Search and Analytics Language Query Specification

An example of a general search query structure looks like the following:

```
SELECT * | expression[, expression[, ...]]
FROM datatype_name
[WHERE expression]
[WITH TAXONOMY taxonomy_name]
[GROUP BY [PERMUTED] alias | fieldname [, fieldname | alias | fieldname [, ...]]]
[HAVING expression]
[ORDER BY alias | fieldname [ASC | DESC][, alias | fieldname [, ...]]]
[LIMIT limit]
```

A query is a sequence of tokens. Tokens are separated by whitespace, except for non-alphanumeric operators (e.g. +, -), which are self-delimiting.

Below are examples of valid expressions:

```
SELECT time_recv + +1 FROM logmsgs;
SELECT time_recv + -1 FROM logmsgs;
```

Below are examples of valid expressions that have the self-delimiter + operator:

```
SELECT time_recv++1 FROM logmsgs;
SELECT time_recv+-1 FROM logmsgs;
```

A valid query must have a condition bound to the time of the message because you can only search within a specific time frame.. The condition must be entered in the time field (e.g. time_recv) from the primary section of the data type (e.g. logmsgs). Some common data types and their time fields are:

Type	Time field
auditmsgs	`event_ts`

Type	Time field
bdrmsgs	`create_ts`
fimdata	`ts`
hoststate	`create_ts`
logmsgs	`time_recv`
observation	`ts`
secmsgs	`ts`
telemetry	`create_ts`
udrmsgs	`create_ts`
vpcflow	`create_ts`

A search can only be executed against one data type (e.g. `logmsgs`) and one account ID.

Identifier

Identifiers are used as a name for something, such as field, alias, function, taxonomy or datatypes. Identifiers matching the regular expression `/[a-zA-Z@_][a-z.A-Z0-9_]*/` are specified as is. Anything else requires quoting with `"` where `"` and `\` escaped with a backslash (`\`).

Identifiers that collide with keywords (e.g. function names) also must be quoted to avoid ambiguity.

Below are examples of identifiers:

Identifier	Description
foo	Common case
"foo"	Not necessary, but valid and identical to above
"from", "count"	Collides with keyword, so it must be escaped
"foo.bar"	This is not json path, but rather identifier with dot in name
"\""	Exists only for completeness

Constant identifier

`false`, `true` and `null` are well-known language constants.

Strings are single quoted (with `'`), where `'` and `\` escaped by a backslash (`\`).

Numbers are `[sign]digits[.digits]` (e.g. the regular expression `/^[-+]?[0-9]+(.[0-9]+)?/`.

Field names identifier

Search is built as a data type agnostic service. Field identifiers are validated against a data type schema, which is defined by `ingest` service dynamically.

Using a field as part of a search query is as simple as specifying its name. There is no risk of ambiguity because every search query is executed in the context for a data type and the field name is unique within a section.

Below is an example of using field as an identifier:

```
SELECT ts, event_id, proto WHERE ts BETWEEN 1483228800 AND 1484265600
```

Json-path identifier

Indirect fields or derived fields, which return complex types (defined as `map` output), can be accessed in query with Jpath syntax, such as the example below:

```
SELECT ts, event_id, payload WHERE payload[0].ts BETWEEN 1483228800 AND 1484265600
```

An asterisk can be used in jpath fields for generating collection of child objects, like the example below:

```
SELECT payload[*].data WHERE ts BETWEEN 1483228800 AND 1484265600
```

Other language elements

Other tokens, used in specific constructions are below:

Token	Description
`.`	jpath expressions
`,`	Separator in `SELECT`, `GROUP BY`, `ORDER BY` statements and function arguments
`[`	jpath expressions
`]`	jpath expressions
`(`	function calls, expression grouping
`)`	function calls, expression grouping

Expressions

Expressions are assembled from non-keyword tokens and used in statements.

Some expression examples are below:

Expression	Description
field_name	Identifier is a valid expression

Expression	Description
metadata.dict.dict.some_meta[123]	jpath expression
COUNT()	Function
GEOIP(source_ip, 'country')	Another function, with arguments
COUNT() > 10 AND AVG(foo) < 10	Functions and operators
GEOIP(parsed.tokens.src_ip).country	Syntactically correct, but not supported now

Case sensitivity

Field names are case-sensitive, e.g. `parsed.json.foo` and `parsed.json.Foo,` are different valid field names.

All functions are case-insensitive while working with string literals or values stored in fields, e.g. `CAST(xyz, 'iNtEgEr')` and `CAST(xyz, 'integer')` are same valid names. The only exceptions to this as for now is string literals used for regular expression, e.g. `FooBar*` and `foobar*` are different valid regular expression patterns.

Alias names are case-sensitive, e.g. `somealias` and `someAlias` are different valid alias names.

Taxonomy names are case-insensitive, e.g. `SeCuRiTy` and `security` are same valid taxonomy identifiers.

Parity exceptions

This document describes common SQL used for search and analytics queries. All queries should be compatible between these two entities, however there are some specific cases where full parity is not achieved. Such use cases are defined here.

TBD

1. Structure

1. Structure - AS

1.0.0

Specifies a name for the result of a calculation or renames a field.

```
expression | fieldname AS identifier
```

Aliases can be declared in the SELECT clause only. If a user requires special sorting or grouping by a transformed field then the aliased identifier of such transformation should be passed into other query parts (e.g. `ORDER_BY`, `GROUP_BY` or `HAVING`)

Creating Aliases

In most cases, aliases are used to specify the name of the field for display. Aliases can also be used for the results of functions and complex expressions, such as the example below:

```
SELECT foo AS bar FROM ...

SELECT SUM(foo) AS SumFoo FROM ...

SELECT (foo * 2) AS DoubledFoo FROM ...
```

If an alias contains:

- any characters other than ASCII Latin letters and digits ([a-zA-Z0-9]),
- starts with a character other than a letter,
- the alias is a keyword (SELECT, AS, FROM, WITH, TAXONOMY, WHERE, GROUP, BY, PERMUTED, HAVING, ORDER),
- or function name

hen it must be enclosed in double quotes, such as the example below:

```
SELECT foo AS "My [beautiful] field!" FROM ...

SELECT foo AS "Group", COUNT(bar) AS "Count" FROM ...
```

Using names of datatype fields that were already used in the same query as an alias is prohibited. The following are examples of invalid queries:

```
SELECT foo AS foo FROM ...
```

```
SELECT foo, bar AS foo FROM ...
```

Declaring a single alias for multiple fields is an error is an invalid query:

```
SELECT foo AS "My Field", bar AS "My Field" FROM ...
```

However, aliases are case-sensitive, so the following query is valid:

```
SELECT foo AS "My Field", bar AS "my field" FROM ...
```

If you do not specify a field or expression name, a name is chosen automatically. If the expression of the field is a simple datatype field reference, then the chosen name is the same as that name of the field. In more complex cases, the system may fall back on a generated name.

Remember that you should not rely on automatically generated names, as they may be changed in the future. If you want to refer to fields or expressions, you must use aliases.

Using Aliases

Search allows the use of aliases to display in the result fields, and also in expressions of clauses SELECT, WHERE, GROUP BY / GROUP BY PERMUTED, HAVING and ORDER BY, such as the example below.:

```
SELECT foo AS Bar FROM ... WHERE Bar > 0 ...

SELECT foo AS Bar, (Bar + Bar) AS DoubledBar FROM ...

SELECT abc AS FieldName, SUM(foo) AS SumFoo FROM ... GROUP BY FieldName ...

SELECT foo AS Bar FROM ... ORDER BY Bar

SELECT abc AS FieldName, SUM(foo) AS SumFoo FROM ... WHERE FieldName = 'Some Name' GR
OUP BY FieldName HAVING SumFoo > 100 ORDER BY SumFoo

SELECT foo AS "Very Special(!!!) Alias" FROM ... WHERE "Very Special(!!!) Alias" > 0
...
```

Aliases can be used immediately after the declaration. For example, the following query is valid:

```
SELECT x AS Y, Y AS Z FROM ... WHERE Z = ...
```

And the following is an invalid query:

```
SELECT Y AS Z, x AS Y FROM ... WHERE Z = ...
```

Partial Aliases

If the type of expression for which the alias is specified is an object (or JSON), the alias can be used as part of another expression in WHERE and ORDER BY clauses, such as the example below:

```
SELECT GEOIP(foo) AS Geo FROM ... WHERE Geo.country = 'us' ...

SELECT some_json AS Root FROM ... ORDER BY Root.main_field

some_json.nested_json AS "My JSON" FROM ... WHERE "My JSON".int_value > 0 ... ORDER B
Y "My JSON".name
```

Named expressions

Aliases can be declared to display fields, and for use as a reference in the SELECT and WHERE clauses. Such aliases are called named expressions. To create a named expression, enclose the alias declaration in parentheses, such as the example below:

```
SELECT (GEOIP(foo) AS Geo).city_name FROM ... WHERE Geo.country = 'us' ...

SELECT ((foo AS MainField) * 10) AS Bar, (MainField + 100) AS Baz FROM ... WHERE Main
Field > 0 ...
```

Named expressions apply the same rules that apply to all aliases.

Structure - FROM

1.0.0
Specifies the data type for query.

```
FROM datatype_name
```

The mandatory `FROM` statement specifies well-known searchable data types.

FROM examples:

Example	Description
FROM logmsgs	Use `logmsgs` data type
FROM fimdata	Use `fimdata` data type
FROM snmsgs	Use `snmsgs` data type
FROM observation	Use `observation` data type

QUERY

```
FROM
```

1. Structure - GROUP BY [PERMUTED]

1.0.0
Specifies grouping for aggregated results.

```
[GROUP BY [PERMUTED] alias | fieldname [, fieldname | alias | fieldname [, ...]]]
```

The optional `GROUP BY` statement specifies the group criteria for aggregated searches.

If `GROUP BY` is not set, all final records are aggregated into single output row by default.

`PERMUTED` option specifies an alternative version of `GROUP BY`. GROUP BY PERMUTED uses array elements as group keys instead of using array values directly and grouping by full array values.

Therefore, there is a group for each unique value in array and each input row can fall into multiple groups, rather than one.

`GROUP BY` examples

GROUP BY field1, field2	Aggregate records that have the same field1, field2 combination
GROUP BY PERMUTED array_field1, array_field2	Aggregate records that have the full union of values from field1 and field2 combination

QUERY

```
GROUP_BY
```

1. Structure - HAVING

1.0.0

```
[HAVING expression]
```

The optional `HAVING` statement specifies the condition that messages are satisfied to be returned after aggregation. If not set, all records will be filtered (`expression = true` is used).

expression should return a boolean value and contain only transforms functions over original field names or aliases over aggregated functions.

`HAVING` examples

Example	Description
SELECT COUNT() AS cnt ... HAVING cnt > 5	Intended usage
SELECT field ... GROUP BY field HAVING field > 5	Not too useful, but works too
SELECT field.a GROUP BY field ... HAVING field.a.b	Variation of above example

QUERY

```
HAVING
```

1. Structure - LIMIT

1.0.0

Specifies maximum number of final records in results.

```
[LIMIT limit]
```

If `LIMIT` is not set, maximum value of 500000 is used as default.

QUERY

```
LIMIT
```

1. Structure - ORDER BY

1.0.0
Defines the sort specification for the results.

```
[ORDER BY alias | fieldname [ASC | DESC][, alias | fieldname [, ...]]]
```

The Optional `ORDER BY` statement specifies the order and sort directions of the results.

If `ORDER BY` is not specified, natural time-specific primary key(s) are used as defaults for regular searches (for example `time_recv DESC` for log messages)

If the `ORDER BY` direction is not set, `DESC` is used by default.

The order of messages that have the same sort key is undefined and should not be expected to be stable.

QUERY

```
ORDER_BY
```

1. Structure - SELECT

1.0.0
Specifies fields, functions, or aggregators requested to be extracted.

```
SELECT * | expression[, expression[, ...]]
```

The mandatory `SELECT` statement specifies values you want returned. It is either `*`, or list of expressions with optional aliases.

The presence of aggregation functions (such as `SUM` or `AVG`) in expressions makes the search aggregated. The `SELECT` clause of an aggregated search can contain only aggregation functions or expressions present in `GROUP BY` statement.

See aggregators and transforms sections for exact functions specified in language.

`SELECT` examples:

Example	Description
SELECT *	Return all fields, according to schema
SELECT field1, field2, metadata.dict.dict.derivedField	Return explicitly specified fields
SELECT field1 AS alias1	Rename field
SELECT "Fancy named field" AS "Fancy alias"	Non-trivial names require quoting
SELECT SUM(field1) AS sum_of_field1	Aggregation functions are another kind of expression

QUERY

```
SELECT
```

1. Structure - WHERE

1.0.0

Specifies condition for query.

```
[WHERE expression]
```

The optional `WHERE` statement specifies conditions that messages are satisfied to be returned during initial data extraction. If not set, all records will be extracted (i.e. `WHERE true` is used).

expression must return a boolean value and contain conditions on record time key for time range to be inferred. This condition cannot contain aggregation functions (as it is applied to individual record).

QUERY

```
WHERE
```

1. Structure - WITH TAXONOMY

1.0.0
Specifies, which token taxonomy to use.

```
[WITH TAXONOMY taxonomy_name]
```

The optional `WITH TAXONOMY` statement specifies a well-known token taxonomy name. If not set, the default taxonomy 'Default' is used

`WITH TAXONOMY` examples:

Example	Description
WITH TAXONOMY security	Use `security` taxonomy

QUERY

```
WITH_TAXONOMY
```

2. Transforms

2. Transforms - !=

1.0.0
SUMMARY

Inequality test.

SYNOPSIS

a **!=** b

a is float|integer|map|list|string|boolean

b is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

!= returns true if a and b are both non-NULL and a is not equal to b. It returns false otherwise.

EXAMPLES

Filter records where port is not equal to 22:

... WHERE port != 22

TRANSFORMS

```
!=
```

2. Transforms - *

1.0.0

SUMMARY

Numerical multiplication.

SYNOPSIS

a * b

a is float|integer

b is float|integer

return type is float|integer

DESCRIPTION

* performs multiplication of its two operands. If the operands differ in type then the lower will be promoted to the type of the higher, where the decreasing order of types is float, unsigned integer and signed integer. The operator returns the type of its operands or NULL if at least one of the operands is NULL or is neither an integer nor a float.

EXAMPLES

Select the product of a and b:

SELECT a * b ...

TRANSFORMS

```
*
```

2. Transforms - +

1.0.0

SUMMARY

Numerical addition.

SYNOPSIS

a + b

a is float|integer

b is float|integer

return type is float|integer

DESCRIPTION

+ performs addition of its two operands. If the operands differ in type then the lower will be promoted to the type of the higher, where the decreasing order of types is float, unsigned integer and signed integer. The operator returns the type of its operands or NULL if at least one of the operands is NULL or is neither an integer nor a float.

EXAMPLES

Select the sum of a and b:

SELECT a + b ...

TRANSFORMS

```
+
```

2. Transforms - -

1.0.0
SUMMARY

Numerical subtraction.

SYNOPSIS

a - b

a is float|integer

b is float|integer

return type is float|integer

DESCRIPTION

- performs subtraction of its two operands. If the operands differ in type then the lower will be promoted to the type of the higher, where the decreasing order of types is float, unsigned integer and signed integer. The operator returns the type of its operands or NULL if at least one of the operands is NULL or is neither an integer nor a float.

EXAMPLES

Select the difference between a and b:

SELECT a - b ...

TRANSFORMS

```
-
```

2. Transforms - <

1.0.0

SUMMARY

Less than test.

SYNOPSIS

a **<** b

a is float|integer|map|list|string|boolean

b is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

< returns true if a and b are both non-NULL and a is less than b. It returns false otherwise.

EXAMPLES

Filter records where port is less than 22:

... WHERE port < 22

TRANSFORMS

```
<
```

2. Transforms - <=

1.0.0

SUMMARY

Less than or equal to test.

SYNOPSIS

a **<=** b

a is float|integer|map|list|string|boolean

b is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

<= returns true if a and b are both non-NULL and a is less than or equal to b or if both arguments are NULL. It returns false otherwise.

EXAMPLES

Filter records where port is less than or equal to 22:

... WHERE port <= 22

TRANSFORMS

```
<=
```

2. Transforms - =

1.0.0
SUMMARY

Equality test.

SYNOPSIS

a **=** b

a is float|integer|map|list|string|boolean

b is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

= returns true if a and b are both non-NULL and a is equal to b or if both arguments are NULL. It returns false otherwise.

EXAMPLES

Filter records where port is equal to 22:

... WHERE port = 22

TRANSFORMS

```
=
```

2. Transforms - >

1.0.0

SUMMARY

Greater than test.

SYNOPSIS

a **>** b

a is float|integer|map|list|string|boolean

b is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

> returns true if a and b are both non-NULL and a is greater than b. It returns false otherwise.

EXAMPLES

Filter records where port is greater than 22:

... WHERE port > 22

TRANSFORMS

```
>
```

2. Transforms - >=

1.0.0

SUMMARY

Greater than or equality test.

SYNOPSIS

a **>=** b

a is float|integer|map|list|string|boolean

b is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

>= returns true if a and b are both non-NULL and a is greater than or equal to b or if both arguments are NULL. It returns false otherwise.

EXAMPLES

Filter records where port is greater than or equal to 22:

... WHERE port >= 22

TRANSFORMS

```
>=
```

2. Transforms - ALL_IN

1.0.0

SUMMARY

subset test.

SYNOPSIS

set_a **all_in** set_b

set_a is list of float|integer|map|list|string|boolean

set_b is list of float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

ALL_IN returns true if all are of the elements in set_a are in set_b. The function returns false if either set_a or set_b is NULL or is not a list.

EXAMPLES

Filter records where all are of the members of addresses are in watchlist:

... WHERE addresses all_in watchlist

TRANSFORMS

```
ALL_IN
```

2. Transforms - AND

1.0.0

SUMMARY

Logical conjunction.

SYNOPSIS

a **and** b

a is boolean

b is boolean

return type is boolean

DESCRIPTION

AND returns true if both a and b are true; otherwise it returns false.

EXAMPLES

Filter records in which both a and b are true:

... WHERE a and b

TRANSFORMS

```
AND
```

2. Transforms - ANY_IN

1.0.0

SUMMARY

set intersection test.

SYNOPSIS

set_a **any_in** set_b

set_a is list of float|integer|map|list|string|boolean

set_b is list of float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

ANY_IN returns true if any are of the elements in set_a are in set_b. The function returns false if either set_a or set_b is NULL or is not a list.

EXAMPLES

Filter records where any are of the members of addresses are in watchlist:

... WHERE addresses any_in watchlist

TRANSFORMS

```
ANY_IN
```

2. Transforms - ARRAY_ALL

1.0.0

SUMMARY

universal criterion acceptance.

SYNOPSIS

array_all(list, condition)

list is list of float|integer|map|list|string|boolean

condition is boolean

return type is boolean

DESCRIPTION

ARRAY_ALL iterates over list, evaluating condition for each element. Where @ appears in condition it is substituted by the current element. If the result of condition is false, then iteration ceases and the function returns false. The function returns true if condition is always true.

The array iterator functions — ARRAY_ANY, ARRAY_ALL and ARRAY_NONE — may be nested, in which case there will be multiple "current" elements. If nesting occurs then each "current" element appears in condition as @n where n is its nesting depth, starting at 1. A single @ always refers to the current level of nesting.

The function returns false if list is NULL or is not a list.

EXAMPLES

Filter records where every element of ports is 22:

... WHERE array_all(ports, @ = 22)

If entries are a list of objects of the form

```
{ "ports" : [ 2, 3, 5, 7, ... ], "max" : 11 }
```

then filter records for which there exists at least one object where at least one member of ports is greater than max:

... WHERE ARRAY_ANY(entries, ARRAY_ANY(@.ports, @ > @1.max))

TRANSFORMS

ARRAY_ALL

2. Transforms - ARRAY_ANY

1.0.0

SUMMARY

partial criterion acceptance.

SYNOPSIS

array_any(list, condition)

list is list of float|integer|map|list|string|boolean

condition is boolean

return type is boolean

DESCRIPTION

ARRAY_ANY iterates over list, evaluating condition for each element. Where @ appears in condition it is substituted by the current element. If the result of condition is true, then iteration ceases and the function returns true. The function returns false if condition is never true.

The array iterator functions — ARRAY_ANY, ARRAY_ALL and ARRAY_NONE — may be nested, in which case there will be multiple "current" elements. If nesting occurs then each "current" element appears in condition as @n where n is its nesting depth, starting at 1. A single @ always refers to the current level of nesting.

The function returns false if list is NULL or is not a list.

EXAMPLES

Filter records where any element of ports is 22:

... WHERE array_any(ports, @ = 22)

If entries are a list of objects of the form

```
{ "ports" : [ 2, 3, 5, 7, ... ], "max" : 11 }
```

then filter records for which there exists at least one object where at least one member of ports is greater than max:

... WHERE ARRAY_ANY(entries, ARRAY_ANY(@.ports, @ > @1.max))

TRANSFORMS

ARRAY_ANY

2. Transforms - ARRAY_NONE

1.0.0

SUMMARY

partial criterion acceptance.

SYNOPSIS

array_none(list, condition)

list is list of float|integer|map|list|string|boolean

condition is boolean

return type is boolean

DESCRIPTION

ARRAY_NONE iterates over list, evaluating condition for each element. Where @ appears in condition it is substituted by the current element. If the result of condition is true, then iteration ceases and the function returns false. The function returns true if condition is never true.

The array iterator functions — ARRAY_ANY, ARRAY_ALL and ARRAY_NONE — may be nested, in which case there will be multiple "current" elements. If nesting occurs then each "current" element appears in condition as @n where n is its nesting depth, starting at 1. A single @ always refers to the current level of nesting.

The function returns false if list is NULL or is not a list.

EXAMPLES

Filter records where no element of ports is 22:

... WHERE array_none(ports, @ = 22)

If entries are a list of objects of the form

```
{ "ports" : [ 2, 3, 5, 7, ... ], "max" : 11 }
```

then filter records for which there exists at least one object where at least one member of ports is greater than max:

... WHERE ARRAY_ANY(entries, ARRAY_ANY(@.ports, @ > @1.max))

TRANSFORMS

```
ARRAY_NONE
```

2. Transforms - AS

1.0.0

SUMMARY

SYNOPSIS

DESCRIPTION

EXAMPLES

TRANSFORMS

```
AS
```

2. Transforms - BETWEEN

1.0.0

SUMMARY

Interval test.

SYNOPSIS

between(a, b, c)

a is float|integer|map|list|string|boolean

b is float|integer|map|list|string|boolean

c is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

Erratum: the synopsis above is incorrect. The correct form is a **BETWEEN** b AND c.

BETWEEN returns true if a is greater than or equal to b and less than c. It returns false otherwise or if any argument is NULL.

EXAMPLES

Filter records in which age is the interval [10,20):

... WHERE age BETWEEN 10 AND 20

TRANSFORMS

```
BETWEEN
```

2. Transforms - CAST

1.0.0

SUMMARY

Change type.

SYNOPSIS

cast(source, destination)

source is integer|string

destination is enumeration of {integer}

return type is integer

cast(source, destination)

source is string

destination is enumeration of {ip}

return type is string

cast(source, destination)

source is integer|string

destination is enumeration of {string}

return type is string

DESCRIPTION

cast transforms source to the destination type. If the first argument is NULL or does not match the expected type, then the function returns NULL. The supported values for destination are:

Destination	Type	Transformation	Notes

integer	integer	A decimal string to a signed, 64-bit integer.	If the resulting value underflow or overflow, then NULL is returned. If the argument is an integer, then no transformation is applied.
ip	string	A IPv4 or IPv6 address, optionally base64-encoded, to a canonical form.	If the input is not a valid address, then NULL is returned.
string	string	A signed, 64-bit integer to a decimal string representation.	If the argument is a string, then no transformation is applied.

EXAMPLES

Filter records where address is equivalent to 1.2.3.4:

... WHERE cast(address, 'ip') = '1.2.3.4'

TRANSFORMS

CAST

2. Transforms - CHAR_LENGTH

1.0.0

SUMMARY

Return string length.

SYNOPSIS

char_length(text)

text is string

return type is integer

DESCRIPTION

CHAR_LENGTH returns the number of characters in text. It returns NULL if text is not a string or is NULL.

EXAMPLES

Filter records where the length of password is less than eight:

... WHERE char_length(password) < 8

TRANSFORMS

```
CHAR_LENGTH
```

2. Transforms - CIDR_MATCH

1.0.0

SUMMARY

Subnet membership test.

SYNOPSIS

cidr_match(address, CIDR)

address is list|string

CIDR is static list of string|string

return type is boolean

DESCRIPTION

CIDR_MATCH returns true if address belongs to the network or list of networks specified by CIDR. CIDR, or each of its elements if a list, is a string in CIDR notation and must be an expression that can be evaluated at query compilation. It cannot, for example, be a field even if the field were to contain a valid CIDR specification.

EXAMPLES

Filter records where address is in the subnet 131.111.96.0/20:

... WHERE cidr_match(address, '131.111.96.0/20')

Filter records where address is in any of the subnets in a tag set value:

... WHERE cidr_match(address, TAGS:LOOKUP('CidrSet', 'watchlist'))

TRANSFORMS

```
CIDR_MATCH
```

2. Transforms - COALESCE

1.0.0

SUMMARY

Return the first non-NULL entry.

SYNOPSIS

coalesce(list)

list is list of float|integer|map|list|string|boolean

return type is float|integer|map|list|string|boolean

DESCRIPTION

COALESCE evaluates each member of list, stopping at, and returning, the first non-NULL result. It returns NULL if list is NULL or if every one of its elements is NULL.

EXAMPLES

Filter records in which the first extant field of "user", "username" and "login" is equal to 'admin':

... WHERE coalesce([user, username, login]) = 'admin'

TRANSFORMS

```
COALESCE
```

2. Transforms - CONCAT

1.0.0

SUMMARY

Concatenate string elements.

SYNOPSIS

concat(list)

list is list of float|integer|map|list|string|boolean

return type is string

DESCRIPTION

CONCAT returns the string formed by concatenating all string elements found in list. The function returns NULL if list is NULL.

EXAMPLES

Select all strings in users as a single field:

SELECT concat(users) ...

TRANSFORMS

```
CONCAT
```

2. Transforms - CONCAT_WS

1.0.0

SUMMARY

Concatenate string elements.

SYNOPSIS

concat_ws(list, delimiter)

list is list of float|integer|map|list|string|boolean

delimiter is static string

return type is string

DESCRIPTION

CONCAT_WS returns the string formed by concatenating all string elements found in list using delimiter. The function returns NULL if list is NULL.

EXAMPLES

Select all strings in users as a single field and delimit using ",":

SELECT concat_ws(users, ',') ...

TRANSFORMS

```
CONCAT_WS
```

2. Transforms - CONTAINS

1.0.0

SUMMARY

Single substring test.

SYNOPSIS

haystack **contains** needle

haystack is list|string

needle is static string

return type is boolean

DESCRIPTION

CONTAINS returns true if haystack contains needle. The function returns NULL if haystack is NULL.

EXAMPLES

Filter records in which message contains the string "denied":

... WHERE message contains 'denied'

TRANSFORMS

```
CONTAINS
```

2. Transforms - CONTAINS_ALL

1.0.0

SUMMARY

Universal substring test.

SYNOPSIS

haystack **contains_all** needles

haystack is list|string

needles is static list of string

return type is boolean

DESCRIPTION

CONTAINS_ALL returns true if haystack contains every element of needles and false otherwise. The function returns NULL if haystack is NULL.

EXAMPLES

Filter records in which message contains every of the strings "denied", "refused" and "failed":

... WHERE message contains_all ['denied', 'refused', 'failed']

TRANSFORMS

```
CONTAINS_ALL
```

2. Transforms - CONTAINS_ANY

1.0.0

SUMMARY

Partial substring test.

SYNOPSIS

haystack **contains_any** needles

haystack is list|string

needles is static list of string

return type is boolean

DESCRIPTION

CONTAINS_ANY returns true if haystack contains any element of needles and false otherwise. The function returns NULL if haystack is NULL.

EXAMPLES

Filter records in which message contains any of the strings "denied", "refused" and "failed":

... WHERE message contains_any ['denied', 'refused', 'failed']

TRANSFORMS

```
CONTAINS_ANY
```

2. Transforms - DECODE

1.0.0

SUMMARY

Decodes encoded binary data.

SYNOPSIS

decode(source, format)

source is string

format is enumeration of {base64}

return type is string

DESCRIPTION

DECODE transforms source based on the specified format. If the first argument is NULL, then the function returns NULL. The supported values for format are: base64.

EXAMPLES

Select the result of decoding the base64-encoded string encoded:

SELECT decode(encoded, 'base64')

TRANSFORMS

```
DECODE
```

2. Transforms - ENDS_WITH

1.0.0

SUMMARY

String suffix test.

SYNOPSIS

ends_with(text, suffix)

text is list|string

suffix is list of string|string

return type is boolean

DESCRIPTION

ENDS_WITH returns true if suffix is a string and text ends with suffix or if suffix is a list and text ends with any of its elements. The function otherwise returns NULL.

EXAMPLES

Filter records in which message ends with 'foo':

... WHERE ends_with(message, 'foo')

TRANSFORMS

```
ENDS_WITH
```

2. Transforms - EXISTS

1.0.0

SUMMARY

Test for presence.

SYNOPSIS

exists(argument)

argument is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

exists returns true if argument is a field and is present, or if argument is an expression and is true. It returns false otherwise.

EXAMPLES

Filter records where field is present:

... WHERE exists(field)

TRANSFORMS

```
EXISTS
```

2. Transforms - FROM_EPOCHTIME

1.0.0

SUMMARY

Format date and time.

SYNOPSIS

from_epochtime(time, format)

time is integer

format is static string

return type is string

DESCRIPTION

FROM_EPOCHTIME produces a string representation of time, the number of seconds since the UNIX epoch, according to the specification in format. Characters in the format string are copied to the output one at a time until a % is encountered. A % indicates that the following letter is a conversion specification and the two characters are substituted by the corresponding translation from the table below. The use of any other character following a % is an error.

If time is NULL or is not an integer, then the function returns NULL.

Specification	Description
B	month name in full (January to December)
F	full date (%Y-%m-%d)
H	hour (00 to 23)
M	minutes (00 to 59)
S	seconds (00 to 59)
T	time in 24 hour format (hh:mm:ss)
W	weekday name in full (Sunday to Saturday)
Y	year as a numeric, 4-digit value
a	abbreviated weekday name (Sun to Sat)
b	abbreviated month name (Jan to Dec)
d	month day of as a numeric value (01 to 31)
h	hour (01 to 12)
j	day of the year (001 to 366)
m	month name as a numeric value (01 to 12)
p	AM or PM

r	time as 12 hour AM/PM (hh:mm:ss AM/PM)
w	weekday where Sunday=0 and Saturday=6
y	year as a numeric, 2-digit value

EXAMPLES

Select the time of day corresponding to time_recv:

SELECT from_epochtime(time_recv, '%H:%M') ...

Filter records where time_recv corresponds to a Wednesday:

... WHERE from_epochtime(time_recv, '%W') = 'wednesday'

TRANSFORMS

```
FROM_EPOCHTIME
```

2. Transforms - GEOIP

1.0.0

SUMMARY

geographical network address translation

SYNOPSIS

geoip(address)

address is string

return type is map

geoip(address, attribute)

address is string

attribute is enumeration of {city_name,country_name,country_iso_code,country_name_rr,country_iso_code_rr,aso}

return type is string

geoip(address, attribute)

address is string

attribute is enumeration of {asn}

return type is integer

DESCRIPTION

In its single-argument form, **GEOIP** transforms the internet address into a map where each key is a geographical attribute from the table below and the corresponding value is the translation of address. If **GEOIP** is given attribute as a second argument, then it returns the specific translation as a scalar value. If address is NULL or if no suitable translation exists, then the function returns NULL.

Attribute	Type	Description
city_name	string	the city name
country_name	string	the country name
country_iso_code	string	the country ISO code
country_name_rr	string	the name of the represented/registered/geographic cou
country_iso_code_rr	string	the ISO code of the represented/registered/geographic c
asn	integer	the Autonomous System Number
aso	string	the Autonomous System Organisation

EXAMPLES

Select all geographical information for address:

SELECT geoip(address)...

Select the city name and country ISO code corresponding to address:

SELECT (geoip(address) AS geoip).city_name, geoip.country_iso_code ...

Filter records where the country corresponding to address is the United States:

... WHERE geoip(address, 'country_iso_code') = 'US'

TRANSFORMS

```
GEOIP
```

2. Transforms - IF

1.0.0

SUMMARY

Conditional evaluation.

SYNOPSIS

if(condition, then, else)

condition is boolean

then is float|integer|map|list|string|boolean

else is float|integer|map|list|string|boolean

return type is float|integer|map|list|string|boolean

DESCRIPTION

if evaluates the Boolean expression condition. /If the result is true it evaluates and returns the result of then, otherwise it evaluates and returns the result of else.

EXAMPLES

Select the field to project as "destination" according to whether the protocol is TCP:

SELECT if(proto = 6, tcp.port, ip_dest) AS destination ...

TRANSFORMS

```
IF
```

2. Transforms - IN

1.0.0

SUMMARY

Set membership test.

SYNOPSIS

element **in** set

element is integer|map|string|boolean

set is list of float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

IN returns true if the scalar element (which may be NULL) is a member of the list set. It returns false otherwise.

EXAMPLES

Filter records where address is in watchlist:

... WHERE address in watchlist

TRANSFORMS

```
IN
```

2. Transforms - INTERVAL

1.0.0

SUMMARY

Round time down to interval.

SYNOPSIS

interval(time, interval)

time is integer

interval is integer

return type is integer

DESCRIPTION

INTERVAL returns the result of rounding down time to the nearest multiple of interval. The function returns NULL if either argument is NULL or if interval is zero.

EXAMPLES

Select the start of the hour corresponding to time_recv:

SELECT interval(time_recv, 3600) ...

TRANSFORMS

```
INTERVAL
```

2. Transforms - ISNULL

1.0.0

SUMMARY

Test for absence.

SYNOPSIS

isnull(argument)

argument is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

isnull returns true if argument is a field and is absent or if argument is an expression and is false. It returns false otherwise.

EXAMPLES

Filter records where field is absent:

... WHERE isnull(field)

TRANSFORMS

```
ISNULL
```

2. Transforms - IS_INTERNAL_ADDRESS

1.0.0

SUMMARY

Internal network membership test.

SYNOPSIS

is_internal_address(address)

address is string

return type is boolean

DESCRIPTION

IS_INTERNAL_ADDRESS returns true if address is an IP address on an internet-internal network and false otherwise. The following network ranges are considered internal:

10.0.0.0/8
100.64.0.0/10
127.0.0.0/8
169.254.0.0/16
172.16.0.0/12
192.168.0.0/16
::1/128
fc00::/7
fe80::/10

EXAMPLES

Filter records where address is on an internal network:

... WHERE is_internal_address(address)

TRANSFORMS

```
IS_INTERNAL_ADDRESS
```

2. Transforms - IS_NOT_NULL

1.0.0

SUMMARY

Test for presence.

SYNOPSIS

argument **is_not_null**

argument is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

is_not_null returns true if argument is a field and is present or if argument is an expression and is true. It returns false otherwise.

EXAMPLES

Filter records where field is present:

... WHERE field is_not_null

TRANSFORMS

```
IS_NOT_NULL
```

2. Transforms - IS_NULL

1.0.0

SUMMARY

Test for absence.

SYNOPSIS

argument **is_null**

argument is float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

is_null returns true if argument is a field and is absent or if argument is an expression and is false. It returns false otherwise.

EXAMPLES

Filter records where field is absent:

... WHERE field is_null

TRANSFORMS

```
IS_NULL
```

2. Transforms - LENGTH

1.0.0

SUMMARY

List length.

SYNOPSIS

length(list)

list is list of float|integer|map|list|string|boolean

return type is integer

DESCRIPTION

LENGTH returns the number of elements in list. It returns NULL if list is NULL or is not a list.

EXAMPLES

Select the length of the list addresses:

SELECT length(addresses) ...

TRANSFORMS

```
LENGTH
```

2. Transforms - LIKE

1.0.0

SUMMARY

Simple pattern matching.

SYNOPSIS

string **like** pattern

string is list|string

pattern is static string

return type is boolean

DESCRIPTION

LIKE returns true if string is non-NULL and matches the case-insensitive pattern. It returns false otherwise. Pattern matching characters are:

Character	Description

%	Zero or more characters.
_	A single character.

These special characters must otherwise be escaped with "\".

EXAMPLES

Filter records where program contains the string "foo":

... WHERE program like '%FOO%'

TRANSFORMS

```
LIKE
```

2. Transforms - LOWER

1.0.0

SUMMARY

Convert string to lowercase.

SYNOPSIS

lower(text)

text is string

return type is string

DESCRIPTION

lower converts upper-case characters in text to their lower-case equivalents and returns the result.

EXAMPLES

Select the lower-case equivalent of name:

SELECT lower(name)

TRANSFORMS

```
LOWER
```

2. Transforms - NONE_IN

1.0.0

SUMMARY

disjoint set test.

SYNOPSIS

set_a **none_in** set_b

set_a is list of float|integer|map|list|string|boolean

set_b is list of float|integer|map|list|string|boolean

return type is boolean

DESCRIPTION

NONE_IN returns true if none is of the elements in set_a are in set_b. The function returns false if either set_a or set_b is NULL or is not a list.

EXAMPLES

Filter records where none is of the members of addresses are in watchlist:

... WHERE addresses none_in watchlist

TRANSFORMS

```
NONE_IN
```

2. Transforms - NOT

1.0.0

SUMMARY

Logical negation.

SYNOPSIS

not a

a is boolean

return type is boolean

DESCRIPTION

NOT returns true if a is non-NULL and false, otherwise it returns false.

EXAMPLES

Filter records in which a is non-NULL and false:

... WHERE not a

TRANSFORMS

```
NOT
```

2. Transforms - NOW

1.0.0

SUMMARY

The current time.

SYNOPSIS

now()

return type is integer

DESCRIPTION

NOW returns the current time, expressed as the integer number of seconds since the start of the UNIX epoch.

EXAMPLES

Filter records where time_recv is later than one day ago:

... WHERE time_recv > now() - 86400

TRANSFORMS

```
NOW
```

2. Transforms - OR

1.0.0

SUMMARY

Logical disjunction.

SYNOPSIS

a **or** b

a is boolean

b is boolean

return type is boolean

DESCRIPTION

OR returns true if at least one of a and b is true, otherwise it returns false.

EXAMPLES

Filter records in which at least one of a and b is true:

... WHERE a or b

TRANSFORMS

```
OR
```

2. Transforms - REGEXP_EXTRACT

1.0.0

SUMMARY

Regular expression extraction.

SYNOPSIS

regexp_extract(string, regex)

string is string

regex is static string

return type is map

DESCRIPTION

REGEXP_EXTRACT returns map if string is non-NULL and matches regex, otherwise it returns NULL.

Returned map contains results only for first matched substring. It has one pair per each named capture group, where key is a name of the group, and value is an extracted group or NULL if it is not present.

regex must have at least one named capture group. Names may be up to 32 symbols long and they may contain only ASCII alphanumeric characters and underscores but must start with a non-digit. Names must be unique within regex.

See the documentation for the underlying engine for the specification of regex.

EXAMPLES

Select extracted fields from message:

SELECT regexp_extract(message, 'Login from: (?<ip>([0-9]{1,3}\.){3}[0-9]{1,3})').ip as ip ...

Select extracted user, ip from message and aggregate by them:

SELECT (regexp_extract(message, 'User (?<user>[A-Za-z0-9]+) has logged in from (?<ip>([0-9]{1,3}\.){3}[0-9]{1,3})') AS res).user AS user, res.ip as ip, Count(*) WHERE res IS NOT NULL) GROUP BY user, ip

TRANSFORMS

```
REGEXP_EXTRACT
```

2. Transforms - REGEXP_MATCH

1.0.0

SUMMARY

Regular expression matching.

SYNOPSIS

string **regexp_match** regex

string is list|string

regex is static string

return type is boolean

DESCRIPTION

REGEXP_MATCH returns true if string is non-NULL and matches regex. See the documentation for the underlying engine for the specification of regex.

EXAMPLES

Filter records where program contains either of the strings "foo" and "bar":

... WHERE program regexp_match 'foo|bar'

TRANSFORMS

```
REGEXP_MATCH
```

2. Transforms - SPLIT

1.0.0

SUMMARY

Split a string.

SYNOPSIS

split(text, delimiter)

text is string

delimiter is static string

return type is list of string

DESCRIPTION

SPLIT returns the list of strings created by separating text using one of the characters in delimiter. The function returns NULL if text is NULL.

EXAMPLES

Select the list of colon-delimited fields in passwd:

SELECT split(passwd, ':') ...

TRANSFORMS

```
SPLIT
```

2. Transforms - STARTS_WITH

1.0.0

SUMMARY

String prefix test.

SYNOPSIS

starts_with(text, prefix)

text is list|string

prefix is list of string|string

return type is boolean

DESCRIPTION

STARTS_WITH returns true if prefix is a string and text starts with prefix or if prefix is a list and text starts with any of its elements. The function otherwise returns NULL.

EXAMPLES

Filter records in which message starts with 'foo':

... WHERE starts_with(message, 'foo')

TRANSFORMS

```
STARTS_WITH
```

2. Transforms - TAGS:EXISTS

1.0.0

SUMMARY

Tag set key test.

SYNOPSIS

tags:exists(tagset, key)

tagset is static string

key is integer|string

return type is boolean

DESCRIPTION

TAGS:EXISTS returns true if tagset contains key, otherwise, it returns false.

EXAMPLES

Filter records in which the tag set 'Users' contains the key 'watchlist':

... WHERE tags:exists('Users', 'watchlist')

TRANSFORMS

```
TAGS:EXISTS
```

2. Transforms - TAGS:LOOKUP

1.0.0

SUMMARY

Return tag set value.

SYNOPSIS

tags:lookup(tagset, key)

tagset is static string

key is integer|string

return type is float|integer|map|list|string|boolean

tags:lookup(tagset, key, default)

tagset is static string

key is integer|string

default is float|integer|map|list|string|boolean

return type is float|integer|map|list|string|boolean

DESCRIPTION

TAGS:LOOKUP returns the value associated with key in tagset. If key is NULL or not present then the function will return default, if provided, or NULL if not.

EXAMPLES

Filter records in which address is in the list associated with the key 'allowed' in the tag set 'watchlists':

... WHERE address IN tags:lookup('watchlists', 'allowed')

TRANSFORMS

```
TAGS : LOOKUP
```

2. Transforms - TO_EPOCHTIME

1.0.0

SUMMARY

Convert date/time string to UNIX time.

SYNOPSIS

to_epochtime(date/time, format)

date/time is string

format is enumeration of {ISO8601}

return type is integer

DESCRIPTION

TO_EPOCHTIME transforms a formatted string containing a date and time into the corresponding number of seconds since the UNIX epoch. The supported values for format are ISO8601.

EXAMPLES

Interpret _date_ as an ISO8601 string and select the equivalent Unix time:

SELECT to_epochtime(_date_, 'ISO8601') ...

TRANSFORMS

```
TO_EPOCHTIME
```

2. Transforms - TUNE:THRESHOLD

1.0.0

SUMMARY

Return tag set value.

SYNOPSIS

tune:threshold(tagset, key)

tagset is static string

key is static string

return type is integer|list of integer|string|boolean|string|boolean

DESCRIPTION

TUNE:THRESHOLD returns the value associated with key in tagset. It returns NULL if key is NULL.

EXAMPLES

Filter records in which score is greater than the value associated with 'cvss' in the tag set 'tuneables':

... WHERE score > tune:threshold('tuneables', 'cvss')

TRANSFORMS

```
TUNE:THRESHOLD
```

2. Transforms - URI_PARSE

1.0.0

SUMMARY

Parse a URI.

SYNOPSIS

uri_parse(uri)

uri is string

return type is map

DESCRIPTION

URI_PARSE parses uri and returns a map whose values are the fields within it; if uri is NULL or cannot be parsed successfully then the function returns NULL. The map's keys are the following:

Key	Value type	Value notes
scheme	string	NULL if uri is relative, i.e. if it starts with a path.
authority	string	The host and, optionally, the user information and port. NULL if absent.
path	string	May be an empty (zero-length) string.
query	string	NULL if absent.
query_kvs	map	A map whose keys and values represent the unescaped parameters in the query string. NULL if absent.
fragment	string	NULL if absent.

EXAMPLES

Select URI path from uri :

SELECT uri_parse(uri).path ...

Filter records where the query string in uri contains the key/value pairs "action=login" and "type=user":

... WHERE (uri_parse(uri).query_kvs AS qs).action = 'login' AND qs.type = 'user'

TRANSFORMS

```
URI_PARSE
```

2. Transforms - WINDOW

1.0.0

SUMMARY

Sliding window time interval.

SYNOPSIS

window(time, window)

time is integer

window is integer

return type is integer

DESCRIPTION

In search, **WINDOW** behaves exactly like **INTERVAL**: it returns the result of rounding down time to the nearest multiple of window. In Log Correlation and Streaming analytics, window is a sliding interval. The function returns NULL if either argument is NULL or if window is zero.

EXAMPLES

Select the start of the hour corresponding to time_recv:

SELECT window(time_recv, 3600) ...

TRANSFORMS

```
WINDOW
```

3. Aggregators

3. Aggregators - AVG

1.0.0

SUMMARY

Returns the average of a numeric value for a group.

SYNOPSIS

DESCRIPTION

EXAMPLES

AGGREGATORS

```
AVG
```

3. Aggregators - COUNT

1.0.0

SUMMARY

Returns number of non-null elements in a group.

SYNOPSIS

DESCRIPTION

EXAMPLES

AGGREGATORS

```
COUNT
```

3. Aggregators - LSET

1.0.0

SUMMARY

Returns set of sorted unique non-null elements for each aggregated group limited up to specified value.

SYNOPSIS

DESCRIPTION

EXAMPLES

AGGREGATORS

```
LSET
```

3. Aggregators - LUCOUNT

1.0.0

SUMMARY

Returns number of unique non-null elements in a group limited by specified value.

SYNOPSIS

DESCRIPTION

EXAMPLES

AGGREGATORS

```
LUCOUNT
```

3. Aggregators - MAX

1.0.0

SUMMARY

Returns the maximum of a numeric value for a group.

SYNOPSIS

DESCRIPTION

EXAMPLES

AGGREGATORS

```
MAX
```

3. Aggregators - MIN

1.0.0

SUMMARY

Returns the minimum of a numeric value for a group.

SYNOPSIS

DESCRIPTION

EXAMPLES

AGGREGATORS

```
MIN
```

3. Aggregators - SET

1.0.0

SUMMARY

Returns set of unique non-null elements for each aggregated group.

SYNOPSIS

DESCRIPTION

EXAMPLES

AGGREGATORS

```
SET
```

3. Aggregators - SUM

1.0.0

SUMMARY

Returns the sum of a numeric value for a group.

SYNOPSIS

DESCRIPTION

EXAMPLES

AGGREGATORS

```
SUM
```

3. Aggregators - UCOUNT

1.0.0

SUMMARY

Returns number of non-null elements in a group.

SYNOPSIS

DESCRIPTION

EXAMPLES

AGGREGATORS

```
UCOUNT
```