

Demystifying KQL



Session Learning Objectives

At the end of this session, you will have...

- A foundational understanding of Log Analytics and the Kusto Query Language
- An appreciation of the purpose of the most common KQL operations and functions
- The ability to construct security-related queries using these common operators and functions

Agenda



Section 1

- What is KQL?

Section 2

- How do we use it?

Section 3

- Kusto Queries 101

Section 4

- Pop Quiz

Section 5

- Hands On Labs

What is KQL?

What is KQL and What does it stand for?

Kusto Query Language
Built for Performance
Read-only

Importance:

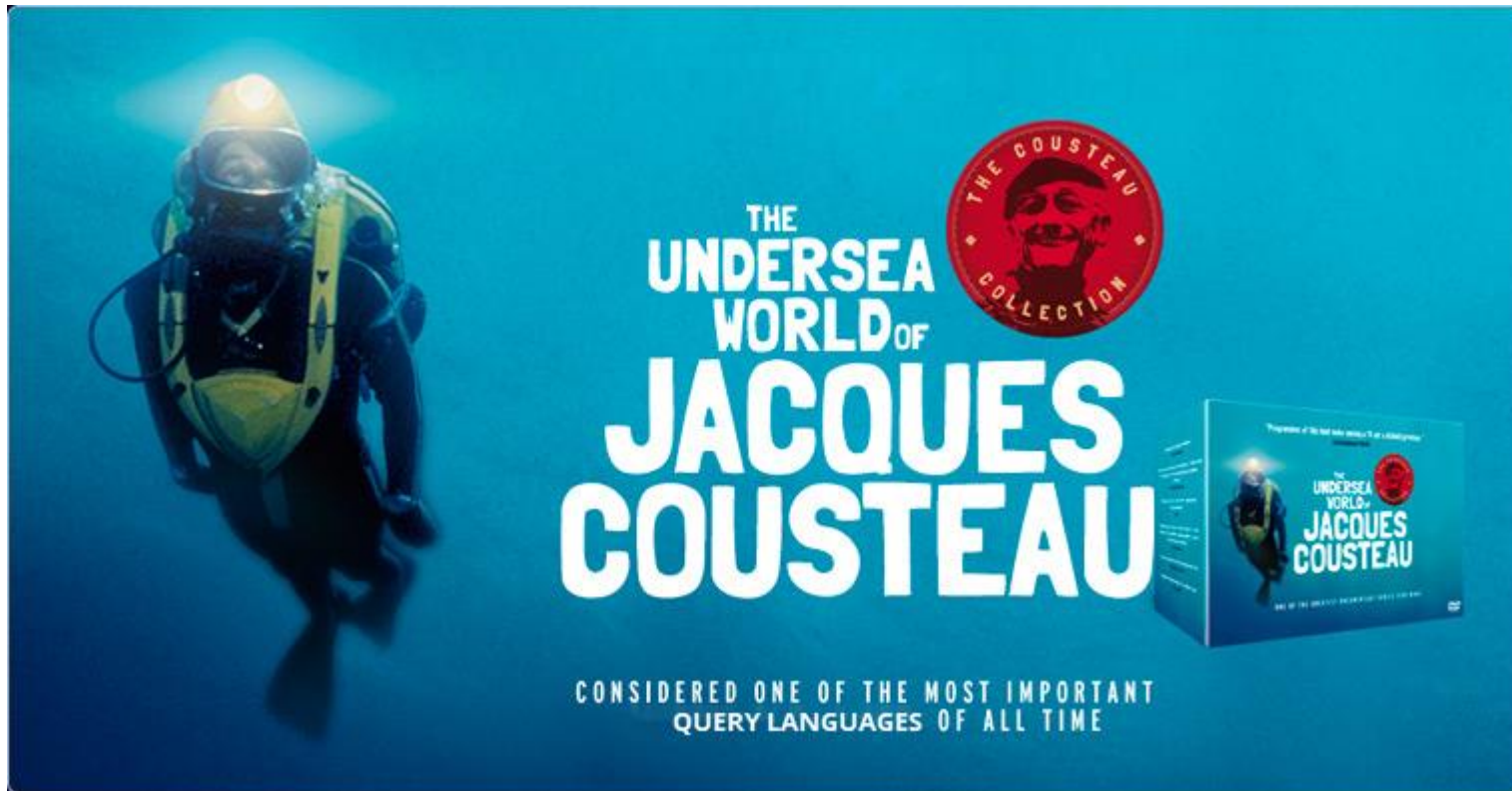
Log Analytics
Azure Monitor
Azure Security services
Across all of Azure

It's the language used to query the Azure log databases: Azure Monitor Logs, Azure Monitor Application Insights and others.



What's Kusto?

Kusto was the original codename for the Azure Application Insights platform that Azure Monitor is now based on and... is named after the famous explorer.



Jacques Cousteau



Actual references to **Jacques** in
the Kusto documentation

Example

Kusto

Copy

```
datatable (Date:datetime, Event:string)
[datetime(1910-06-11), "Born",
 datetime(1930-01-01), "Enters Ecole Navale",
 datetime(1953-01-01), "Published first book",
 datetime(1997-06-25), "Died"]
| where strlen(Event) > 4
```

<https://docs.microsoft.com/en-us/azure/kusto/query/datatableoperator?pivots=azuremonitor>

KQL is related to SQL

SQL to Log Analytics query (KQL) - Structured

Description	SQL Query	
Select all data from a table	SELECT * FROM dependencies	
Select specific columns from a table	SELECT name, resultCode FROM dependencies	
Select 100 records from a table	SELECT TOP 100 * FROM dependencies	
Null evaluation		
String comparison: equality		
String comparison: substring		
String comparison: wildcard		
Date comparison: last 1 day		
Date comparison: date range		dependencies where timestamp between (datetime(2016-10-01) .. datetime(2016-10-01))
Boolean comparison	SELECT * FROM dependencies where success == "False"	dependencies where success == "False"
Sort	SELECT * FROM dependencies order by timestamp asc	dependencies order by timestamp asc
Distinct	SELECT DISTINCT * FROM dependencies	dependencies summarize by name, type
Grouping, Aggregation	SELECT operation_Name, AVG(duration) FROM dependencies GROUP BY name	dependencies summarize avg(duration) by name
Column aliases, Extend	SELECT operation_Name as Name, AVG(duration) as AvgD FROM dependencies GROUP BY name	dependencies summarize AvgD=avg(duration) by operation_Name project Name=operation_Name, AvgD

Kusto for Splunkers
aka.ms/SPL2KQL

How do we use it?

How do we use Log Analytic Queries?

Using Log Analytics queries is the way you manipulate at the data in Microsoft Sentinel.

Log Analytics provides the following features for working with log queries:

- **Multiple tabs** – create separate tabs to work with multiple queries
- **Rich visualizations** – variety of charting options
- **Improved Intellisense** and **language auto-completion (tab complete)**
- **Syntax highlighting** – improves readability of queries
- **Query explorer** – access saved queries and functions
- **Schema view** – review the structure of your data to assist in writing queries
- **Share** – create links to queries, or pin queries to any shared Azure dashboard
- **Smart Analytics** - identifies spikes in your charts and a quick analysis of the cause
- **Column selection** – sort and group columns in the query results

Query Window Overview

Home > Azure Sentinel | Logs

Azure Sentinel | Logs
Selected workspace: 'CyberSecurityDemo'

New Query 1* +

CyberSecurityDemo

Time range : Last 24 hours | Save | Copy link | New alert rule | Export | Pin to dashboard | Prettify query

Run

SecurityEvent
where EventID == 4625

Query window

The ribbon

Results | Chart | Columns | Add bookmark | Display time (UTC+00:00)

Completed. Showing partial results from the last 24 hours. 00:00:06.015 10,000 records

Tables

Search

Group by: Solution Filters: not selected

Favorites

- BehaviorAnalyticsInsights
- Change Tracking
- ContainerInsights
- DNS Analytics (Preview)
- LogManagement
- Network Performance Monitor
- Office 365 Analytics (Preview)
- Security and Audit
 - CommonSecurityLog
 - Activity (string)
 - AdditionalExtensions (string)
 - ApplicationProtocol (string)
 - CommunicationDirection (string)
 - Computer (string)
 - DestinationDnsDomain (string)

Column chooser

	TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
> <input type="checkbox"/>	4/22/2020, 10:31:04.483 AM	\OSLADMIN	User	VictimPC.Contoso.Az...	Microsoft-Windows-Security-Auditing	Security
> <input type="checkbox"/>	4/22/2020, 10:31:04.760 AM	\OUJADM	User	VictimPC.Contoso.Az...	Microsoft-Windows-Security-Auditing	Security
▼ <input type="checkbox"/>	4/22/2020, 10:31:04.903 AM	\SVCADM	User	VictimPC.Contoso.Az...	Microsoft-Windows-Security-Auditing	Security
...						
	TenantId	ab86c959-1ba3-495c-a00d-ced30d8825d3				
	TimeGenerated [UTC]	2020-04-22T10:31:04.903Z				
	System	OpsManager				
	Account	\SVCADM				
	AccountType	User				
	Computer	VictimPC.Contoso.Azure				
	EventSourceName	Microsoft-Windows-Security-Auditing				

Results pane

Typed columns

Learn By Doing...

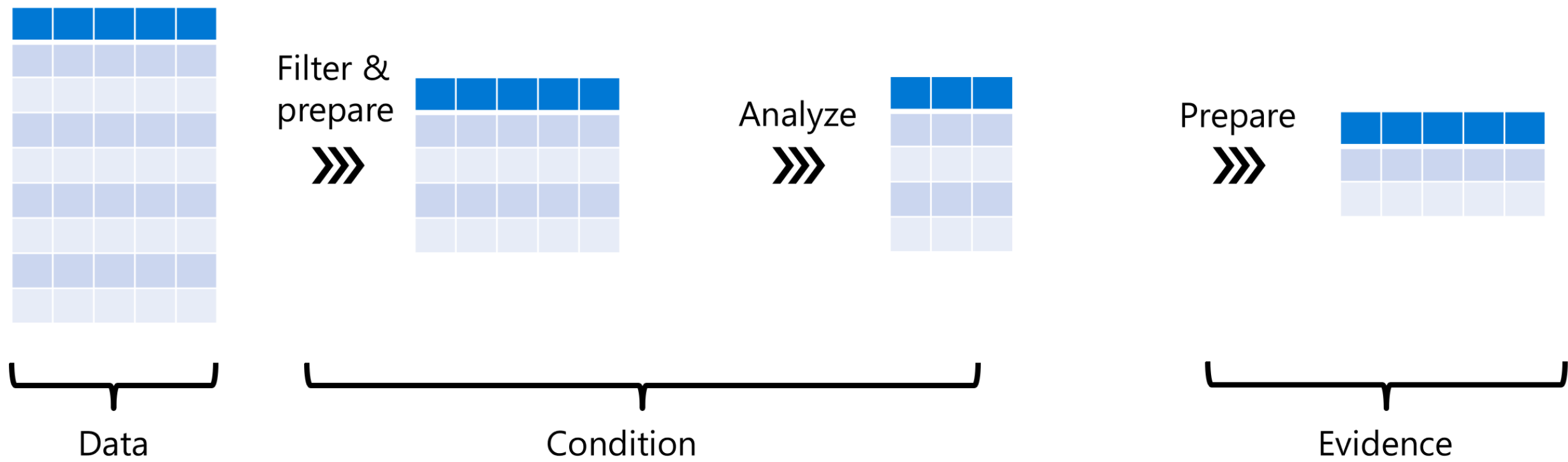
Follow along...

aka.ms/LADemo

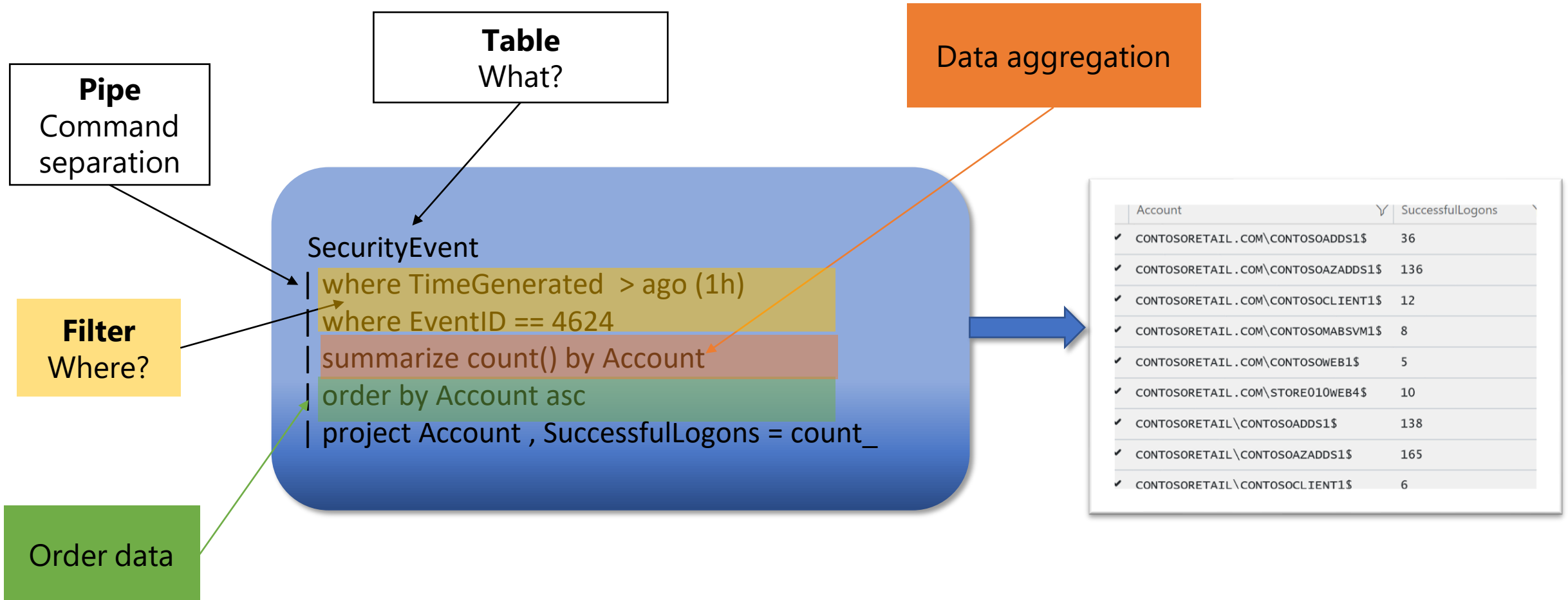
Kusto Queries 101

Understanding the pipe

```
SecurityEvent | where EventID == "4624" | summarize count() by Account | top 10 by _count
```



(The Essence) Structure of KQL Queries



KQL Scalar Data Types

'int' data type

The int data type represents a signed, 32-bit wide, integer.

Example: *SecurityEvent*
 | *where EventID == 4625*

- These are simply whole numbers.
- Common examples include: EventID, LogonType, ErrorCode, KeyLength
- Can hold a value of -2,147,483,648 to 2,147,483,647

Column	Value
EventID	4625
LogonType	5
ErrorCode	1222
KeyLength	6

'long' data type

The long data type represents a signed, 64-bit wide, integer.

Example:

NetworkMonitoring
| *where* MemTotalInMB == *7,677,000*

Column	Value
MemTotalInMB	7,677,000
BytePerSecond	5,534,123

- This is best used if a number is too small or large for int to handle.
- Can hold a value range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

'string' data type

The string data type represents a sequence of zero or more Unicode characters.

Example:

```
SecurityEvent  
| where Account == @"domain\account"
```

- Strings are simply words or phrases.
- Most strings are contained inside "" or '' .
- If the string contains a "/" or "\", it must be preceded by an "@" symbol to show that it's a string.

Column	Value
Account	@ "NA\Admin"
AccountDomain	"NA"
Channel	"Security"
IPAddress	"172.1.4.6"
FilePath	@ "C:\Windows"

'timespan' data type

The timespan (time) data type represents a time interval.

Syntax: *T* | *time* <*time*>

Example: *SecurityEvent* | *time* (5h)

- Two values of timespan can be added, subtracted, or divided.

Value	Length of Time
2d	2 days
1.5h	1.5 hours
30m	30 minutes
10s	10 seconds
100ms	100 milliseconds
time (15 seconds)	15 seconds
time(2)	2 days

'datetime' data type

The datetime (date) data type represents an instant in time, typically expressed as a date and time of day.

Syntax: `T | time <time>`
Example: `SecurityEvent`
`| where TimeGenerated ago(7d)`

- These values are always in UTC time.
- You can use “..” to indicate a time range.
- Most commonly used with the TimeGenerated column.

Example	Value
datetime(2023-03-17)	2023-03-17
ago(7d)	7 days ago
ago(14d)..ago(7d)	14 days to 7 days ago
now()	Current UTC Time
now(offset)	Current Time - Offset

In-Line Commenting

In-Line Commenting

This allows you to comment out lines of code that you want to exclude from running or add comments about the query for contextual understanding.

Example: *SecurityEvent*
 // project Account, Computer

- Uses a double `"/"` or `//` to indicate a comment or exclusion.

Operators and Functions

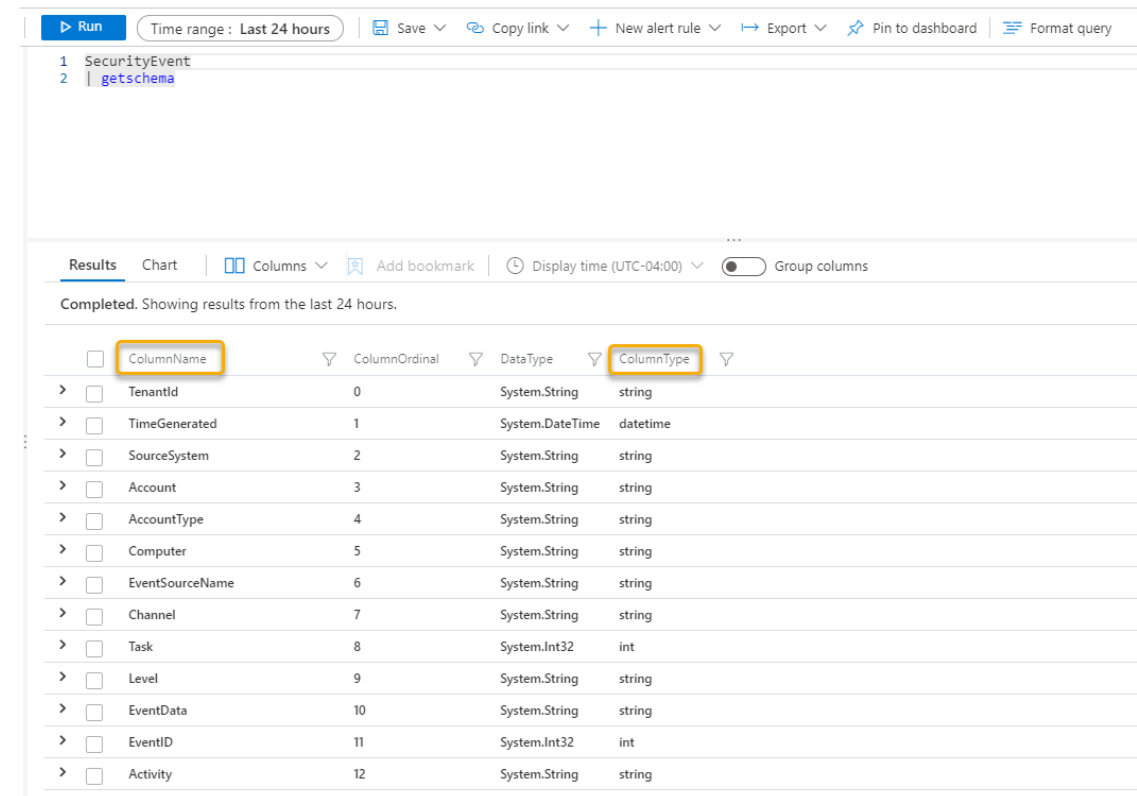
'getschema' operator

Shows the schema (Data-Types) for the table that you are working with.

Syntax: *Table* | *getschema*

Examples: *SecurityEvent* | *getschema*

- This shows what columns and data types are available to work with.
- This information is also viewable from the "Tables" pane in the UI



Run Time range: Last 24 hours Save Copy link New alert rule Export Pin to dashboard Format query

```
1 SecurityEvent
2 | getschema
```

Results Chart Columns Add bookmark Display time (UTC-04:00) Group columns

Completed. Showing results from the last 24 hours.

<input type="checkbox"/>	ColumnName	ColumnOrdinal	DataType	ColumnType
> <input type="checkbox"/>	TenantId	0	System.String	string
> <input type="checkbox"/>	TimeGenerated	1	System.DateTime	datetime
> <input type="checkbox"/>	SourceSystem	2	System.String	string
> <input type="checkbox"/>	Account	3	System.String	string
> <input type="checkbox"/>	AccountType	4	System.String	string
> <input type="checkbox"/>	Computer	5	System.String	string
> <input type="checkbox"/>	EventSourceName	6	System.String	string
> <input type="checkbox"/>	Channel	7	System.String	string
> <input type="checkbox"/>	Task	8	System.Int32	int
> <input type="checkbox"/>	Level	9	System.String	string
> <input type="checkbox"/>	EventData	10	System.String	string
> <input type="checkbox"/>	EventID	11	System.Int32	int
> <input type="checkbox"/>	Activity	12	System.String	string

'search' operator

Easy to use. Use interactively for threat hunting but not in analytic rules.
Filter first to avoid long search times.

Syntax: *Table* | *search*

Examples: *search "administrator"*

- "T |" and "in (Tables)" are optional. If no table is specified, it will search all tables.
- Filter first before search to save time.
- The result set will include a "\$table" field and will indicate the table name in the output, if more than one table is searched.
- **If you are going to use a file-path in your search, you will need to "@" before the file-path ..i.e.(@"C:/Windows/System32/hacked.dll")**

Multi-Table Searches – 3 Questions To Ask

The *search* operator provides a multi-table/multi-column search experience.

▶ RunTime range : Last 24 hoursSaveCopy linkNew alert ruleExportPin to dashbo

1 search "sixmilliondollarman"

Does it exist?

ResultsChartColumnsAdd bookmarkDisplay time (UTC-04:00)Group columns

Completed. Showing results from the last 24 hours.

<input type="checkbox"/>	TimeGenerated [Local Time]	<input type="text" value="\$table"/>	Corr
> <input type="checkbox"/>	9/20/2020, 2:48:24.130 PM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/20/2020, 3:47:20.177 PM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/20/2020, 3:47:16.083 PM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/20/2020, 5:41:14.832 PM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/20/2020, 6:37:12.069 PM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/21/2020, 8:00:51.671 AM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/21/2020, 8:00:49.872 AM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/21/2020, 8:22:19.356 AM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/21/2020, 8:30:08.023 AM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/21/2020, 8:55:34.398 AM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/21/2020, 9:09:34.245 AM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/21/2020, 10:42:04.033 AM	AADNonInteractiveUserSignInLogs	
> <input type="checkbox"/>	9/21/2020, 1:07:24.306 PM	AADNonInteractiveUserSignInLogs	

▶ RunTime range : Last 24 hoursSaveCopy linkNew alert ruleExportPin to das

1 search "sixmilliondollarman"
2 | distinct \$table

Where does it exist?

ResultsChartColumnsAdd t

Completed. Showing results from the last 24 hours.

<input type="checkbox"/>	\$table
> <input type="checkbox"/>	SignInLogs
> <input type="checkbox"/>	IntuneDeviceComplianceOrg
> <input type="checkbox"/>	AADNonInteractiveUserSignInLogs
> <input type="checkbox"/>	OfficeActivity
> <input type="checkbox"/>	Event
> <input type="checkbox"/>	AzureActivity

▶ RunTime range : Last 24 hoursSaveCopy linkNew alert ruleExport

1 search in (OfficeActivity) "sixmilliondollarman"

Why does it exist?

ResultsChartColumnsAdd bookmarkDisplay time (UTC-04:00)Group

Completed. Showing results from the last 24 hours.

<input type="checkbox"/>	TimeGenerated [Local Time]	<input type="text" value="\$table"/>	<input type="text" value="RecordType"/>	<input type="text" value="Operation"/>
> <input type="checkbox"/>	9/21/2020, 8:51:43.000 AM	OfficeActivity	50	MailItemsAccessed
> <input type="checkbox"/>	9/21/2020, 8:55:36.000 AM	OfficeActivity	ExchangeItemGroup	MoveToDeletedItems

'limit' / 'take' operators

Return up to the specified number of rows.

Syntax: *T* | *limit* <number>

Example: *SecurityEvent* | *limit* 5

- Sort is not guaranteed to be preserved.
- Consistent result is not guaranteed (when running the same query twice)
- Very useful when trying out new queries.
- Default limit is 30,000.

'limit/take' example

```
SecurityEvent
```

```
| limit 10
```

```
SecurityEvent
```

```
| where TimeGenerated > ago(1h)
```

```
| where EventID == 4624
```

```
| where AccountType =~ "user"
```

```
| take 10
```

'top' operator

Returns a list of the first n records sorted by specified column/s

Syntax: $T \mid \text{top } \langle \text{number} \rangle \text{ by } \langle \text{Column} \rangle$

Example: $\text{SecurityEvent} \mid \text{top } 10 \text{ by Account}$

- Great for looking for anomalies, data spikes, and anything else that is unusual.

'top' example

```
SecurityEvent  
| where TimeGenerated > ago(1h)  
| where EventID == 4624  
| summarize count() by Account  
| top 10 by _count
```

'project' operator

Select the columns to include, rename or drop, and insert new computed columns.

Syntax: $T \mid \textit{project ColumnName [= Expression] [, ...]}$

Example: $\textit{SecurityEvent} \mid \textit{project TimeGenerated, Computer}$

Additional Project operators:

'| project-away' – Removed specified column/s.

'| project-rename' – Rename specified column/s.

'| project-keep' – Select what columns from the input to keep in the output.

'| project-reorder' – Reorders columns in the result output

'project' example

SecurityEvent

```
| where EventID == 4624 or EventID == 4625  
| project TimeGenerated, Computer, Account  
| take 10
```

'distinct' operator

Produces a table with the distinct or unique combination of the provided columns of the input table..

Syntax: *T* | *distinct* *Column1*, *Column2*

Example: *SecurityEvent* | *distinct* *Computer*

'distinct' example

```
SecurityEvent
```

```
| where EventID == 4625
```

```
| distinct Computer
```

```
ThreatIntelligenceIndicator
```

```
| where ThreatType contains "BotNet"
```

```
| distinct NetworkSourceIP
```

Filtering Data

'where' operator

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

Syntax: *Table* | *where* *Predicate*

Examples: *SecurityEvent*

| *where* *EventID* == 1102

| *summarize* *LogClearedCount* = *count()* by *Computer*

- **String:** has, contains, startswith, endswith, matches regex, etc
- **Numeric/Date:** ==, !=, <, >, <=, >=
- **Lookup:** in, !in, has_any
- **Empty:** isempty(), notempty(), isnull(), notnull()

Note: contains = *Word* ; has = "Word"

'where' example

SecurityEvent

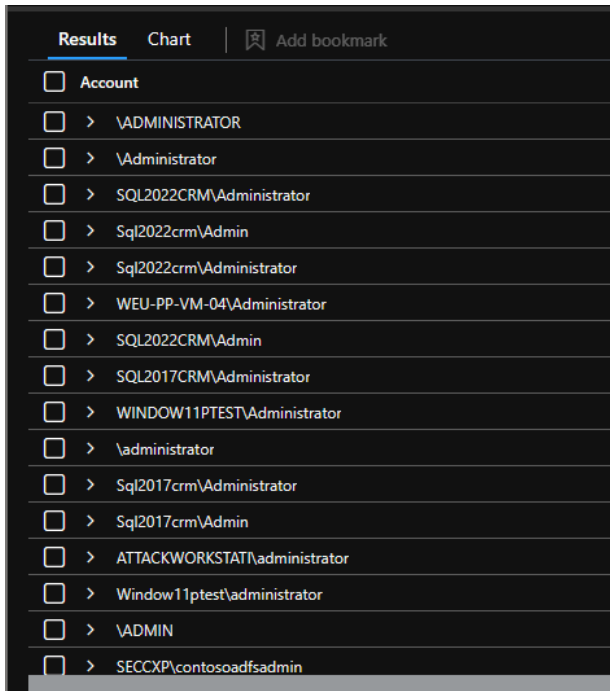
```
| where TimeGenerated > ago(1h)  
| where EventID == 4624 // Successful logon  
| where AccountType =~ "user" // case insensitive
```

'contains' filtering

The 'contains' filter has no word boundary around the phrase being searched. It will return any results that contains 'phrase' in the result.

Syntax: *Table | **where** Predicate contains 'phrase'*

Examples: *SecurityEvent
| **where** Account contains 'admin'*



The screenshot shows a search results interface with a dark theme. At the top, there are tabs for 'Results' and 'Chart', and a link to 'Add bookmark'. Below the tabs is a list of search results, each preceded by a checkbox. The results are accounts that contain the word 'admin'.

<input type="checkbox"/> Account
<input type="checkbox"/> > \ADMINISTRATOR
<input type="checkbox"/> > \Administrator
<input type="checkbox"/> > SQL2022CRM\Administrator
<input type="checkbox"/> > Sql2022crm\Admin
<input type="checkbox"/> > Sql2022crm\Administrator
<input type="checkbox"/> > WEU-PP-VM-04\Administrator
<input type="checkbox"/> > SQL2022CRM\Admin
<input type="checkbox"/> > SQL2017CRM\Administrator
<input type="checkbox"/> > WINDOW11PTEST\Administrator
<input type="checkbox"/> > \administrator
<input type="checkbox"/> > Sql2017crm\Administrator
<input type="checkbox"/> > Sql2017crm\Admin
<input type="checkbox"/> > ATTACKWORKSTAT\administrator
<input type="checkbox"/> > Window11ptest\administrator
<input type="checkbox"/> > \ADMIN
<input type="checkbox"/> > SECCXP\contosoadmin

'has' filtering

The 'has' filter has a word boundary around the phrase being searched. It will only return results that contains the exact 'phrase' in the result.

Syntax: *Table | **where** Predicate has 'phrase'*

Examples: *SecurityEvent
| **where** Account has 'Admin'*

<input type="checkbox"/>	Account
<input type="checkbox"/>	> Sql2022crm\Admin
<input type="checkbox"/>	> SQL2017CRM\Admin
<input type="checkbox"/>	> Sql2017crm\Admin
<input type="checkbox"/>	> SQL2022CRM\Admin
<input type="checkbox"/>	> \ADMIN
<input type="checkbox"/>	> WEU-PP-VM-04\admin
<input type="checkbox"/>	> \admin
<input type="checkbox"/>	> ATTACKWORKSTAT\admin
<input type="checkbox"/>	> \Admin
<input type="checkbox"/>	> ATTACKWORKSTATION\Admin
<input type="checkbox"/>	> \ADMIN!
<input type="checkbox"/>	> \SUPER_ADMIN
<input type="checkbox"/>	> \yjlt_admin
<input type="checkbox"/>	> \integrity-admin
<input type="checkbox"/>	> \spc_admin

'Count' operator

Returns the number of records

Syntax: T | *count*

Example: *SecurityEvent* | *count*



'Count' example

SecurityEvent

```
| where TimeGenerated > ago(1h)  
| where EventID == 4624  
| count
```

Analyzing Data

'Summarize' operator

Produces a table that aggregates the content of the input table.

Syntax: T | *summarize*

Examples: *SecurityEvent* | *summarize count() by Account*

Often used with the count() operator.

Simple aggregation functions: count(), sum(), avg(), min(), max(),

Advanced summarizations are covered in the Advanced KQL module.

'Summarize' example

```
//Logons with clear text password by target  
account
```

```
SecurityEvent
```

```
| where EventID == 4624 and Logontype == 8  
| summarize count() by TargetComputer
```

'Percentile' function

Calculates an estimate for the specified nearest-rank percentile of the population defined by *Expr*.

Syntax: *T* | *summarize percentile(Expr, Percentile)*

Examples: *Perf*
| *where CounterName == "Available Mbytes"*
| *summarize percentile(CounterValue, 90) by Computer*

Always used with the summarize operator

Results		Chart
Computer		percentile_CounterValue_90
>	DC01.na.contosohotels.com	5,183.375
>	DC10.na.contosohotels.com	5,204.25
>	AppFE000012N	5,908
>	SQL00.na.contosohotels.com	11,957.25
>	AppBE00.na.contosohotels.com	4,841
>	SQL01.na.contosohotels.com	11,461.6
>	AppBE01.na.contosohotels.com	5,253
>	DC11.na.contosohotels.com	6,053.625
>	AppFE0000002	6,006.167
>	RETAILVM01	5,712.5
>	AppFE0000003	6,345.375
>	SQL12.na.contosohotels.com	11,827
>	CH1-AVSMGMTVM	3,736.2
>	DC00.na.contosohotels.com	5,170.2
>	JBOX00	6,195

'Percentile' example

Perf

| *where* CounterName == "% Processor Time"

| *summarize* percentile(CounterValue, 90) *by* Computer

Presenting Data

'extend' operator

Create calculated columns and append them to the result set.

Syntax: *T | extend ColumnName [= Expression] [, ...]*

Example: *SecurityEvent | extend ComputerNameLength = strlen(Computer)*

- The new added column is not indexed.
- To only change a column name, use 'project-rename'.
- Useful functions for 'extend': iff, extract

'extend' example

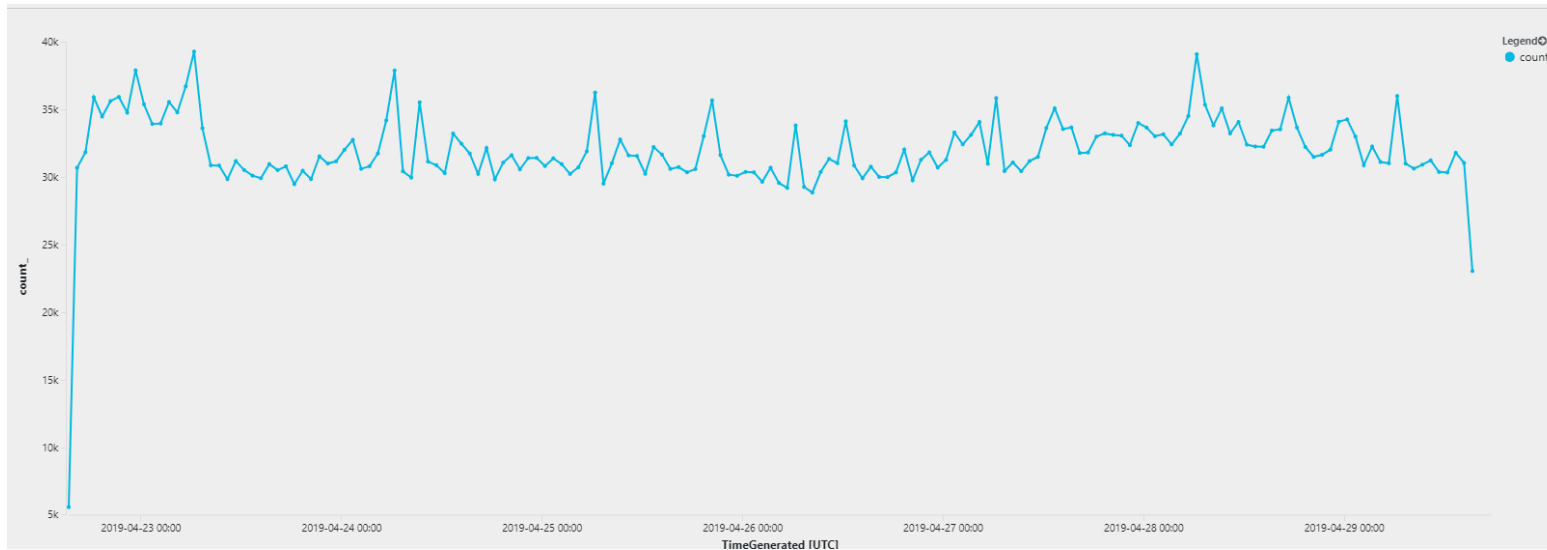
Perf

```
| where CounterName == "Free Megabytes"  
| extend FreeKB = CounterValue * 1024  
| extend FreeGB = CounterValue / 1024  
| extend FreeMB = CounterValue  
| project Computer, CounterName, FreeGB, FreeMB, FreeKB
```

'Summarize' : bin and time series

A very useful summarize operation is creating time series:

`SecurityEvent | summarize count() by bin(TimeGenerated, 1h) | render timechart`



Other time measurements: 1h, 5d, 10m (defaults to 1h)

Can create multiple legends by aggregating additional field

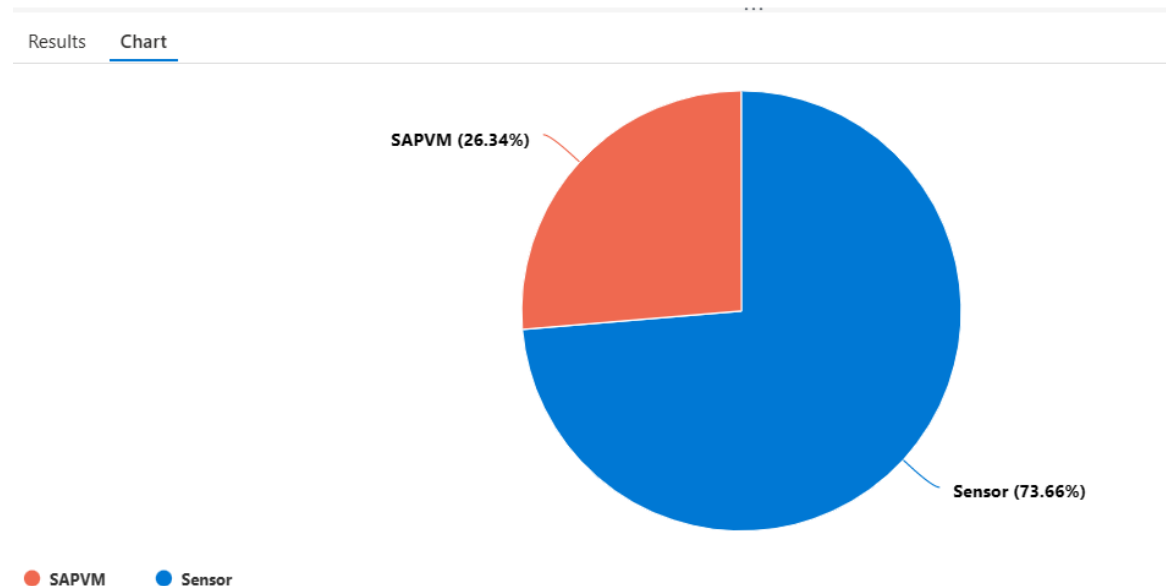
'Render' Operator

Generates a visualization of the query results.

Syntax: $T \mid \text{render Visualization [with (PropertyName = PropertyValue [, ...])]}$

Supported visualizations:

- Areachart
- Barchart
- Columnchart
- Piechart
- Scatterchart
- timechart



'bin' and 'render' example

SecurityEvent

```
| where TimeGenerated > ago(7d)  
| summarize count() by bin(TimeGenerated, 1d)  
| render barchart
```

'let' statement

The **let** command creates a temporary variable that can hold a single value, list, or table

- Used as a traditional variable before a complex query
- Used for allow, deny, and exclusion lists
- Used as a temporary container for a table
- Useful for targeted threat hunting

For good examples of **let** variables and complex queries, check out the Sentinel scheduled rule templates

'let' statement example

```
let suspiciousAccounts = datatable(account: string) [  
    @"\\administrator",  
    @"NT AUTHORITY\\SYSTEM"  
];  
SecurityEvent | where Account in (suspiciousAccounts)
```

Joining Data

'union' operator

Takes two or more tables and returns the rows of all of them.

Example: *SecurityEvent | union (SecurityAlert | where AlertSeverity == "high")*

- kind=inner(common columns), outer (all columns- default)
- Supports wildcard to union multiple tables (union Security*)
- Can union between tables from different clusters (or workspaces)

'union' example

```
SecurityEvent  
| union Heartbeat  
| summarize count() by Computer
```

'join' operator

Merge the rows of two tables to form a new table by matching values of the specified column(s) from each table.

Syntax: LeftTable | join [JoinParameters] (RightTable) on Attributes

Example: *SecurityEvent* | join (*SecurityAlert* | where *AlertSeverity* == "high") on *Status*

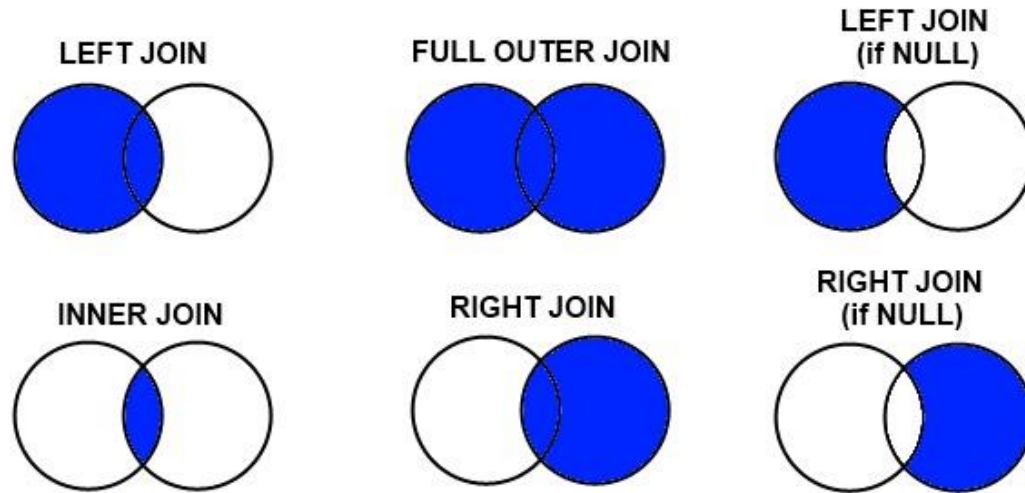


Table1 | join (Table2) on CommonColumn, \$left.Col1 == \$right.Col2

'join' example

```
SecurityEvent  
| join Heartbeat on Computer  
| where EventID == "4688"  
| project Computer, OSType, OSMajorVersion,  
Version
```

Syntax Review

- Each command is separated by a pipe "|" (like PowerShell).
- KQL queries are case-insensitive by default.
- Lines are not delimited, no (;) to mark the end of a line. (Except for Let statements)
- Command are typically 'stacked' one per line (though single line is acceptable).
- Extra spaces between commands are ignored.
- Data types include basic, int (long), Boolean, string, datetime, timestamp, or dynamic (JSON).
- Operators include ==, !=, contains, has, !in, startswith, <=, >=, and many more.
- In-line commenting "--" are supported.
- Anything in quotes is treated as a string.
- Numbers with no quotes are treated as a long integer.

Kusto Queries Summary

where – Reduces or filters the number of rows returned.

project – Modifies columns by limiting and renaming the columns displayed.

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

extend – Adds a new column using static or dynamic inputs.

summarize – Groups or aggregates data (key to visualization).

search – Queries all tables or specific tables for keywords.

count - Returns the number of records in the input record set.

take - Returns the specified number of rows (same as limit).

render – Lets you define the chart for consistent viewing.

let – Defines a variable that can store anything from a single value to an entire table.

join - Merge the rows of two tables to form a new table based on a matching column.

union – Stack two or more tables (**type=isfuzzy** will ignore empty tables).

iff() and **case()** – Allow conditional responses, often combined with extend.

ago() – Core component for date-time evaluation and lookback.

split() and **parse()** – Breaks up columns using a delaminated value into an array (ex. comma separated).

Pop Quiz Time!



Question 1:

Find the mistakes

```
SecurityEvents  
| join Heartbeat on Computer  
| where EventID == "4688"  
| project computer, OSType, OSMajorVersion,  
Version
```


Question 2:

Correct the mistakes

```
SecurityEvent  
| where EventID == "4624"  
| project TimeGenerated, Accounts, Computer
```

Question 3:

True or False?

When using the 'has' filter, a query will return any record that includes the phrase.

Question 4:

Finish the Query

```
SecurityEvent
```

```
| where TimeGenerated >= ____ (7d)
```

```
| where _____ == 4624
```

```
| _____ TimeGenerated, Computer, Activity
```

Question 5:

Find the Mistakes

```
SignInLogs  
| where Timestamp > ago(1d)  
| count() by AppDisplayName  
| render piechart
```

Question 6:

True or False?

In order to find out what columns are in the table, you can use the mapschema operator to show the table schema.

Question 7:

Finish the Query

```
__ suspiciousIPs = datatable(IPAddress: string) [  
    "10.34.56.3",  
    "192.168.2.3"  
];  
SecurityEvent | where _____ in (suspiciousIPs)
```

Question 8:

True or False?

There is only one way to rename a column in KQL?

Question 9:

True or False?

KQL is short for Kusto Query Language.

Question 10:

Find the Mistake

```
SecurityEvent  
| where EventID = 4624  
| Project TimeGenerated, Computer, Account
```

Hands-on Lab



1st Scenario

Your SOC has just discovered that a hacker has been using brute force attacks against your network for the past 7 days.

You need to find the count of failed logins for each user account and computer being attacked during that period.

Hints and guidelines:

- Use the SecurityEvent table to begin your search.
- How would you find the EventID for failed logins using KQL?
- What column would have the account name?



1st Lab Exercise

```
// Find the count of failed logins by Account Name
// run parts of the query, adding a line at the time, to learn more
```

SecurityEvent

```
| where TimeGenerated <= ago(7d)
| where EventID == 4625
| summarize count() by TargetAccount, Computer
```

Results Chart		
TargetAccount	Computer	count_
> NA\sqlservice	SQL12.na.contosohotels.com	73481
> na.contosohotels.com\sh360DB\$	SQL00.na.contosohotels.com	8292
> na.contosohotels.com\SQLc\$	SQL00.na.contosohotels.com	3196
> na.contosohotels.com\sqlc\$	SQL01.na.contosohotels.com	2491
> NA\sqlservice	SQL01.na.contosohotels.com	133
> \ADMINISTRATOR	JBOX00	128
> \timadmin	SQL01.na.contosohotels.com	36
> \ADMIN	JBOX00	12
> \USER	JBOX00	12
> \PC	JBOX00	11
> \timadmin	CH1-AVSMGMTVM	11
> \HP	JBOX00	11
> \STUDENT	JBOX00	6

2nd Scenario

A hacker has compromised your network and has successfully logged into your network. To find the intruder, you need to find all Windows logon events starting 2 weeks ago until 1 week ago that occurred on a computer with a name which starts with "App" .

- Hints and guidelines:
- Windows security events are stored in the table "SecurityEvent"
- The logon event id is 4624. What is the name of the field which contains the event ID?
- What is the name of the field which represents the computer name?
- What should be the order of the commands for better performance?
- **Bonus:** Can you find the count per computer as well?

2nd Lab Exercise

```
// Find all Windows logon events starting 2 weeks ago until 1 week ago that occurred on a computer with name which starts with "App"
```

```
SecurityEvent | limit 100 // Find relevant fields: Activity, EventID, Computer
```

```
SecurityEvent | summarize by Activity // find the Event signaling login
```

```
SecurityEvent  
| where TimeGenerated between (ago(14d)..ago(7d)) // start with the time filter  
| where EventID == "4624"  
| where Computer startswith "App" // case insensitive  
// This is the solution, but there are so many results
```

```
SecurityEvent  
| where TimeGenerated between (ago(14d)..ago(7d))  
| where EventID == "4624"  
| where Computer startswith "App"  
| summarize count() by Computer  
// so let's count per computer
```

Results		Chart
Computer	count_	
> AppBE01.na.contosohotels.com	1896	
> AppBE00.na.contosohotels.com	1590	
> AppFE0000C3Y	364	
> AppFE0000C3W	382	

3rd Scenario

**An APT has installed malware on your network .
In order to find the traces of malware, you need to find out how many times
each process ran per computer.**

Hints and guidelines:

- Event 4688 logs process creation.
- Which column represents the processes created?
- Which computer was it ran on?



3rd Lab Exercise

```
// Find how many times each process ran per computer
```

```
SecurityEvent | summarize by Activity // Let's find the event that includes  
process names
```

```
SecurityEvent | where EventID == "4688" | limit 10  
// find the relevant field, in this case "Process"
```

```
SecurityEvent  
| where EventID == "4688"  
| summarize count() by Process, Computer
```

Results			Chart
Process	Computer	count_	
> conhost.exe	DC01.na.contosohotels.com	2683	
> conhost.exe	DC00.na.contosohotels.com	2685	
> conhost.exe	DC10.na.contosohotels.com	2699	
> conhost.exe	DC11.na.contosohotels.com	2798	
> conhost.exe	JBOX00	4218	
> conhost.exe	JBOX10	4229	
> conhost.exe	AppBE01.na.contosohotels.com	3585	
> conhost.exe	AppBE00.na.contosohotels.com	3139	

4th Scenario

Your SOC has just discovered a Crypto-Mining Agent has been installed on one of your domain controllers.

You need to chart the rate of process creation on all domain controllers in order to discover which DC has been compromised .

- Hints and guidelines:
- Process creation is Windows event 4688
- All Domain controller names start with "DC"
- This will be a time chart. (Think bin...)

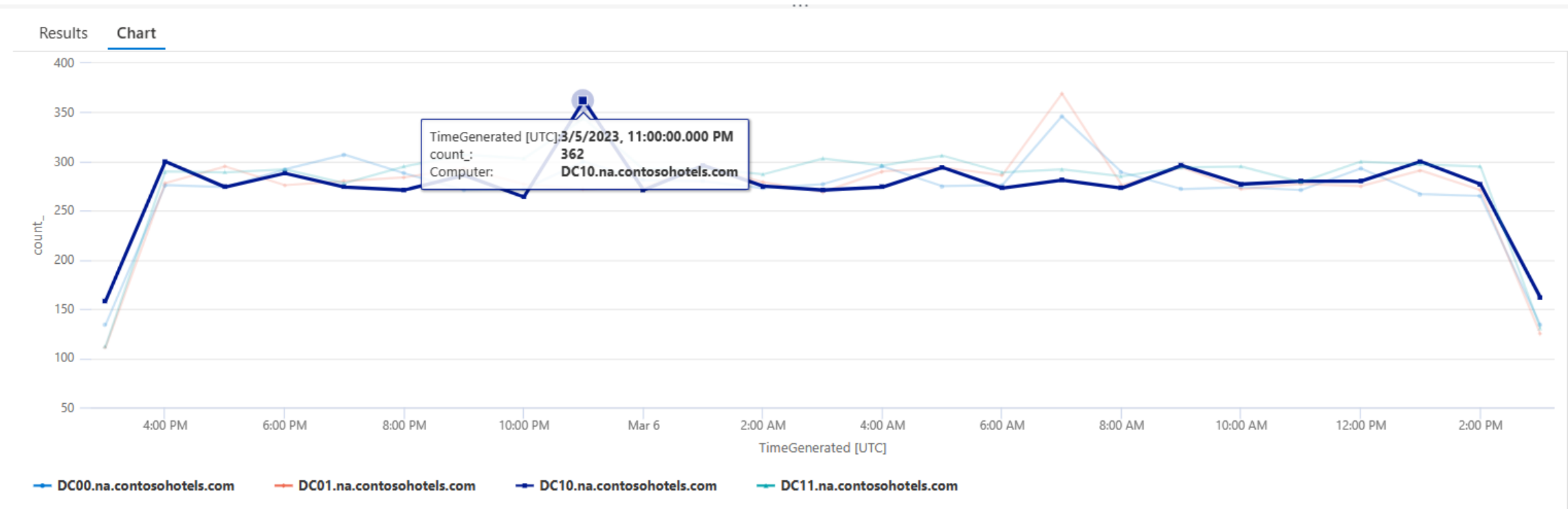


4th Lab Exercise

// Chart the rate of process creation on all domain controllers.

SecurityEvent

```
| where Computer startswith "DC"  
| where EventID == "4688" | summarize count() by Computer, bin(TimeGenerated, 1h)  
| render timechart
```



5th Scenario

A hacker has run an encoded PowerShell command on your network and has started exfiltrating data. You need to find the encoded PowerShell command that was ran on a computer that **endswith CSK that was ran in the past 90 days.**

Hints and Guidelines:

- Use the VMProcess table for your hunt.
- There *should* only be one result.



5th Lab Exercise

// Find the encoded PowerShell command that was ran on a computer that ends with App for the past 60 days.

VMProcess

```
| where TimeGenerated >= ago(90d)  
| where Computer endswith "CSK"  
| where CommandLine contains "encoded"
```

6th Scenario

As a part of your post incident response, you need to compare the successful and failed logons to determine what day the password spray took place.

You will need to render a graph of successful vs failed logons over the last 30 days, use alias for the legend ("Success", "Failed") to find your answer.

Hints and guideline:

- Utilize Countif for each EventID
- Remember this is a time chart.



6th Lab Exercise

```
// Render graph of successful vs failed logons over the last 30 days, use alias for the legend ("Success", "Failed")  
// run parts of the query, adding a line at the time, to learn more
```

```
SecurityEvent  
| where TimeGenerated > ago(30d)  
| summarize  
    Success=countif(EventID == 4624),  
    Failed=countif(EventID == 4625)  
    by bin(TimeGenerated, 1h)  
| render timechart
```

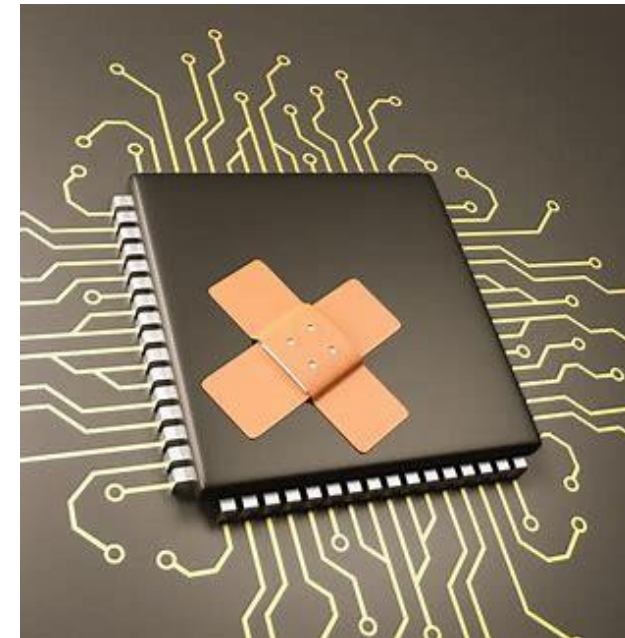
7th Scenario

Your company has hired a Pentester to audit your security environment, the results of the test determined that missing patches have resulted in a successful attack against your systems.

You need to find the missing **critical security updates for the VM that starts with App.**

Hints and guideline:

- Use the Updates table to begin your hunt.
- How would you find patch information?



7th Lab Exercise

//Find the missing critical security updates for the VM that starts with App.

Update

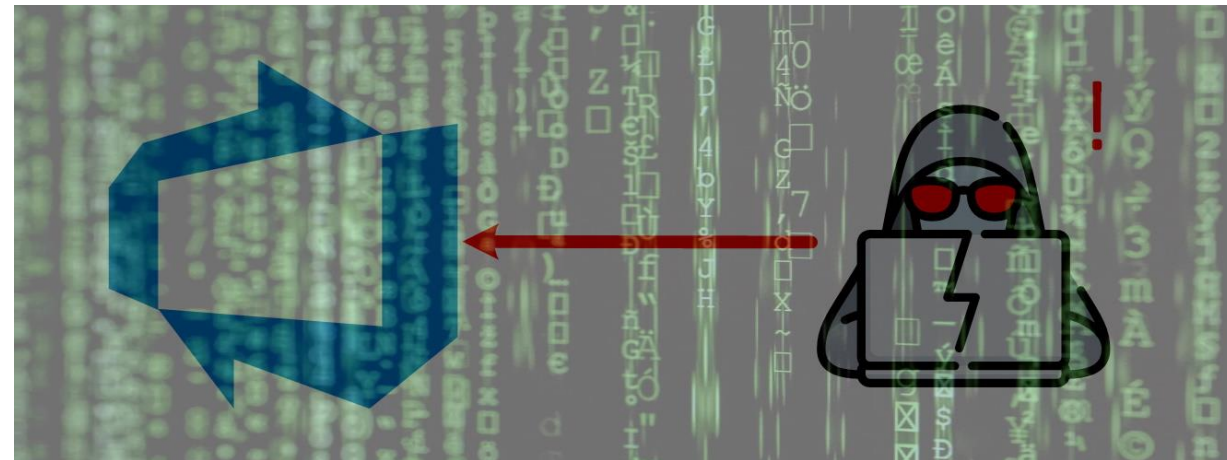
```
| where Classification == "Security Updates" and UpdateState == "Needed" and  
MSRCSeverity == "Critical"  
| where Resource startswith "App"
```


8th Scenario

Your Azure Environment has suffered an attack with multiple brute force attacks. No compromise took place; however, you need to know what user account was used and what country the attacker attempted to log in from. The results should only show the timestamp, Status, Username and the app name.

Hints and guideline:

- This will involve more than one table.
- The company did have MFA enabled.
- Limit your search to the past 60 days.



8th Lab Exercise

// You need to know what user account was used to sign in and what country they signed in from in the past 60 days.

```
union SigninLogs, AADNonInteractiveUserSignInLogs
| where TimeGenerated >= ago(60d)
| where Status_dynamic contains "User did not pass the MFA challenge."
| where AppDisplayName contains "Azure Portal"
| project TimeGenerated, Status_dynamic, AppDisplayName
```

9th Scenario

Your Azure Environment has successfully defended an attack from an outside entity. As a part of your IR Report, you need to find the top 3 source IP addresses which were blocked by your firewall.

Hints and guideline:

- Which table would you use?
- What way does the data “flow” ?



9th Lab Exercise

// Find the top 3 source IP addresses which were blocked by your firewall.

```
AzureNetworkAnalytics_CL
```

```
| where FlowStatus_s == "D"
```

```
| where FlowDirection_s == "I"
```

```
| where isnotempty(SrcIP_s)
```

```
| summarize count() by SrcIP_s
```

Fun with Functions



The Anatomy of a Function:

Function name *

WeeklySecurityEvent

The Function Name

Code

```
SecurityEvent
| where TimeGenerated >= ago(7d)
| summarize count() by Activity
```

The Function Code

Legacy category *

Security

Unused category.
(Insert anything)

☐ Save as computer group ⓘ

Parameters

Type

Name

Default value

Select type



Type name

Type default value

Function Parameters

Save

Cancel

Save / Cancel Buttons

The Anatomy of a Function with Parameters:

The screenshot shows a web-based interface for defining a function. It includes a 'Function name' field with the value 'FindEventID', a 'Code' block containing a SQL query, a 'Legacy category' dropdown set to 'Security', a 'Parameters' table with one row for 'TERM', and 'Save' and 'Cancel' buttons at the bottom. Annotations with arrows point to these elements from the right side of the image.

Function name *

FindEventID

The Function Name

Code

```
SecurityEvent
| where Activity contains TERM
| distinct Activity
```

The Function Code

Legacy category *

Security

Unused category.
(Insert anything)

☐ Save as computer group ⓘ

Parameters

Type	Name	Default value
string	TERM	

Function Parameter
(must match the
PARAM in code)

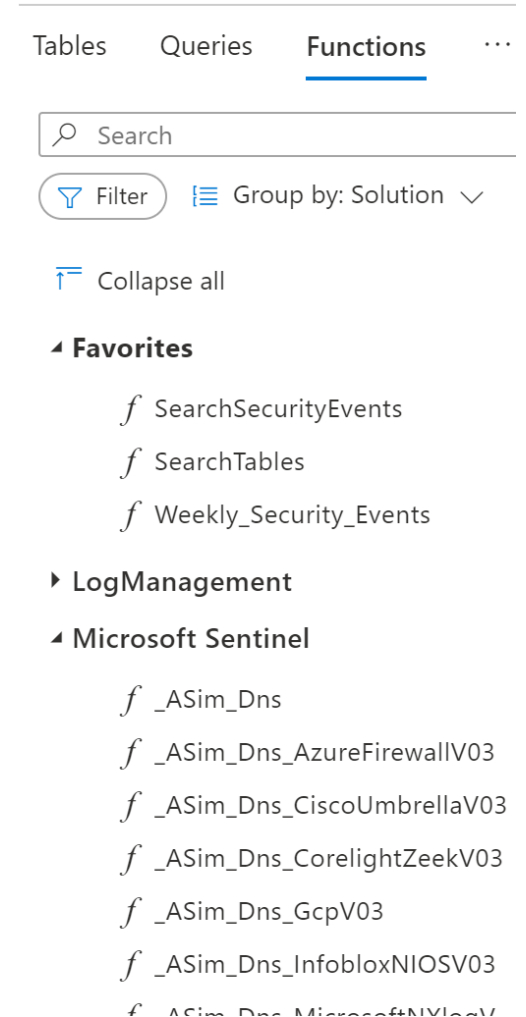
Select type ▼ Type name Type default value

Save Cancel

Save / Cancel Buttons

Creating a Function:

1. Give your function a purpose.
2. Create a query for the function in Logs.
3. Save the query as a function.
4. Add Parameters if needed.
5. Name and Save the function.



Follow along in your environment for this next part.

Function 1: WeeklySecurityEvents

Query Code:

```
SecurityEvent
| where TimeGenerated >= ago(7d)
| summarize count() by Activity
```

Example: WeeklySecurityEvents

Save as function ✕

Function name *

WeeklySecurityEvents ✓

Code

SecurityEvent
| where TimeGenerated >= ago(7d)
| summarize count() by Activity

Legacy category *

Threat Hunting ✓

☐ Save as computer group ⓘ

Parameters

Type	Name	Default value
<div>Select type ▼</div>	<div>Type name</div>	<div>Type default value</div>

Save

Cancel

Function 2: SearchTables

Query Code:

search TERM
| summarize Count=count() by Table=\$table

* Note the Parameter 'TERM' that is used.

Example: `SearchTables("BadGuy")`

Edit function details

Function name *

SearchTables

Code

```
//.create-or-alter function with (docstring = "Search for TERM (a string) across the whole database and all of its tables/all cells, and summarize/count the number of hits per table. This can be slow to run. NOTE: this function uses the \'search\' operator, which uses the logic of \'has\' - not \'contains\' underneath.",folder = "Utility")  
search TERM
```

Legacy category *

Hunting

☐

 Save as computer group ⓘ

Parameters

Type	Name	Default value
string	TERM	
<div>Select type</div>	<div>Type name</div>	<div>Type default value</div>

Save

Cancel

Function 3: SearchSecurityEvents

Query Code:

SecurityEvent

| where Activity contains TERM
| project TimeGenerated, Account ,Computer, Activity

Example: 1 SearchSecurityEvents("Failed")

Function name *

SearchSecurityEvents

Code

SecurityEvent
| where Activity contains TERM
| project TimeGenerated, Account, AccountType, Computer, EventSourceName, Channel, Type , EventID, Activity, SourceComputerId, AuthenticationPackageName, FailureReason, IpAddress, IpPort, LogonProcessName, LogonTypeName, SubjectUserSid,

Legacy category *

Utility

☐

 Save as computer group ⓘ

Parameters

Type	Name	Default value
string	TERM	

Select type

Type name

Type default value

Save

Cancel

Function 4: FindNewProcessCount

Query Code:

```
search in (SecurityEvent) EventID == 4688
| summarize ExecutionCount = count() by NewProcessName
```

Example: 1 FindNewProcessCount

Function name *

FindNewProcessCount

Code

search in (SecurityEvent) EventID == 4688
| summarize ExecutionCount = count() by NewProcessName

Legacy category *

Threat Hunting ✓

☐ Save as computer group ⓘ

Parameters

Type	Name	Default value
Select type ▼	Type name	Type default value

Save

Cancel

Function 4: SearchSecurityAlerts

Query Code:

SecurityAlert
| where AlertSeverity has TERM

* Note the Parameter 'TERM' that is used.

Example: `SearchSecurityAlerts("Medium")`

Function name *

SearchSecurityAlerts ✓

Code

SecurityAlert
| where AlertSeverity has TERM

Legacy category *

Hunting ✓

☐ Save as computer group ⓘ

Parameters

Type	Name	Default value
string ✓	TERM ✓	Type default value
Select type ✓	Type name	Type default value

Save

Cancel

Function 4: FindEventID

Query Code:

```
SecurityEvent  
| where Activity contains TERM  
| distinct Activity
```

* Note the Parameter 'TERM' that is used.

Example: `1 FindEventID("fail")`

Function name *

FindEventId ✓

Code

SecurityAlert
| where Activity contains TERM
| distinct Activity

Legacy category *

Utility ✓

☐ Save as computer group ⓘ

Parameters

Type	Name	Default value
string ✓	TERM ✓	Type default value
Select type ✓	Type name	Type default value

Save

Cancel

Further Reading



1. KQL
 1. [rod-trent/MustLearnKQL: Code included as part of the MustLearnKQL blog series \(github.com\)](#)
 2. [Kusto King | The go to shop for KQL](#)
 3. [KQL Tutorial Series | Straight Basics | EP1 - YouTube](#)
 4. [reprise99/Sentinel-Queries: Collection of KQL queries \(github.com\)](#)
 5. [Sentinel-Queries/Query Pack at main · reprise99/Sentinel-Queries \(github.com\)](#)
 6. [inodee/spl-to-kql: The idea is simply to save some quick notes that will make it easier for Splunk users to leverage KQL \(Kusto\), especially giving projects requiring both technologies \(Splunk and Azure/Sentinel\) or any other hybrid environments. Feel free to add/suggest entries. \(github.com\)](#)

Questions?