DateTime/timespan functions

Function Name	Syntax	Description
ago()	ago(*timespan*)	Subtracts the given timespan from the current UTC clock time.
datetime_add()	datetime_add(*period*, *amount*, *datetime*)	Calculates a new datetime from a specified datepart multiplied by a specified amount, added to a specified datetime.
datetime_diff()	datetime_diff(*period*, *datetime1*, *datetime2*)	Calculates the number of the specified periods between two datetime values.
datetime_local_to_utc()	datetime_local_to_utc(*from*, *timezone*)	Converts local datetime to UTC datetime using a time-zone specification.
datetime_part()	datetime_part(*part*, *datetime*)	Extracts the requested date part as an integer value.
datetime_utc_to_local()	datetime_utc_to_local(*from*, *timezone*)	Converts UTC datetime to local datetime using a time-zone specification.
dayofmonth()	dayofmonth(*date*)	Returns the integer number representing the day number of the given month.
dayofweek()	dayofweek(*date*)	Returns the integer number of days since the preceding Sunday, as a timespan.
dayofyear()	dayofyear(*date*)	Returns the integer number represents the day number of the given year.
endofday()	endofday(*date* [, *offset*])	Returns the end of the day containing the date, shifted by an offset, if provided.
endofmonth()	endofmonth(*date* [, *offset*])	Returns the end of the month containing the date, shifted by an offset, if provided.

Function Name	Syntax	Description
endofweek()	endofweek(*date* [, *offset*])	Returns the end of the week containing the date, shifted by an offset, if provided. Start of the week is considered to be a Sunday.
endofyear()	endofyear(*date* [, *offset*])	Returns the end of the year containing the date, shifted by an offset, if provided.
format_datetime()	format_datetime(*date*, *format*)	Formats a datetime parameter based on the format pattern parameter.
format_timespan()	format_timespan(*timespan*, *format*)	Formats a format-timespan parameter based on the format pattern parameter.
getyear()	getyear(*date*)	Returns the year part of the datetime argument.
hourofday()	hourofday(*date*)	Returns the integer number representing the hour number of the given date.
make_datetime()	make_datetime(year, month, day[, hour, minute[, second]])	Creates a datetime scalar value from the specified date and time.
make_timespan()	make_timespan([day,] *hour*, *minute[, second]*)	Creates a timespan scalar value from the specified time period.
monthofyear()	monthofyear(*date*)	Returns the integer number that represents the month number of the given year.
now()	now([*offset*])	Returns the current UTC clock time, optionally offset by a given timespan.
startofday()	startofday(*date* [, *offset*])	Returns the start of the day containing the date, shifted by an offset, if provided.
startofmonth()	startofmonth(*date* [, *offset*])	Returns the start of the month containing the date, shifted by an offset, if provided.

Function Name	Syntax	Description
startofweek()	startofweek(*date* [, *offset*])	Returns the start of the week containing the date, shifted by an offset, if provided. Start of the week is considered to be a Sunday.
startofyear()	startofyear(*date* [, *offset*])	Returns the start of the year containing the date, shifted by an offset, if provided.
todatetime()	todatetime(*value*)	Converts input to datetime scalar.
totimespan()	totimespan(*value*)	Converts input to timespan scalar.
unixtime_microseconds_todatetime()	unixtime_microseconds_todatetime(*microseconds*)	Converts unix-epoch [from 1970-01-01 00:00:00] microseconds to UTC datetime.
unixtime_milliseconds_todatetime()	unixtime_milliseconds_todatetime(*milliseconds*)	Converts unix-epoch milliseconds to UTC datetime.
unixtime_nanoseconds_todatetime()	unixtime_nanoseconds_todatetime(nanoseconds)	Converts unix-epoch nanoseconds to UTC datetime.
unixtime_seconds_todatetime()	unixtime_seconds_todatetime(seconds)	Converts unix-epoch seconds to UTC datetime.
weekofyear()	week_of_year(*date*)	Returns an integer representing the week number.

Window scalar functions

Function Name	Syntax	Description
next()	next(*column*, [*offset*, *default_value*])	For the serialized row set, returns a value of a specified column from the later row according to the offset.
prev()	prev(*column*, [*offset*], [*default_value*])	For the serialized row set, returns a value of a specified column from the earlier row according to the offset.
row_cumsum()	row_cumsum(*term* [, *restart*])	Calculates the cumulative sum of a column.
row_number()	row_number([*StartingIndex* [, *Restart*]])	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()	row_rank_dense (*Term*)	Returns a row's dense rank in the serialized row set.
row_rank_min()	row_rank_min (*Term*)	Returns a row's minimal rank in the serialized row set.

Mathematical functions

Function Name	Syntax	Description
abs()	abs(*x*)	Calculates the absolute value of the input.
acos()	acos(*x*)	Returns the angle whose cosine is the specified number (the inverse operation of cos()).
asin()	asin(*x*)	Returns the angle whose sine is the specified number (the inverse operation of sin()).
atan()	atan(*x*)	Returns the angle whose tangent is the specified number (the inverse operation of tan()).
atan2()	atan2(*y*, *x*)	Calculates the angle, in radians, between the positive x-axis and the ray from the origin to the point (y, x).
beta_cdf()	beta_cdf(*x*, *alpha*, *beta*)	Returns the standard cumulative beta distribution function.
beta_inv()	beta_inv(*probability*, *alpha*, *beta*)	Returns the inverse of the beta cumulative probability beta density function.
beta_pdf()	beta_pdf(*x*, *alpha*, *beta*)	Returns the probability density beta function.
cos()	cos(*number*)	Returns the cosine function.
cot()	cot(*number*)	Calculates the trigonometric cotangent of the specified angle, in radians.
degrees()	degrees(*radians*)	Converts angle value in radians into value in degrees, using formula degrees = (180 / PI) * angle-in-radians.
erf()	erf(*x*)	Returns the error function.
erfc()	erfc(*x*)	Returns the complementary error function.
exp()	exp(*x*)	The base-e exponential function of x, which is e raised to the power x: e^x.
exp10()	exp10(*x*)	The base-10 exponential function of x, which is 10 raised to the power x: 10^x.
exp2()	exp2(*x*)	The base-2 exponential function of x, which is 2 raised to the power x: 2^x.
gamma()	gamma(*number*)	Computes gamma function.

Function Name	Syntax	Description
isfinite()	isfinite(*number*)	Returns whether input is a finite value (isn't infinite or NaN).
isinf()	isinf(*number*)	Returns whether input is an infinite (positive or negative) value.
isnan()	isnan(*number*)	Returns whether input is Not-a-Number (NaN) value.
log()	log(*number*)	Returns the natural logarithm function.
log10()	log10(*number*)	Returns the common (base-10) logarithm function.
log2()	log2(*number*)	Returns the base-2 logarithm function.
loggamma()	loggamma(*number*)	Computes log of absolute value of the gamma function.
not()	not(*expr*)	Reverses the value of its bool argument.
pi()	pi()	Returns the constant value of Pi (π).
pow()	pow(*base*, *exponent*)	Returns a result of raising to power.
radians()	radians(*degrees*)	Converts angle value in degrees into value in radians, using formula radians = (PI / 180) * angle-in-degrees.
rand()	rand() for a number from 0 to 1, or rand(N) for an integer from 0 to N-1.	Returns a random number.
range()	range(*start*, *stop* [, *step*])	Generates a dynamic array holding a series of equally spaced values.
round()	round(*number* [, *precision*])	Returns the rounded source to the specified precision.
sign()	sign(*number*)	Sign of a numeric expression.
sin()	sin(*number*)	Returns the sine function.
sqrt()	sqrt(*number*)	Returns the square root function.
tan()	tan(*x*)	Returns the tangent function.
welch_test()	welch_test(*mean1*, *variance1*, *count1*, *mean2*, *variance2*, *count2*)	Computes the p-value of the Welch-test function.

Rounding functions

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

Conditional functions

Function Name	Syntax	Description
case()	case(*predicate_1*, *then_1*, [*predicate_2*, *then_2*, ...] *else*)	Evaluates a list of predicates and returns the first result expression whose predicate is satisfied.
coalesce()	coalesce(*arg*, *arg_2*, [*arg_3*, ...])	Evaluates a list of expressions and returns the first non-null (or non-empty for string) expression.
iff()	iff(*if*, *then*, *else*)	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()	max_of(*arg*, *arg_2*, [*arg_3*, ...])	Returns the maximum value of several evaluated numeric expressions.
min_of()	min_of (*arg*, *arg_2*, [*arg_3*, ...])	Returns the minimum value of several evaluated numeric expressions.

String functions

Function Name	Syntax	Description
base64_encode_tostring()	base64_encode_tostring(*string*)	Encodes a string as base64 string.
base64_encode_fromguid()	base64_encode_fromguid(*guid*)	Encodes a GUID as base64 string.
base64_decode_tostring()	base64_decode_tostring(*base64_string*)	Decodes a base64 string to a UTF-8 string.
base64_decode_toarray()	base64_decode_toarray(*base64_string*)	Decodes a base64 string to an array of long values.
base64_decode_toguid()	base64_decode_toguid(*base64_string*)	Decodes a base64 string to a GUID.
countof()	countof(*source*, *search* [, *kind*])	Counts occurrences of a substring in a string. Plain string matches may overlap; regex matches don't.
extract()	extract(*regex*, *captureGroup*, *source* [, *typeLiteral*])	Get a match for a regular expression from a text string.
extract_all()	extract_all(*regex*, [*captureGroups*,] *source*)	Get all matches for a regular expression from a text string.
extract_json()	extract_json(*jsonPath*, *dataSource*, *type*)	Get a specified element out of a JSON text using a path expression.
has_any_index()	has_any_index (*source*, *values*)	Searches the string for items specified in the array and returns the position of the first item found in the string.
indexof()	indexof(*string*, *match*[, *start*[, *length*[, *occurrence*]]])	Function reports the zero-based index of the first occurrence of a specified string within input string.
isempty()	isempty(*value*)	Returns true if the argument is an empty string or is null.
isnotempty()	isnotempty(*value*)	Returns true if the argument isn't an empty string or a null.
isnotnull()	isnotnull(*value*)	Returns true if the argument is not null.
isnull()	isnull(*Expr*)	Evaluates its sole argument and returns a bool value indicating if the argument evaluates to a null value.

Function Name	Syntax	Description
parse_command_line()	parse_command_line(*command_line*, *parser_type*)	Parses a Unicode command line string and returns an array of the command line arguments.
parse_csv()	parse_csv(*csv_text*)	Splits a given string representing comma-separated values and returns a string array with these values.
parse_ipv4()	parse_ipv4(*ip*)	Converts input to long (signed 64-bit) number representation.
parse_ipv4_mask()	parse_ipv4_mask(*ip*, *prefix*)	Converts input string and IP-prefix mask to long (signed 64-bit) number representation.
parse_ipv6()	parse_ipv6(*ip*)	Converts IPv6 or IPv4 string to a canonical IPv6 string representation.
parse_ipv6_mask()	parse_ipv6_mask(*ip*, *prefix*)	Converts IPv6 or IPv4 string and netmask to a canonical IPv6 string representation.
parse_json()	parse_json(*json*)	Interprets a string as a JSON value and returns the value as dynamic.
parse_url()	parse_url(*url*)	Parses an absolute URL string and returns a dynamic object contains all parts of the URL.
parse_urlquery()	parse_urlquery(*query*)	Parses a url query string and returns a dynamic object contains the Query parameters.
parse_version()	parse_version (*version*)	Converts input string representation of version to a comparable decimal number.
replace_regex()	replace_regex(*source*, *lookup_regex*, *rewrite_pattern*)	Replace all regex matches with another string.
replace_string()	replace_string(*text*, *lookup*, *rewrite*)	Replace all single string matches with a specified string.
replace_strings()	replace_strings(*text*, *lookups*, *rewrites*)	Replace all multiple strings matches with specified strings.
punycode_from_string()	punycode_from_string('input_string')	Encodes domain name to Punycode form.
punycode_to_string()	punycode_to_string('input_string')	Decodes domain name from Punycode form.
reverse()	reverse(*value*)	Function makes reverse of input string.

Function Name	Syntax	Description
split()	split(*source*, *delimiter* [, *requestedIndex*])	Splits a given string according to a given delimiter and returns a string array with the contained substrings.
strcat()	strcat(*argument1*, *argument2* [, *argument3* ...])	Concatenates between 1 and 64 arguments.
strcat_delim()	strcat_delim(*delimiter*, *argument1*, *argument2*[, *argumentN*])	Concatenates between 2 and 64 arguments, with delimiter, provided as first argument.
strcmp()	strcmp(*string1*, *string2*)	Compares two strings.
strlen()	strlen(*source*)	Returns the length, in characters, of the input string.
strrep()	strrep(*value*, *multiplier*, [*delimiter*])	Repeats given string provided number of times (default - 1).
substring()	substring(*source*, *startingIndex* [, *length*])	Extracts a substring from a source string starting from some index to the end of the string.
toupper()	toupper(*value*)	Converts a string to upper case.
translate()	translate(*searchList*, *replacementList*, *source*)	Replaces a set of characters ('searchList') with another set of characters ('replacementList') in a given a string.
trim()	trim(*regex*, *source*)	Removes all leading and trailing matches of the specified regular expression.
trim_end()	trim_end(*regex*, *source*)	Removes trailing match of the specified regular expression.
trim_start()	trim_start(*regex*, *source*)	Removes leading match of the specified regular expression.
url_decode()	url_decode(*encoded_url*)	The function converts encoded URL into a regular URL representation.
url_encode()	url_encode(*url*)	The function converts characters of the input URL into a format that can be transmitted over the Internet.

Conversion functions

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

Units conversion functions

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

Flow control functions

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

Series element-wise functions

Function Name	Syntax	Description
series_abs()	series_abs(*series*)	Calculates the element-wise absolute value of the numeric series input.
series_acos()	series_acos(*series*)	Calculates the element-wise arccosine function of the numeric series input.
series_add()	series_add(*series1*, *series2*)	Calculates the element-wise addition of two numeric series inputs.
series_asin()	series_asin(*series*)	Calculates the element-wise arcsine function of the numeric series input.
series_atan()	series_atan(*series*)	Calculates the element-wise arctangent function of the numeric series input.
series_ceiling()	series_ceiling(*series*)	Calculates the element-wise ceiling function of the numeric series input.
series_cos()	series_cos(*series*)	Calculates the element-wise cosine function of the numeric series input.
series_divide()	series_divide(*series1*, *series2*)	Calculates the element-wise division of two numeric series inputs.
series_equals()	series_equals (*series1*, *series2*)	Calculates the element-wise equals (==) logic operation of two numeric series inputs.
series_exp()	series_exp(*series*)	Calculates the element-wise base-e exponential function (e^x) of the numeric series input.
series_floor()	series_floor(*series*)	Calculates the element-wise floor function of the numeric series input.

Function Name	Syntax	Description
series_greater()	series_greater(*series1*, *series2*)	Calculates the element-wise greater (>) logic operation of two numeric series inputs.
series_greater_equals()	series_greater_equals(*series1*, *series2*)	Calculates the element-wise greater or equals (>=) logic operation of two numeric series inputs.
series_less()	series_less(*series1*, *series2*)	Calculates the element-wise less (<) logic operation of two numeric series inputs.
series_less_equals()	series_less_equals(*series1*, *series2*)	Calculates the element-wise less or equal (<=) logic operation of two numeric series inputs.
series_log()	series_log(*series*)	Calculates the element-wise natural logarithm function (base-e) of the numeric series input.
series_multiply()	series_multiply(*series1*, *series2*)	Calculates the element-wise multiplication of two numeric series inputs.
series_not_equals()	series_not_equals(*series1*, *series2*)	Calculates the element-wise not equals (!=) logic operation of two numeric series inputs.
series_pow()	series_pow(*series1*, *series2*)	Calculates the element-wise power of two numeric series inputs.
series_sign()	series_sign(*series*)	Calculates the element-wise sign of the numeric series input.
series_sin()	series_sin(*series*)	Calculates the element-wise sine function of the numeric series input.
series_subtract()	series_subtract(*series1*, *series2*)	Calculates the element-wise subtraction of two numeric series inputs.
series_tan()	series_tan(*series*)	Calculates the element-wise tangent function of the numeric series input.

Series processing functions

Function Name	Syntax	Description
series_cosine_similarity()	series_cosine_similarity(*series1*, *series2*)	Calculates the cosine similarity of two numeric series.
series_decompose()	series_decompose(*Series*, [*Seasonality*, *Trend*, *Test_points*, *Seasonality_threshold*])	Does a decomposition of the series into components.
series_decompose_anomalies()	series_decompose_anomalies (*Series*, [*Threshold*, *Seasonality*, *Trend*, *Test_points*, *AD_method*, *Seasonality_threshold*])	Finds anomalies in a series based on series decomposition.
series_decompose_forecast()	series_decompose_forecast(*Series*, *Points*, [*Seasonality*, *Trend*, *Seasonality_threshold*])	Forecast based on series decomposition.
series_dot_product()	series_dot_product(*series1/numeric*, *series2/numeric*)	Calculates the dot product of two numeric series.
series_fill_backward()	series_fill_backward(*series*[, *missing_value_placeholder*])	Performs backward fill interpolation of missing values in a series.
series_fill_const()	series_fill_const(*series*, *constant_value*, [*missing_value_placeholder*])	Replaces missing values in a series with a specified constant value.
series_fill_forward()	series_fill_forward(*series*, [*missing_value_placeholder*])	Performs forward fill interpolation of missing values in a series.
series_fill_linear()	series_fill_linear(*series*, [*missing_value_placeholder* [, *fill_edges* [, *constant_value*]]])	Performs linear interpolation of missing values in a series.
series_fft()	series_fft(*x_real* [, *x_imaginary*])	Applies the Fast Fourier Transform (FFT) on a series.
series_fir()	series_fir(*series*, *filter* [, *normalize*[, *center*]])	Applies a Finite Impulse Response filter on a series.
series_fit_2lines()	project series_fit_2lines(*series*) OR project/extend (rs, si, v)=series_fit_2lines(series)	Applies two segments linear regression on a series, returning multiple columns.

Function Name	Syntax	Description
series_fit_2lines_dynamic()	series_fit_2lines_dynamic(*series*)	Applies two segments linear regression on a series, returning dynamic object.
series_fit_line()	series_fit_line(*series*)	Applies linear regression on a series, returning multiple columns.
series_fit_line_dynamic()	series_fit_line_dynamic(*series*)	Applies linear regression on a series, returning dynamic object.
series_fit_poly()	T | extend series_fit_poly(*y_series* [, *x_series*, *degree*])	Applies polynomial regression on a series, returning multiple columns.
series_ifft()	series_ifft(*fft_real* [, *fft_imaginary*])	Applies the Inverse Fast Fourier Transform (IFFT) on a series.
series_iir()	series_iir(*series*, *numerators*, *denominators*)	Applies an Infinite Impulse Response filter on a series.
series_magnitude()	series_magnitude(*series*)	Calculates the magnitude of the numeric series.
series_outliers()	series_outliers(*series* [, *kind*] [, *ignore_val*] [, *min_percentile*] [, *max_percentile*])	Scores anomaly points in a series.
series_pearson_correlation()	series_pearson_correlation(*series1*, *series2*)	Calculates the Pearson correlation coefficient of two series.
series_periods_detect()	series_periods_detect(*series*, *min_period*, *max_period*, *num_periods*)	Finds the most significant periods that exist in a time series.
series_periods_validate()	series_periods_validate(*series*, *period1* [, *period2*, . . .])	Checks whether a time series contains periodic patterns of given lengths.
series_seasonal()	series_seasonal(*series* [, *period*])	Finds the seasonal component of the series.
series_stats()	... | extend (*Name*, ...) = series_stats (*series* [, *ignore_nonfinite*])	Returns statistics for a series in multiple columns.
series_stats_dynamic()	series_stats_dynamic(*series* [, *ignore_nonfinite*])	Returns statistics for a series in dynamic object.
series_sum()	series_sum(*series*)	Calculates the sum of numeric series elements.

Binary functions

Function Name	Syntax	Description
binary_and()	binary_and(*value1*, *value2*)	Returns a result of the bitwise and operation between two values.
binary_not()	binary_not(*value*)	Returns a bitwise negation of the input value.
binary_or()	binary_or(*value1*, *value2*)	Returns a result of the bitwise or operation of the two values.
binary_shift_left()	binary_shift_left(*value*, *shift*)	Returns binary shift left operation on a pair of numbers: a << n.
binary_shift_right()	binary_shift_right(*value*, *shift*)	Returns binary shift right operation on a pair of numbers: a >> n.
binary_xor()	binary_xor(*value1*, *value2*)	Returns a result of the bitwise xor operation of the two values.
bitset_count_ones()	bitset_count_ones(*value*)	Returns the number of set bits in the binary representation of a number.

Dynamic/array functions

Function Name	Syntax	Description
array_concat()	array_concat(*arr* [, ...])	Concatenates a number of dynamic arrays to a single array.
array_iff()	array_iff(*condition_array*, *when_true*, *when_false*)	Applies element-wise iif function on arrays.
array_index_of()	array_index_of(*array*, *value* [, *start* [, *length* [, *occurence*]]])	Searches the array for the specified item, and returns its position.
array_length()	array_length(*array*)	Calculates the number of elements in a dynamic array.
array_reverse()	array_reverse(*value*)	Reverses the order of the elements in a dynamic array.
array_rotate_left()	array_rotate_left(*array*, *rotate_count*)	Rotates values inside a dynamic array to the left.
array_rotate_right()	array_rotate_right(*array*, *rotate_count*)	Rotates values inside a dynamic array to the right.
array_shift_left()	array_shift_left(*array*, *shift_count* [, *default_value*])	Shifts values inside a dynamic array to the left.
array_shift_right()	array_shift_right(*array*, *shift_count* [, *default_value*])	Shifts values inside a dynamic array to the right.
array_slice()	array_slice(*array*, *start*, *end*)	Extracts a slice of a dynamic array.
array_sort_asc()	array_sort_asc(*array1*[, ..., *arrayN*][, *nulls_last*])	Sorts a collection of arrays in ascending order.
array_sort_desc()	array_sort_desc(*array1*[, ..., *argumentN*][, *nulls_last]*)	Sorts a collection of arrays in descending order.
array_split()	array_split(*array*, *index*)	Builds an array of arrays split from the input array.
array_sum()	array_sum(*array*)	Calculates the sum of a dynamic array.
bag_has_key()	bag_has_key(*bag*, *key*)	Checks whether a dynamic bag column contains a given key.
bag_keys()	bag_keys(*object*)	Enumerates all the root keys in a dynamic property-bag object.
bag_merge()	bag_merge(*bag1*, *bag2*[, *bag3*, ...])	Merges dynamic property-bags into a dynamic property-bag with all properties merged.

Function Name	Syntax	Description
bag_pack()	bag_pack(*key1*, *value1*, *key2*, *value2*, …)	Creates a dynamic object (property bag) from a list of names and values.
bag_pack_columns()	bag_pack_columns(*column1*, *column2*, …)	Creates a dynamic object (property bag) from a list of columns.
bag_remove_keys()	bag_remove_keys(*bag*, *keys*)	Removes keys and associated values from a dynamic property-bag.
bag_set_key()	bag_set_key(*bag*, *key*, *value*)	Sets a given key to a given value in a dynamic property-bag.
jaccard_index()	jaccard_index(*set1*, *set2*)	Computes the Jaccard index of two sets.
pack_all()	pack_all([*ignore_null_empty*])	Creates a dynamic object (property bag) from all the columns of the tabular expression.
pack_array()	pack_array(*value1*, [*value2*, …]) or pack_array(*)	Packs all input values into a dynamic array.
repeat()	repeat(*value*, *count*)	Generates a dynamic array holding a series of equal values.
set_difference()	set_difference(*set1*, *set2* [, *set3*, …])	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()	set_has_element(*set*, *value*)	Determines whether the specified array contains the specified element.
set_intersect()	set_intersect(*set1*, *set2* [, *set3*, …])	Returns an array of the set of all distinct values that are in all arrays.
set_union()	set_union(*set1*, *set2* [, *set3*, …])	Returns an array of the set of all distinct values that are in any of provided arrays.
treepath()	treepath(*object*)	Enumerates all the path expressions that identify leaves in a dynamic object.
zip()	zip(*arrays*)	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.

IPv4/ IPv6 functions

Function Name	Syntax	Description
ipv4_compare()	ipv4_compare(*Expr1*, *Expr2*[, *PrefixMask*])	Compares two IPv4 strings.
ipv4_is_in_range()	ipv4_is_in_range(*Ipv4Address*, *Ipv4Range*)	Checks if IPv4 string address is in IPv4-prefix notation range.
ipv4_is_in_any_range()	ipv4_is_in_any_range(Ipv4Address, Ipv4Range/Ranges [, Ipv4Range ...])	Checks if IPv4 string address is any of the IPv4-prefix notation ranges.
ipv4_is_match()	ipv4_is_match(*ip1*, *ip2*[, *prefix*])	Matches two IPv4 strings.
ipv4_is_private()	ipv4_is_private(*ip*)	Checks if IPv4 string address belongs to a set of private network IPs.
ipv4_netmask_suffix	ipv4_netmask_suffix(*ip*)	Returns the value of the IPv4 netmask suffix from IPv4 string address.
parse_ipv4()	parse_ipv4(*ip*)	Converts input string to long (signed 64-bit) number representation.
parse_ipv4_mask()	parse_ipv4_mask(*ip*, *prefix*)	Converts input string and IP-prefix mask to long (signed 64-bit) number representation.
ipv4_range_to_cidr_list()	ipv4_range_to_cidr_list(*StartAddress*, *EndAddress*)	Converts IPv4 address range to a list of CIDR ranges.
ipv6_compare()	ipv6_compare(*ip1*, *ip2*[, *prefix*])	Compares two IPv4 or IPv6 strings.
ipv6_is_match()	ipv6_is_match(*ip1*, *ip2*[, *prefix*])	Matches two IPv4 or IPv6 strings.
parse_ipv6()	parse_ipv6(*ip*)	Converts IPv6 or IPv4 string to a canonical IPv6 string representation.
parse_ipv6_mask()	parse_ipv6_mask(*ip*, *prefix*)	Converts IPv6 or IPv4 string and netmask to a canonical IPv6 string representation.
format_ipv4()	format_ipv4(*ip* [, *prefix*])	Parses input with a netmask and returns string representing IPv4 address.

Function Name	Syntax	Description
format_ipv4_mask()	format_ipv4_mask(*ip* [, *prefix*])	Parses input with a netmask and returns string representing IPv4 address as CIDR notation.
ipv6_is_in_range()	ipv6_is_in_range(*Ipv6Address*, *Ipv6Range*)	Checks if an IPv6 string address is in IPv6-prefix notation range.
ipv6_is_in_any_range()	ipv6_is_in_any_range(*Ipv6Address*, *Ipv6Range/Ranges* [, *Ipv6Range* ...])	Checks if an IPv6 string address is in any of the IPv6-prefix notation ranges.
geo_info_from_ip_address()	geo_info_from_ip_address(*IpAddress*)	Retrieves geolocation information about IPv4 or IPv6 addresses.

IPv4 text match functions

Function Name	Syntax	Description
has_ipv4()	has_ipv4(*source*, *ip_address*)	Searches for an IPv4 address in a text.
has_ipv4_prefix()	has_ipv4_prefix(*source*, *ip_address_prefix*)	Searches for an IPv4 address or prefix in a text.
has_any_ipv4()	has_any_ipv4(*source*, *ip_address* [, *ip_address_2*, ...])	Searches for any of the specified IPv4 addresses in a text.
has_any_ipv4_prefix()	has_any_ipv4_prefix(*source*, *ip_address_prefix* [, *ip_address_prefix_2*, ...])	Searches for any of the specified IPv4 addresses or prefixes in a text.

Metadata functions

Function Name	Syntax	Description
column_ifexists()	column_ifexists(*columnName*, *defaultValue*)	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()	current_cluster_endpoint()	Returns the current cluster running the query.
current_database()	current_database()	Returns the name of the database in scope.
current_principal()	current_principal()	Returns the current principal running this query.
current_principal_details()	current_principal_details()	Returns details of the principal running the query.
current_principal_is_member_of()	current_principal_is_member_of(*group*)	Checks group membership or principal identity of the current principal running the query.
cursor_after()	cursor_after(*RHS*)	Used to access to the records that were ingested after the previous value of the cursor.
estimate_data_size()	estimate_data_size(*columns*)	Returns an estimated data size of the selected columns of the tabular expression.
extent_id()	extent_id()	Returns a unique identifier that identifies the data shard ("extent") that the current record resides in.
extent_tags()	extent_tags()	Returns a dynamic array with the tags of the data shard ("extent") that the current record resides in.
ingestion_time()	ingestion_time()	Retrieves the record's $IngestionTime hidden datetime column, or null.

Type functions

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

Scalar aggregation functions

Function Name	Syntax	Description
dcount_hll()	dcount_hll(*hll*)	Calculates the dcount from hll results (which was generated by hll or hll-merge).
hll_merge()	hll_merge(*hll*, *hll2*, [*hll3*, ...])	Merges hll results (scalar version of the aggregate version hll-merge()).
percentile_tdigest()	percentile_tdigest(*expr*, *percentile1*, *typeLiteral*)	Calculates the percentile result from tdigest results (which was generated by tdigest or merge_tdigest).
percentile_array_tdigest()	percentiles_array_tdigest(*tdigest*, *percentile1* [, *percentile2*, ...]) OR percentiles_array_tdigest(tdigest, Dynamic array [, typeLiteral])	Calculates the percentile array result from tdigest results (which was generated by tdigest or merge_tdigest).
percentrank_tdigest()	percentrank_tdigest(*digest*, *value*)	Calculates the percentage ranking of a value in a dataset.
rank_tdigest()	rank_tdigest(*digest*, *value*)	Calculates relative rank of a value in a set.
merge_tdigest()	merge_tdigest(*exprs*)	Merge tdigest results (scalar version of the aggregate version tdigest-merge()).

Geospatial functions

Function Name	Syntax	Description
geo_angle()	geo_angle(*p1_longitude*, *p1_latitude*, *p2_longitude*, *p2_latitude*, *p3_longitude*, *p3_latitude*)	Calculates clockwise angle in radians between two lines on Earth.
geo_azimuth()	geo_azimuth(*p1_longitude*, *p1_latitude*, *p2_longitude*, *p2_latitude*)	Calculates clockwise angle in radians between the line from point1 to true north and a line from point1 to point2 on Earth.
geo_distance_2points()	geo_distance_2points(*p1_longitude*, *p1_latitude*, *p2_longitude*, *p2_latitude*)	Calculates the shortest distance between two geospatial coordinates on Earth.
geo_distance_point_to_line()	geo_distance_point_to_line(*longitude*, *latitude*, *lineString*)	Calculates the shortest distance between a coordinate and a line or multiline on Earth.
geo_distance_point_to_polygon()	geo_distance_point_to_polygon(*longitude*, *latitude*, *polygon*)	Calculates the shortest distance between a coordinate and a polygon or multipolygon on Earth.
geo_intersects_2lines()	geo_intersects_2lines(*lineString1*, *lineString2*)	Calculates whether the two lines or multilines intersects.
geo_intersects_2polygons()	geo_intersects_2polygons(*polygon1*, *polygon2*)	Calculates whether the two polygons or multipolygons intersects.
geo_intersects_line_with_polygon()	geo_intersects_line_with_polygon(*lineString*, *polygon*)	Calculates whether the line or multiline intersects with polygon or multipolygon.
geo_intersection_2lines()	geo_intersection_2lines(*lineString1*, *lineString2*)	Calculates the intersection of two lines or multilines.
geo_intersection_2polygons()	geo_intersection_2polygons(*polygon1*, *polygon2*)	Calculates the intersection of two polygons or multipolygons.
geo_intersection_line_with_polygon()	geo_intersection_line_with_polygon(*lineString*, *polygon*)	Calculates the intersection of line or multiline with polygon or multipolygon.

Function Name	Syntax	Description
geo_point_buffer()	geo_point_buffer(*longitude*, *latitude*, *radius*, *tolerance*)	Calculates polygon that contains all points within the given radius of the point on Earth.
geo_point_in_circle()	geo_point_in_circle(*p_longitude*, *p_latitude*, *pc_longitude*, *pc_latitude*, *c_radius*)	Calculates whether the geospatial coordinates are inside a circle on Earth.
geo_point_in_polygon()	geo_point_in_polygon(*longitude*, *latitude*, *polygon*)	Calculates whether the geospatial coordinates are inside a polygon or a multipolygon on Earth.
geo_point_to_geohash()	geo_point_to_geohash(*longitude*, *latitude*, [*accuracy*])	Calculates the Geohash string value for a geographic location.
geo_point_to_s2cell()	geo_point_to_s2cell(*longitude*, *latitude*, [*level*])	Calculates the S2 Cell token string value for a geographic location.
geo_point_to_h3cell()	geo_point_to_h3cell(*longitude*, *latitude*, [*resolution*])	Calculates the H3 Cell token string value for a geographic location.
geo_line_buffer()	geo_line_buffer(*lineString*, *radius*, *tolerance*)	Calculates polygon or multipolygon that contains all points within the given radius of the input line or multiline on Earth.
geo_line_centroid()	geo_line_centroid(*lineString*)	Calculates the centroid of line or a multiline on Earth.
geo_line_densify()	geo_line_densify(*lineString*, *tolerance*, [*preserve_crossing*])	Converts planar line edges to geodesics by adding intermediate points.
geo_line_length()	geo_line_length(*lineString*)	Calculates the total length of line or a multiline on Earth.
geo_line_simplify()	geo_line_simplify(*lineString*, *tolerance*)	Simplifies line or a multiline by replacing nearly straight chains of short edges with a single long edge on Earth.
geo_line_to_s2cells()	geo_line_to_s2cells(*lineString* [, *level*[, *radius*]])	Calculates S2 cell tokens that cover a line or multiline on Earth. Useful geospatial join tool.

Function Name	Syntax	Description
geo_polygon_area()	geo_polygon_area(*polygon*)	Calculates the area of polygon or a multipolygon on Earth.
geo_polygon_buffer()	geo_polygon_buffer(*polygon*, *radius*, *tolerance*)	Calculates polygon or multipolygon that contains all points within the given radius of the input polygon or multipolygon on Earth.
geo_polygon_centroid()	geo_polygon_centroid(*polygon*)	Calculates the centroid of polygon or a multipolygon on Earth.
geo_polygon_densify()	geo_polygon_densify(*polygon*, *tolerance*, [*preserve_crossing*])	Converts polygon or multipolygon planar edges to geodesics by adding intermediate points.
geo_polygon_perimeter()	geo_polygon_perimeter(*polygon*)	Calculates the length of the boundary of polygon or a multipolygon on Earth.
geo_polygon_simplify()	geo_polygon_simplify(*polygon*, *tolerance*)	Simplifies polygon or a multipolygon by replacing nearly straight chains of short edges with a single long edge on Earth.
geo_polygon_to_s2cells()	geo_polygon_to_s2cells(*polygon* [, *level*[, *radius*]])	Calculates S2 Cell tokens that cover a polygon or multipolygon on Earth. Useful geospatial join tool.
geo_polygon_to_h3cells()	geo_polygon_to_h3cells(*polygon* [, *resolution*[, *radius*]])	Converts polygon to H3 cells. Useful geospatial join and visualization tool.
geo_geohash_to_central_point()	geo_geohash_to_central_point(*geohash*)	Calculates the geospatial coordinates that represent the center of a Geohash rectangular area.
geo_geohash_neighbors()	geo_geohash_neighbors(*geohash*)	Calculates the geohash neighbors.
geo_geohash_to_polygon()	geo_geohash_to_polygon(*geohash*)	Calculates the polygon that represents the geohash rectangular area.

Function Name	Syntax	Description
geo_s2cell_to_central_point()	geo_s2cell_to_central_point(*s2cell*)	Calculates the geospatial coordinates that represent the center of an S2 Cell.
geo_s2cell_neighbors()	geo_s2cell_neighbors(*s2cell*)	Calculates the S2 cell neighbors.
geo_s2cell_to_polygon()	geo_s2cell_to_polygon(*s2cell*)	Calculates the polygon that represents the S2 Cell rectangular area.
geo_h3cell_to_central_point()	geo_h3cell_to_central_point(*h3cell*)	Calculates the geospatial coordinates that represent the center of an H3 Cell.
geo_h3cell_neighbors()	geo_h3cell_neighbors(*h3cell*)	Calculates the H3 cell neighbors.
geo_h3cell_to_polygon()	geo_h3cell_to_polygon(*h3cell*)	Calculates the polygon that represents the H3 Cell rectangular area.
geo_h3cell_parent()	geo_h3cell_parent(*h3cell*, *resolution*)	Calculates the H3 cell parent.
geo_h3cell_children()	geo_h3cell_children(*h3cell*, *resolution*)	Calculates the H3 cell children.
geo_h3cell_level()	geo_h3cell_level(*h3cell*)	Calculates the H3 cell resolution.
geo_h3cell_rings()	geo_h3cell_rings(*h3cell*, *distance*)	Calculates the H3 cell Rings.
geo_simplify_polygons_array()	geo_simplify_polygons_array(*polygons*, *tolerance*)	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()	geo_union_lines_array(*lineStrings*)	Calculates the union of lines or multilines on Earth.
geo_union_polygons_array()	geo_union_polygons_array(*polygons*)	Calculates the union of polygons or multipolygons on Earth.

Hash functions

Function Name	Syntax	Description
hash()	hash(*source* [, *mod*])	Returns a hash value for the input value.
hash_combine()	hash_combine(*h1*, *h2* [, *h3* ...])	Combines two or more hash values.
hash_many()	hash_many(*s1*, *s2* [, *s3* ...])	Returns a combined hash value of multiple values.
hash_md5()	hash_md5(*source*)	Returns an MD5 hash value for the input value.
hash_sha1()	hash_sha1(*source*)	Returns a SHA1 hash value for the input value.
hash_sha256()	hash_sha256(*source*)	Returns a SHA256 hash value for the input value.
hash_xxhash64()	hash_xxhash64(*source* [, *mod*])	Returns an XXHASH64 hash value for the input value.