Demystifying KQL

A Part of the Kusto Ninja Series

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

- Intro to KQL

Section 2

- Sentinel Logs Overview

Section 3

- Kusto Queries 101

Section 4

- Knowledge Check

Section 5

- Real World Scenarios

Intro to KQL

What is KQL and What does it stand for?

Kusto Query Language

- Performance Based
- Similar to SQL
- Read Only

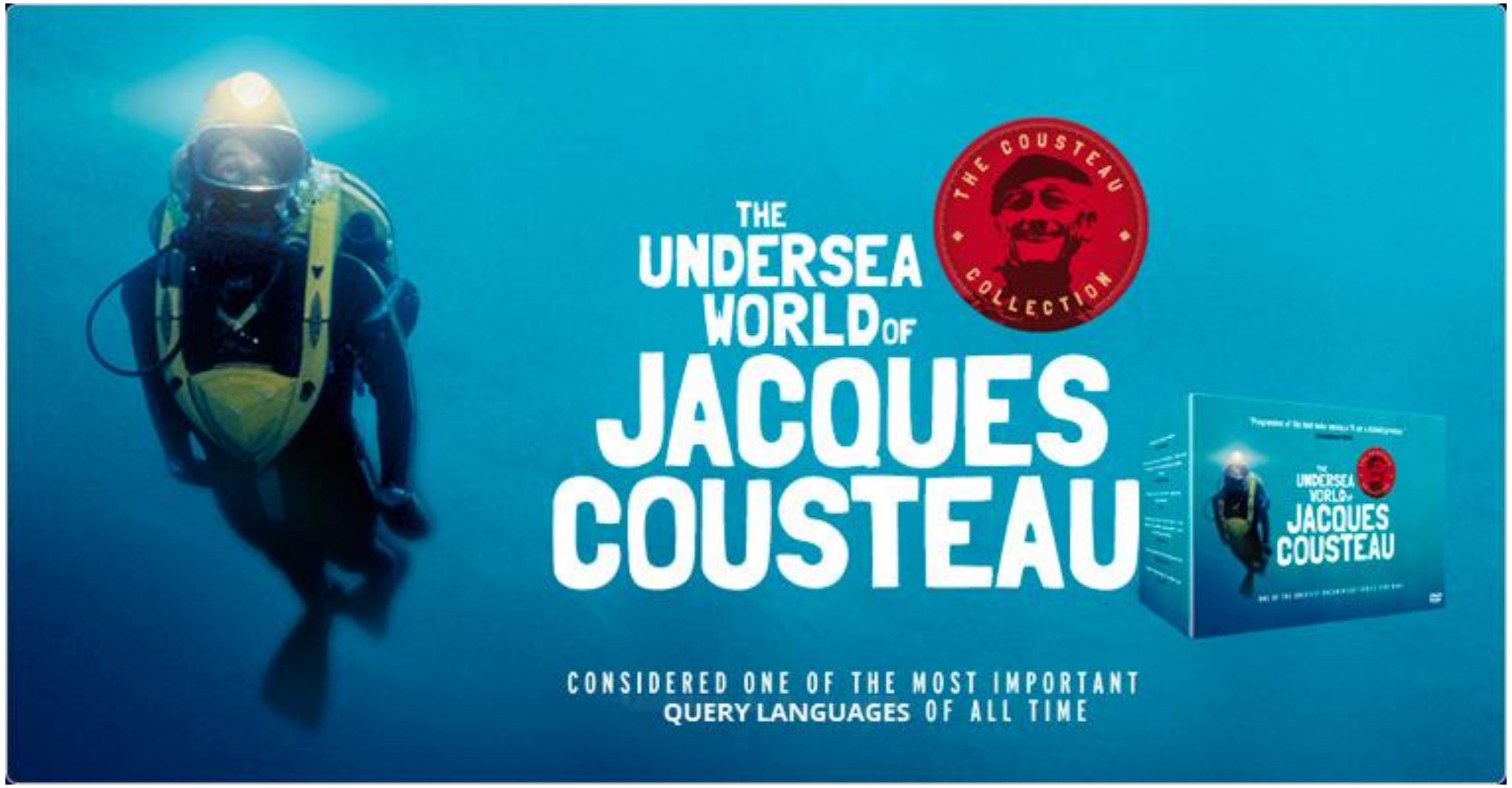
Importance:

- 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.

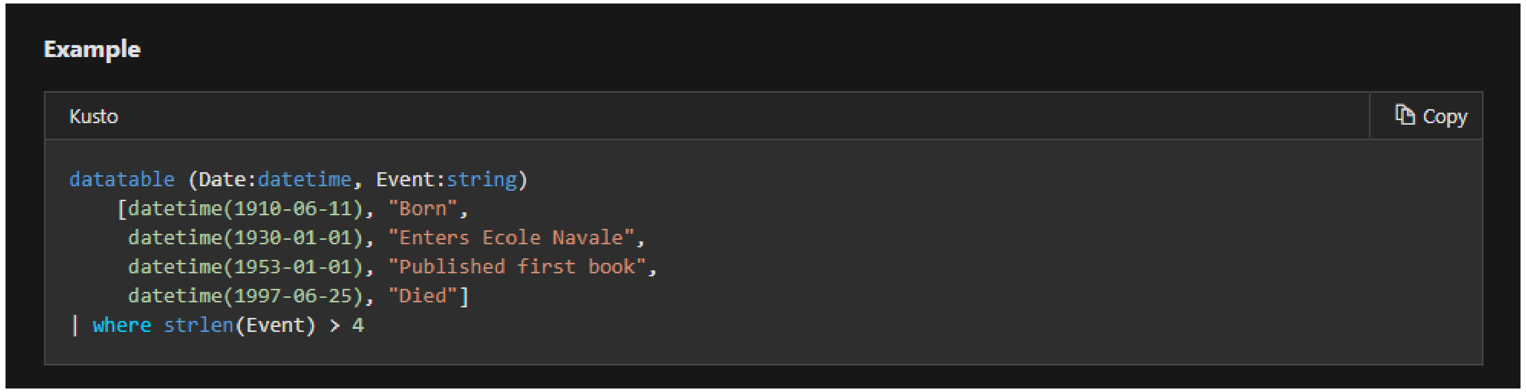
What’s Kusto?

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.


THE COUSTEAU COLLECTION
THE UNDERSEA WORLD OF JACQUES COUSTEAU
CONSIDERED ONE OF THE MOST IMPORTANT
QUERY LANGUAGES OF ALL TIME

Jacques Cousteau

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



```
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 Q*

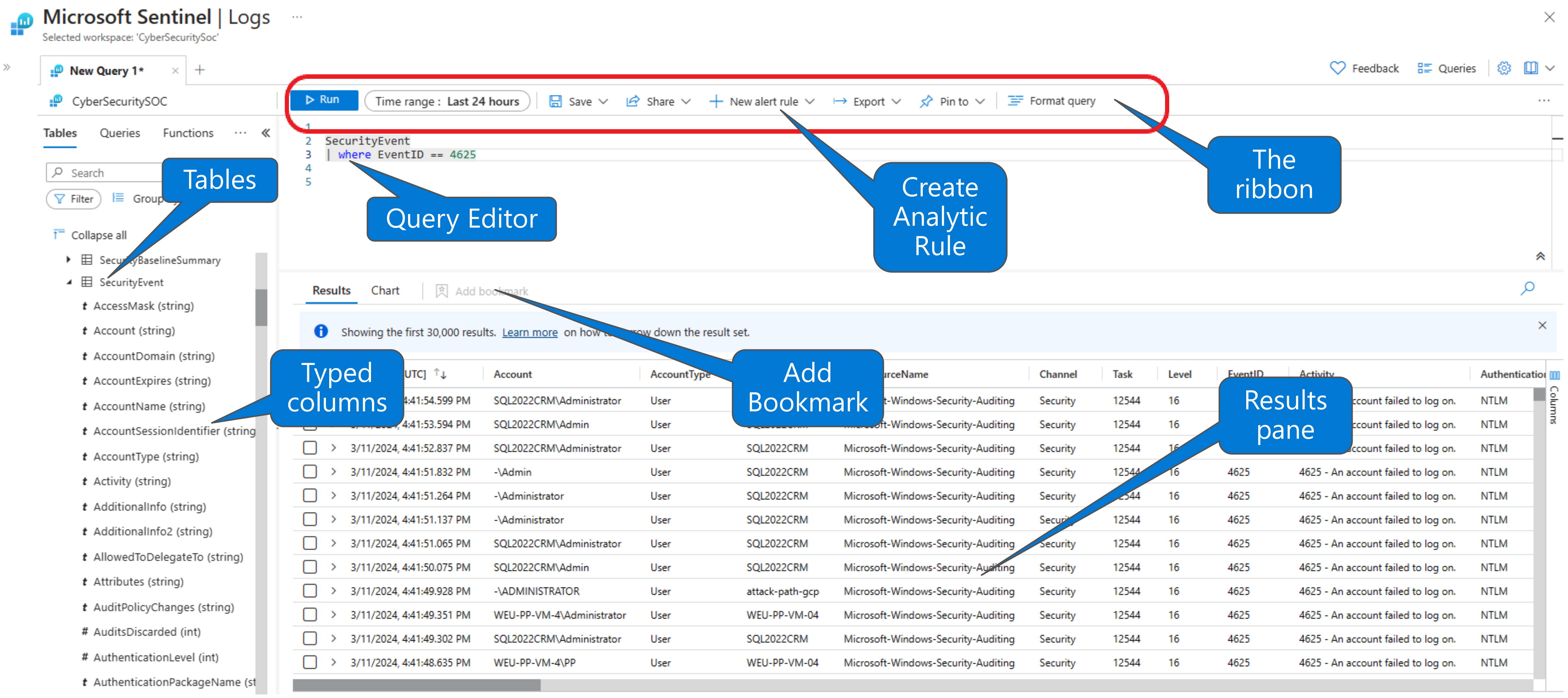
Description	SQL Query	
Select all data from a table	SELECT * FROM dependencies	
Select specific columns from a table	SELECT name, resultCode FROM	
Select 100 records from a table	SELECT T	
Null evaluation		
String comparison: equality		
String comparison: substring		
String comparison: wildcard		
Date comparison: last 1 day		(1d)
Date comparison: date range		ies where timestamp between (datetime(2016-10-01) .. datetime(2016-10-01))
Boolean comparison	SE	dependencies | where success == "False"
Sort	SELE p asc	dependencies | order by timestamp asc
Distinct	SELEC ndencies	dependencies | summarize by name, type
Grouping, Aggregation	SELECT G(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

Sentinel Logs Overview

Sentinel Logs Overview



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

- SecurityBaselineSummary
- SecurityEvent
  - t AccessMask (string)
  - t Account (string)
  - t AccountDomain (string)
  - t AccountExpires (string)
  - t AccountName (string)
  - t AccountSessionIdentifier (string
  - t AccountType (string)
  - t Activity (string)
  - t AdditionalInfo (string)
  - t AdditionalInfo2 (string)
  - t AllowedToDelegateTo (string)
  - t Attributes (string)
  - t AuditPolicyChanges (string)
  - # AuditsDiscarded (int)
  - # AuthenticationLevel (int)
  - t AuthenticationPackageName (st

Showing the first 30,000 results. Learn more on how to narrow down the result set.

UTC]	Account	AccountType		rceName	Channel	Task	Level	EventID	Activity	Authenticatio
4:41:54.599 PM	SQL2022CRM\Administrator	User		t-Windows-Security-Auditing	Security	12544	16		ccount failed to log on.	NTLM
4:41:53.594 PM	SQL2022CRM\Admin	User		ft-Windows-Security-Auditing	Security	12544	16		ccount failed to log on.	NTLM
3/11/2024, 4:41:52.837 PM	SQL2022CRM\Administrator	User	SQL2022CRM	Microsoft-Windows-Security-Auditing	Security	12544	16		ccount failed to log on.	NTLM
3/11/2024, 4:41:51.832 PM	-\Admin	User	SQL2022CRM	Microsoft-Windows-Security-Auditing	Security	12544	16	4625	4625 - An account failed to log on.	NTLM
3/11/2024, 4:41:51.264 PM	-\Administrator	User	SQL2022CRM	Microsoft-Windows-Security-Auditing	Security	12544	16	4625	4625 - An account failed to log on.	NTLM
3/11/2024, 4:41:51.137 PM	-\Administrator	User	SQL2022CRM	Microsoft-Windows-Security-Auditing	Security	12544	16	4625	4625 - An account failed to log on.	NTLM
3/11/2024, 4:41:51.065 PM	SQL2022CRM\Administrator	User	SQL2022CRM	Microsoft-Windows-Security-Auditing	Security	12544	16	4625	4625 - An account failed to log on.	NTLM
3/11/2024, 4:41:50.075 PM	SQL2022CRM\Admin	User	SQL2022CRM	Microsoft-Windows-Security-Auditing	Security	12544	16	4625	4625 - An account failed to log on.	NTLM
3/11/2024, 4:41:49.928 PM	-\ADMINISTRATOR	User	attack-path-gcp	Microsoft-Windows-Security-Auditing	Security	12544	16	4625	4625 - An account failed to log on.	NTLM
3/11/2024, 4:41:49.351 PM	WEU-PP-VM-4\Administrator	User	WEU-PP-VM-04	Microsoft-Windows-Security-Auditing	Security	12544	16	4625	4625 - An account failed to log on.	NTLM
3/11/2024, 4:41:49.302 PM	SQL2022CRM\Administrator	User	SQL2022CRM	Microsoft-Windows-Security-Auditing	Security	12544	16	4625	4625 - An account failed to log on.	NTLM
3/11/2024, 4:41:48.635 PM	WEU-PP-VM-4\PP	User	WEU-PP-VM-04	Microsoft-Windows-Security-Auditing	Security	12544	16	4625	4625 - An account failed to log on.	NTLM

Kusto Queries 101

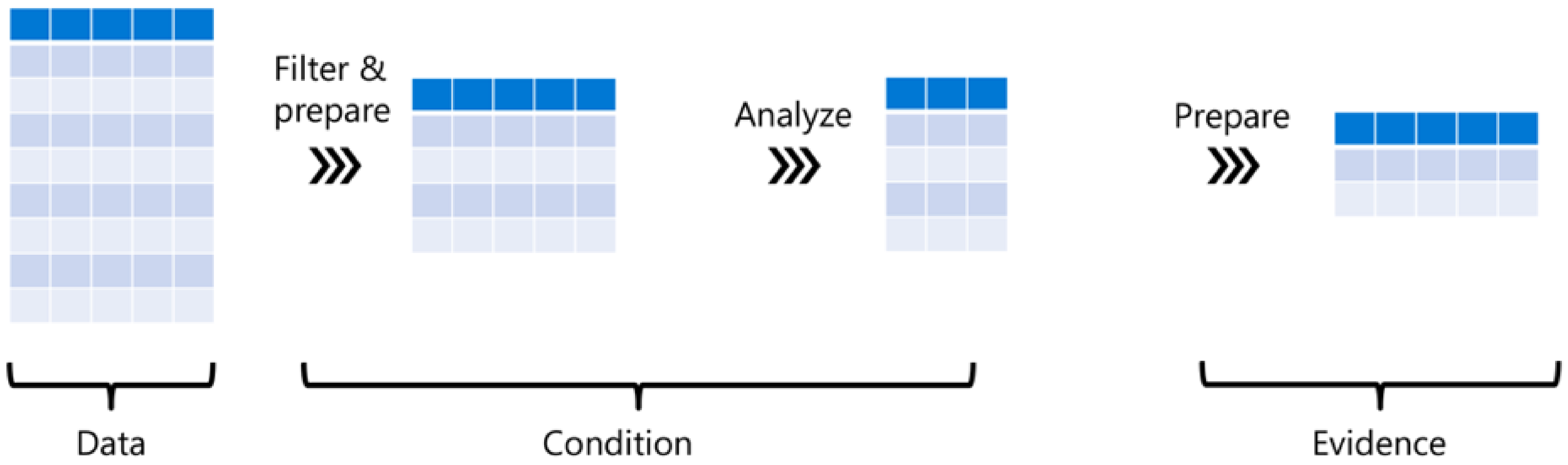
Learn By Doing…

Follow along two different ways…

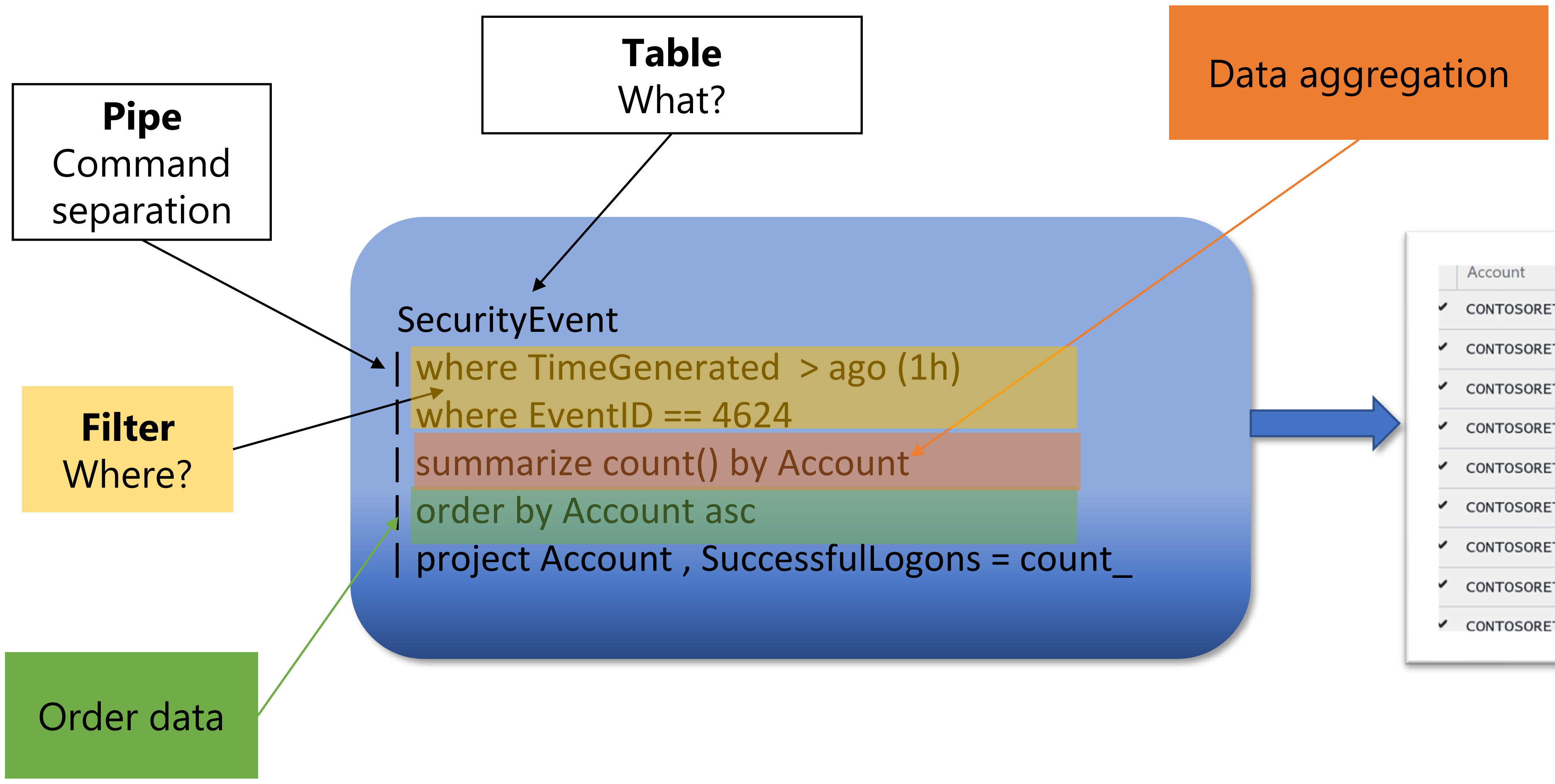
1. Use your own dataset with Microsoft Sentinel
2. Use the Log Analytics Demo Environment at : aka.ms/LADemo

Understanding the flow

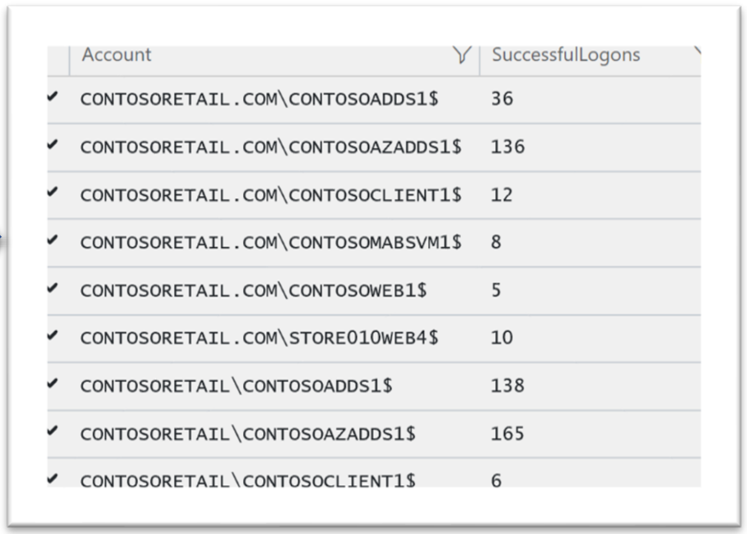
```
SecurityEvent | where EventID == "4264" | summarize count() by Account | top 10 by count_
```



(The Essence) Structure of KQL Queries



```
SecurityEvent
| where TimeGenerated  > ago (1h)
| where EventID == 4624
| summarize count() by Account
| order by Account asc
| project Account , SuccessfulLogons = count_
```



	Account	SuccessfulLogons
✓	CONTOSORETAIL.COM\CONTOSOADDS1$	36
✓	CONTOSORETAIL.COM\CONTOSOAZADDS1$	136
✓	CONTOSORETAIL.COM\CONTOSOCLIENT1$	12
✓	CONTOSORETAIL.COM\CONTOSOMABSVM1$	8
✓	CONTOSORETAIL.COM\CONTOSOWEB1$	5
✓	CONTOSORETAIL.COM\STORE010WEB4$	10
✓	CONTOSORETAIL\CONTOSOADDS1$	138
✓	CONTOSORETAIL\CONTOSOAZADDS1$	165
✓	CONTOSORETAIL\CONTOSOCLIENT1$	6

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

‘**bool**’ data type

The bool data type can have two states: *true* or *false*.

Example:

```
SecurityAlert
| where AlertName contains ‘fail’ and
  IsIncident == true
```

- true and bool(true): Representing trueness
- false and bool(false): Representing falsehood
- bool(null): See null values

Column	Value
Bool(true)	*true*
Bool(false)	*false*

‘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 ‘’ .
- <u>If the string contains a “/” or “\”, it must be preceded by an “@” symbol to show that it’s a string.</u>

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 used with the TimeGenerated column.

Example Usage	**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

Example Columns
TimeGenerated
TimeStamp
LastActivityTime
FirstModifiedTime
CreatedTime

In-Line Commenting

Adding Comments to KQL

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.

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

Example 1:

```
SecurityEvent
//  project Account, Computer
```

Example 2:

```
// This query looks for failed logons
SecurityEvent
| where EventID == 4625
```

Tabular Operators

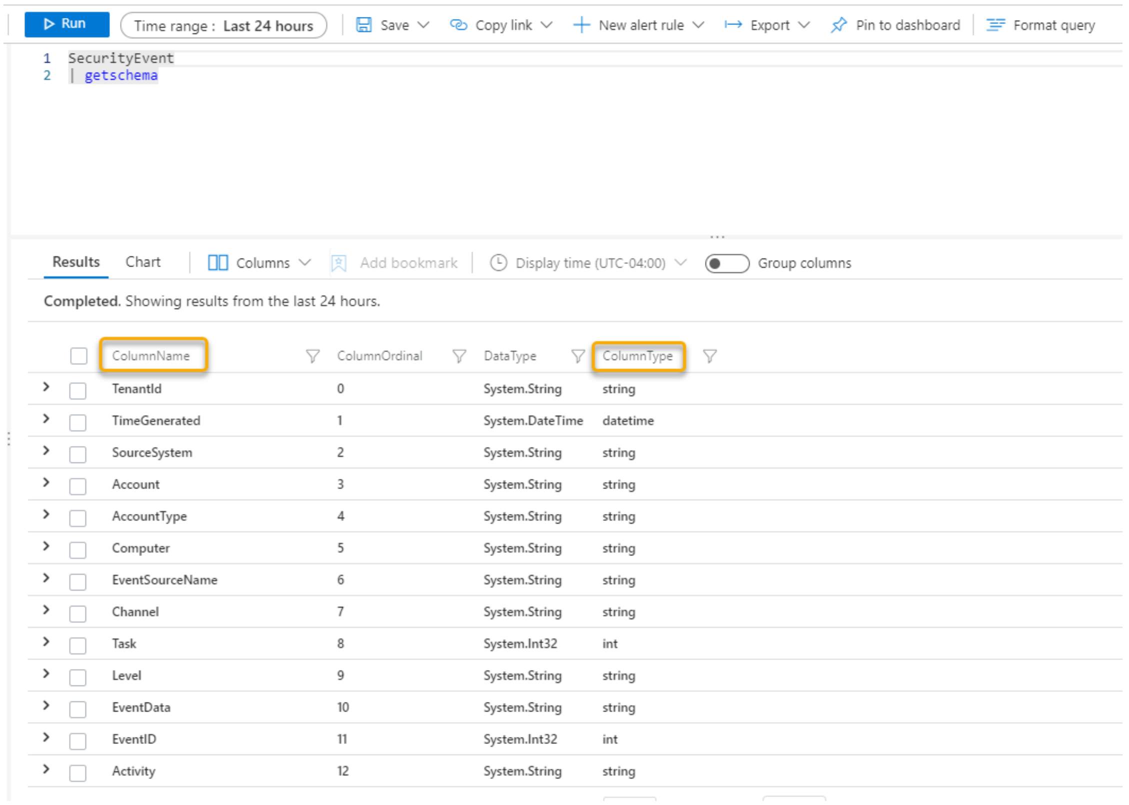
‘**getschema**’ operator

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 for query creation

- This information is also viewable from the “Tables” pane in the UI



```
SecurityEvent
| getschema
```

Completed. Showing results from the last 24 hours.

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

‘search’ operator

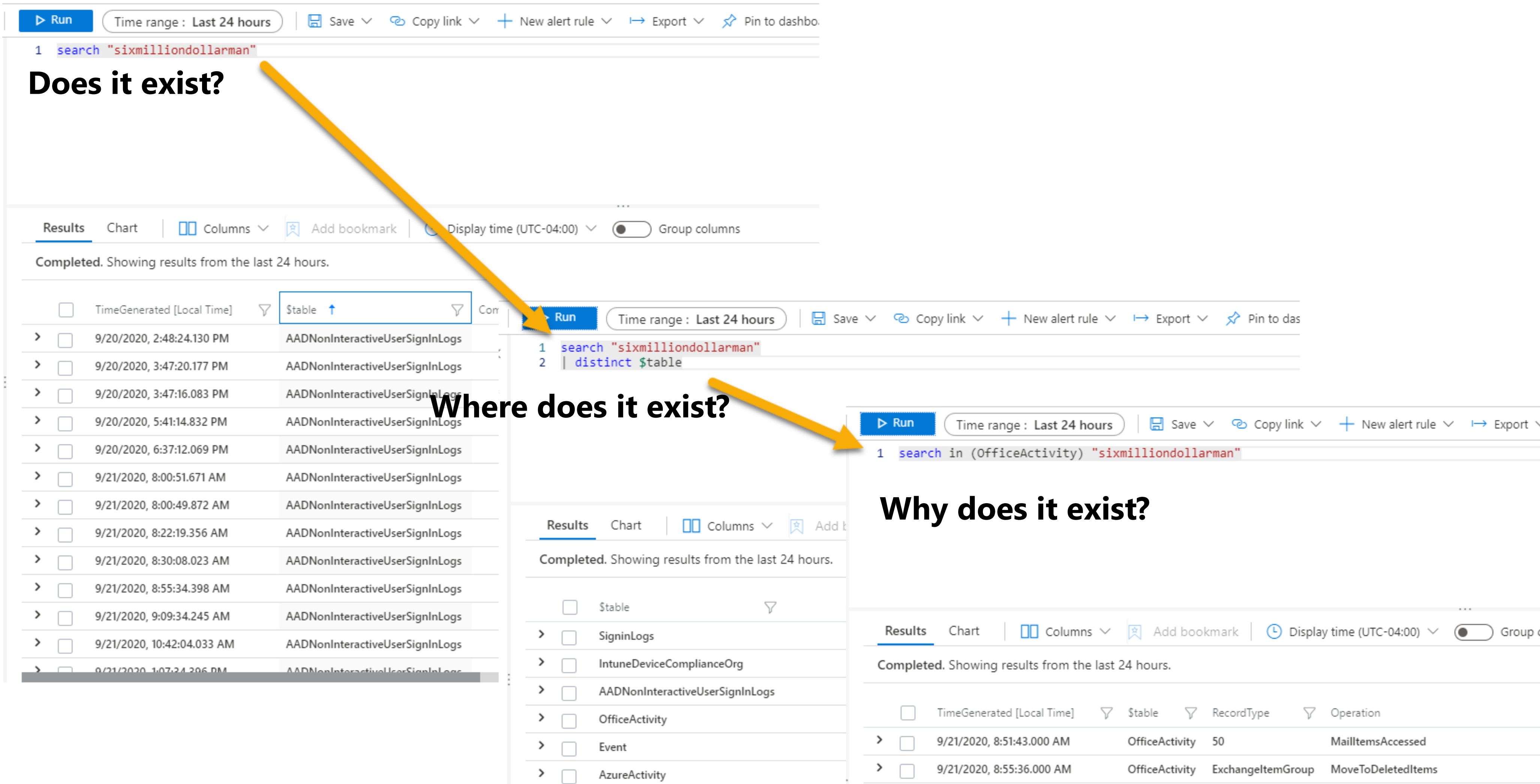
Easy to use. Use interactively for threat hunting but not in analytic rules.
Filter first to avoid long search times.

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 add “@” before the file-path ..i.e.(@”C:/Windows/System32/mimilib.dll”)**

Multi-Table Searches –3 Questions To Ask

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



```
search "sixmilliondollarman"
```

Does it exist?

Completed. Showing results from the last 24 hours.

TimeGenerated [Local Time]	$table
9/20/2020, 2:48:24.130 PM	AADNonInteractiveUserSignInLogs
9/20/2020, 3:47:20.177 PM	AADNonInteractiveUserSignInLogs
9/20/2020, 3:47:16.083 PM	AADNonInteractiveUserSignInLogs
9/20/2020, 5:41:14.832 PM	AADNonInteractiveUserSignInLogs
9/20/2020, 6:37:12.069 PM	AADNonInteractiveUserSignInLogs
9/21/2020, 8:00:51.671 AM	AADNonInteractiveUserSignInLogs
9/21/2020, 8:00:49.872 AM	AADNonInteractiveUserSignInLogs
9/21/2020, 8:22:19.356 AM	AADNonInteractiveUserSignInLogs
9/21/2020, 8:30:08.023 AM	AADNonInteractiveUserSignInLogs
9/21/2020, 8:55:34.398 AM	AADNonInteractiveUserSignInLogs
9/21/2020, 9:09:34.245 AM	AADNonInteractiveUserSignInLogs
9/21/2020, 10:42:04.033 AM	AADNonInteractiveUserSignInLogs

```
search "sixmilliondollarman"
| distinct $table
```

Where does it exist?

Completed. Showing results from the last 24 hours.

$table
SigninLogs
IntuneDeviceComplianceOrg
AADNonInteractiveUserSignInLogs
OfficeActivity
Event
AzureActivity

```
search in (OfficeActivity) "sixmilliondollarman"
```

Why does it exist?

Completed. Showing results from the last 24 hours.

TimeGenerated [Local Time]	$table	RecordType	Operation
9/21/2020, 8:51:43.000 AM	OfficeActivity	50	MailItemsAccessed
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 for Log Analytics.

‘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* | top <*number*> by <*Column*>
Example: *SecurityEvent* | top 10 by Account

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

‘top’ example

```
Security Event
| where TimeGenerated > ago(1h)
| where EventID == 4624
| summarize count() by Account
| top 10 by count_
```

‘**top-hitters**’ operator

Returns an approximation for the most popular distinct values, or the values with the largest sum, in the input.

Syntax: *T* | top-hitters <*NumberOfValues*> *of* <*ValueExpression*> *[by SummingExpression]*

Example: *SecurityEvent* | top-hitters 10 of Account by EventID

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

‘top-hitters’ example

```
WindowsFirewall
| top-hitters 10 of SourcePort
```

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

’where’ example

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

Filtering Data - Exclude Filters

Using the “!” Symbol

Putting a “!” symbol before an operator or filter function will exclude a specific string or integer from the result set.

Example 1:

```
// Excludes any results where the EventID is not 4624
SecurityEvent
| where EventID != 4624
```

Example 2:

```
// Excludes any results where account name does not contain ‘svc’
SecurityEvent
| where Account !contains ‘svc’
```

Exclude Filter Examples

Example Usage	Value
Not Equal to	!=
Does Not Contain	!contains
Not In List	!in
Does Not Start With	!startswith
Does Noy End With	!endswith
Does Not Have	!has

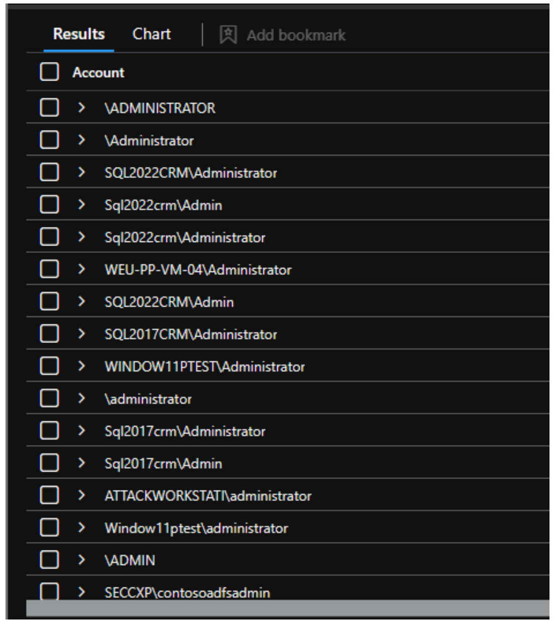
Filtering Data – Contains vs Has Filtering

‘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’



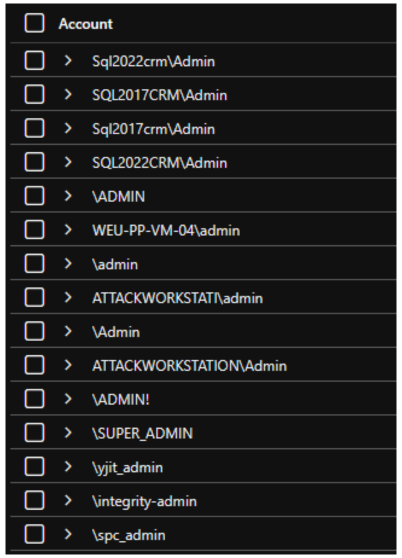
```
Account
\ADMINISTRATOR
\Administrator
SQL2022CRM\Administrator
Sql2022crm\Admin
Sql2022crm\Administrator
WEU-PP-VM-04\Administrator
SQL2022CRM\Admin
SQL2017CRM\Administrator
WINDOW11PTEST\Administrator
\administrator
Sql2017crm\Administrator
Sql2017crm\Admin
ATTACKWORKSTATI\administrator
Window11ptest\administrator
\ADMIN
SECCXP\contosoadfsadmin
```

‘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’



```
Account
Sql2022crm\Admin
SQL2017CRM\Admin
Sql2017crm\Admin
SQL2022CRM\Admin
\ADMIN
WEU-PP-VM-04\admin
\admin
ATTACKWORKSTATI\admin
\Admin
ATTACKWORKSTATION\Admin
\ADMIN!
\SUPER_ADMIN
\yjit_admin
\integrity-admin
\spc_admin
```

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 TargetAccount
```

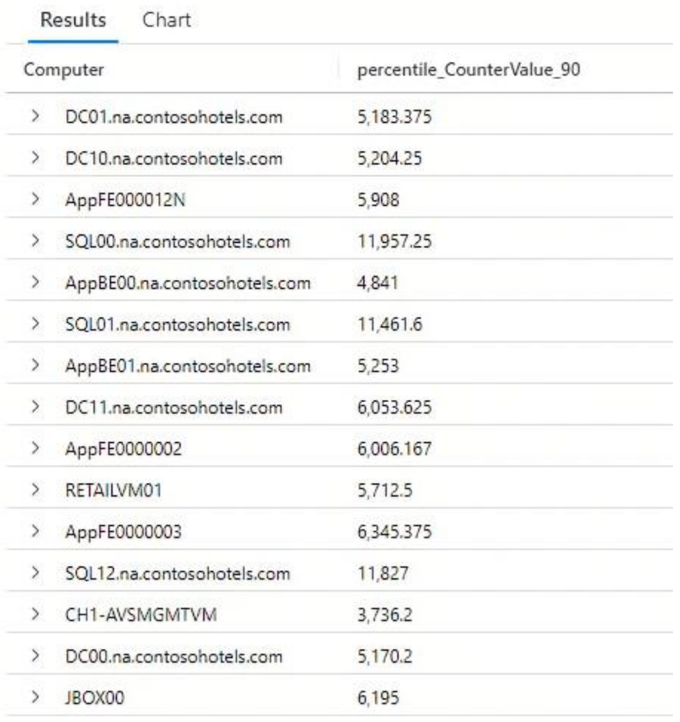
'percentile' function

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
```



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

Always used with the summarize operator

‘percentile’ example

```
Perf
| where CounterName == “% Processor Time”
| summarize percentile(CounterValue, 90) by Computer
```

‘count’ operator

Returns the number of records for the result set.

Syntax:	*T* | *count*
Example:	*SecurityEvent* | *count*

‘count’ example

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

‘dcount’ operator

Returns the number of distinct or unique records for the result set.

Syntax:	*T | summarize dcount(Predicate)*
Example:	*SecurityEvent | summarize dcount(Account)*

‘dcount’ example

```
SecurityEvent
| where TimeGenerated > ago(1h)
| summarize dcount(IpAddress)
```

’countif()’ function

Counts the rows in which predicate evaluates to true.

Syntax: *T | summarize countif(predicate)*

Examples:

```
SecurityEvent
| summarize countif(Account == ‘admin’)
```

‘countif()’ example

```
SecurityEvent
| summarize
             System = countif(Account == @‘NT AUTHORITY\SYSTEM’),
             UserAcct = countif(Account == @‘CH1-VM-CTS\chVmAdminUser’)
             by bin(TimeGenerated, 1h)
|  render timechart
```

Presenting Data

‘project’ operator

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

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

Example: *SecurityEvent | project TimeGenerated, Computer*

<u>Additional Project operators:</u>

‘| 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
```

‘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
```

Time Series Analysis

‘bin()’ function

Rounds a datetime or timespan value down to the nearest time unit.

Syntax: *bin(value, roundTo)*
Example: *SecurityEvent | summarize count() by bin(TimeGenerated, 1h)*

- bin() is often used when summarizing data over a period of time, such as creating a time chart
- Useful for analyzing login trends, brute force attacks, processor trends, etc
- Example Time Values: 1h, 5d, 10m (defaults to 1h)
- Can create multiple legends by aggregating additional field

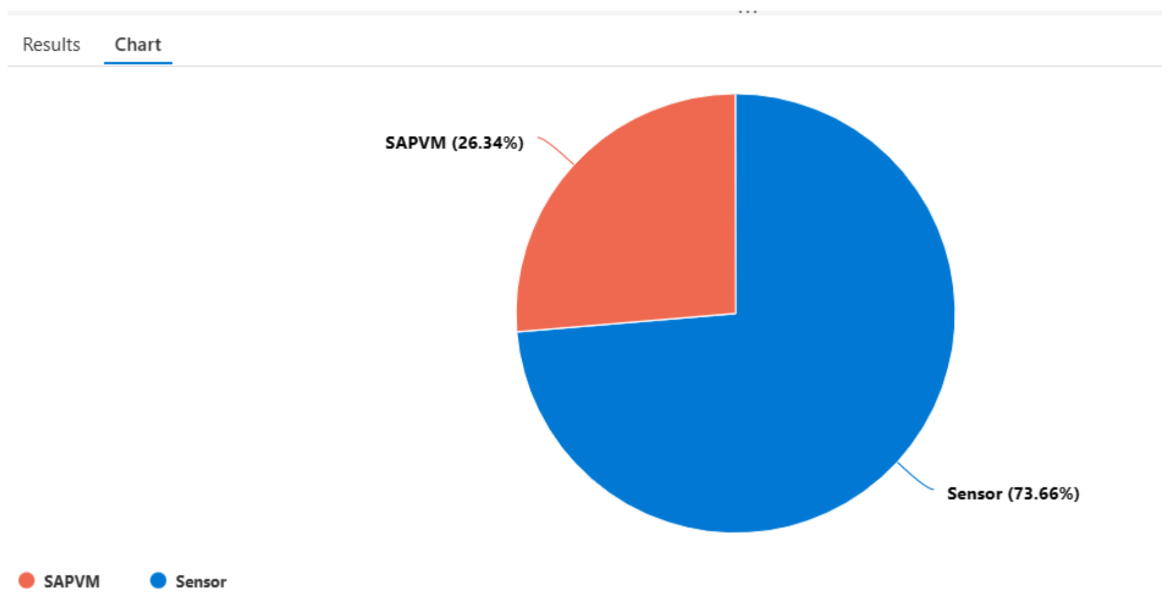
‘render’ operator

Generates a visualization of the query results.

Syntax: *T* | render *Visualization [with (PropertyName = PropertyValue [, ...])]*

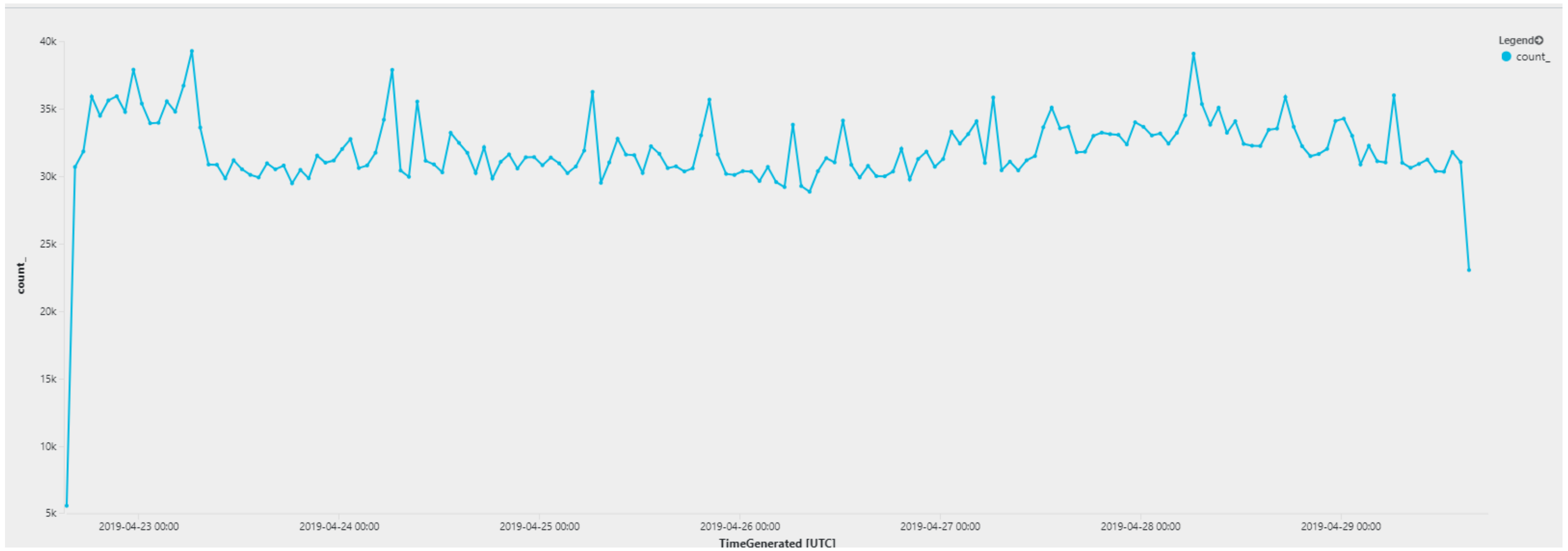
Supported visualizations:

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



‘bin() and render’ example

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



Declaring Variables

‘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*

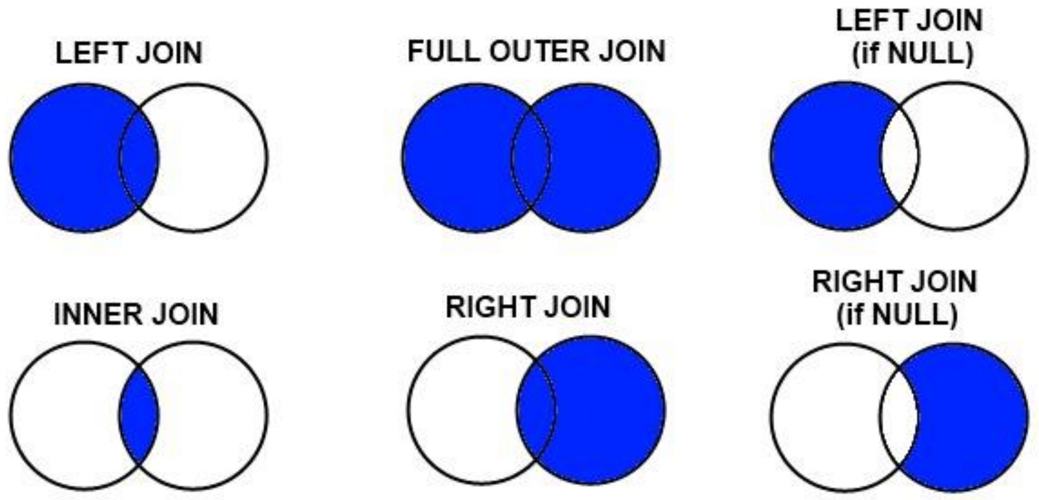


Table1 | join (Table2) on CommonColumn, $left.Col1 == $right.Col2

‘join’ example

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

Let’s Recap..

KQL Syntax Review

- Like SQL…but…Easier to Use!
- 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), bool, string, datetime, timestamp, or dynamic (JSON).
- Operators include ==, !=, contains, has, !in, startswith, <=, >=, and many more.
- The ! operator is used for “not”. (i.e… != is not equal to)
- Queries can be made insensitive by using ‘~’ .
- In-line commenting “\\” are supported.
- Anything in quotes is treated as a string.
- Numbers with no quotes are treated as a long integer.

Top Kusto Operators for SecOps

where – Filter data to look for specific entities

project – Reduce results clutter by only showing required columns

distinct - Remove duplicates from results.

extend – Creates new columns from results.

summarize – Group data into useful sets for aggregation

count – Count # of records in results

dcount – Count # of distinct/unique records in results

search – Search for specific words, phrases, or results

top-hitters - Returns the top 10 results of a query

limit/take - Returns the specified number of rows

union- Merges multiple tables together for a combined result set

let- Creates a temporary variable for holding data

render- Creates visualizations and charts

Pop Quiz Time!

Question 1: Find the Mistakes

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

Question 1: Answer

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

Question 2: Find the Mistakes

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

Question 2: Answer

```
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 3: Answer is 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 4: Answer

```
SecurityEvent
| where TimeGenerated >= ago(7d)
| where EventID == 4624
| Project TimeGenerated, Computer, Activity
```

Question 5: Find the Mistakes

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

Question 5: Answer

```
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 6: Answer

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

False: The correct syntax is: getschema

Question 7: Finish the Query

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

Question 7: Answer

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

Question 8: True or False?

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

Question 8: Answer

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

False: There are two ways to rename a column.

1. project-rename
2. Count = count_

Question 9: True or False?

KQL is short for Kusto Query Language.

Question 9: Answer

KQL is short for Kusto Query Language.

The answer is True .

Question 10: Find the Mistake

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

Question 10: Answer

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

Real World Scenario Labs

1st Scenario

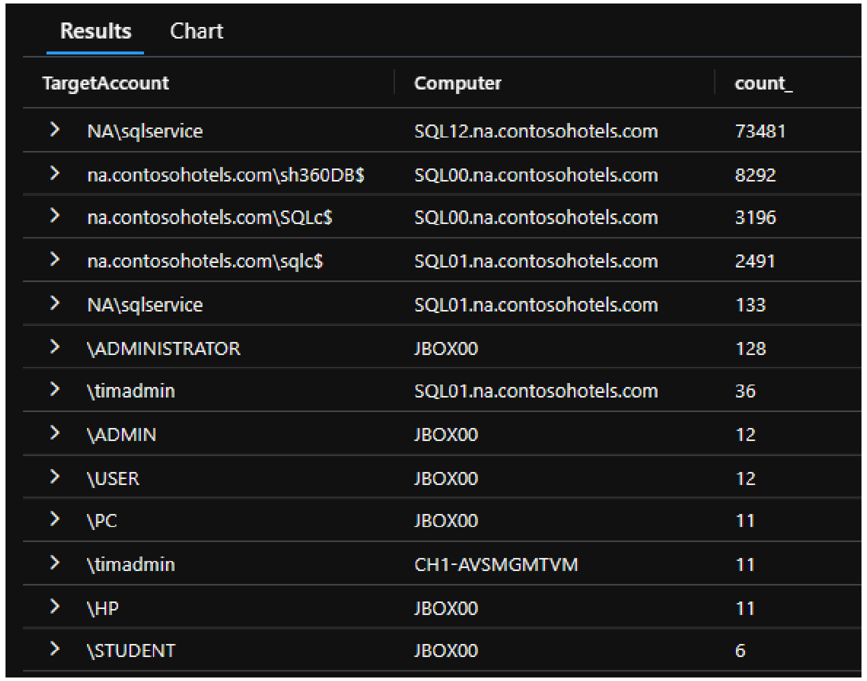
Your SOC has just discovered that a hacker has been using brute force attacks against your network for the past 7 days.
You need to find the count of failed logins for each user account and computer being attacked during that period.

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

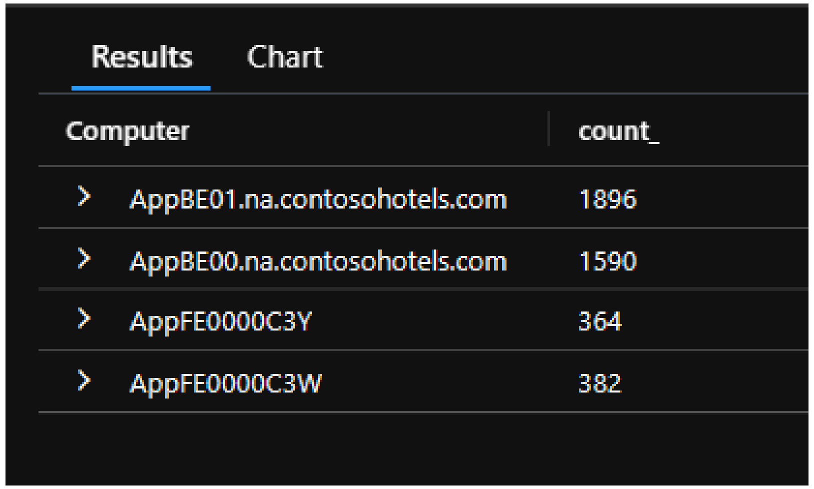
2nd Scenario

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
```



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

3rd Scenario

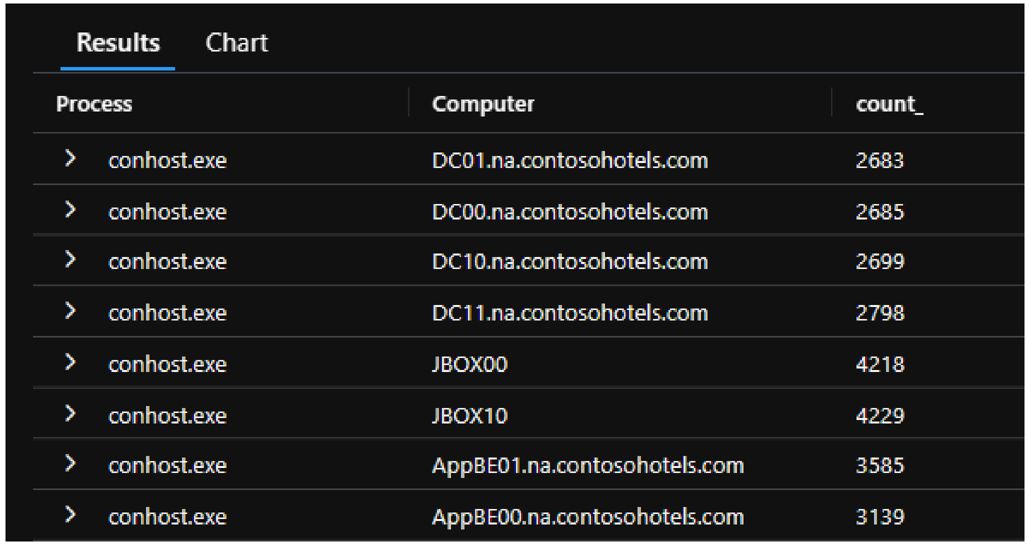
An APT has installed malware on your network .
In order to find the traces of malware, you need to find out how many times each process ran per computer.

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

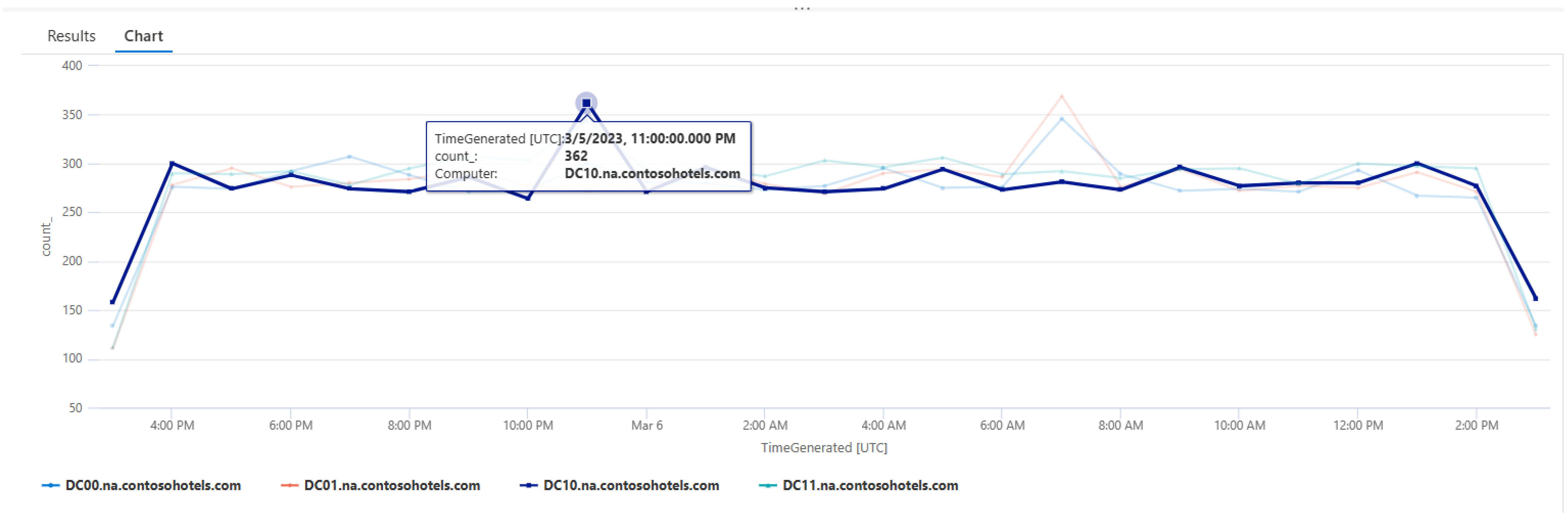
4th Scenario

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
```



```
TimeGenerated [UTC]: 3/5/2023, 11:00:00.000 PM
count_:              362
Computer:            DC10.na.contosohotels.com
```

5th 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.

5th 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
```

6th 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:

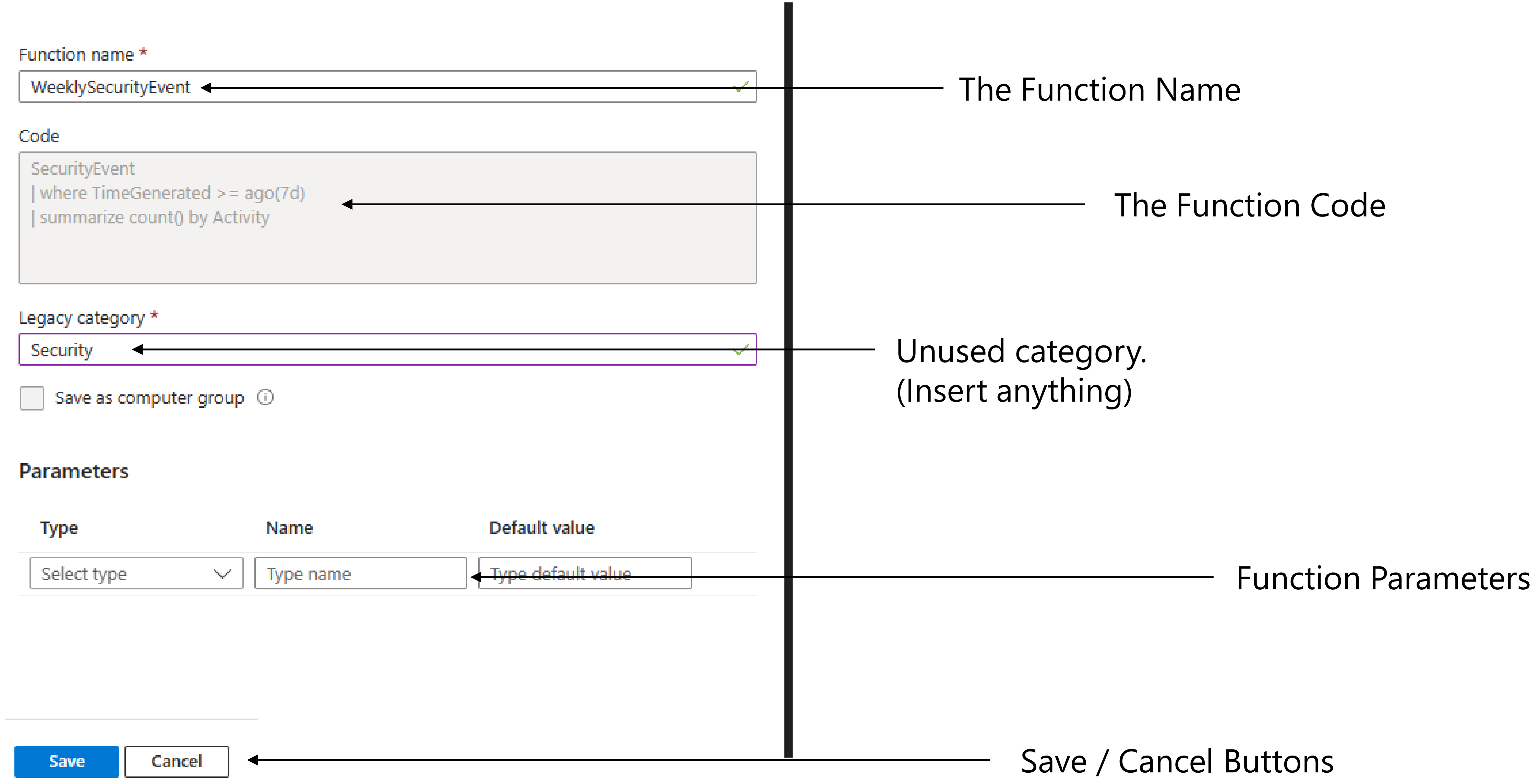
- Which table would you use?
- What way does the data "flow" ?

6th 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
```

Creating Shortcuts with Functions

The Anatomy of a Function:



Function name *

WeeklySecurityEvent

Code

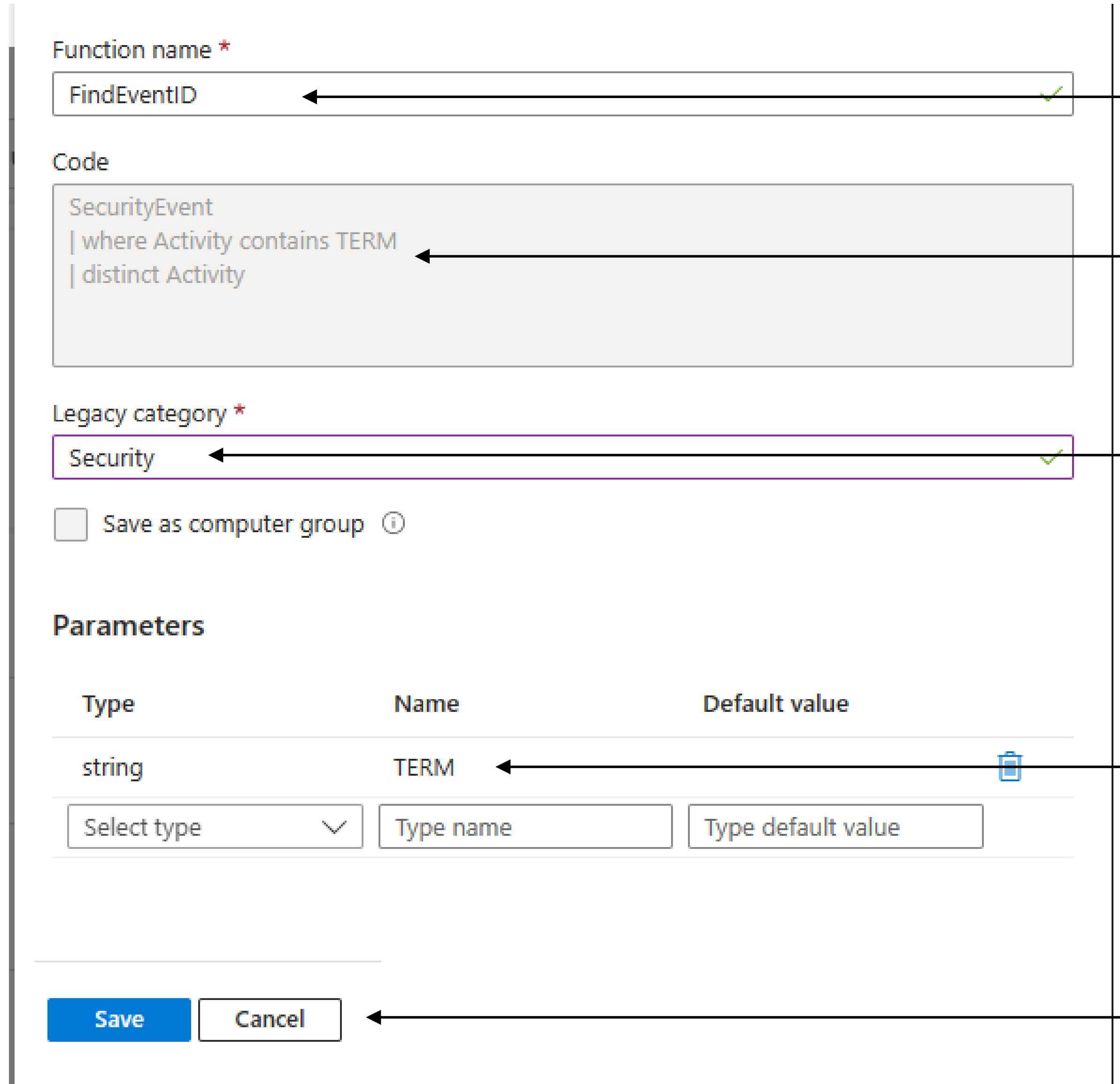
```
SecurityEvent
| where TimeGenerated >= ago(7d)
| summarize count() by Activity
```

Legacy category *

Security

Save as computer group

The Anatomy of a Function with Parameters:



Function name *

FindEventID

Code

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

Legacy category *

Security

☐ Save as computer group

Parameters

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

The Function Name

The Function Code

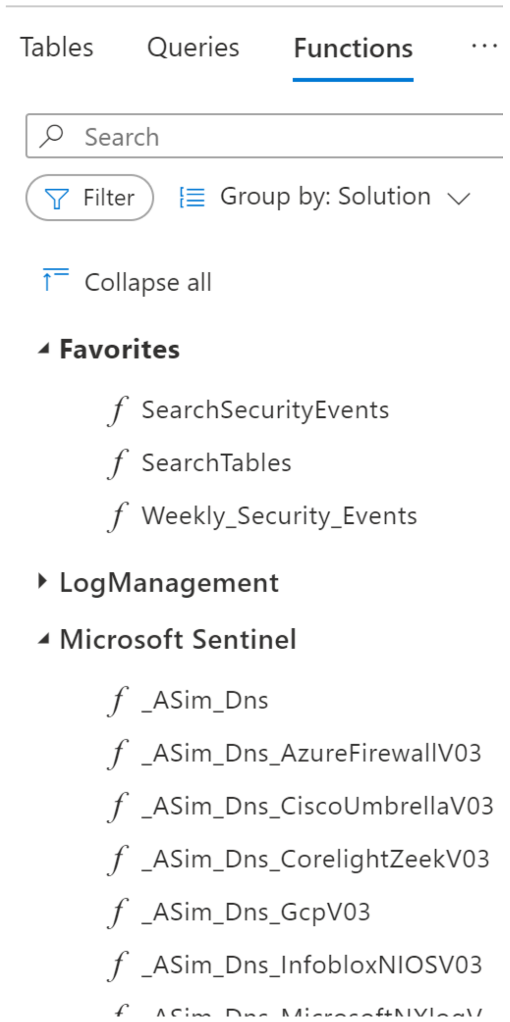
Unused category.
(Insert anything)

Function Parameter
(must match the
PARAM in code)

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.



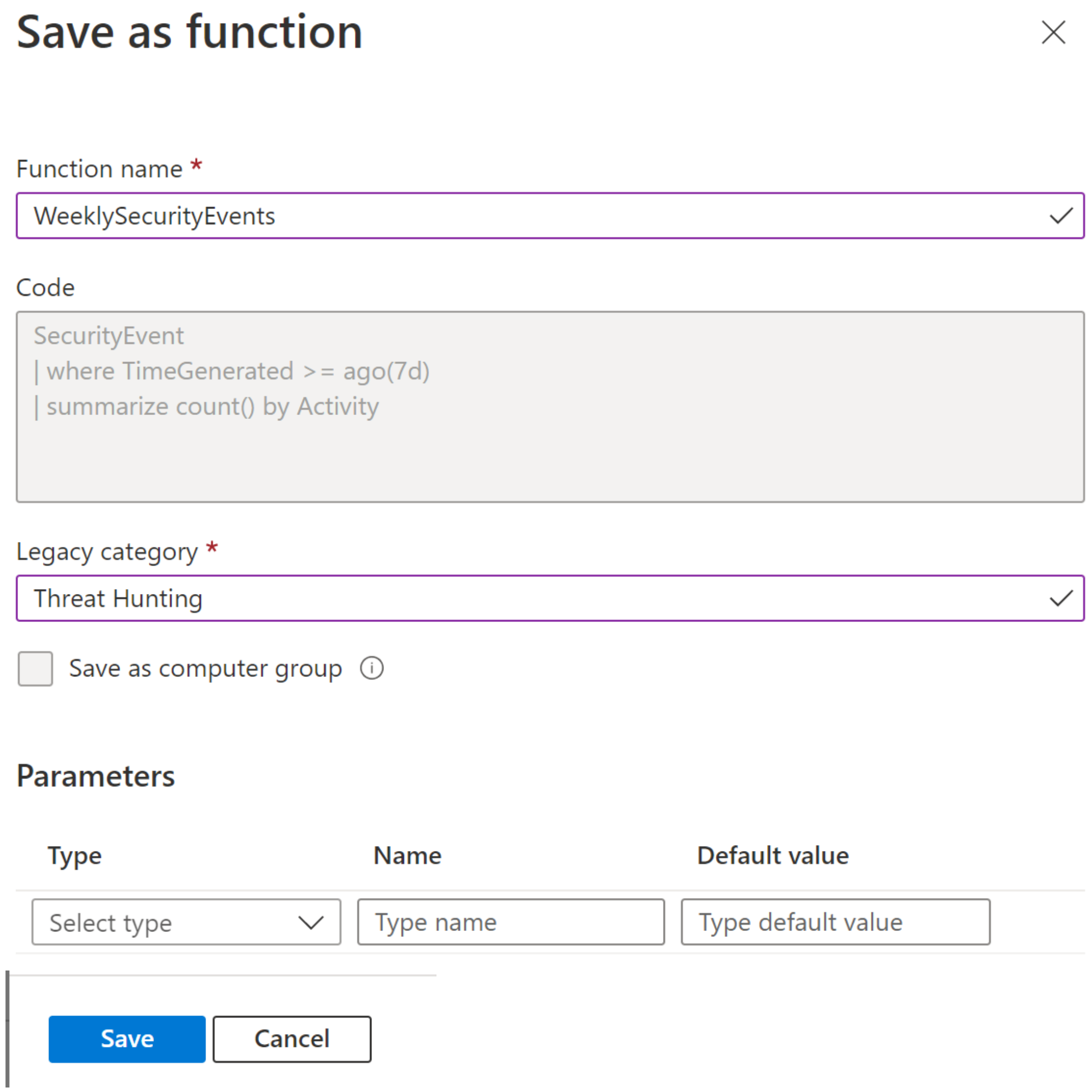
Follow along in your environment for this next part.

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
Select type	Type name	Type default value

Function 2: SearchTables

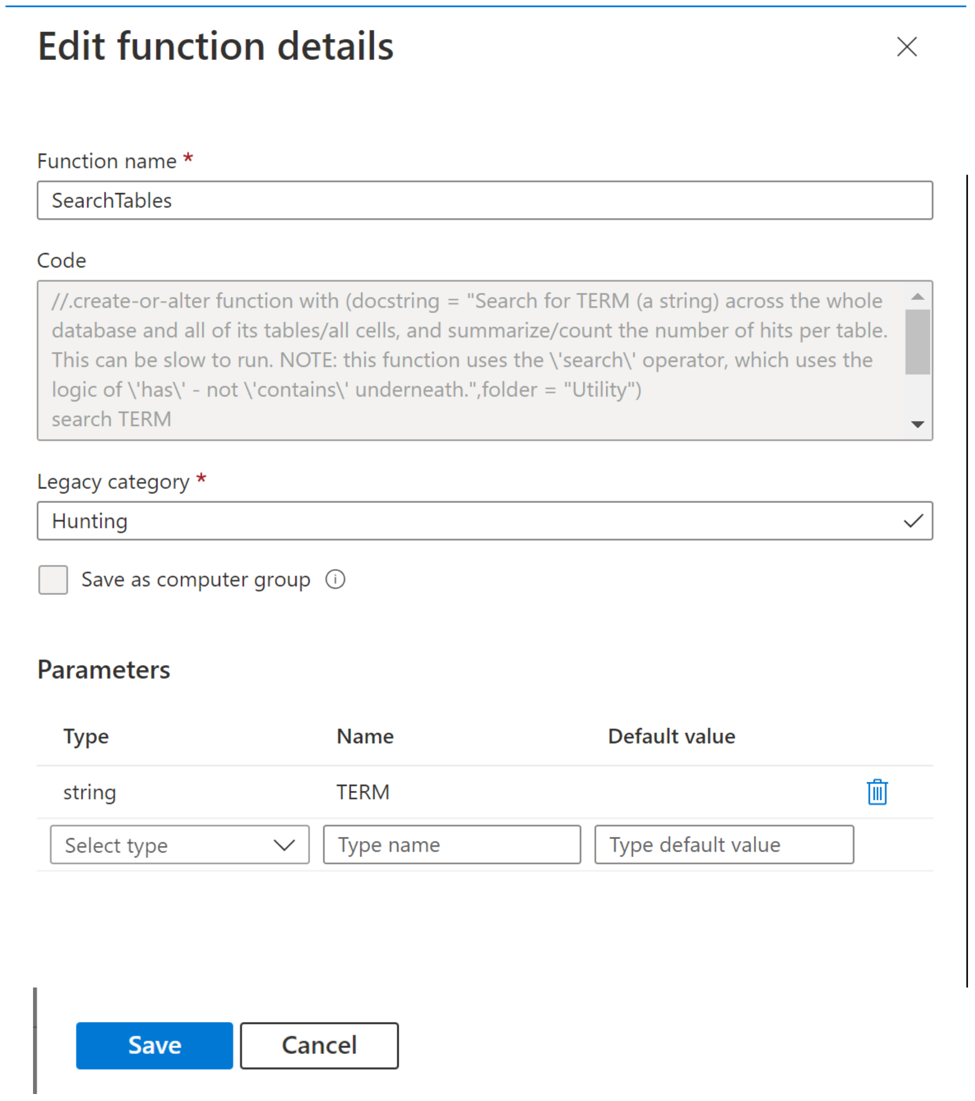
Query Code:

```
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	
Select type	Type name	Type default value

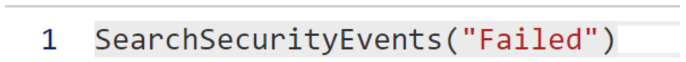
Save | Cancel

Function 3: SearchSecurityEvents

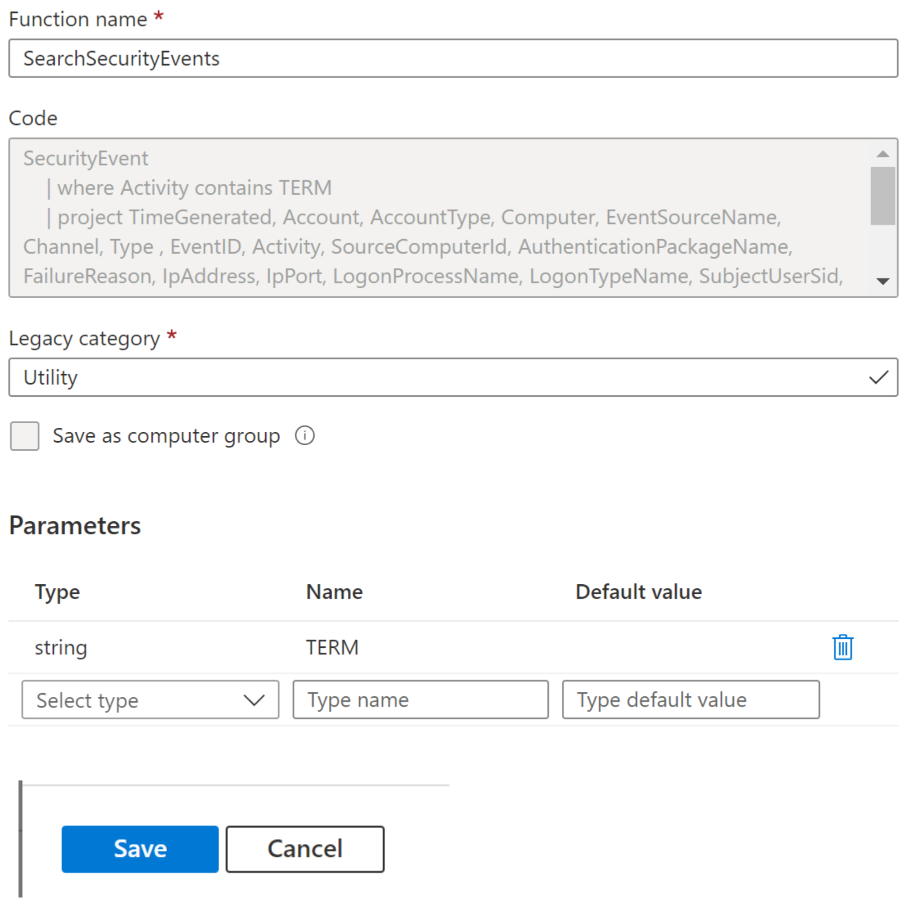
Query Code:

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

Example:



```
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

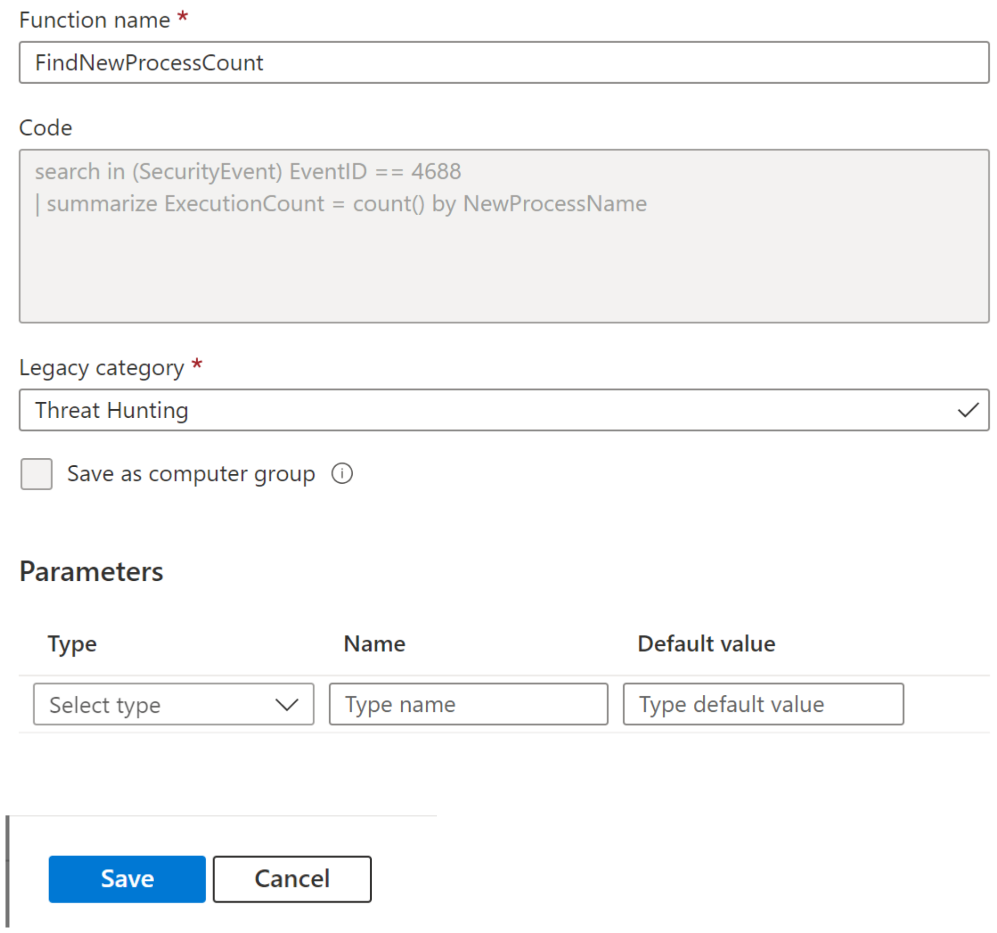
Parameters

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

Function 4: FindNewProcessCount

Query Code:

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



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

Example:

```
FindNewProcessCount
```

Function 5: SearchSecurityAlerts

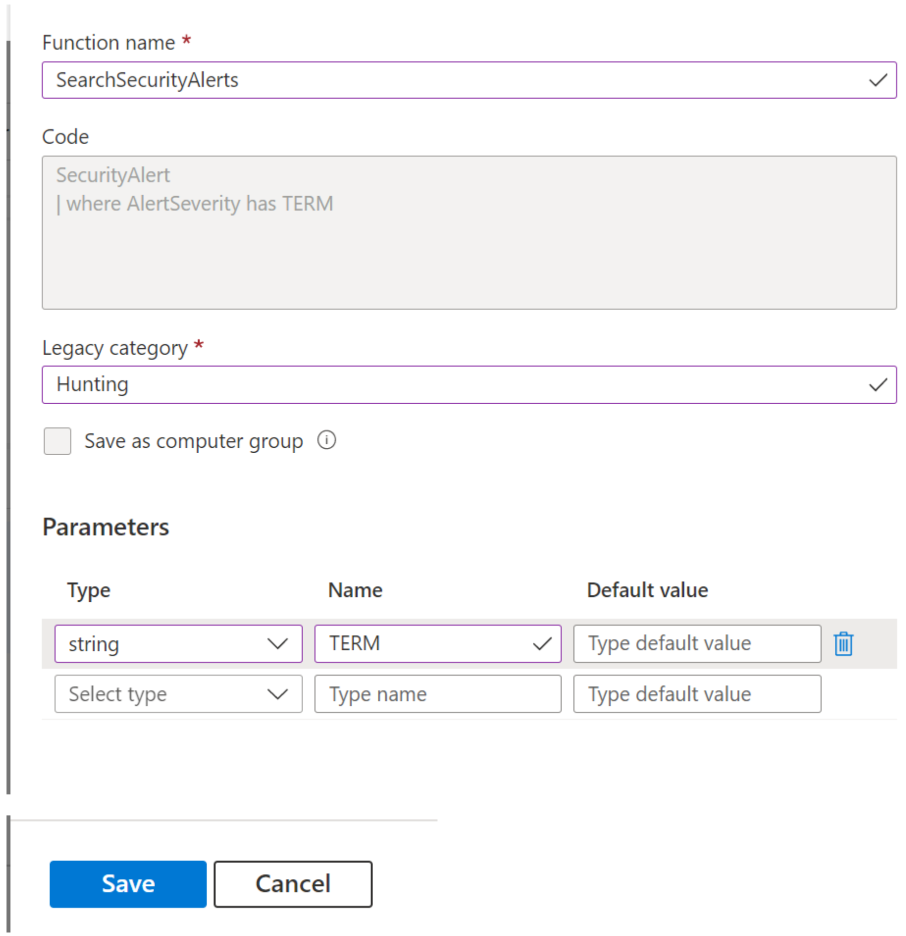
Query Code:

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

Function 6: FindEventID

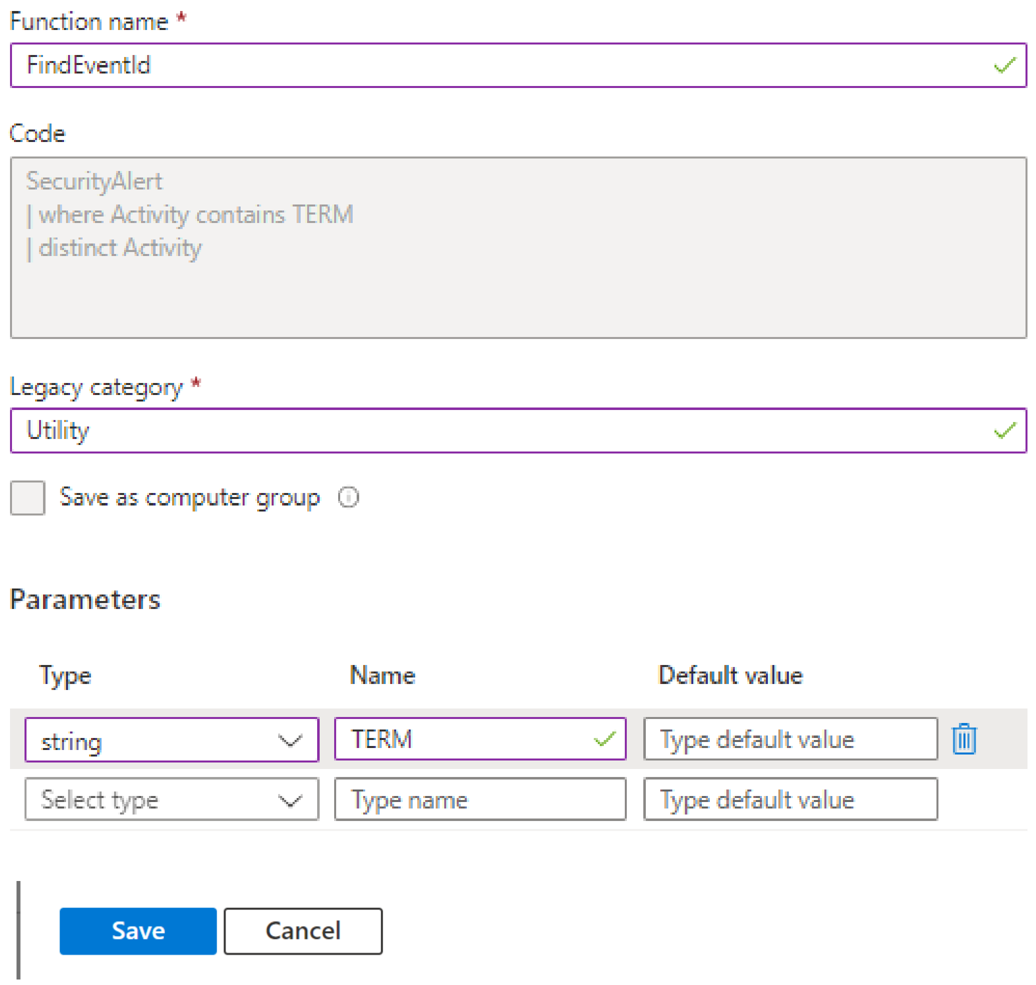
Query Code:

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

* Note the Parameter ‘TERM’ that is used.

Example:

```
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

Resources

Become a Kusto Ninja

- **Must Learn KQL Series** can be found at aka.ms/mustlearnkql
- Take the **KQL Learning Path** at SC-200: Create queries for Microsoft Sentinel using Kusto Query Language
- Become a **Kusto Detective** at the **Kusto Detective Agency** at https://detective.kusto.io/
- **Solve** Mysteries with the **KQL Mysteries Series** at http://aka.ms/KQLMysteries
- Convert **SPL to KQL (for Splunk Users)** using Kusto for Splunkers
- Watch the **KQL Straight Basics Video Series** on YouTube using KQL Tutorial Series | Straight Basics |
- Read the **KQL Reference Documents** at Keyword Query Language (KQL) syntax reference | Microsoft Learn

Questions?