kql Cheat Sheet

by markwoan via cheatography.com/93412/cs/20629/

Search

`Test | search "Fred"`	Searches all columns in the table "Test" for the value "Fred"
`Test | search "fred"`	Same as above, since **search** is not by default case sensitive
`Test | search kind=case_sensitive "fred"`	Searches all columns in the table "Test" for the value "Fred", now requiring a match on the case
`search "fred"`	Searches across all tables for the value "Fred"
`search in (Process, Autoruns) "Fred"`	Searches across the tables "Proc", "Autoruns" for the value "Fred"
`Processes | search ProcName=="explorer.exe`	Searches the "Processes" table on the column named "ProcName" for a value of "explorer.exe"
`Processes | search ProcName:"svchost`	Searches the "Processes" table on the column named "ProcName" for a value containing "svchost"
`Processes | search "svchost.exe"`	Searches the "Processes" table for a value containing exactly "svchost.exe"
`Processes | search "net*"`	Searches the "Processes" table for a value that contains "net"
`Processes | search * startswith "net"`	Searches the "Processes" table for a value that starts with "net"
`Processes | search * endswith "net"`	Searches the "Processes" table for a value that ends with "net"
`Processes | search "Powershell.exe" and " -encodedCommand"`	Searches the "Processes" table for both "Powershell.exe" and "-encodedCommand"
`Processes | search * matches regex "[A-Z]:\\\\Program\\sFiles"`	Searches the "Processes" table for values that match the regex

Search operator provides a multi-table/multi-column search experience

Where

`Processes | where ProcName =="explorer.exe"`	Limits search to the "ProcName" column and a specific value
`Processes | where ProcName =="explorer.exe" and ParentProcName=="Word.exe"`	Limits search to the "ProcName" and "ParentProcName" columns and specific values for each
`Processes | where ProcName =="explorer.exe" and ParentProcName=="Word.exe" and Host=="DESKTOP1"`	Additional "and" operators
`Processes | where ProcName =="explorer.exe" and (Host=="DESKTOP1" or Host=="SERVER1"`	"or" operator logic

Where (cont)

`Processes | where ProcName =="explorer.exe" | ParentProcName=="Word.exe"`	"where" operators stacked, so that each data set is reduced. Used when performing additional operations between each "where"
`Processes | where * hasprefix "svchost"`	Has "svchost" at the start of a column value
`Processes | where * hassuffix ".exe"`	Has ".exe" at the end of a column value
`Processes | where * contains "svchost"`	Has "svchost" some where in a column value
`Processes | where CommandLine matches regex "[A-Z]:\\\\Program\\sFiles"`	Can use regex for the matching logic

Filters a table to the subset of rows that satisfy a predicate.

Take

`Processes | take 5`	Retrieves 5 rows at random from the "Processes" table
`Processes | where ProcName=="Powershell.exe" and Host=="DESKTOP1" | take 5`	Combines "where" and "and" operators to retrieve 5 rows at random from the "Processes" table
`Processes | limit 5`	The "limit" operator has the same effect as "take"

Return up to the specified number of rows

Count

`Proc | count`	Returns the count of rows within the "Procs" table
`Proc | where ProcName=="explorer.exe" | count`	Returns the count of rows within the "Procs" table, limited by the "where" operator

Returns the number of records in the input record set

Summarize

`Procs | summarize count() by ProcName`	Summarize **Processes** table (like SQL group by) the row counts, by **ProcName**
`Procs | summarize count() by ProcName, Host`	Summarize **Processes** table (like SQL group by) the row counts, by **ProcName** and **Host**
`Procs | summarize ProcCount=count() by ProcName, Host`	Summarize **Processes** table (like SQL group by) the row counts (as **ProcCount**), by **ProcName** and **Host**
`Procs | summarize Num=count(), AvgTime=avg(ProcDuration) by ProcName`	Summarize **Processes** table (like SQL group by) the row counts (as **Num**), by **ProcName** and **Host**

Summarize (cont)

`Procs | summarize Num=count(), by ProcName, bin(TimeGenerated, 1d), Host`	Summarize **Processes** table (like SQL group by) the row counts (as **Num**), by each day (using **bin** function which separates into smaller values e.g. days, hours etc),**ProcName** and **Host**

Summarize operator produces a table that aggregates the content of the input table

Extend

`Procs | extend FileSizeKb = FileSizeMB/1000`	Adds new **FileSizeKb** column by dividing existing FileSizeMb column value
`Procs | extend FileSizeKb = FileSizeMB/1000, FileSizeB = FileSizeMB/1000000`	Adds new **FileSizeKb**, **FileSizeB** columns by dividing existing FileSizeMb column value
`Procs | extend FullPath = strcat(FilePath, "\", FileName)`	Adds new **FullPath** column by concatenating strings from two columns (strcat)

Create calculated columns and append them to the result set

Project

`Procs | project PID, ProcName, Host`	Allows reduced column selection (PID, ProcName, Host)
`Procs | extend FileSizeKb = FileSizeMB/1000 | project ProcName, FileSizeKb`	Used **extend** function to add a new column (**FileSizeKb**) using a field not required (FileSizeMb) in output
`Procs | project FileSizeKb = FileSizeMB/1000, ProcName, FileSizeKb`	Used **project** to add a new column using a field not required in output, without using **extend**
`Procs | project-away PID, ParentPID`	Show all columns apart from **PID** and **ParentPID** using the **project-away** function
`Procs | project-rename Computer=Host`	Rename **Host** column to **Computer** and display the rest of the columns

Select (project) the columns to include, rename or drop, and insert new computed columns

Select (project-away) what columns in the input to exclude from the output

Renames (project-rename) columns in the result output

Distinct

`Procs | distinct ProcName`	Returns a uniqued list of **ProcName** values
`Procs | where ParentProcName=="Explorer.exe" | distinct ProcName`	Using **distinct** function to limit the results returned

Produces a table with the distinct combination of the provided columns of the input table

Top

`Procs | top 100 by ProcDuration`	Top returns N rows from the data set, using the **by** clause to sort
`Procs | top 100 by ProcDuration asc`	Top returns N rows from the data set, using the **by** clause to sort, and the **asc** clause to sort in ascending values

Returns the first N records sorted by the specified columns

Ago

`print ago(1s)`	Prints a timestamp in the past e.g. 1 second. Can use **d** = days, **h** = hours, **m** = minutes, **s** = seconds, **ms** = milliseconds, **microsecond** as is, and **tick** = nanosecond
`print ago(2m)`	Prints a timestamp in the past e.g. 2 minutes
`print ago(3h)`	Prints a timestamp in the past e.g. 3 hours
`print ago(4d)`	Prints a timestamp in the past e.g. 4 days
`print ago(-3d)`	Print a timestamp in the future e.g. today + 3 days
`print ago(-12h)`	Print a timestamp in the future e.g. today + 12 hours

Subtracts the given timespan from the current UTC clock time

Print

`print "We love KQL"`	Prints **We love KQL** as the result set output
`print 10+5`	Prints **15** as the result set output
`print 10\5`	Prints **2** as the result set output
`print Calc=5+15`	Prints **20** as the result set output and names the column as **Calc**

Outputs single-row with one or more scalar expressions

Sort/Order

`Procs | project ProcName, PID sort by TimeStarted`	Sorts the data set by the column **TimeStarted**. Defaults to **desc**
`Procs | project ProcName, PID sort by TimeStarted asc`	Sorts the data set by the column **TimeStarted** in ascending order
`Procs | project ProcName, PID order by TimeStarted`	Orders the data set by the column **TimeStarted** in ascending order. Same functionality as **sort**

Sort the rows of the input table into order by one or more columns

Extract

`W3CIISLog | extend Domain = extract("http://(.*)/", 1, FullUrl)`	Creates a new column (Domain), and uses a regex group to extract just the domain from a full URL. Note that the second parameter (1 in this instance), is used to specify which regex group is returned. A value of 0 will return the entire value

Get a match for a regular expression from a text string

Parse

`SecurityEvent | parse Fqbn with "O=" user ", L=" location "," | project user, location`	Parses the **Fqbn** column into two new columns (User, Location) from column string **O=MICROSOFT CORPORATION, L=REDMOND, S=WASHINGTON, C=US\MICROSOFT ® WINDOWS SCRIPT HOST\CSCRIPT.EXE\5.812.10240.16384**

Evaluates a string expression and parses its value into one or more calculated columns.

Date/Time Calculations

`SecurityEvent | extend TimePast = (now() - TimeGenerated)`	Adds a new column (TimePast) with the duration of time since the event occurred
`Process | extend Duration= (EndTime - StartTime) | project PID, FullPath, StartTime , EndTime, Duration`	Adds new column (Duration), that calculates the duration between two timestamps (EndTime, StartTime)

Startof

`print startofday(now())`	Prints the start of day for today
`print startofday(now(), 1)`	Prints the start of day for tomorrow
`print startofday(now(), -1)`	Prints the start of day for yesterday
`print startofweek(now())`	Prints the start of the current week
`print startofweek(now(), 1)`	Prints the start of week for the next week
`print startofweek(now(), -1)`	Prints the start of the week for last week
`print startofmonth(now())`	Prints the start of the current month
`print startofmonth(now(), 1)`	Prints the start of the next month
`print startofmonth(now(), -1)`	Prints the start of the previous month
`print startofyear(now())`	Prints the start of the current year
`print startofyear(now(), 1)`	Prints the start of the next year
`print startofyear(now(), -1)`	Prints the start of the previous year

Returns the start of the day, week, month, year containing the date, shifted by an offset, if provided.

Endof

`print endofday(now())`	Prints the end of day for today
`print endofday(now(), 1)`	Prints the end of day for tomorrow
`print endofday(now(), -1)`	Prints the end of day for yesterday
`print endofweek(now())`	Prints the end of the current week
`print endofweek(now(), 1)`	Prints the end of week for the next week
`print endofweek(now(), -1)`	Prints the end of the week for last week
`print endofmonth(now())`	Prints the end of the current month
`print endofmonth(now(), 1)`	Prints the end of the next month
`print endofmonth(now(), -1)`	Prints the end of the previous month
`print endofyear(now())`	Prints the end of the current year
`print endofyear(now(), 1)`	Prints the end of the next year
`print endofyear(now(), -1)`	Prints the end of the previous year

Returns the end of the day, week, month, year containing the date, shifted by an offset, if provided.

Between

`Process | where PID between (1 .. 1000)`	Returns the processes that have a PID between 1 and 1000
`Procs | where TimeStarted between (datetime("2019-10-01 00:00:00") .. datetime("2019-10-01 12:00:00"))`	Returns the processes that started between the two timestamps
`Procs | where PID !between (1 .. 1000)`	Returns the processes that are **not** between 1 and 1000

Matches the input that is inside the inclusive range

Format DateTime

`format_datetime(datetime(2017-01-29 09:00:05), 'yy-MM-dd [HH:mm:ss]'), 'yy-MM-dd [HH:mm:ss]')`	Returns timestamp as **17-01-29 [09:00:05]**
`format_datetime(datetime(2017-01-29 09:00:05), , 'yyyy-M-dd [H:mm:ss]')`	Returns timestamp as **2017-1-29 [9:00:05]**
`format_datetime(datetime(2017-01-29 09:00:05), 'yy-MM-dd [hh:mm:ss tt]')`	Returns timestamp as **17-01-29 [09:00:05 AM]**

Formats a datetime parameter based on the format pattern parameter