**Shortcuts**

**<enter> //search  
<ctrl>+<space> //suggestions  
<shift>+<enter> //new Line**

**Examples**

***source* | *select* | *where* | *limit* | *order by* | *group* | *'|'external functions***

**source in ('*13151*', '*13158*')  
source in ('*Microsoft Security*', '*Palo Alto*')  
source like '*Microsoft Secu%*'  
source = '*Microsoft Security*'  
sourcetype = '*Security*'  
sourcetag = '*SourceTag*'  
select \* //returns all columns and rows  
select *col1*, *col2* //returns the selected column values  
where '*palo alto*' //performs full-text search  
where *col1* > *20* or (*col2* = '*palo*' and *col3* != '*host*') //multiple conditional statement  
where *col1* between *123* and *456*  
select *col1*, *col2* group  
select *col1*, *col2*, count(\*) group having count(\*) > *20*  
select *col1*, *col2* order by *col1* asc, *col2* desc**

**source**

Used for source name filtering

**source = [*source name or source id*]  
source in ([*source name or source id, ...*])  
sourcetype //filters by source definition code  
sourcetag //filters by source tag  
sourcecategory //filters by source category  
lookup = "\*" //returns lookup data  
geolocation = "\*" //returns geographic data  
alert = "\*" //returns created alerts data**

**Examples**

**source = *'Microsoft Security'*  
source in (*'logsource1'*, *'logsource2'*)  
lookup = "*lookup1*"**

**select**

The column names to be retrieved are specified with the 'select' command

**select [\*|*columns...*|*aggregation functions...*|*as*]**

Examples

**select \* //returns all columns and rows  
select *col1*, *col2* //returns the selected column values  
select *col1*, *col2* as '*newcol2*' //changes col2 name as newcol2**

**where**

Used to filter records

| **Operator** | **Description** | **Usage** |
| --- | --- | --- |
| = | Equal | where *col1* = '*foo*' or *col1* = *123* |
| != | Not Equal | where *col1* != '*foo*' or *col1* != *123* |
| > | Greater Than | where *col1* > *123* |
| < | Less Than | where *col1* < *123* |
| >= | Greater Than or Equal | where *col1* >= *123* |
| <= | Less Than or Equal | where *col1* <= *123* |
| between | Between two values | where *[column]* beetween *[value1]* and *[value2]* |
| in | If it is in the list | where *[column]* int (*[value1]*, *[value2]*, ...) |
| notin | If it is not in the list | where *[column]* notin (*[value1]*, *[value2]*, ...) |
| like | Like operator | where *[column]* like *'[value]%'* **//%value, value%, val%ue can be used.** |
| notlike | Not like operator | where *[column]* notlike *'[value]%'* |

Examples

**where '*palo alto*' //performs full-text search  
where *col1* > *20* or (*col2* = '*palo*' and *col3* != '*host*') //multiple conditional statement  
where *col1* in ('*foo*', '*bar*') //col1 with values foo and bar  
where *col1* notin (*'foo'*, *'bar'*)  
where *col1* between *123* and *456*  
where *col1* like *'host%'* //starting with 'host'**

**group**

Groups the result set by one or more columns

If used with aggregate functions, 'group' keyword can be omitted. 'having' can be used to filter the result after aggregation functions

**select [*group columns...*, *aggregation functions...*] group [*group columns*] having [*group conditional expression*] top [*number*]**

Aggregation Functions

| **Name** | **Description** | **Usage** |
| --- | --- | --- |
| count | Calculates row count | select count([*column*|\*]) |
| sum | Calculates the sum of the numeric values in the column | select sum(*column*) |
| avg | Calculates the average of the numeric values in the column | select avg(*column*) |
| min | Get the lowest value for the selected column | select min(*column*) |
| max | Get the highest value for the selected column | select max(*column*) |

Examples

**select count(\*)  
select count(\*), max(*col1*), avg(*col2*), sum(*col3*), min(*col4*)  
select *col1*, count(\*)  
select *col1*, *col2*, count(\*) group *col1*, *col2*  
select *col1*, *col2*, count(\*) group having count(\*) > *20*  
select *col1*, *col2*, count(\*), sum(*col1*) group having count(\*) > *20* and sum(*col1*) < *4000* top *10*  
select *col1*, *col2*, count(\*), sum(*col1*) having count(\*) > *20* and sum(*col1*) < *4000***

**order by**

Used to sort columns in ascending or descending order

**select [*columns...*] order by [*columns...*] [asc|desc]**

Examples

**select *col1*, *col2* order by *col1* desc  
select *col1*, *col2* order by *col1* asc, *col2* desc  
select *col1*, *col2* order by *col1* asc, *col2* desc  
select *col1*, *count(\*)* order by *count(\*)* desc //use on a aggregation function  
select *col1*, *ts*, *count(\*)* order by *ts* desc**

**limit**

Limits the result set

**limit [*number*]**

Examples

**select \* limit *10***

**External Functions**

fter the pipe (|) character, external functions loaded into the system can be used. These functions operates on the records received as a result of the query, so the result of the previous operation restricts the data set. Than passes the result to the next function, if any.

**\* | [external functions] [parameters] | ...**

**uniq**

Groups the result and add 'count' column

**uniq *col1, col2, ...***

**uniq *col1*, *col2***

Returns col1, col2, count columns

**uniq**

Returns all columns and the 'count' column

**head**

Returns the first N rows of the result

**head *number***

**head *5***

Returns the first 5 row of the result

**eval**

Runs the expression given in the selected column. If the 'column' column exists, it is overwritten, otherwise a new column is created. Arithmetic operations + - \* % or column expressions can be combined

**eval *column* = *expression* | eval *func*(*args*) | ...**

**eval *resultcol* = *col1* + *col2* - *5***

Evaluates 'col1+col2-5' and write the result to 'resultcol'

**eval *resultcol* = toint(*col1*) + toint(*col2*)**

Evaluates 'col1+col2' as integer and write the result to 'resultcol'

**eval *resultcol* = toupper(*col1*)**

Converts the 'col1' column to uppercase and write the result to 'resultcol'

**eval *resultcol* = tolower(*col1*)**

Converts the 'col1' column to lowercase and write the result to 'resultcol'

**eval *resultcol* = replace(*col1, "value1", "value2")***

Replaces 'value1' with 'value2' in column 'col1' and write the result to 'resultcol'

**eval *resultcol* = split(*col1*, *"\*"*, *1*)**

It parses column 'col1' with '\*' and takes the given part and write the result to 'resultcol'

**eval *resultcol* = replacemulti(*col1*, *"aaa=one,bbb=two,ccc=three,\*=other"*)**

Replaces one value with other.'\*' replaces all other values with the given value if used. And write the result to 'resultcol'

**eval *resultcol* = like(*col1*, *"%value1%"*, *"true"*, *"false"*)**

If 'col1' contains 'value1' returns 'true' otherwise 'false' and write the result to 'resultcol'

**eval *resultcol* = md5(*col1*)**

Computes md5 hash of 'col1' and write the result to 'resultcol'

**eval *resultcol* = hash(*col1*, *"sha256"*)**

Computes hash of 'col1' and write the result to 'resultcol'

**eval *resultcol* = list\_count(*col1*)**

Count list of 'col1' and write the result to 'resultcol'

**eval *resultcol* = list\_filter(*col1*, *"%value%"*)**

If list of 'col1' contains 'value' write the result to 'resultcol'

**eval *resultcol* = list\_join(*col1*, *"value"*)**

list of 'col1' joins 'value' write the result to 'resultcol'

**eval tail(*5*)**

Returns last 5 row of the data

**eval removecolumn(*col1*)**

Removes the 'col1' column from the data

**eval fields(*col1*, *col2*, ...)**

Returns the wanted columns

**eval list(*col1*, *col2*)**

Groups 'col2' by 'col1'

**regex**

Matches the given regular expression. If the column value matches, than it will be taken otherwise not

**regex *column*, *regular expression***

**regex *eventdate*, *"\S+\s+\S+"***

If the value of the given column matches the regular expression (Regex), that row is retrieved, otherwise not

**top**

Gets the top N value for the given column

**top *column* limit=*number***

**top *col1* limit=*10***

Gets highest 10 values for column 'col1'

**rename**

Renames columns

**rename *col1* as *newcol1*, ...**

**rename *col1* as *threat\_name***

Renames 'col1' column as 'threat\_name'

**linecount**

Return one column and one value that contains the row count

**linecount**

**sum**

Returns the sum of each column

**sum *column1*, *column2*, ...**

**sum *col1***

Returns the sum of 'col1'

**avg**

Returns the average of each column

**avg *column1*, *column2*, ...**

**avg *col1***

Returns the average of 'col1'

**min**

Returns the minimum of each column

**min *column1*, *column2*, ...**

**min *col1***

Returns the minimum of 'col1'

**max**

Returns the maximum of each column

**max *column1*, *column2*, ...**

**max *col1***

Returns the maximum of 'col1'

**count**

Returns the count of each column (each column gives same result)

**count *column1*, *column2*, ...**

**count *col1***

Returns the count of 'col1'

**std**

Returns the standard deviation of each column

**std *column1*, *column2*, ...**

**std *col1***

Returns the standard deviation of 'col1'