv6 or IPv4 string to a canonical IPv6 string representation.
parse_ipv6_mask()	Converts IPv6 or IPv4 string and netmask to a canonical IPv6 string representation.
parse_json()	Interprets a string as a JSON value and returns the value as dynamic.
parse_url()	Parses an absolute URL string and returns a dynamic object contains all parts of the URL.
parse_urlquery()	Parses a url query string and returns a dynamic object contains the Query parameters.
parse_version()	Converts input string representation of version to a comparable decimal number.
replace_regex()	Replace all regex matches with another string.
replace_string()	Replace all single string matches with a specified string.
replace_strings()	Replace all multiple strings matches with specified strings.
punycode_from_string()	Encodes domain name to Punycode form.
punycode_to_string()	Decodes domain name from Punycode form.
reverse()	Function makes reverse of input string.
split()	Splits a given string according to a given delimiter and returns a string array with the contained substrings.
strcat()	Concatenates between 1 and 64 arguments.
strcat_delim()	Concatenates between 2 and 64 arguments, with delimiter, provided as first argument.
strcmp()	Compares two strings.
strlen()	Returns the length, in characters, of the input string.

Function Name	Description
strrep()	Repeats given string provided number of times (default - 1).
substring()	Extracts a substring from a source string starting from some index to the end of the string.
toupper()	Converts a string to upper case.
translate()	Replaces a set of characters ('searchList') with another set of characters ('replacementList') in a given a string.
trim()	Removes all leading and trailing matches of the specified regular expression.
trim_end()	Removes trailing match of the specified regular expression.
trim_start()	Removes leading match of the specified regular expression.
url_decode()	The function converts encoded URL into a regular URL representation.
url_encode()	The function converts characters of the input URL into a format that can be transmitted over the Internet.

IPv4/IPv6 functions

Function Name	Description
ipv4_compare()	Compares two IPv4 strings.
ipv4_is_in_range()	Checks if IPv4 string address is in IPv4-prefix notation range.
ipv4_is_in_any_range()	Checks if IPv4 string address is any of the IPv4-prefix notation ranges.
ipv4_is_match()	Matches two IPv4 strings.
ipv4_is_private()	Checks if IPv4 string address belongs to a set of private network IPs.
ipv4_netmask_suffix	Returns the value of the IPv4 netmask suffix from IPv4 string address.
parse_ipv4()	Converts input string to long (signed 64-bit) number representation.
parse_ipv4_mask()	Converts input string and IP-prefix mask to long (signed 64-bit) number representation.
ipv4_range_to_cidr_list()	Converts IPv4 address range to a list of CIDR ranges.

Function Name	Description
ipv6_compare()	Compares two IPv4 or IPv6 strings.
ipv6_is_match()	Matches two IPv4 or IPv6 strings.
parse_ipv6()	Converts IPv6 or IPv4 string to a canonical IPv6 string representation.
parse_ipv6_mask()	Converts IPv6 or IPv4 string and netmask to a canonical IPv6 string representation.
format_ipv4()	Parses input with a netmask and returns string representing IPv4 address.
format_ipv4_mask()	Parses input with a netmask and returns string representing IPv4 address as CIDR notation.
ipv6_is_in_range()	Checks if an IPv6 string address is in IPv6-prefix notation range.
ipv6_is_in_any_range()	Checks if an IPv6 string address is in any of the IPv6-prefix notation ranges.
geo_info_from_ip_address()	Retrieves geolocation information about IPv4 or IPv6 addresses.

IPv4 text match functions

Function Name	Description
has_ipv4()	Searches for an IPv4 address in a text.
has_ipv4_prefix()	Searches for an IPv4 address or prefix in a text.
has_any_ipv4()	Searches for any of the specified IPv4 addresses in a text.
has_any_ipv4_prefix()	Searches for any of the specified IPv4 addresses or prefixes in a text.

Type functions

Function Name	Description
gettype()	Returns the runtime type of its single argument.

Scalar aggregation functions

Function Name	Description
dcount_hll()	Calculates the dcount from hll results (which was generated by hll or hll-merge).
hll_merge()	Merges hll results (scalar version of the aggregate version hll-merge()).
percentile_tdigest()	Calculates the percentile result from tdigest results (which was generated by tdigest or merge_tdigest).
percentile_array_tdigest()	Calculates the percentile array result from tdigest results (which was generated by tdigest or merge_tdigest).
percentrank_tdigest()	Calculates the percentage ranking of a value in a dataset.
rank_tdigest()	Calculates relative rank of a value in a set.
merge_tdigest()	Merge tdigest results (scalar version of the aggregate version tdigest-merge()).

Geospatial functions

Function Name	Description
geo_distance_2points()	Calculates the shortest distance between two geospatial coordinates on Earth.
geo_distance_point_to_line()	Calculates the shortest distance between a coordinate and a line or multiline on Earth.
geo_distance_point_to_polygon()	Calculates the shortest distance between a coordinate and a polygon or multipolygon on Earth.
geo_intersects_2lines()	Calculates whether the two lines or multilines intersects.
geo_intersects_2polygons()	Calculates whether the two polygons or multipolygons intersects.
geo_intersects_line_with_polygon()	Calculates whether the line or multiline intersects with polygon or multipolygon.
geo_intersection_2lines()	Calculates the intersection of two lines or multilines.
geo_intersection_2polygons()	Calculates the intersection of two polygons or multipolygons.
geo_intersection_line_with_polygon()	Calculates the intersection of line or multiline with polygon or multipolygon.

Function Name	Description
geo_point_buffer()	Calculates polygon that contains all points within the given radius of the point on Earth.
geo_point_in_circle()	Calculates whether the geospatial coordinates are inside a circle on Earth.
geo_point_in_polygon()	Calculates whether the geospatial coordinates are inside a polygon or a multipolygon on Earth.
geo_point_to_geohash()	Calculates the Geohash string value for a geographic location.
geo_point_to_s2cell()	Calculates the S2 Cell token string value for a geographic location.
geo_point_to_h3cell()	Calculates the H3 Cell token string value for a geographic location.
geo_line_buffer()	Calculates polygon or multipolygon that contains all points within the given radius of the input line or multiline on Earth.
geo_line_centroid()	Calculates the centroid of line or a multiline on Earth.
geo_line_densify()	Converts planar line edges to geodesics by adding intermediate points.
geo_line_length()	Calculates the total length of line or a multiline on Earth.
geo_line_simplify()	Simplifies line or a multiline by replacing nearly straight chains of short edges with a single long edge on Earth.
geo_polygon_area()	Calculates the area of polygon or a multipolygon on Earth.
geo_polygon_buffer()	Calculates polygon or multipolygon that contains all points within the given radius of the input polygon or multipolygon on Earth.
geo_polygon_centroid()	Calculates the centroid of polygon or a multipolygon on Earth.
geo_polygon_densify()	Converts polygon or multipolygon planar edges to geodesics by adding intermediate points.
geo_polygon_perimeter()	Calculates the length of the boundary of polygon or a multipolygon on Earth.
geo_polygon_simplify()	Simplifies polygon or a multipolygon by replacing nearly straight chains of short edges with a single long edge on Earth.

Function Name	Description
geo_polygon_to_s2cells()	Calculates S2 Cell tokens that cover a polygon or multipolygon on Earth. Useful geospatial join tool.
geo_geohash_to_central_point()	Calculates the geospatial coordinates that represent the center of a Geohash rectangular area.
geo_geohash_neighbors()	Calculates the geohash neighbors.
geo_geohash_to_polygon()	Calculates the polygon that represents the geohash rectangular area.
geo_s2cell_to_central_point()	Calculates the geospatial coordinates that represent the center of an S2 Cell.
geo_s2cell_neighbors()	Calculates the S2 cell neighbors.
geo_s2cell_to_polygon()	Calculates the polygon that represents the S2 Cell rectangular area.
geo_h3cell_to_central_point()	Calculates the geospatial coordinates that represent the center of an H3 Cell.
geo_h3cell_neighbors()	Calculates the H3 cell neighbors.
geo_h3cell_to_polygon()	Calculates the polygon that represents the H3 Cell rectangular area.
geo_h3cell_parent()	Calculates the H3 cell parent.
geo_h3cell_children()	Calculates the H3 cell children.
geo_h3cell_level()	Calculates the H3 cell resolution.
geo_h3cell_rings()	Calculates the H3 cell Rings.
geo_simplify_polygons_array()	Simplifies polygons by replacing nearly straight chains of short edges with a single long edge, while ensuring mutual boundaries consistency related to each other, on Earth.
geo_union_lines_array()	Calculates the union of lines or multilines on Earth.
geo_union_polygons_array()	Calculates the union of polygons or multipolygons on Earth.

Hash functions

Function Name	Description

Function Name	Description
hash()	Returns a hash value for the input value.
hash_combine()	Combines two or more hash values.
hash_many()	Returns a combined hash value of multiple values.
hash_md5()	Returns an MD5 hash value for the input value.
hash_sha1()	Returns a SHA1 hash value for the input value.
hash_sha256()	Returns a SHA256 hash value for the input value.
hash_xxhash64()	Returns an XXHASH64 hash value for the input value.

Units conversion functions

Function Name	Description
convert_angle()	Returns the input value converted from one angle unit to another
convert_energy()	Returns the input value converted from one energy unit to another
convert_force()	Returns the input value converted from one force unit to another
convert_length()	Returns the input value converted from one length unit to another
convert_mass()	Returns the input value converted from one mass unit to another
convert_speed()	Returns the input value converted from one speed unit to another
convert_temperature()	Returns the input value converted from one temperature unit to another
convert_volume()	Returns the input value converted from one volume unit to another

Feedback

abs()

Article • 03/23/2023

Calculates the absolute value of the input.

Syntax

`abs(`*x*`)`

Parameters

Name	Type	Required	Description
x	int, real, or timespan	✓	The value to make absolute.

Returns

Absolute value of x.

Example

Run the query

Kusto

```
print abs(-5)
```

Output

print_0
5

Feedback

Was this page helpful? Yes No

acos()

Article • 04/11/2023

Calculates the angle whose cosine is the specified number. Inverse operation of cos().

Syntax

`acos(x)`

Parameters

Name	Type	Required	Description
x	real	✓	The value used to calculate the arc cosine.

Returns

The value of the arc cosine of `x`. The return value is `null` if `x` < -1 or `x` > 1.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

ago()

Article • 03/22/2023

Subtracts the given timespan from the current UTC time.

Like `now()`, if you use `ago()` multiple times in a single query statement, the current UTC time being referenced will be the same across all uses.

Syntax

`ago(`*timespan*`)`

Parameters

Name	Type	Required	Description
timespan	timespan	✓	The interval to subtract from the current UTC clock time `now()`.

Returns

A datetime value `now() - a_timespan`

Example

All rows with a timestamp in the past hour:

```
Kusto
```

```
T | where Timestamp > ago(1h)
```

Feedback

around()

Article • 12/28/2022

Creates a `bool` value indicating if the first argument is within a range around the center value.

Syntax

`around(` *value* `,` *center* `,` *delta* `)`

Parameters

Name	Type	Required	Description
value	int, long, real, datetime, or timespan	✓	The value to compare to the *center*.
center	int, long, real, datetime, or timespan	✓	The center of the range defined as [(`center` - `delta`) .. (`center` + `delta`)].
delta	int, long, real, datetime, or timespan	✓	The delta value of the range defined as [(`center` - `delta`) .. (`center` + `delta`)].

Returns

Returns `true` if the value is within the range, `false` if the value is outside the range. Returns `null` if any of the arguments is `null`.

Example: Filtering values around a specific timestamp

The following example filters rows around specific timestamp.

Run the query

Kusto

```
range dt
    from datetime(2021-01-01 01:00)
    to datetime(2021-01-01 02:00)
```

```
    step 1min
| where around(dt, datetime(2021-01-01 01:30), 1min)
```

Output

dt
2021-01-01 01:29:00.0000000
2021-01-01 01:30:00.0000000
2021-01-01 01:31:00.0000000

Feedback

array_concat()

Article • 12/28/2022

Concatenates many dynamic arrays to a single array.

Syntax

`array_concat(`*arr*`[`, `*arr2*, ...`])`

Parameters

Name	Type	Required	Description
arr1...arrN	dynamic	✓	The arrays to concatenate into a dynamic array.

Returns

Returns a dynamic array of arrays with arr1, arr2, ... , arrN.

Example

The following example shows concatenated arrays.

Run the query

Kusto

```
range x from 1 to 3 step 1
| extend y = x * 2
| extend z = y * 2
| extend a1 = pack_array(x,y,z), a2 = pack_array(x, y)
| project array_concat(a1, a2)
```

Output

Column1
[1,2,4,1,2]
[2,4,8,2,4]

Column1
[3,6,12,3,6]

See also

- pack_array()

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

array_iff()

Article • 02/06/2023

Element-wise iif function on dynamic arrays.

> The `array_iff()` and `array_iif()` functions are equivalent

Syntax

`array_iff(`*condition_array*, *when_true*, *when_false*`)`

Parameters

Name	Type	Required	Description
condition_array	dynamic	✓	An array of *boolean* or numeric values.
when_true	dynamic or scalar	✓	An array of values or primitive value. This will be the result when *condition_array* is *true*.
when_false	dynamic or scalar	✓	An array of values or primitive value. This will be the result when *condition_array* is *false*.

ⓘ Note

- The length of the return value will be the same as the input *condition_array*.
- Numeric condition values are considered `true` if not equal to 0.
- Non-numeric and non-boolean condition values will be null in the corresponding index of the return value.
- If *when_true* or *when_false* is shorter than *condition_array*, missing values will be treated as null.

Returns

Returns a dynamic array of the values taken either from the *when_true* or *when_false* array values, according to the corresponding value of the condition array.

Examples

Run the query

Kusto

```
print condition=dynamic([true,false,true]), if_true=dynamic([1,2,3]),
if_false=dynamic([4,5,6])
| extend res= array_iff(condition, if_true, if_false)
```

Output

condition	if_true	if_false	res
[true, false, true]	[1, 2, 3]	[4, 5, 6]	[1, 5, 3]

Numeric condition values

Run the query

Kusto

```
print condition=dynamic([1,0,50]), if_true="yes", if_false="no"
| extend res= array_iff(condition, if_true, if_false)
```

Output

condition	if_true	if_false	res
[1, 0, 50]	yes	no	[yes, no, yes]

Non-numeric and non-boolean condition values

Run the query

Kusto

```
print condition=dynamic(["some string value", datetime("01-01-2022"),
null]), if_true=1, if_false=0
| extend res= array_iff(condition, if_true, if_false)
```

Output

condition	if_true	if_false	res

condition	if_true	if_false	res
[true, false, true]	1	0	[null, null, null]

Mismatched array lengths

Run the query

Kusto

```
print condition=dynamic([true,true,true]), if_true=dynamic([1,2]),
if_false=dynamic([3,4])
| extend res= array_iff(condition, if_true, if_false)
```

Output

condition	if_true	if_false	res
[true, true, true]	[1, 2]	[3, 4]	[1, 2, null]

Feedback

array_index_of()

Article • 12/28/2022

Searches an array for the specified item, and returns its position.

Syntax

`array_index_of(`*array*`,`*value*`)`

Parameters

Name	Type	Required	Description
array	dynamic	✓	The array to search.
value	long, integer, double, datetime, timespan, decimal, string, guid, or boolean	✓	The value to lookup.
start	number		The search start position. A negative value will offset the starting search value from the end of the array by `abs(start_index)` steps.
length	int		The number of values to examine. A value of -1 means unlimited length.
occurrence	int	The number of the occurrence. The default is 1.	

Returns

Returns a zero-based index position of lookup. Returns -1 if the value isn't found in the array. Returns *null* for irrelevant inputs (*occurrence* < 0 or *length* < -1).

Example

The following example shows the position number of specific words within the array.

Run the query

Kusto

```
let arr=dynamic(["this", "is", "an", "example", "an", "example"]);
print
 idx1 = array_index_of(arr,"an")    // lookup found in input string
 , idx2 = array_index_of(arr,"example",1,3) // lookup found in researched
range
 , idx3 = array_index_of(arr,"example",1,2) // search starts from index 1,
but stops after 2 values, so lookup can't be found
 , idx4 = array_index_of(arr,"is",2,4) // search starts after occurrence of
lookup
 , idx5 = array_index_of(arr,"example",2,-1)  // lookup found
 , idx6 = array_index_of(arr, "an", 1, -1, 2)   // second occurrence found
in input range
 , idx7 = array_index_of(arr, "an", 1, -1, 3)   // no third occurrence in
input array
 , idx8 = array_index_of(arr, "an", -3)   // negative start index will look
at last 3 elements
 , idx9 = array_index_of(arr, "is", -4)   // negative start index will look
at last 3 elements
```

Output

idx1	idx2	idx3	idx4	idx5	idx6	idx7	idx8	idx9
2	3	-1	-1	3	4	-1	4	-1

See also

Use set_has_element(arr, value) to check whether a value exists in an array. This function will improve the readability of your query. Both functions have the same performance.

Feedback

array_length()

Article • 12/28/2022

Calculates the number of elements in a dynamic array.

> **Deprecated aliases:** arraylength()

Syntax

`array_length(`*array*`)`

Parameters

Name	Type	Required	Description
array	dynamic	✓	The array for which to calculate length.

Returns

Returns the number of elements in *array*, or `null` if *array* isn't an array.

Examples

The following example shows the number of elements in the array.

Run the query

Kusto

```
print array_length(dynamic([1, 2, 3, "four"]))
```

Output

print_0
4

Feedback

array_reverse()

Article • 12/28/2022

Reverses the order of the elements in a dynamic array.

Syntax

`array_reverse(`*value*`)`

Parameters

Name	Type	Required	Description
value	dynamic	✓	The array to reverse.

Returns

Returns an array that contains the same elements as the input array in reverse order.

Example

This example shows an array of words reversed.

Run the query

Kusto

```
print arr=dynamic(["this", "is", "an", "example"])
| project Result=array_reverse(arr)
```

Output

Result
["example","an","is","this"]

Feedback

array_rotate_left()

Article • 12/28/2022

Rotates values inside a `dynamic` array to the left.

Syntax

`array_rotate_left(`*array*, *rotate_count*`)`

Parameters

Name	Type	Required	Description
array	dynamic	✓	The array to rotate.
rotate_count	integer	✓	The number of positions that array elements will be rotated to the left. If the value is negative, the elements will be rotated to the right.

Returns

Dynamic array containing the same elements as the original array with each element rotated according to *rotate_count*.

Examples

Rotating to the left by two positions:

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_rotated=array_rotate_left(arr, 2)
```

Output

arr	arr_rotated
[1,2,3,4,5]	[3,4,5,1,2]

Rotating to the right by two positions by using negative rotate_count value:

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_rotated=array_rotate_left(arr, -2)
```

Output

arr	arr_rotated
[1,2,3,4,5]	[4,5,1,2,3]

See also

- To rotate an array to the right, use array_rotate_right().
- To shift an array to the left, use array_shift_left().
- To shift an array to the right, use array_shift_right()

Feedback

array_rotate_right()

Article • 12/28/2022

Rotates values inside a `dynamic` array to the right.

Syntax

`array_rotate_right(`*array*, *rotate_count*`)`

Parameters

Name	Type	Required	Description
array	dynamic	✓	The array to rotate.
rotate_count	integer	✓	The number of positions that array elements will be rotated to the right. If the value is negative, the elements will be rotated to the Left.

Returns

Dynamic array containing the same elements as the original array with each element rotated according to *rotate_count*.

Examples

Rotating to the right by two positions:

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_rotated=array_rotate_right(arr, 2)
```

Output

arr	arr_rotated
[1,2,3,4,5]	[4,5,1,2,3]

Rotating to the left by two positions by using negative rotate_count value:

Run the query

Results

```
Kusto
```

```
print arr=dynamic([1,2,3,4,5])
| extend arr_rotated=array_rotate_right(arr, -2)
```

Output

arr	arr_rotated
[1,2,3,4,5]	[3,4,5,1,2]

See also

- To rotate an array to the left, use array_rotate_left().
- To shift an array to the left, use array_shift_left().
- To shift an array to the right, use array_shift_right().

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

array_shift_left()

Article • 12/28/2022

Shifts the values inside a `dynamic` array to the left.

Syntax

`array_shift_left(`*array*, *shift_count* `[,` *default_value* `])`

Parameters

Name	Type	Required	Description
array	dynamic	✓	The array to shift.
shift_count	integer	✓	The number of positions that array elements will be shifted to the left. If the value is negative, the elements will be shifted to the right.
default_value	scalar		The value used for an element that was shifted and removed. The default is null or an empty string depending on the type of elements in the *array*.

Returns

Returns a dynamic array containing the same number of elements as in the original array. Each element has been shifted according to *shift_count*. New elements that are added in place of removed elements will have a value of *default_value*.

Examples

Shifting to the left by two positions:

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_shift=array_shift_left(arr, 2)
```

Output

`arr`	`arr_shift`
[1,2,3,4,5]	[3,4,5,null,null]

Shifting to the left by two positions and adding default value:

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_shift=array_shift_left(arr, 2, -1)
```

Output

`arr`	`arr_shift`
[1,2,3,4,5]	[3,4,5,-1,-1]

Shifting to the right by two positions by using negative *shift_count* value:

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_shift=array_shift_left(arr, -2, -1)
```

Output

arr	**arr_shift**
[1,2,3,4,5]	[-1,-1,1,2,3]

See also

- To shift an array to the right, use array_shift_right().
- To rotate an array to the right, use array_rotate_right().
- To rotate an array to the left, use array_rotate_left().

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

array_shift_right()

Article • 12/28/2022

Shifts the values inside a dynamic array to the right.

Syntax

`array_shift_right(`*array*, *shift_count* [, *default_value*]`)`

Parameters

Name	Type	Required	Description
array	dynamic	✓	The array to shift.
shift_count	integer	✓	The number of positions that array elements will be shifted to the right. If the value is negative, the elements will be shifted to the left.
default_value	scalar		The value used for an element that was shifted and removed. The default is null or an empty string depending on the type of elements in the *array*.

Returns

Returns a dynamic array containing the same amount of the elements as in the original array. Each element has been shifted according to *shift_count*. New elements that are added instead of the removed elements will have a value of *default_value*.

Examples

Shifting to the right by two positions:

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_shift=array_shift_right(arr, 2)
```

Output

arr	arr_shift
[1,2,3,4,5]	[null,null,1,2,3]

Shifting to the right by two positions and adding a default value:

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_shift=array_shift_right(arr, 2, -1)
```

Output

arr	arr_shift
[1,2,3,4,5]	[-1,-1,1,2,3]

Shifting to the left by two positions by using a negative shift_count value:

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_shift=array_shift_right(arr, -2, -1)
```

Output

arr	arr_shift
[1,2,3,4,5]	[3,4,5,-1,-1]

See also

- To shift an array to the left, use array_shift_left().
- To rotate an array to the right, use array_rotate_right().
- To rotate an array to the left, use array_rotate_left().

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

array_slice()

Article • 03/20/2023

Extracts a slice of a dynamic array.

Syntax

`array_slice`(*array*, *start*, *end*)

Parameters

Name	Type	Required	Description
array	dynamic	✓	The array from which to extract the slice.
start	int	✓	The start index of the slice (inclusive). Negative values are converted to `array_length` + `start`.
end	int	✓	The last index of the slice. (inclusive). Negative values are converted to `array_length` + `end`.

> ⓘ Note
>
> Out of bounds indices are ignored.

Returns

Returns a dynamic array of the values in the range [`start..end`] from `array`.

Examples

The following examples return a slice of the array.

Run the query

Kusto

```
print arr=dynamic([1,2,3])
| extend sliced=array_slice(arr, 1, 2)
```

Output

arr	sliced
[1,2,3]	[2,3]

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend sliced=array_slice(arr, 2, -1)
```

Output

arr	sliced
[1,2,3,4,5]	[3,4,5]

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend sliced=array_slice(arr, -3, -2)
```

Output

arr	sliced
[1,2,3,4,5]	[3,4]

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

array_sort_asc()

Article • 12/28/2022

Receives one or more arrays. Sorts the first array in ascending order. Orders the remaining arrays to match the reordered first array.

Syntax

`array_sort_asc(`*array1*[, ..., *argumentN*]`)`

`array_sort_asc(`*array1*[, ..., *argumentN*]`,` *nulls_last*`)`

If *nulls_last* isn't provided, a default value of `true` is used.

Parameters

Name	Type	Required	Description
array1...arrayN	dynamic	✓	The array or list of arrays to sort.
nulls_last	bool		Determines whether `null`s should be last.

Returns

Returns the same number of arrays as in the input, with the first array sorted in ascending order, and the remaining arrays ordered to match the reordered first array.

`null` will be returned for every array that differs in length from the first one.

If an array contains elements of different types, it will be sorted in the following order:

- Numeric, `datetime`, and `timespan` elements
- String elements
- Guid elements
- All other elements

Example 1 - Sorting two arrays

Run the query

Kusto

```
let array1 = dynamic([1,3,4,5,2]);
let array2 = dynamic(["a","b","c","d","e"]);
print array_sort_asc(array1,array2)
```

Output

array1_sorted	array2_sorted
[1,2,3,4,5]	["a","e","b","c","d"]

> **Note**
>
> The output column names are generated automatically, based on the arguments to the function. To assign different names to the output columns, use the following syntax: `... | extend (out1, out2) = array_sort_asc(array1,array2)`

Example 2 - Sorting substrings

Run the query

Kusto

```
let Names = "John,Paul,George,Ringo";
let SortedNames = strcat_array(array_sort_asc(split(Names, ",")), ",");
print result = SortedNames
```

Output

result
George,John,Paul,Ringo

Example 3 - Combining summarize and array_sort_asc

Run the query

Kusto

```
datatable(command:string, command_time:datetime, user_id:string)
[
    'chmod',   datetime(2019-07-15),   "user1",
    'ls',      datetime(2019-07-02),   "user1",
    'dir',     datetime(2019-07-22),   "user1",
    'mkdir',   datetime(2019-07-14),   "user1",
    'rm',      datetime(2019-07-27),   "user1",
    'pwd',     datetime(2019-07-25),   "user1",
    'rm',      datetime(2019-07-23),   "user2",
    'pwd',     datetime(2019-07-25),   "user2",
]
| summarize timestamps = make_list(command_time), commands =
make_list(command) by user_id
| project user_id, commands_in_chronological_order =
array_sort_asc(timestamps, commands)[1]
```

Output

user_id	commands_in_chronological_order
user1	["ls", "mkdir", "chmod", "dir", "pwd", "rm"]
user2	["rm", "pwd"]

> ⓘ Note
>
> If your data may contain `null` values, use **make_list_with_nulls** instead of **make_list**.

Example 4 - Controlling location of `null` values

By default, `null` values are put last in the sorted array. However, you can control it explicitly by adding a `bool` value as the last argument to `array_sort_asc()`.

Example with default behavior:

Run the query

```
Kusto
print array_sort_asc(dynamic([null,"blue","yellow","green",null]))
```

Output

print_0
["blue","green","yellow",null,null]

Example with non-default behavior:

Run the query

```
Kusto
print array_sort_asc(dynamic([null,"blue","yellow","green",null]), false)
```

Output

print_0
[null,null,"blue","green","yellow"]

See also

To sort the first array in descending order, use array_sort_desc().

Feedback

array_sort_desc()

Article • 12/28/2022

Receives one or more arrays. Sorts the first array in descending order. Orders the remaining arrays to match the reordered first array.

Syntax

`array_sort_desc(`*array1*[, ..., *argumentN*]`)`

`array_sort_desc(`*array1*[, ..., *argumentN*]`,` *nulls_last*`)`

If *nulls_last* isn't provided, a default value of `true` is used.

Parameters

Name	Type	Required	Description
array1...arrayN	dynamic	✓	The array or list of arrays to sort.
nulls_last	bool		Determines whether `null`s should be last.

Returns

Returns the same number of arrays as in the input, with the first array sorted in ascending order, and the remaining arrays ordered to match the reordered first array.

`null` will be returned for every array that differs in length from the first one.

If an array contains elements of different types, it will be sorted in the following order:

- Numeric, `datetime`, and `timespan` elements
- String elements
- Guid elements
- All other elements

Example 1 - Sorting two arrays

Run the query

Kusto

```
let array1 = dynamic([1,3,4,5,2]);
let array2 = dynamic(["a","b","c","d","e"]);
print array_sort_desc(array1,array2)
```

Output

array1_sorted	array2_sorted
[5,4,3,2,1]	["d","c","b","e","a"]

> **Note**
>
> The output column names are generated automatically, based on the arguments to the function. To assign different names to the output columns, use the following syntax: `... | extend (out1, out2) = array_sort_desc(array1,array2)`

Example 2 - Sorting substrings

Run the query

Kusto

```
let Names = "John,Paul,George,Ringo";
let SortedNames = strcat_array(array_sort_desc(split(Names, ",")), ",");
print result = SortedNames
```

Output

result
Ringo,Paul,John,George

Example 3 - Combining summarize and array_sort_desc

Run the query

Kusto

```
datatable(command:string, command_time:datetime, user_id:string)
[
    'chmod',   datetime(2019-07-15),   "user1",
    'ls',      datetime(2019-07-02),   "user1",
    'dir',     datetime(2019-07-22),   "user1",
    'mkdir',   datetime(2019-07-14),   "user1",
    'rm',      datetime(2019-07-27),   "user1",
    'pwd',     datetime(2019-07-25),   "user1",
    'rm',      datetime(2019-07-23),   "user2",
    'pwd',     datetime(2019-07-25),   "user2",
]
| summarize timestamps = make_list(command_time), commands =
make_list(command) by user_id
| project user_id, commands_in_chronological_order =
array_sort_desc(timestamps, commands)[1]
```

Output

user_id	commands_in_chronological_order
user1	["rm", "pwd", "dir", "chmod", "mkdir", "ls"]
user2	["pwd", "rm"]

> ⓘ Note
>
> If your data may contain `null` values, use **make_list_with_nulls** instead of **make_list**.

Example 4 - Controlling location of `null` values

By default, `null` values are put last in the sorted array. However, you can control it explicitly by adding a `bool` value as the last argument to `array_sort_desc()`.

Example with default behavior:

Run the query

```
print array_sort_desc(dynamic([null,"blue","yellow","green",null]))
```

Output

print_0
["yellow","green","blue",null,null]

Example with non-default behavior:

Run the query

```
print array_sort_desc(dynamic([null,"blue","yellow","green",null]), false)
```

Output

print_0
[null,null,"yellow","green","blue"]

See also

To sort the first array in ascending order, use array_sort_asc().

Feedback

array_split()

Article • 12/28/2022

Splits an array to multiple arrays according to the split indices and packs the generated array in a dynamic array.

Syntax

`array_split`(*array*, *index*)

Parameters

Name	Type	Required	Description
array	dynamic	✓	The array to split.
index	integer or dynamic	✓	An integer or dynamic array of integers used to indicate the location at which to split the array. The start index of arrays is zero. Negative values are converted to `array_length` + `value`.

Returns

Returns a dynamic array containing N+1 arrays with the values in the range `[0..i1), [i1..i2), ... [iN..array_length)` from `array`, where N is the number of input indices and `i1...iN` are the indices.

Examples

This following example shows how to split and array.

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_split=array_split(arr, 2)
```

Output

arr	arr_split
[1,2,3,4,5]	[[1,2],[3,4,5]]

Run the query

Kusto

```
print arr=dynamic([1,2,3,4,5])
| extend arr_split=array_split(arr, dynamic([1,3]))
```

Output

arr	arr_split
[1,2,3,4,5]	[[1],[2,3],[4,5]]

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

array_sum()

Article • 12/28/2022

Calculates the sum of elements in a dynamic array.

Syntax

`array_sum`(*array*)

Parameters

Name	Type	Required	Description
array	dynamic	✓	The array to sum.

Returns

Returns a double type value with the sum of the elements of the array.

> ⓘ Note
>
> If the array contains elements of non-numeric types, the result is `null`.

Example

This following example shows the sum of an array.

Run the query

Kusto

```
print arr=dynamic([1,2,3,4])
| extend arr_sum=array_sum(arr)
```

Output

arr	arr_sum

arr	arr_sum
[1,2,3,4]	10

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

asin()

Article • 12/28/2022

Calculates the angle whose sine is the specified number, or the arc sine. This is the inverse operation of sin().

Syntax

`asin(`*x*`)`

Parameters

Name	Type	Required	Description
x	real	✓	A real number in range [-1, 1] used to calculate the arc sine.

Returns

Returns the value of the arc sine of `x`. Returns `null` if `x` < -1 or `x` > 1.

Example

Run the query

Kusto

```
asin(0.5)
```

Output

result
1.2532358975033751

Feedback

Was this page helpful? Yes No

assert()

Article • 05/01/2023

Checks for a condition. If the condition is false, outputs error messages and fails the query.

> ⓘ Note
>
> The `assert` function gets evaluated during the query analysis phase, before optimizations such as constant-folding and predicate short-circuiting get applied.

> ⓘ Note
>
> The parameters given to `assert` must be evaluated to constants during the query analysis phase. In other words, it can be constructed from other expressions referencing constants only, and can't be bound to row-context.

Syntax

`assert(`*condition*`,`*message*`)`

Parameters

Name	Type	Required	Description
condition	bool	✓	The conditional expression to evaluate. The condition must be evaluated to constant during the query analysis phase.
message	string	✓	The message used if assertion is evaluated to `false`.

Returns

Returns `true` if the condition is `true`. Raises a semantic error if the condition is evaluated to `false`.

Examples

The following query defines a function `checkLength()` that checks input string length, and uses `assert` to validate input length parameter (checks that it's greater than zero).

Run the query

Kusto

```
let checkLength = (len:long, s:string)
{
    assert(len > 0, "Length must be greater than zero") and
    strlen(s) > len
};
datatable(input:string)
[
    '123',
    '4567'
]
| where checkLength(len=long(-1), input)
```

Running this query yields an error: `assert() has failed with message: 'Length must be greater than zero'`

Example of running with valid `len` input:

Run the query

Kusto

```
let checkLength = (len:long, s:string)
{
    assert(len > 0, "Length must be greater than zero") and strlen(s) > len
};
datatable(input:string)
[
    '123',
    '4567'
]
| where checkLength(len=3, input)
```

Output

input
4567

The following query will always fail, demonstrating that the `assert` function gets evaluated even though the `where b` operator returns no data when `b` is `false`:

Kusto

```
let b=false;
print x="Hello"
| where b
| where assert(b, "Assertion failed")
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

atan()

Article • 12/28/2022

Returns the angle whose tangent is the specified number. This is the inverse operation of tan().

Syntax

`atan(x)`

Parameters

Name	Type	Required	Description
x	real	✓	The number used to calculate the arc tangent.

Returns

The value of the arc tangent of `x`.

Example

Run the query

Kusto

```
atan(0.5)
```

Output

result
0.46364760900080609

Feedback

Was this page helpful? Yes No

atan2()

Article • 12/28/2022

Calculates the angle, in radians, between the positive x-axis and the ray from the origin to the point (y, x).

Syntax

`atan2(`*y*`,`*x*`)`

Parameters

Name	Type	Required	Description
y	real	✓	The Y coordinate.
x	real	✓	The X coordinate.

Returns

Returns the angle in radians between the positive x-axis and the ray from the origin to the point (y, x).

Examples

The following example returns the angle measurements in radians.

Run the query

Kusto

```
print atan2_0 = atan2(1,1) // Pi / 4 radians (45 degrees)
| extend atan2_1 = atan2(0,-1) // Pi radians (180 degrees)
| extend atan2_2 = atan2(-1,0) // - Pi / 2 radians (-90 degrees)
```

Output

atan2_0	atan2_1	atan2_2
0.785398163397448	3.14159265358979	-1.5707963267949

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

bag_has_key()

Article • 12/28/2022

Checks whether a dynamic property bag object contains a given key.

Syntax

`bag_has_key(`*bag*`,`*key*`)`

Parameters

Name	Type	Required	Description
bag	dynamic	✓	The property bag to search.
key	string	✓	The key for which to search. Search for a nested key using the JSONPath notation. Array indexing isn't supported.

Returns

True or false depending on if the key exists in the bag.

Examples

Run the query

Kusto

```
datatable(input: dynamic)
[
    dynamic({'key1' : 123, 'key2': 'abc'}),
    dynamic({'key1' : 123, 'key3': 'abc'}),
]
| extend result = bag_has_key(input, 'key2')
```

Output

input	result

input	result

{
"key1": 123,
"key2": "abc"
}
	true

{
"key1": 123,
"key3": "abc"
}
	false

Search using a JSONPath key

Run the query

Kusto

```
datatable(input: dynamic)
[
    dynamic({'key1': 123, 'key2': {'prop1' : 'abc', 'prop2': 'xyz'}, 'key3':
[100, 200]}),
]
| extend result = bag_has_key(input, '$.key2.prop1')
```

Output

input	result

{
"key1": 123,
"key2": {
"prop1": "abc",
"prop2": "xyz"
},
"key3": [
100,
200
]
}
	true

Feedback

Was this page helpful? Yes No

bag_keys()

Article • 12/28/2022

Enumerates all the root keys in a dynamic property bag object.

Syntax

`bag_keys(`*object*`)`

Parameters

Name	Type	Required	Description
object	dynamic	✓	The property bag object for which to enumerate keys.

Returns

An array of keys, order is undetermined.

Example

Run the query

Kusto

```
datatable(index:long, d:dynamic) [
    1, dynamic({'a':'b', 'c':123}),
    2, dynamic({'a':'b', 'c':{'d':123}}),
    3, dynamic({'a':'b', 'c':[{'d':123}]}),
    4, dynamic(null),
    5, dynamic({}),
    6, dynamic('a'),
    7, dynamic([])
]
| extend keys = bag_keys(d)
```

Output

index	d	keys

index	d	keys

1	{
	"a": "b",
	"c": 123
	}
		[
		"a",
		"c"
		]

2	{
	"a": "b",
	"c": {
	"d": 123
	}
	}
		[
		"a",
		"c"
		]

3	{
	"a": "b",
	"c": [
	{
	"d": 123
	}
	]
	}
		[
		"a",
		"c"
		]

4		
5	{}	[]
6	a	
7	[]	

Feedback

bag_merge()

Article • 12/28/2022

Merges `dynamic` property bags into a `dynamic` property bag object with all properties merged.

Syntax

`bag_merge(`*bag1*`,`*bag2*`[,*bag3*, ...])`

Parameters

Name	Type	Required	Description
bag1...*bagN*	dynamic	✓	The property bags to merge. The function accepts between 2 to 64 arguments.

Returns

Returns a `dynamic` property bag. Results from merging all of the input property bag objects. If a key appears in more than one input object, an arbitrary value out of the possible values for this key will be chosen.

Example

Run the query

Kusto

```
print result = bag_merge(
   dynamic({'A1':12, 'B1':2, 'C1':3}),
   dynamic({'A2':81, 'B2':82, 'A1':1}))
```

Output

result

result
{ "A1": 12, "B1": 2, "C1": 3, "A2": 81, "B2": 82 }

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

bag_pack()

Article • 01/10/2023

Creates a dynamic property bag object from a list of keys and values.

> **Deprecated aliases**: pack(), pack_dictionary()

Syntax

`bag_pack(`*key1*`,` *value1*`,` *key2*`,` *value2*`,... )`

Parameters

Name	Type	Required	Description
key	string	✓	The key name.
value	string	✓	The key value.

> **Note**
>
> The *key* and *value* strings are an alternating list the total length of the list must be even.

Returns

Returns a `dynamic` property bag object from the listed *key* and *value* inputs.

Examples

Example 1

The following example creates and returns a property bag from an alternating list of keys and values.

Run the query

Kusto

```
print bag_pack("Level", "Information", "ProcessID", 1234, "Data", bag_pack("url",
"www.bing.com"))
```

Results

print_0
{"Level":"Information","ProcessID":1234,"Data":{"url":"www.bing.com"}}

Example 2

The following example uses two tables, *SmsMessages* and *MmsMessages*, and returns their common columns and a property bag from the other columns. The tables are created ad-hoc as part of the query.

SmsMessages

SourceNumber	TargetNumber	CharsCount
555-555-1234	555-555-1212	46
555-555-1234	555-555-1213	50
555-555-1212	555-555-1234	32

MmsMessages

SourceNumber	TargetNumber	AttachmentSize	AttachmentType	AttachmentName
555-555-1212	555-555-1213	200	jpeg	Pic1
555-555-1234	555-555-1212	250	jpeg	Pic2
555-555-1234	555-555-1213	300	png	Pic3

Run the query

Kusto

```
let SmsMessages = datatable (
    SourceNumber: string,
    TargetNumber: string,
    CharsCount: string
) [
    "555-555-1234", "555-555-1212", "46",
    "555-555-1234", "555-555-1213", "50",
    "555-555-1212", "555-555-1234", "32"
];
let MmsMessages = datatable (
    SourceNumber: string,
    TargetNumber: string,
    AttachmentSize: string,
    AttachmentType: string,
    AttachmentName: string
) [
    "555-555-1212", "555-555-1213", "200", "jpeg", "Pic1",
    "555-555-1234", "555-555-1212", "250", "jpeg", "Pic2",
    "555-555-1234", "555-555-1213", "300", "png", "Pic3"
];
SmsMessages
| join kind=inner MmsMessages on SourceNumber
| extend Packed=bag_pack("CharsCount", CharsCount, "AttachmentSize", AttachmentSize,
"AttachmentType", AttachmentType, "AttachmentName", AttachmentName)
| where SourceNumber == "555-555-1234"
| project SourceNumber, TargetNumber, Packed
```

Results

SourceNumber	TargetNumber	Packed
555-555-1234	555-555-1213	{"CharsCount":"50","AttachmentSize":"250","AttachmentType":"jpeg","AttachmentName":"Pic2"}
555-555-1234	555-555-1212	{"CharsCount":"46","AttachmentSize":"250","AttachmentType":"jpeg","AttachmentName":"Pic2"}

SourceNumber	TargetNumber	Packed
555-555-1234	555-555-1213	{"CharsCount":"50","AttachmentSize":"300","AttachmentType":"png","AttachmentName":"Pic3"}
555-555-1234	555-555-1212	{"CharsCount":"46","AttachmentSize":"300","AttachmentType":"png","AttachmentName":"Pic3"}

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

bag_pack_columns()

Article • 03/16/2023

Creates a dynamic property bag object from a list of columns.

Syntax

`bag_pack_columns(`*column1*`,` *column2*`,... )`

Parameters

Name	Type	Required	Description
column	scalar	✓	A column to pack. The name of the column is the property name in the property bag.

Returns

Returns a `dynamic` property bag object from the listed *columns*.

Examples

The following example creates a property bag that includes the `Id` and `Value` columns:

Run the query

Kusto

```
datatable(Id: string, Value: string, Other: long)
[
    "A", "val_a", 1,
    "B", "val_b", 2,
    "C", "val_c", 3
]
| extend Packed = bag_pack_columns(Id, Value)
```

Id	Value	Other	Packed

Id	Value	Other	Packed

A	val_a	1	{
			"Id": "A",
			"Value": "val_a"
			}

B	val_b	2	{
			"Id": "B",
			"Value": "val_b"
			}

C	val_c	3	{
			"Id": "C",
			"Value": "val_c"
			}

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

bag_remove_keys()

Article • 05/31/2023

Removes keys and associated values from a `dynamic` property bag.

Syntax

`bag_remove_keys(`*bag*`,`*keys*`)`

Parameters

Name	Type	Required	Description
bag	dynamic	✓	The property bag from which to remove keys.
keys	dynamic	✓	List of keys to be removed from the input. The keys are the first level of the property bag. You can specify keys on the nested levels using JSONPath notation. Array indexing isn't supported.

Returns

Returns a `dynamic` property bag without specified keys and their values.

Examples

Run the query

Kusto

```
datatable(input:dynamic)
[
    dynamic({'key1' : 123,     'key2': 'abc'}),
    dynamic({'key1' : 'value', 'key3': 42.0}),
]
| extend result=bag_remove_keys(input, dynamic(['key2', 'key4']))
```

Output

input	result

input	result

{
"key1": 123,
"key2": "abc"
}
	{
	"key1": 123
	}

{
"key1": "value",
"key3": 42.0
}
	{
	"key1": "value",
	"key3": 42.0
	}

Remove inner properties of dynamic values using JSONPath notation

Run the query

Kusto

```
datatable(input:dynamic)
[
    dynamic({'key1': 123, 'key2': {'prop1' : 'abc', 'prop2': 'xyz'}, 'key3':
[100, 200]}),
]
| extend result=bag_remove_keys(input, dynamic(['$.key2.prop1', 'key3']))
```

Output

input	result

{
"key1": 123,
"key2": {
"prop1": "abc",
"prop2": "xyz"
},
"key3": [
100,
200
]
}
	{
	"key1": 123,
	"key2": {
	"prop2": "xyz"
	}
	}

Feedback

bag_set_key()

Article • 04/18/2023

bag_set_key() receives a `dynamic` property-bag, a key and a value. The function sets the given key in the bag to the given value. The function will override any existing value in case the key already exists.

Syntax

`bag_set_key(`*bag*`,`*key*`,`*value*`)`

Parameters

Name	Type	Required	Description
bag	dynamic	✓	The property bag to modify.
key	string	✓	The key to set. Either a JSON path (you can specify a key on the nested levels using JSONPath notation) or the key name for a root level key. Array indexing or root JSON path aren't supported.
value	any scalar data type	✓	The value to which the key is set.

Returns

Returns a `dynamic` property-bag with specified key-value pairs. If the input bag isn't a property-bag, a `null` value is returned.

> ⓘ Note
>
> To treat `null`s as empty bags, use `coalesce(x, dynamic({}))`.

Examples

Use a root-level key

Run the query

Kusto

```
datatable(input: dynamic) [
    dynamic({'key1': 1, 'key2': 2}),
    dynamic({'key1': 1, 'key3': 'abc'}),
]
| extend result = bag_set_key(input, 'key3', 3)
```

input	result
{ "key1": 1, "key2": 2 }	{ "key1": 1, "key2": 2, "key3": 3 }
{ "key1": 1, "key3": "abc" }	{ "key1": 1, "key3": 3 }

Use a JSONPath key

Run the query

Kusto

```
datatable(input: dynamic)[
    dynamic({'key1': 123, 'key2': {'prop1': 123, 'prop2': 'xyz'}}),
    dynamic({'key1': 123})
]
| extend result = bag_set_key(input, '$.key2.prop1', 'abc')
```

input	result
{ "key1": 123, "key2": { "prop1": 123, "prop2": "xyz" } }	{ "key1": 123, "key2": { "prop1": "abc", "prop2": "xyz" } }

input	result

{
"key1": 123
}
	{
	"key1": 123,
	"key2": {
	"prop1": "abc"
	}
	}

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

bag_zip()

Article • 05/01/2023

Creates a dynamic property-bag from two input dynamic arrays. In the resulting property-bag, the values from the first input array are used as the property keys, while the values from the second input array are used as corresponding property values.

Syntax

`bag_zip(`*KeysArray*`,` *ValuesArray*`)`

Parameters

Name	Type	Required	Description
KeysArray	dynamic	✓	An array of strings. These strings represent the property names for the resulting property-bag.
ValuesArray	dynamic	✓	An array whose values will be the property values for the resulting property-bag.

> ⓘ Note
>
> - If there are more keys than values, missing values are filled with null.
> - If there are more values than keys, values with no matching keys are ignored.
> - Keys that aren't strings are ignored.

Returns

Returns a dynamic property-bag.

Examples

In the following example, the array of keys and the array of values are the same length and are zipped together into a dynamic property bag.

Run the query

```
```

Kusto

```
let Data = datatable(KeysArray: dynamic, ValuesArray: dynamic) [
    dynamic(['a', 'b', 'c']), dynamic([1, '2', 3.4])
];
Data
| extend NewBag = bag_zip(KeysArray, ValuesArray)
```

KeysArray	ValuesArray	NewBag
['a','b','c']	[1,'2',3.4]	{'a': 1,'b': '2','c': 3.4}

More keys than values

In the following example, the array of keys is longer than the array of values. The missing values are filled with nulls.

Run the query

Kusto

```
let Data = datatable(KeysArray: dynamic, ValuesArray: dynamic) [
    dynamic(['a', 'b', 'c']), dynamic([1, '2'])
];
Data
| extend NewBag = bag_zip(KeysArray, ValuesArray)
```

KeysArray	ValuesArray	NewBag
['a','b','c']	[1,'2']	{'a': 1,'b': '2','c': null}

More values than keys

In the following example, the array of values is longer than the array of keys. Values with no matching keys are ignored.

Run the query

Kusto

```
let Data = datatable(KeysArray: dynamic, ValuesArray: dynamic) [
    dynamic(['a', 'b']), dynamic([1, '2', 2.5])
];
Data
| extend NewBag = bag_zip(KeysArray, ValuesArray)
```

KeysArray	ValuesArray	NewBag
['a','b']	[1,'2',2.5]	{'a': 1,'b': '2'}

Non-string keys

In the following example, there are some values in they keys array that aren't of type string. The non-string values are ignored.

Run the query

Kusto

```
let Data = datatable(KeysArray: dynamic, ValuesArray: dynamic) [
    dynamic(['a', 8, 'b']), dynamic([1, '2', 2.5])
];
Data
| extend NewBag = bag_zip(KeysArray, ValuesArray)
```

KeysArray	ValuesArray	NewBag
['a',8,'b']	[1,'2',2.5]	{'a': 1,'b': 2.5}

Fill values with null

In the following example, the parameter that is supposed to be an array of values isn't an array, so all values are filled with nulls.

Run the query

Kusto

```
let Data = datatable(KeysArray: dynamic, ValuesArray: dynamic) [
    dynamic(['a', 8, 'b']), dynamic(1)
];
Data
| extend NewBag = bag_zip(KeysArray, ValuesArray)
```

KeysArray	ValuesArray	NewBag
['a',8,'b']	1	{'a': null,'b': null}

Null property-bag

In the following example, the parameter that is supposed to be an array of keys isn't an array, so the resulting property-bag is null.

Run the query

Kusto

```
let Data = datatable(KeysArray: dynamic, ValuesArray: dynamic) [
    dynamic('a'), dynamic([1, '2', 2.5])
];
Data
| extend NewBag = bag_zip(KeysArray, ValuesArray)
| extend IsNewBagEmpty=isnull(NewBag)
```

KeysArray	ValuesArray	NewBag	IsNewBagEmpty
a	[1,'2',2.5]		TRUE

See also

- zip function

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

base64_decode_toarray()

Article • 12/28/2022

Decodes a base64 string to an array of long values.

Syntax

`base64_decode_toarray(`*base64_string*`)`

Parameters

Name	Type	Required	Description
base64_string	string	✓	The value to decode from base64 to an array of long values.

Returns

Returns an array of long values decoded from a base64 string.

Example

Run the query

Kusto

```
print Quine=base64_decode_toarray("S3VzdG8=")
// 'K', 'u', 's', 't', 'o'
```

Output

Quine
[75,117,115,116,111]

See also

- To decode base64 strings to a UTF-8 string, see base64_decode_tostring()
- To encode strings to a base64 string, see base64_encode_tostring()

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

base64_decode_tostring()

Article • 12/28/2022

Decodes a base64 string to a UTF-8 string.

> **Deprecated aliases:** base64_decodestring()

Syntax

`base64_decode_tostring(`*base64_string*`)`

Parameters

Name	Type	Required	Description
base64_string	string	✓	The value to decode from base64 to UTF-8 string.

Returns

Returns UTF-8 string decoded from base64 string.

Example

Run the query

```
Kusto
print Quine=base64_decode_tostring("S3VzdG8=")
```

Output

Quine
Kusto

Trying to decode a base64 string that was generated from invalid UTF-8 encoding will return null:

Run the query

Kusto

```
print Empty=base64_decode_tostring("U3RyaW5n0KHR0tGA0L7Rh9C60LA=")
```

Output

Empty

See also

- To decode base64 strings to an array of long values, see base64_decode_toarray()
- To encode strings to base64 string, see base64_encode_tostring()

Feedback

base64_decode_toguid()

Article • 12/28/2022

Decodes a base64 string to a GUID.

Syntax

`base64_decode_toguid(` *base64_string* `)`

Parameters

Name	Type	Required	Description
base64_string	string	✓	The value to decode from base64 to a GUID.

Returns

Returns a GUID decoded from a base64 string.

Example

Run the query

Kusto

```
print Quine = base64_decode_toguid("JpbpECu8dUy7Pv5gbeJXAA==")
```

Output

Quine
10e99626-bc2b-754c-bb3e-fe606de25700

If you try to decode an invalid base64 string, "null" will be returned:

Run the query

Kusto

```
print Empty = base64_decode_toguid("abcd1231")
```

See also

To encode a GUID to a base64 string, see base64_encode_fromguid().

Feedback

base64_encode_fromarray()

Article • 12/28/2022

Encodes a base64 string from a bytes array.

Syntax

`base64_encode_fromarray(` *base64_string_decoded_as_array* `)`

Parameters

Name	Type	Required	Description
base64_string_decoded_as_array	dynamic	✓	The bytes array to be encoded into a base64 string.

Returns

Returns the base64 string encoded from the bytes array.

Examples

Run the query

Kusto

```
let bytes_array = toscalar(print base64_decode_toarray("S3VzdG8="));
print decoded_base64_string = base64_encode_fromarray(bytes_array)
```

Output

decoded_base64_string
S3VzdG8=

Trying to encode a base64 string from an invalid bytes array that was generated from invalid UTF-8 encoded string will return null:

Run the query

Kusto

```
let empty_bytes_array = toscalar(print
base64_decode_toarray("U3RyaW5n0KHR0tGA0L7Rh9C60LA"));
print empty_string = base64_encode_fromarray(empty_bytes_array)
```

Output

empty_string

See also

- For decoding base64 strings to a UTF-8 string, see base64_decode_tostring()
- For encoding strings to a base64 string see base64_encode_tostring()
- This function is the inverse of base64_decode_toarray()

Feedback

base64_encode_tostring()

Article • 12/28/2022

Encodes a string as base64 string.

> **Deprecated aliases:** base64_encodestring()

Syntax

`base64_encode_tostring(`*string*`)`

Parameters

Name	Type	Required	Description
string	string	✓	The value to encode as a base64 string.

Returns

Returns *string* encoded as a base64 string.

Example

Run the query

```
Kusto
print Quine=base64_encode_tostring("Kusto")
```

Output

Quine
S3VzdG8=

See also

- To decode base64 strings to UTF-8 strings, see base64_decode_tostring().

- To decode base64 strings to an array of long values, see base64_decode_toarray().

Feedback

base64_encode_fromguid()

Article • 12/28/2022

Encodes a GUID to a base64 string.

Syntax

`base64_encode_fromguid(`*guid*`)`

Parameters

Name	Type	Required	Description
guid	guid	✓	The value to encode to a base64 string.

Returns

Returns a base64 string encoded from a GUID.

Example

Run the query

Kusto

```
print Quine = base64_encode_fromguid(toguid("ae3133f2-6e22-49ae-b06a-16e6a9b212eb"))
```

Output

Quine
8jMxriJurkmwahbmqbIS6w==

If you try to encode anything that isn't a GUID as below, an error will be thrown:

Run the query

Kusto

```
print Empty = base64_encode_fromguid("abcd1231")
```

See also

- To decode a base64 string to a GUID, see base64_decode_toguid().
- To create a GUID from a string, see toguid().

Feedback