

KQL Intermediate Training

Data Types	Summarize	Arg / Min	Rounding	union
Variables (let)	Bin / Floor	Arg / max	Null/Empty values	join
Rendering Graphs and Charts				

Data Types:

getschema operator : Produce a table that represents a tabular schema of the input. You can list the data type of each column of the table.

The screenshot shows the KQL Editor interface. At the top, there is a toolbar with a 'Run' button, a time range selector set to 'Last 24 hours', and other options like 'Save', 'Share', and 'New'. Below the toolbar, the KQL code is displayed:

```
1 SecurityEvent
2 | getschema
```

Under the 'Results' tab, a table is shown with the following columns: ColumnName, ColumnOrdinal, DataType, and ColumnType. The data is as follows:

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

KQL Data Types

Bool	Dynamic	Int	Real	Timespan
Datetime	Guid	Long	String	Decimal

The bool data type

- The `bool` data type can be: `true (1)`, `false (0)`, or `null`.
- The `bool` and `boolean` data types are equivalent.
- The `bool` data type supports all of the [logical operators](#): equality (`==`), inequality (`!=`), logical-and (`and`), and logical-or (`or`).

IsRisky - bool

The screenshot shows a Kusto query results page. At the top, there are buttons for 'Run' (highlighted in blue), 'Save', 'Share', 'New alert rule', and 'Export'. The time range is set to 'Last 24 hours'. The query results table has four columns: ColumnName, ColumnOrdinal, DataType, and ColumnType. The single row shows 'IsRisky' as the column name, ordinal 36, data type System.SByte, and column type bool.

ColumnName	ColumnOrdinal	DataType	ColumnType
IsRisky	36	System.SByte	bool

List the user who performed risky logins

The screenshot shows a Kusto query results page. The query uses the `SigninLogs` table and filters for `IsRisky == true`, then projects the `UserDisplayName`. The results table has four columns: ColumnName, ColumnOrdinal, DataType, and ColumnType, with one row for the `UserDisplayName`.

The datetime data type

The `datetime` data type represents an instant in time, typically expressed as a date and time of day. Values range from 00:00:00 (midnight), January 1, 0001 Anno Domini (Common Era) through 11:59:59 P.M., December 31, 9999 A.D. (C.E.) in the Gregorian calendar.

Time values are measured in 100-nanosecond units called ticks, and a particular date is the number of ticks since 12:00 midnight, January 1, 0001 A.D. (C.E.) in the `GregorianCalendar` calendar (excluding ticks that would be added by leap seconds). For example, a `ticks` value of 312413760000000000 represents the date, Friday, January 01, 0100 12:00:00 midnight. This is sometimes called "a moment in linear time".

The `datetime` and `date` data types are equivalent.

Note: A `datetime` value in Kusto is always in the UTC time zone. If displaying `datetime` values in other time zones is required, use [`datetime_utc_to_local\(\)`](#) or [`datetime_local_to_utc\(\)`](#).

The screenshot shows the Kusto Query Editor interface. At the top, there are buttons for 'Run' (highlighted in blue), 'Save', 'Share', 'New alert rule', 'Export', and 'Pin to'. Below the toolbar, a code editor displays the following Kusto query:

```

1 SecurityEvent
2 | where TimeGenerated >= datetime(2025-01-07)
3 | take 5
4
5

```

Below the code editor, there are two tabs: 'Results' (selected) and 'Chart'. The 'Results' tab displays a table with the following data:

TimeGenerated [UTC]	Account	AccountType	Computer
> 1/9/2025, 7:09:48.159 AM	NA\SQL01\$	Machine	SQL01.na.contosohotels.com
> 1/9/2025, 7:09:48.271 AM	NA\SQL01\$	Machine	SQL01.na.contosohotels.com
> 1/9/2025, 7:09:48.178 AM	NT AUTHORITY\SYSTEM	User	SQL01.na.contosohotels.com
> 1/9/2025, 7:09:48.276 AM	NT AUTHORITY\SYSTEM	User	SQL01.na.contosohotels.com
> 1/9/2025, 7:09:49.427 AM	NT AUTHORITY\SYSTEM	User	SQL01.na.contosohotels.com

The decimal data type

The `decimal` data type represents a 128-bit wide, decimal number.

Caution: Arithmetic operations involving `decimal` values are significantly slower than operations on `real` data type. If your use case doesn't require very high precision, we recommend using `real`.

The dynamic data type

<https://learn.microsoft.com/en-us/kusto/query/scalar-data-types/dynamic?view=microsoft-fabric>

The screenshot shows the Kusto Query Editor interface. At the top, there are buttons for 'Run' (highlighted in blue), 'Save', 'Share', 'New alert rule', 'Export', and 'Pin to'. Below the toolbar, a code editor displays the following Kusto query:

```

1 print o=dynamic({"a":123, "b":"hello", "c":[1,2,3], "d":{}})
2 | extend a=o.a, b=o.b, c=o.c, d=o.d

```

Below the code editor, there are two tabs: 'Results' (selected) and 'Chart'. The 'Results' tab displays a table with the following data:

a	b	c	d	o
> 123	hello	[1,2,3]	0	{"a":123,"b":"hello","c":[1,2,3],"d":{}}

Another screenshot shows a similar setup with the following query:

```

1 print d=dynamic({"a": datetime(1970-05-11)})

```

The 'Results' tab shows the output:

d
> {"a":"1970-05-11T00:00:00.000000Z"}

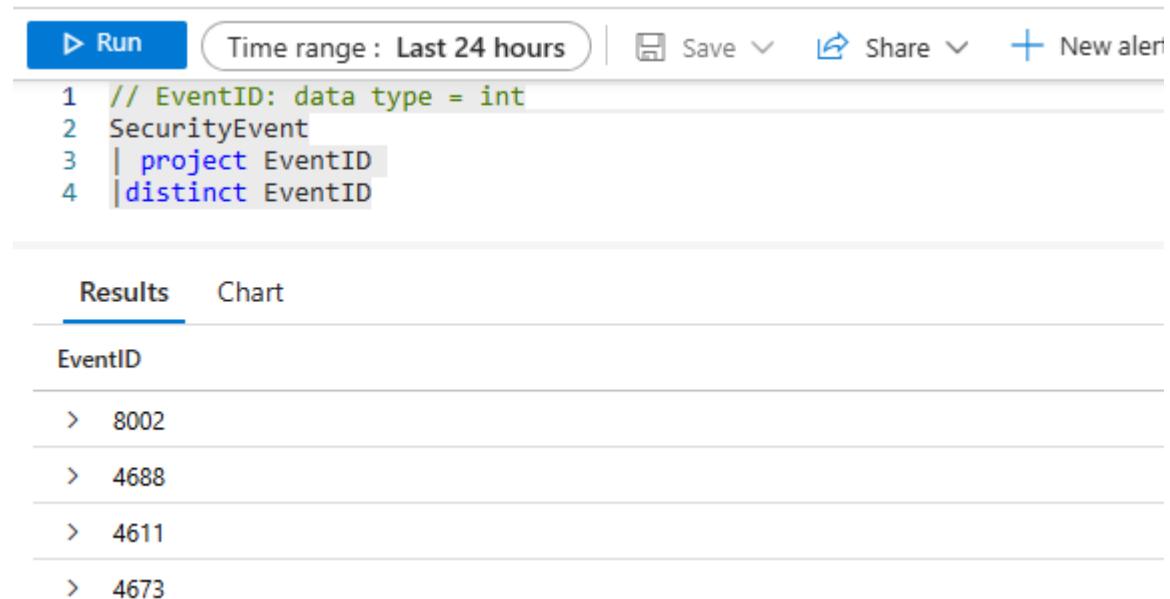
The guid data type

The `guid` data type represents a 128-bit globally unique value. The `guid`, `uuid`, and `uniqueid` data types are equivalent.

guid : 68fg45er-2l5t-67h9-h875-fr0h443g4543

The int data type

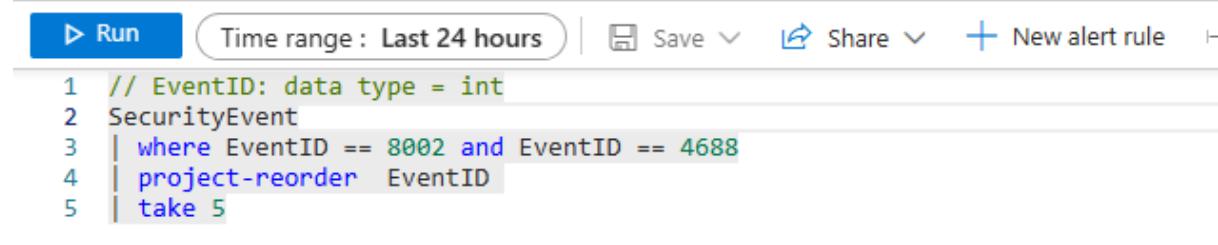
The `int` data type represents a signed, 32-bit wide, integer.



A screenshot of the Kusto Query Editor interface. The top navigation bar includes 'Run' (highlighted in blue), 'Time range: Last 24 hours', 'Save', 'Share', and 'New alert'. Below the editor area, there are two tabs: 'Results' (selected) and 'Chart'. The results section shows a list of EventID values:

```
1 // EventID: data type = int
2 SecurityEvent
3 | project EventID
4 | distinct EventID
```

EventID
> 8002
> 4688
> 4611
> 4673



A screenshot of the Kusto Query Editor interface. The top navigation bar includes 'Run' (highlighted in blue), 'Time range: Last 24 hours', 'Save', 'Share', and 'New alert rule'. Below the editor area, there are two tabs: 'Results' and 'Chart'. The results section shows a list of EventID values filtered by where EventID == 8002 and EventID == 4688:

```
1 // EventID: data type = int
2 SecurityEvent
3 | where EventID == 8002 and EventID == 4688
4 | project-reorder EventID
5 | take 5
```

The long data type

The `long` data type represents a signed, 64-bit wide, integer. By default, integers and integers represented with hexadecimal syntax are of type `long`.

The real data type

The `real` data type represents a 64-bit wide, double-precision, floating-point number. By default, decimal numbers and numbers represented with scientific notation are of type `real`.

The string data type

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

String literal with quotes

A screenshot of a Jupyter Notebook cell. The code is:

```
1 print
2     s1= "string",
3     s2= 'string',
4     s3 = 'string with "double quotes"',
5     s4 = "string with 'single quotes'"
```

The results table shows:

s1	s2	s3	s4
> string	string	string with "double quotes"	string with 'single quotes'

A screenshot of a Jupyter Notebook cell. The code is:

```
1 print " We eat \"apple\" and \"banana\""
```

The results table shows:

print_0
> We eat "apple" and "banana"

String literal with backslash escaping

A screenshot of a Jupyter Notebook cell. The code is:

```
1 print " \\\\"
```

The results pane shows:

print_0

> \\

String literal with Unicode

A screenshot of a Jupyter Notebook cell. The code is:

```
1 print space = "Hello\u00A0World"
```

The results pane shows:

space

> Hello World

Verbatim string literal

The following example creates a path in which the backslashes are part of the path instead of escape characters. To do this, the string @ sign is prepended to the string, creating a [verbatim string literal](#).

A screenshot of a Jupyter Notebook cell. The code is:

```
print myPath = @'C:\Folder\filename.txt'
```

The results pane shows:

myPath

C:\Folder\filename.txt

Multi-line string literal

The following example shows the syntax for a multi-line string literal, which uses newlines and tabs to style a code block. For more information, see [Multi-line string literals](#).

▶ Run Time range : Last 24 hours | Save Share + New alert rule

```
1 print program = ``
2   public class Program {
3     public static void Main() {
4       System.Console.WriteLine("Hello!");
5     }
6   }`
```

Results Chart

program

> public class Program { public static void Main() { System.Console.WriteLine("Hello!"); } }

Concatenated string literals

The following expressions all yield a string of length 13. For more information, see [Concatenation of separated string literals](#).

▶ Run Time range : Last 24 hours | Save Share + New alert rule Export

```
1 print
2 none = strlen("Hello", '@"world!"),
3 whitespace = strlen("Hello ", '@"world!"),
4 whitespaceAndComment = strlen("Hello"
5   // Comment
6   ', '@"world!"`);`
```

Results Chart

none	whitespace	whitespaceAndComment
13	13	13

Obfuscated string literal

In the following query output, the `h` string is visible in your results. However, in tracing or telemetry, the `h` string is stored in an obfuscated form and substituted with asterisks in the log. For more information, see [Obfuscated string literals](#).

The screenshot shows the Azure Log Analytics workspace interface. At the top, there are navigation links: Run, Time range: Last 24 hours, Save, Share, New alert rule, Export, and a favorite icon. Below the header, a code editor displays the following Python-like code:

```
1 print blob="https://contoso.blob.core.windows.net/container/blob.txt?"  
2 h'sv=2012-02-12&se=2013-04-13T0...'
```

Below the code editor, there are two tabs: Results and Chart. The Results tab is selected. The output pane shows the variable `blob` with its value expanded to show the full URL and query parameters.

The timespan data type

The `timespan` data type represents a time interval. The `timespan` and `time` data types are equivalent.

The screenshot shows the Azure Log Analytics workspace interface. At the top, there are navigation links: Run, Time range: Last 24 hours, Save, Share, and New alert. Below the header, a code editor displays the following Python-like code:

```
1 print  
2     result1 = 1d / 1s,  
3     result2 = time(1d) / time(1s),  
4     result3 = 24 * 60 * time(00:01:00) / time(1s)
```

Below the code editor, there are two tabs: Results and Chart. The Results tab is selected. The output pane shows three variables: `result1`, `result2`, and `result3`, each with the value 86400.

Null values

All scalar data types in Kusto have a special value that represents a missing value. This value is called the *null value*, or *null*.

Note: The `string` data type doesn't support null values.

String = empty

Others = null

Null literals

The screenshot shows a Kusto query editor interface. At the top, there are buttons for 'Run' (highlighted in blue), 'Save', 'Share', and 'New'. Below the toolbar, the time range is set to 'Last 24 hours'. The main area contains a code block with the following content:

```
1 print
2     bool(null),
3     datetime(null),
4     dynamic(null),
5     guid(null),
6     int(null),
7     long(null),
8     real(null),
9     double(null),
10    timespan(null)
```

Below the code, there are tabs for 'Results' (which is selected) and 'Chart'. Under the 'Results' tab, the output is shown as:

print_1 [UTC]

>

Predicates on null values

The scalar function `isnull()` can be used to determine if a scalar value is the null value. The corresponding function `isnotnull()` can be used to determine if a scalar value isn't the null value.

Note: Because the `string` type doesn't support null values, we recommend using the [isempty\(\)](#) and the [isnotempty\(\)](#) functions.

Equality and inequality of null values

Equality (==): Applying the equality operator to two null values yields `bool(null)`. Applying the equality operator to a null value and a non-null value yields `bool(false)`.

Inequality (!=): Applying the inequality operator to two null values yields `bool(null)`. Applying the inequality operator to a null value and a non-null value yields `bool(true)`.

For example:

```
datatable(val:int)[5, int(null)]
| extend IsBiggerThan3 = val > 3
| extend IsBiggerThan3OrNull = val > 3 or isnull(val)
| extend isEqualToNull = val == int(null)
| extend IsNotEqualToNull = val != int(null)
```

val	IsBiggerThan3	IsBiggerThan3OrNull	isEqualToString	IsNotEqualToNull
5	true	true	false	true
null	null	true	null	null

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

```
1 datatable(val:int)[5, int(null)]
2 | extend IsBiggerThan3 = val > 3
3 | extend IsBiggerThan3OrNull = val > 3 or isnull(val)
4 | extend isEqualToString = val == int(null)
5 | extend IsNotEqualToNull = val != int(null)
```

Results Chart

val	IsBiggerThan3	IsBiggerThan3OrNull	isEqualToString	IsNotEqualToNull
> 5	true	true	false	true
>		true		

Null values and the where operator

Run Time range : Last 24 hours Save Share New alert rule

```
1 datatable(ival:int, sval:string)[5, "a", int(null), "b"]
2 | where isnull(ival)
```

Results Chart

ival	sval
>	b

Run Time range : Last 24 hours Save Share New alert rule

```
1 datatable(ival:int, sval:string)[5, "a", int(null), "b"]
2 | where isnotnull(ival)
```

Results Chart

ival	sval
> 5	a

Null values and binary operators

Exceptions to this rule

- For the equality (==) and inequality (!=) operators, if one of the values is null and the other value isn't null, then the result is either `bool(false)` or `bool(true)`, respectively.
- For the logical AND (&&) operator, if one of the values is `bool(false)`, the result is also `bool(false)`.
- For the logical OR (||) operator, if one of the values is `bool(true)`, the result is also `bool(true)`.

For example:

```
datatable(val:int)[5, int(null)]
| extend Add = val + 10
| extend Multiply = val * 10
```

val	Add	Multiply
5	15	50
null	null	null

Null values and the logical NOT (!) operator

The logical NOT operator `not()` yields the value `bool(null)` if the argument is the null value.

Null values and the in operator

- The `in operator` behaves like a logical OR of equality comparisons.
- The `!in` operator behaves like a logical AND of inequality comparisons.

Null values and data ingestion

For most data types, a missing value in the data source produces a null value in the corresponding table cell. However, columns of type `string` and CSV (or CSV-like) data formats are an exception to this rule, and a missing value produces an empty string.

Kusto

```
.create table T(a:string, b:int)

.ingest inline into table T
[ , ]
[ , ]
[a,1]

T
| project a, b, isnull_a=isnull(a), isempty_a=isempty(a), strlen_a=strlen(a), isnull_b=isnull(b)
```

a	b	isnull_a	isempty_a	strlen_a	isnull_b
		false	true	0	true
		false	false	1	true
a	1	false	false	1	false

Exercise: LA Demo Environment , SecurityEvent

- 1- Identify datatype of EventID field
- 2- Cast EventID field to String
- 3- Verify the change by checking the schema.

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

```
1 // 1.
2 SecurityEvent
3 |distinct EventID
4 |getschema
5
6 // 2.
7 SecurityEvent
8 |extend EventIDstr = tostring(EventID)
9 |project EventIDstr, EventID
10
11
12 // 3.
13 SecurityEvent
14 |extend EventIDstr = tostring(EventID)
15 |project EventIDstr, EventID
16 |getschema
17
18
```

Results Chart

ColumnName	ColumnOrdinal	DataType	ColumnType
> EventIDstr	0	System.String	string
> EventID	1	System.Int32	int

summarize(): Produces a table that aggregates the content of the input table.

summarize by counting SecurityEvents by EventID

The screenshot shows the Kusto Query Editor interface. At the top, there is a toolbar with a 'Run' button, a time range selector set to 'Last 24 hours', a 'Save' dropdown, and a 'Share' button. Below the toolbar, the query pane contains the following code:

```
1 SecurityEvent
2 | project TimeGenerated, EventID, ProcessId
3 | summarize count() by EventID
```

Under the 'Results' tab, the output table has two columns: 'EventID' and 'count_'. The data rows are:

EventID	count_
> 4688	292724
> 4673	51850
> 8002	309162
> 4670	15111

You can give a name to count result

The screenshot shows the Kusto Query Editor interface. At the top, there is a toolbar with a 'Run' button, a time range selector set to 'Last 24 hours', a 'Save' dropdown, a 'Share' button, and a 'New alert' button. Below the toolbar, the query pane contains the following code:

```
1 SecurityEvent
2 | project TimeGenerated, EventID, ProcessId
3 | summarize TotalEvent = count() by EventID
```

Under the 'Results' tab, the output table has two columns: 'EventID' and 'TotalEvent'. The data rows are:

EventID	TotalEvent
> 8002	309195
> 4634	31975
> 4672	24793
> 4624	34550

summarize by counting SecurityEvents by EventID and Computer

▶ Run Time range : Last 24 hours | Save Share + New alert

```
1 SecurityEvent
2 | project TimeGenerated, EventID, ProcessId, Computer
3 | summarize count() by EventID, Computer
```

Results Chart

EventID	Computer	count_
> 8002	SQL10.na.contosohotels.com	85396
> 4634	DC11.na.contosohotels.com	7710
> 4672	DC11.na.contosohotels.com	5954
> 4624	DC11.na.contosohotels.com	7979
> 4662	DC11.na.contosohotels.com	2318
> 4688	DC11.na.contosohotels.com	7899
> 8002	DC11.na.contosohotels.com	7888
> 4688	JBOX00	13715
> 4673	JBOX00	2418
> 8002	JBOX00	13703

Add sort by to query

▶ Run Time range : Last 24 hours | Save Share + New alert

```
1 SecurityEvent
2 | project TimeGenerated, EventID, ProcessId, Computer
3 | summarize count() by EventID, Computer
4 | sort by EventID, Computer
```

Results Chart

EventID	Computer	count_
> 8002	SQL10.na.contosohotels.com	85381
> 8002	SQL01.na.contosohotels.com	77587
> 8002	SQL00.na.contosohotels.com	77014
> 8002	JBOX10	12425
> 8002	JBOX00	13694
> 8002	DC11.na.contosohotels.com	7861
> 8002	DC10.na.contosohotels.com	7763
> 8002	DC01.na.contosohotels.com	9975
> 8002	DC00.na.contosohotels.com	9099
> 8002	AppBE01.na.contosohotels.com	8344
> 5140	DC11.na.contosohotels.com	729
> 5140	DC10.na.contosohotels.com	1076
> 5140	DC01.na.contosohotels.com	758
> 5140	DC00.na.contosohotels.com	804

How can we combine computers into one set? (Computer, make_set(Computer))

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

```
1 SecurityEvent
2 | project TimeGenerated, EventID, ProcessId, Computer
3 | summarize count(), make_set(Computer) by EventID
4 | sort by EventID
```

Results Chart

EventID	count_	set_Computer
> 8002	309210	["SQL10.na.contosohotels.com", "DC10.na.contosohotels.com", "DC00.na.contosohotels.com", "SQL00.na.contosohotels.com"]
> 5140	3364	["DC00.na.contosohotels.com", "DC11.na.contosohotels.com", "DC01.na.contosohotels.com", "DC10.na.contosohotels.com"]
> 5059	473	["DC11.na.contosohotels.com", "DC01.na.contosohotels.com", "AppBE01.na.contosohotels.com", "SQL00.na.contosohotels.com"]
> 4948	32	["DC00.na.contosohotels.com", "AppBE01.na.contosohotels.com", "DC10.na.contosohotels.com", "DC11.na.contosohotels.com"]

Let's assume that Event ID 8002 indicates malicious activity. How can you find all the Computers where this malicious activity is happening?

Run Time range : Last 24 hours Save Share New alert rule

```
1 SecurityEvent
2 | where EventID == 8002
3 | summarize SuspiciousComputers = make_set(Computer)
4
```

Results Chart

SuspiciousComputers

- ▼ ["AppBE01.na.contosohotels.com", "SQL00.na.contosohotels.com", "DC01.na.contosohotels.com", "SQL01.na.contosohotels.com"]
 - ▼ SuspiciousComputers ["AppBE01.na.contosohotels.com", "SQL00.na.contosohotels.com", "DC01.na.contosohotels.com", "SQL01.na.contosohotels.com"]
 - 0 AppBE01.na.contosohotels.com
 - 1 SQL00.na.contosohotels.com
 - 2 DC01.na.contosohotels.com
 - 3 SQL01.na.contosohotels.com
 - 4 DC11.na.contosohotels.com
 - 5 DC10.na.contosohotels.com
 - 6 DC00.na.contosohotels.com
 - 7 JBOX10
 - 8 SQL10.na.contosohotels.com
 - 9 JBOX00

min() and max()

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

```
1 SecurityEvent
2 | summarize min(TimeGenerated), max(TimeGenerated) by EventID, Computer
3 | sort by EventID
4
```

Results Chart

EventID	Computer	min_TimeGenerated [UTC]	max_TimeGenerated [UTC]
> 8002	SQL10.na.contosohotels.com	1/8/2025, 10:31:14.173 AM	1/9/2025, 10:30:38.752 AM
> 8002	DC10.na.contosohotels.com	1/8/2025, 10:31:26.022 AM	1/9/2025, 10:30:26.015 AM
> 8002	DC01.na.contosohotels.com	1/8/2025, 10:31:37.009 AM	1/9/2025, 10:30:37.022 AM
> 8002	SQL01.na.contosohotels.com	1/8/2025, 10:31:41.904 AM	1/9/2025, 10:31:06.998 AM
> 8002	DC11.na.contosohotels.com	1/8/2025, 10:31:17.009 AM	1/9/2025, 10:31:00.820 AM

Try to summarize DnsNames, Ipv4Addresses, MacAddresses

The screenshot shows a KQL query in the editor:

```
4 VMComputer  
5 | summarize count() by DnsNames
```

A red error message box is displayed, stating: "Summarize group key 'DnsNames' is of a 'dynamic' type. Please use an explicit cast (for example, 'summarize ... by tostring(DnsNames)') as grouping by a 'dynamic' type is not supported. Request id: 12ef877a-8097-4583-ac12-44d59d10d802".

Note: You can not use dynamic data types with summarize.

You can cast it into a string type , if it does your work.

The screenshot shows a KQL query in the editor:

```
1 SigninLogs  
2 | project TimeGenerated, DeviceDetail, Location, AppDisplayName, RiskDetail, UserType  
3 | summarize count() by tostring(DeviceDetail)
```

The results table shows the count of devices for different browser types:

DeviceDetail	count_
{"deviceid": "", "browser": "Rich Client 4.14.0.0"}	192
{"deviceid": "[PII Removed]", "browser": "Edge 117.0.2045", "displayName": "[PII Removed]", "operatingSystem": "Windows10", "isCompliant": true, "isManaged": true, "trustType": "Azure AD joined"}	1
{"deviceid": "", "browser": "Edge 117.0.2045", "operatingSystem": "Windows10"}	1
{"deviceid": "[PII Removed]", "browser": "Chrome 116.0.0", "displayName": "[PII Removed]", "operatingSystem": "Windows10", "isCompliant": true, "isManaged": true, "trustType": "Hybrid Azure AD joined"}	1
{"deviceid": "", "browser": "Rich Client 4.36.0.0", "operatingSystem": "Windows10"}	1

Note: Homework is taken from “Ten Minute KQL” YouTube channel.

Homework

Environment: LADemo
Table: AVSSyslog

1. Last 30 days.
2. Summarizes the count of different severity types for each application name.
3. Sort results by App Name, then by severity.
4. Remove 'empty' App Name values.

The screenshot shows a KQL query in the editor:

```
1 AVSSyslog  
2 | where TimeGenerated >= ago(30d) and isnotempty(AppName)  
3 | summarize count() by Severity,AppName  
4 | sort by AppName, Severity
```

summarize, bin, dcount, countif

VMConnections for Computer “JBOX00”

▶ Run Time range : Last 24 hours | Save Share New alert rule Export Pin to Format

```
1 VMConnection
2 | where Computer == "JBOX00"
3 | sort by TimeGenerated asc
4
5
6
```

Results Chart

Showing the first 30,000 results. [Learn more](#) on how to narrow down the result set.

TimeGenerated [UTC]	Computer	Direction	ProcessName	SourceIP
> 1/8/2025, 11:06:24.192 AM	JBOX00	outbound	MsSense	127.0.0.1
> 1/8/2025, 11:06:24.192 AM	JBOX00	outbound	vmGuestHealthAgent	10.1.0.4
> 1/8/2025, 11:06:24.192 AM	JBOX00	outbound	MsSense	127.0.0.1
> 1/8/2025, 11:06:24.192 AM	JBOX00	outbound	Powershell	10.1.0.4
> 1/8/2025, 11:06:24.192 AM	JBOX00	outbound	svchost	10.1.0.4
> 1/8/2025, 11:06:24.192 AM	JBOX00	inbound	MsSense	127.0.0.1
> 1/8/2025, 11:06:24.192 AM	JBOX00	outbound	WindowsAzureGuestAgent	10.1.0.4
> 1/8/2025, 11:06:24.192 AM	JBOX00	inbound	WindowsAdminCenter	127.0.0.1
> 1/8/2025, 11:06:24.192 AM	JBOX00	outbound	svchost	10.1.0.4
> 1/8/2025, 11:06:24.192 AM	JBOX00	outbound	MsSense	127.0.0.1

Let's summarize that by bin

▶ Run Time range : Last 24 hours | Save Share New alert rule Export

```
1 VMConnection
2 | where Computer == "JBOX00"
3 | summarize count() by bin(TimeGenerated, 1m) // 1 minute
4 | sort by TimeGenerated asc
5
6
7
```

Results Chart

TimeGenerated [UTC]	count_
> 1/8/2025, 11:11:00.000 AM	36
> 1/8/2025, 11:12:00.000 AM	28
> 1/8/2025, 11:13:00.000 AM	28

bin() : Rounds values down to an integer multiple of a given bin size.

Used frequently in combination with `summarize by ...`. If you have a scattered set of values, they'll be grouped into a smaller set of specific values.

The `bin()` and `floor()` functions are equivalent

▶ Run Time range : Last 24 hours | Save Share New alert rule Export

```

1 VMConnection
2 | where Computer == "JBOX00"
3 | summarize count() by bin(TimeGenerated, 1h) // 1 hour
4 | sort by TimeGenerated asc
5
6
7

```

[Results](#) [Chart](#)

TimeGenerated [UTC]	count_
> 1/8/2025, 11:00:00.000 AM	1397
> 1/8/2025, 12:00:00.000 PM	1918
> 1/8/2025, 1:00:00.000 PM	2056

▶ Run Time range : Last 24 hours | Save Share New alert rule

```

1 VMConnection
2 | where Computer == "JBOX00"
3 | summarize count() by bin(TimeGenerated, 1d) // 1 day
4 | sort by TimeGenerated asc
5
6
7

```

[Results](#) [Chart](#)

TimeGenerated [UTC]	count_
> 1/8/2025, 12:00:00.000 AM	24591
> 1/9/2025, 12:00:00.000 AM	22600

How many Guests logged into your system daily in the last month?

▶ Run Time range : Set in query | Save Share New alert rule Export

```

1 SigninLogs
2 | where TimeGenerated >= ago(30d)
3 | where UserType == 'Guest'
4 | summarize count() by bin(TimeGenerated, 1d) // 1 day
5 | sort by TimeGenerated asc
6
7

```

[Results](#) [Chart](#)

TimeGenerated [UTC]	count_
> 1/8/2025, 12:00:00.000 AM	24591
> 1/9/2025, 12:00:00.000 AM	22600

Run Time range : Last 24 hours Save Share New alert rule Export Pin to Format

```

1 VMConnection
2 | project TimeGenerated, BytesSent
3 | summarize count(), min(BytesSent), avg(BytesSent), max(BytesSent) by bin(TimeGenerated, 1h)
4 | sort by TimeGenerated asc
5
6
7

```

Results **Chart**

TimeGenerated [UTC]	count_	min_BytesSent	avg_BytesSent	max_BytesSent
> 1/8/2025, 11:00:00.000 AM	15367	0	59479.00338387454	33106835
> 1/8/2025, 12:00:00.000 PM	25982	0	69261.1486413671	281155523
> 1/8/2025, 1:00:00.000 PM	26223	0	70157.25462380353	287623079

countif - counts only ByteSent is bigger than 0.

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

```

1 VMConnection
2 | project TimeGenerated, BytesSent
3 | summarize count(), countif(BytesSent > 0), min(BytesSent), avg(BytesSent), max(BytesSent) by bin(TimeGenerated, 1h)
4 | sort by TimeGenerated asc
5
6
7

```

Results **Chart**

TimeGenerated [UTC]	count_	countif_	min_BytesSent	avg_BytesSent	max_BytesSent
> 1/8/2025, 11:00:00.000 AM	13436	9466	0	60609.92691277166	33106835
> 1/8/2025, 12:00:00.000 PM	25982	18240	0	69261.1486413671	281155523
> 1/8/2025, 1:00:00.000 PM	26223	18296	0	70157.25462380353	287623079

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

```

1 VMConnection
2 | project TimeGenerated, BytesSent
3 | summarize RecordsPerHour = count(),
4     RecordsGreaterThanZero = countif(BytesSent > 0),
5     min(BytesSent),
6     avg(BytesSent),
7     max(BytesSent),
8     TotalBytesSent = sum(BytesSent) by bin(TimeGenerated, 1h)
9 | sort by TimeGenerated asc
10
11

```

Results **Chart**

TimeGenerated [UTC]	RecordsPerHour	RecordsGreaterThanZero	min_BytesSent	avg_BytesSent	max_BytesS...	TotalBytesS...
> 1/8/2025, 11:00:00.000 AM	9767	6882	0	59127.97184396437	23900836	577502901
> 1/8/2025, 12:00:00.000 PM	25982	18240	0	69261.1486413671	281155523	1799543164
> 1/8/2025, 1:00:00.000 PM	26223	18296	0	70157.25462380353	287623079	1839733688
> 1/8/2025, 2:00:00.000 PM	26688	18611	0	67712.7573441247	206953038	1807118068
> 1/8/2025, 3:00:00.000 PM	26007	18203	0	71084.10081901027	195273389	1848684210
> 1/8/2025, 4:00:00.000 PM	26279	18385	0	69156.61147684463	286906786	1817366593

What SecurityEvents occurred on each Machine for last 24 hours.

Run Time range : Last 24 hours Save Share New alert rule Export Pin to

```
1 //What SecurityEvents occurred on each Machine for last 24 hours.
2 SecurityEvent
3 | where TimeGenerated >= ago(1d)
4 | summarize count(), make_set(Activity) by Computer
5
6
```

Results Chart

Computer	count_	set_Activity
> SQL00.na.contosohotels.com	177450	["8002 - A process was allowed to run.", "4688 - A new process ha
> DC00.na.contosohotels.com	50283	["4688 - A new process has been created.", "4672 - Special privileg
> JBOX10	32755	["8002 - A process was allowed to run.", "4688 - A new process ha
> SQL01.na.contosohotels.com	181571	["8002 - A process was allowed to run.", "4688 - A new process ha
> DC10.na.contosohotels.com	42058	["4688 - A new process has been created.", "4634 - An account wa
> AppBE01.na.contosohotels.com	27879	["4662 - An operation was performed on an object.", "4688 - A ne
> DC01.na.contosohotels.com	53794	["4634 - An account was logged off.", "4688 - A new process has b
> SQL10.na.contosohotels.com	173830	["8002 - A process was allowed to run.", "4673 - A privileged serv
> DC11.na.contosohotels.com	46892	["4688 - A new process has been created.", "8002 - A process was
> JBOX00	35713	["4688 - A new process has been created.", "4673 - A privileged se

Let's assume that you are searching for Activity - 4634. How to narrow down your result.

Run Time range : Last 24 hours Save Share New alert rule Export Pin to

```
1 //What SecurityEvents occurred on each Machine for last 24 hours.
2 SecurityEvent
3 | where Activity startswith '4634'
4 | summarize count(), make_set(Activity) by Computer
5
6
```

Results Chart

Computer	count_	set_Activity
> DC11.na.contosohotels.com	7652	["4634 - An account was logged off."]
> SQL01.na.contosohotels.com	1114	["4634 - An account was logged off."]
> DC00.na.contosohotels.com	7911	["4634 - An account was logged off."]
> DC10.na.contosohotels.com	6768	["4634 - An account was logged off."]
> DC01.na.contosohotels.com	7943	["4634 - An account was logged off."]
> SQL10.na.contosohotels.com	628	["4634 - An account was logged off."]
> SQL00.na.contosohotels.com	105	["4634 - An account was logged off."]
> AppBE01.na.contosohotels.com	16	["4634 - An account was logged off."]

How many 4662 SecurityEvent occurred by Computer, every hour, last 24 hours.

Run Time range : Last 24 hours Save Share New alert rule Export P

```
1 //How many 4662 SecurityEvent occurred by Computer, every hour, last 24 hours
2 SecurityEvent
3 | where Activity startswith '4662'
4 | summarize count() by Computer, bin(TimeGenerated, 1h), Activity
5
6
```

Results Chart

Computer	TimeGenerated [UTC]	Activity	count_
DC11.na.contosohotels.com	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	70
DC10.na.contosohotels.com	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	88
SQL10.na.contosohotels.com	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	23
SQL00.na.contosohotels.com	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	70
AppBE01.na.contosohotels.com	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	213
DC01.na.contosohotels.com	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	97
SQL01.na.contosohotels.com	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	100
JBOX00	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	15
JBOX10	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	15
DC00.na.contosohotels.com	1/9/2025, 11:00:00.000 AM	4662 - An operation was perfor...	34

Taken from “Ten Minute KQL”

Homework

Environment: LADemo
Table: AppEvents

1. Identify number of people from unique states and countries.
2. 'Clicked My Profile Button'
3. For each hour, for the last 24 hours.
4. The 'Name' field is a key field in your query.

Solution:

Run Time range : Set in query Save Share New alert rule Export Pin to

```
1 AppEvents
2 | where TimeGenerated >= ago(24h)
3 | where Name == "Clicked My Profile Button"
4 | summarize count() by bin(TimeGenerated, 1h), ClientStateOrProvince, ClientCountryOrRegion
5 | sort by TimeGenerated asc
6
7
```

render, timechart, areachart, barchart

Results:

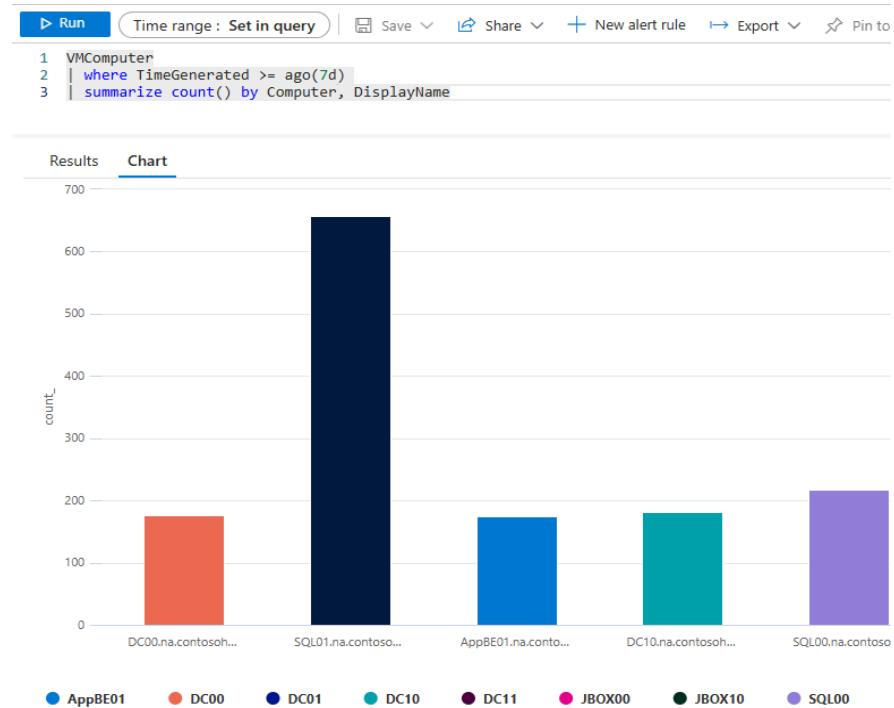
Run Time range : Set in query Save Share New alert rule

```
1 VMComputer
2 | where TimeGenerated >= ago(7d)
3 | summarize count() by Computer, DisplayName
```

Results Chart

Computer	DisplayName	count_
DC00.na.contosohotels.com	DC00	176
SQL01.na.contosohotels.com	SQL01	658
AppBE01.na.contosohotels.com	AppBE01	175
DC10.na.contosohotels.com	DC10	182
SQL00.na.contosohotels.com	SQL00	217
DC11.na.contosohotels.com	DC11	183

When you click to Chart

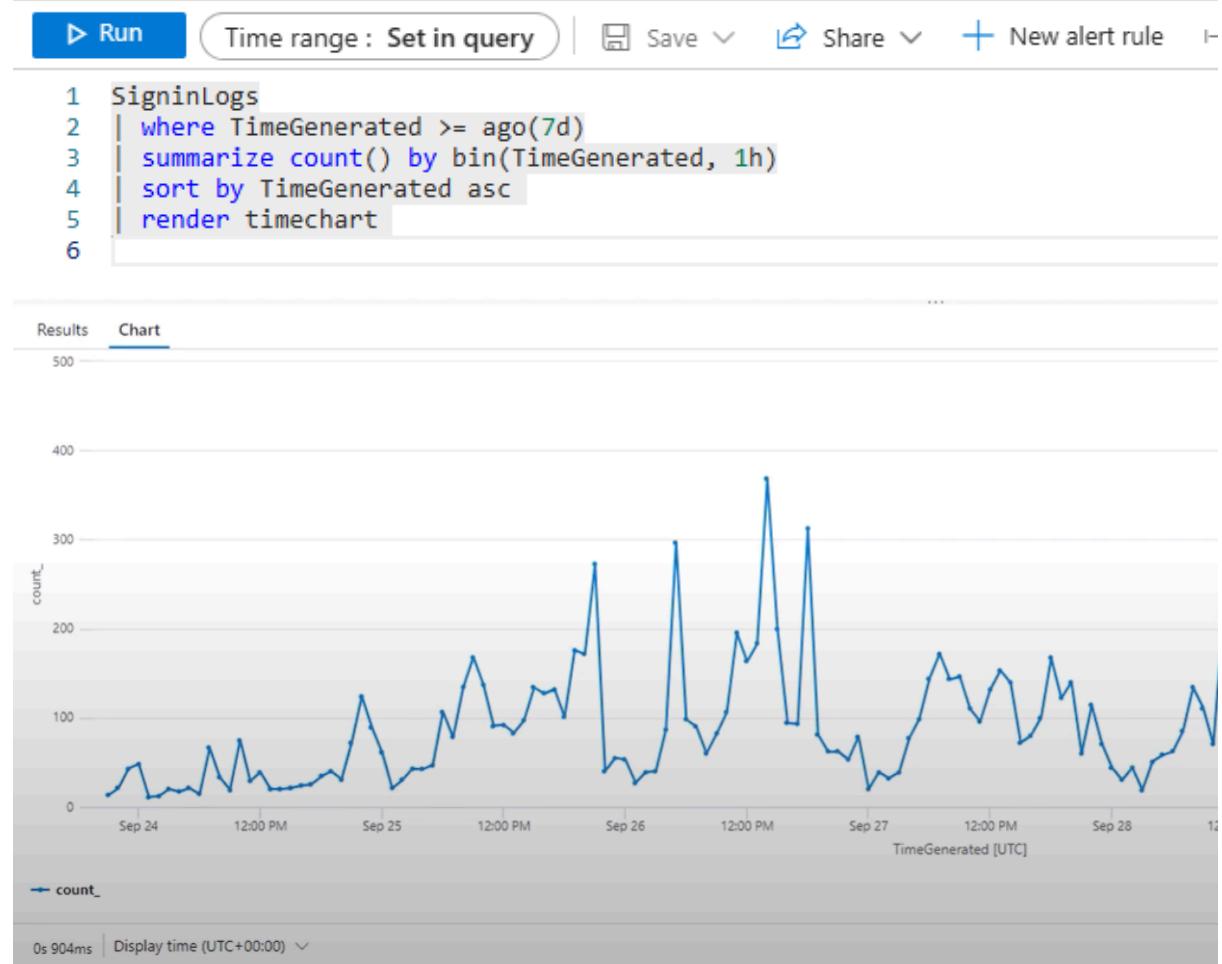


render operator: Instructs the user agent to render a visualization of the query results. The render operator **must be the last operator** in the query, and can only be used with queries that produce a **single tabular data stream result**. The render operator **doesn't modify** data. It injects an annotation ("Visualization") into the result's extended properties. The annotation contains the information provided by the operator in the query. The interpretation of the visualization information is done by the user agent. Different agents, such as Kusto.Explorer or Azure Data Explorer web UI, may support different visualizations.

Time Chart

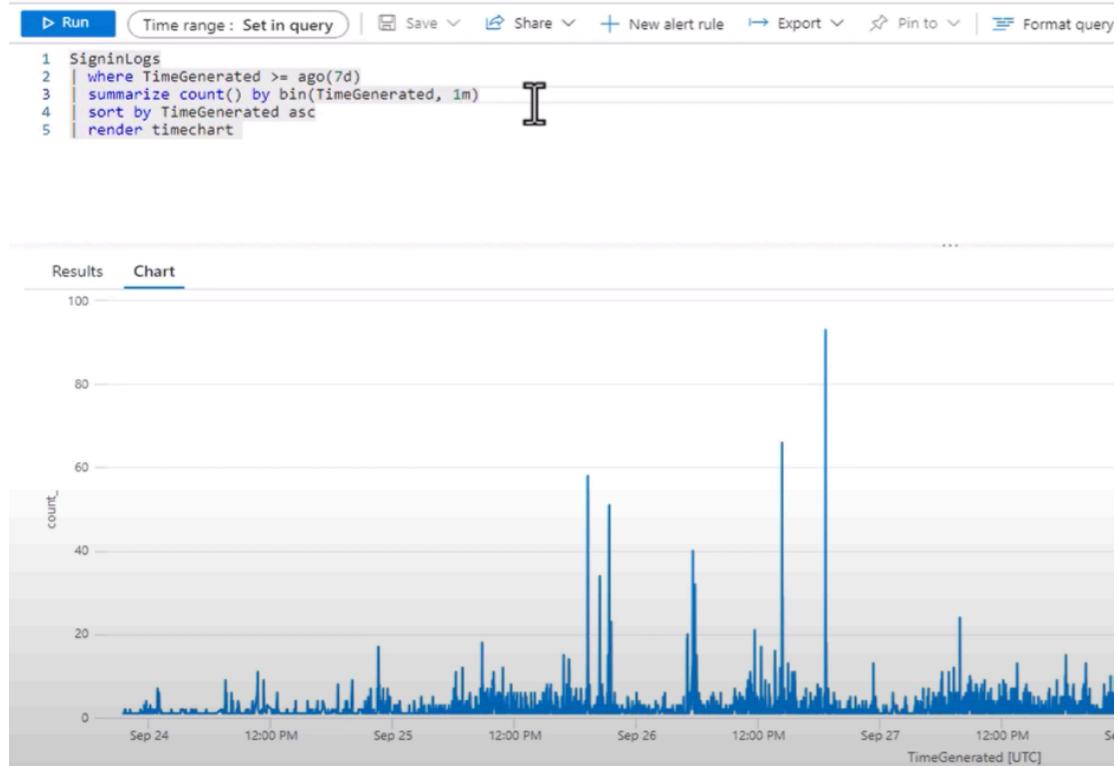
A time chart visual is a type of line graph. The first column of the query is the x-axis, and should be a datetime. Other numeric columns are y-axes. One string column values are used to group the numeric columns and create different lines in the chart. Other string columns are ignored. The time chart visual is like a [line chart](#) except the x-axis is always time.

How do we display logins from the last week in a graph as hourly buckets?



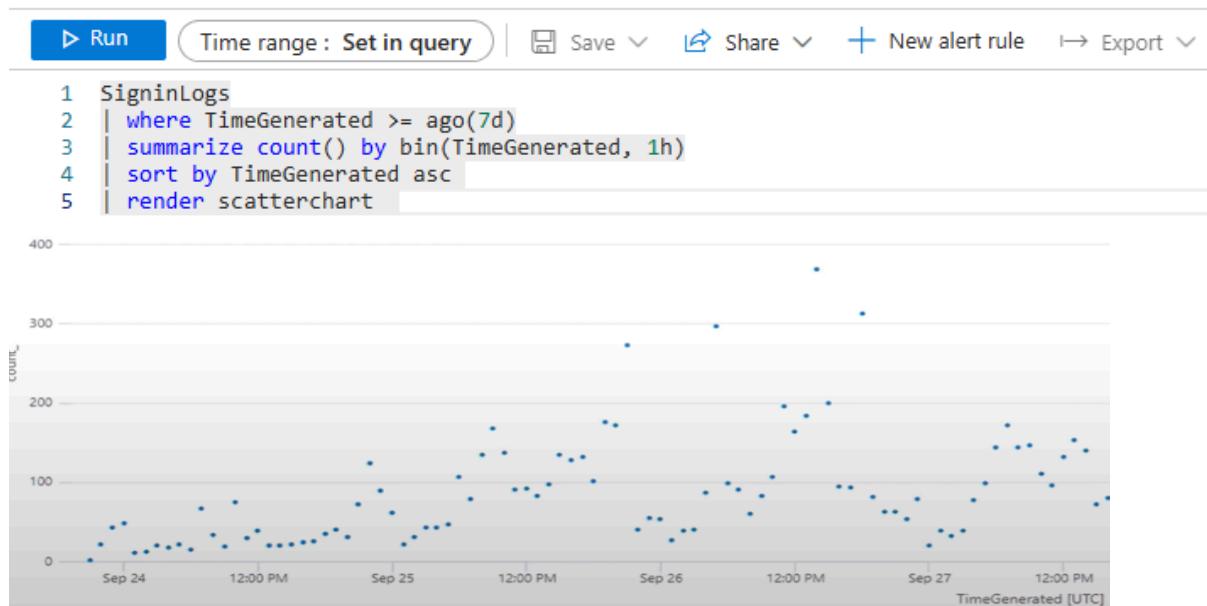
Try for 30 minutes, 6 hours, 12 hours, 1 day and examine the results.

bin(TimeGenerated, 1m)



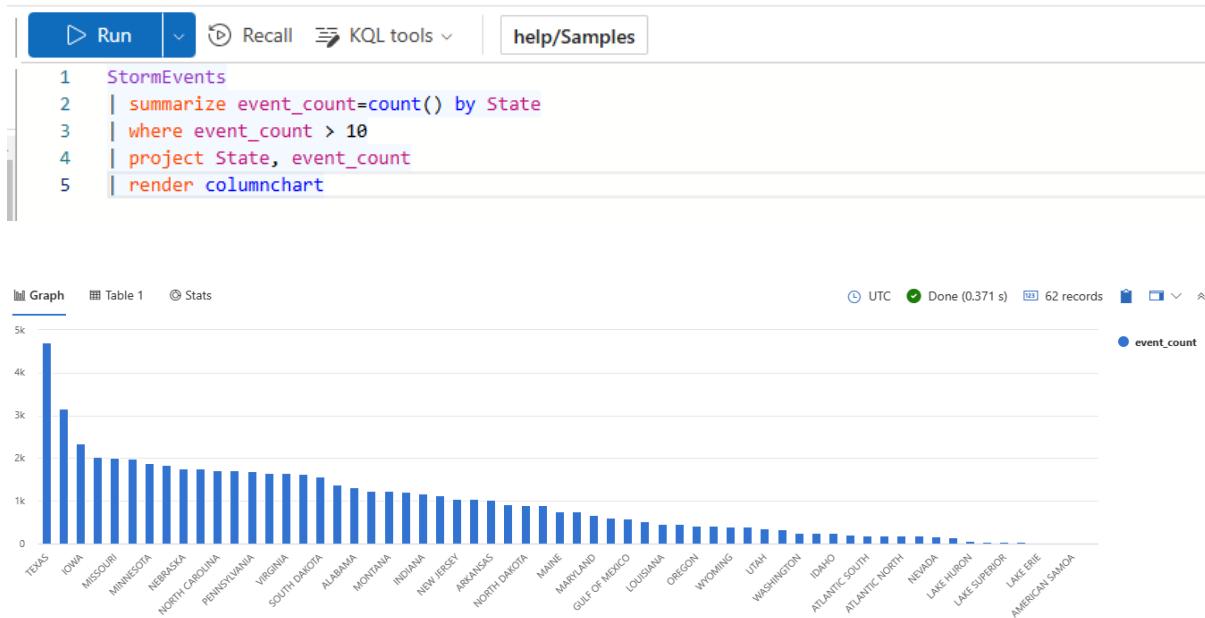
Focus on the spikes, what they are telling us. In some cases those spikes can be very meaningful for us. In some cases, these spikes can be very meaningful to us. Visualization makes the result set easier to read.

Scatter Chart: In a scatter chart visual, the first column is the x-axis and should be a numeric column. Other numeric columns are y-axes. Scatter plots are used to observe relationships between variables. The scatter chart visual can also be used in the context of [Geospatial visualizations](#).



column chart: The column chart visual needs a minimum of two columns in the query result. By default, the first column is used as the x-axis. This column can contain text, datetime, or numeric data types. The other columns are used as the y-axis and contain numeric data types to be displayed as vertical lines. Column charts are used for comparing specific subcategory items in a main category range, where the length of each line represents its value.

ADX - Azure Data Explorer

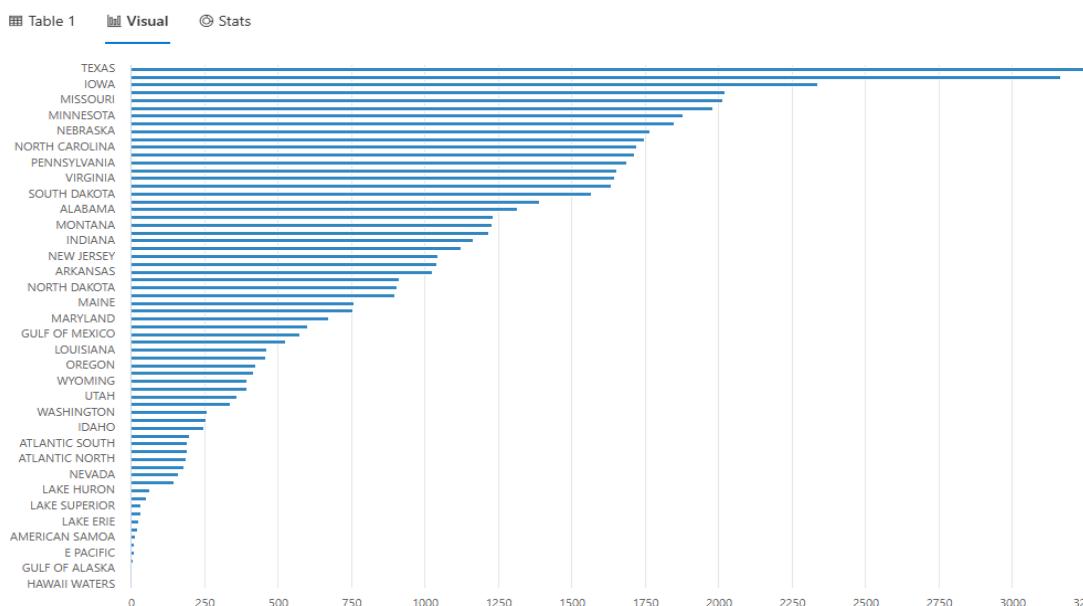


barchart: The bar chart visual needs a minimum of two columns in the query result. By default, the first column is used as the y-axis. This column can contain text, datetime, or numeric data types. The other columns are used as the x-axis and contain numeric data types to be displayed as horizontal lines. Bar charts are used mainly for comparing numeric and nominal discrete values, where the length of each line represents its value.

```

1 StormEvents
2 | summarize event_count=count() by State
3 | project State, event_count
4 | render barchart

```



piechart: The pie chart visual needs a minimum of two columns in the query result. By default, the first column is used as the color axis. This column can contain text, datetime, or numeric data types. Other columns will be used to determine the size of each slice and contain numeric data types. Pie charts are used for presenting a composition of categories and their proportions out of a total.

Two columns data perfect for piechart

Run Recall KQL tools help/Samples

```

1 StormEvents
2 | summarize statecount=count() by State
3 | sort by statecount
4 | limit 10
5 | render piechart with(title="Storm Events by State")

```

Table 1 Visual Stats

State	event_count
TEXAS	4 701
KANSAS	3 166
IOWA	2 337
ILLINOIS	2 022
MISSOURI	2 016
GEORGIA	1 983

Run Recall KQL tools help/Samples

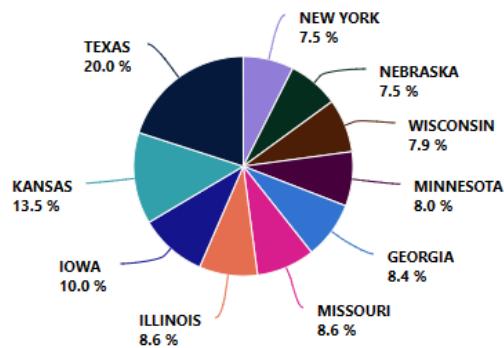
```

1 StormEvents
2 | summarize statecount=count() by State
3 | sort by statecount
4 | limit 10
5 | render piechart with(title="Storm Events by State")

```

Graph Table 1 Stats UTC Done (1.003 s) 10 records

Storm Events by State

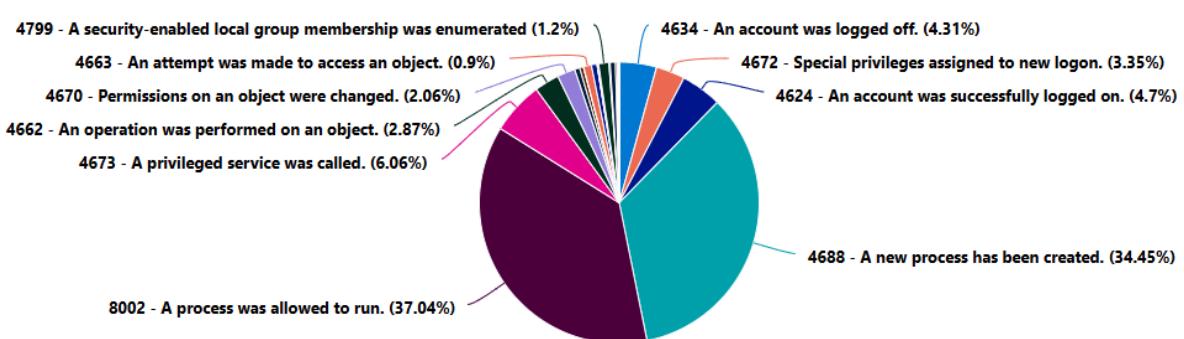


Run Time range : Set in query Save Share New alert rule

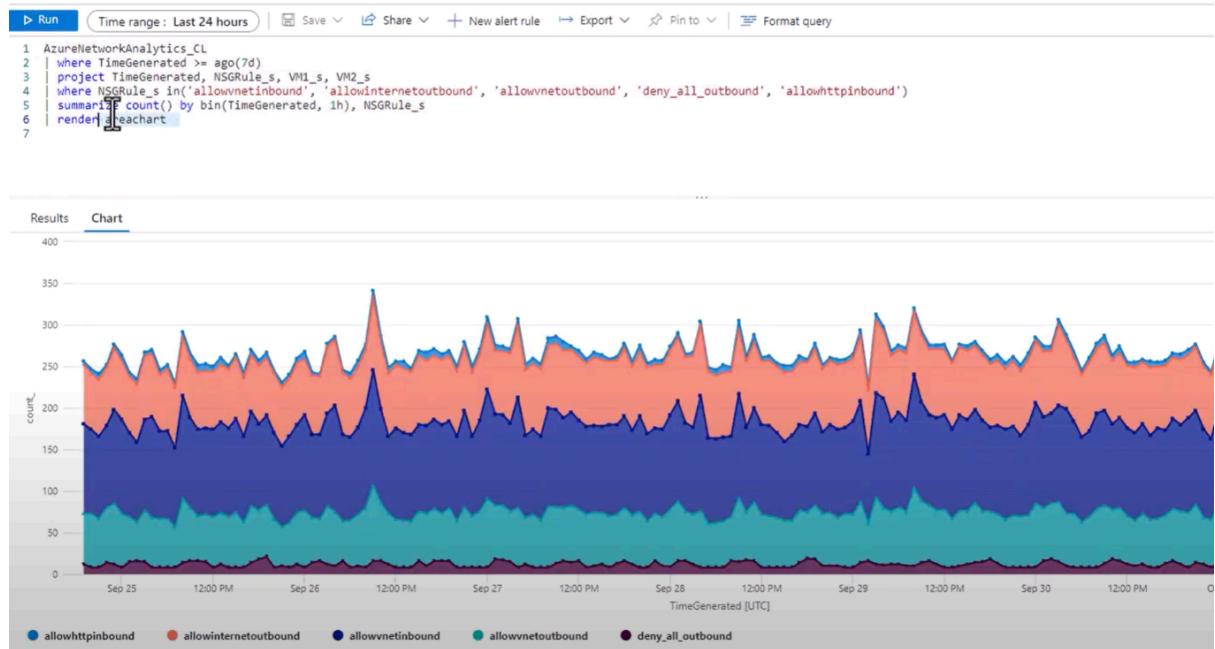
```

1 SecurityEvent
2 | where TimeGenerated >= ago(7d)
3 | summarize Count=count() by Activity
4 | render piechart
5

```



areachart: The area chart visual shows a time-series relationship. The first column of the query should be numeric and is used as the x-axis. Other numeric columns are the y-axes. Unlike line charts, area charts also visually represent volume. Area charts are ideal for indicating the change among different datasets.



Ten Minute KQL:

The slide has a black background with white text. At the top center is the word "Homework" in a large, bold, sans-serif font. Below it, the text "Environment: LADemo" and "Table: SecurityEvent" is displayed. A numbered list of five items is listed below the table information.

1. Build a chart or graph
2. Count of security events
3. For each computer
4. Over the last week
5. Adjust to optimal resolution

Solution:

```
▶ Run Time range : Last 24 hours | Save | Share | New
1 SecurityEvent
2 | where TimeGenerated >= ago(7d)
3     and isnotempty(EventData)
4 | summarize count() by Computer
5 | render piechart
6
7
```

Variables, let, pack_array, has_any

let statement: A `let` statement is used to set a variable name equal to an expression or a function, or to create [views](#).

`let` statements are useful for:

- Breaking up a complex expression into multiple parts, each represented by a variable.
- Defining constants outside of the query body for readability.
- Defining a variable once and using it multiple times within a query.

If the variable previously represented another value, for example in nested statements, the innermost `let` statement applies.

```
▶ Run Recall KQL tools help/Samples
1 let n = 10; // number
2 let place = "Dallas"; // string
3 let cutoff = ago(30d); // datetime
4 StormEvents
5 | where StartTime >= cutoff
6 | and State == place
7 | take n
```

```
▶ Run Time range : Last 24 hours | Save | Share | New alert rule | Export | Pin to | Format query
1 let _DomainController = 'JBOX10';
2 VmComputer
3 | where DisplayName == _DomainController
4
5
```

Results

TimeGenerated [UTC]	Computer	AgentId	Machine	DisplayName
> 1/10/2025, 10:53:53.151 PM	JBOX10	926b5c2d-9031-42d9-b692-07...	m-926b5c2d-9031-42d9-b692-...	JBOX10
> 1/11/2025, 12:53:53.129 AM	JBOX10	926b5c2d-9031-42d9-b692-07...	m-926b5c2d-9031-42d9-b692-...	JBOX10

Run Time range : Last 24 hours Save Share New alert rule Export

```

1 let _add_1_cpu = 1;
2 let _add_2_cpu = 2;
3 VMComputer
4 | extend CPUAdd = Cpus + (_add_1_cpu + _add_2_cpu)
5 | project tolong(Cpus), toint(CPUAdd)
6
7

```

Results Chart

Cpus	CPUAdd
> 2	5
> 4	7
> 2	5

Run Time range : Last 24 hours Save Share New alert rule Export

```

1 let _add_1_cpu = 1;
2 let _add_2_cpu = 2;
3 VMComputer
4 | extend CPUAdd = Cpus + (_add_1_cpu + _add_2_cpu)
5 | project tolong(Cpus), toint(CPUAdd)
6 getschema
7

```

Results Chart

ColumnName	ColumnOrdinal	DataType	ColumnType
> Cpus	0	System.Int64	long
> CPUAdd	1	System.Int32	int

pack_array: Packs all input values into a [dynamic](#) array.

```
range x from 1 to 3 step 1
| extend y = x * 2
| extend z = y * 2
| project pack_array(x, y, z)
```

Column1

[1,2,4]

[2,4,8]

[3,6,12]

```

range x from 1 to 3 step 1
| extend y = tostring(x * 2)
| extend z = (x * 2) * 1s
| project pack_array(x, y, z)

```

Column1

[1,"2","00:00:02"]
[2,"4","00:00:04"]
[3,"6","00:00:06"]

has_any: Filters a record set for data with any set of case-insensitive strings. `has_any` searches for indexed terms, where an indexed `term` is three or more characters. If your term is fewer than three characters, the query scans the values in the column, which is slower than looking up the term in the term index.

For more information about other operators and to determine which operator is most appropriate for your query, see [datatype string operators](#).

The screenshot shows the Kusto Query Editor interface. At the top, there are buttons for 'Run', 'Recall', 'KQL tools', and a 'help/Samples' link. The query window contains the following KQL code:

```

1 StormEvents
2 | where State has_any ("CAROLINA", "DAKOTA", "NEW")
3 | summarize count() by State

```

Below the query window is a table titled 'Table 1'. The table has two columns: 'State' and 'count_'. The data is as follows:

State	count_
NEW YORK	1 750
NORTH CAROLINA	1 721
SOUTH DAKOTA	1 567
NEW JERSEY	1 044
SOUTH CAROLINA	915
NORTH DAKOTA	905
NEW MEXICO	527
NEW HAMPSHIRE	394

```
// The following query shows how to use has_any with a dynamic array.
```

```

StormEvents
| where State has_any (dynamic(['south', 'north']))
| summarize count() by State

```

```

//The same query can also be written with a let statement.

let areas = dynamic(['south', 'north']);
StormEvents
| where State has_any (areas)
| summarize count() by State

//The following query shows how to use has_any with an inline tabular
expression. Notice that an inline tabular expression must be enclosed with
double parentheses.

StormEvents
| where State has_any ((PopulationData | where Population > 5000000 |
project State))
| summarize count() by State

let large_states = PopulationData | where Population > 5000000 | project
State;
StormEvents
| where State has_any (large_states)
| summarize count() by State

```

The screenshot shows the Kusto Query Editor interface. At the top, there are several buttons: Run, Time range: Set in query, Save, Share, New alert rule, Export, Pin to, and Format query. Below the toolbar, the query text is displayed:

```

1 let _machines = pack_array('JBOX00', 'DC10', 'DC11');
2 let _startTime = ago(7d);
3 let _endTime = ago(2d);
4 let _ESTOffset = (5h);
5 ConfigurationChange
6 | where TimeGenerated between (_startTime .. _endTime)
7     and Computer has_any (_machines)
8 | extend EST = TimeGenerated - _ESTOffset
9 | summarize count(), make_set(SvcName) by bin(EST, 1h), Computer

```

Below the query, there are two tabs: Results and Chart. The Results tab is selected, showing a table with the following data:

EST [UTC]	Computer	count_	set_SvcName
> 10/20/2023, 12:00:00.000 AM	JBOX00	6	["TrustedInstaller","smphost","","wmiApSrv","wuauserv"]
> 10/20/2023, 1:00:00.000 AM	JBOX00	6	["TrustedInstaller","wmiApSrv","wuauserv","smphost"]
> 10/20/2023, 12:00:00.000 AM	DC10	4	["smphost","RemoteRegistry","AppXSvc"]
> 10/20/2023, 12:00:00.000 AM	DC10.na.contosohotels.com	2	["WdiSystemHost","wmiApSrv"]
> 10/20/2023, 12:00:00.000 AM	DC11.na.contosohotels.com	3	["smphost","TrustedInstaller"]

Run | Time range : Set in query | Save | Share | New alert rule | Export | Pin to | Format query

```

1 let _suspiciousActivities = pack_array(" 4688 - A new process has been created.",
2                                         "4670 - Permissions on an object were changed.",
3                                         "4672 - Special privileges assigned to new logon.");
4 SecurityEvent
5 | where TimeGenerated >= ago(7d)
6     and Activity has_any (_suspiciousActivities)
7 | summarize make_set(Activity) by Computer

```

Results Chart

Computer	set_Activity
> DC11.na.contosohotels.com	["4670 - Permissions on an object were changed.", "4672 - Special privileges assigned to new logon."]
> DC10.na.contosohotels.com	["4672 - Special privileges assigned to new logon.", "4670 - Permissions on an object were changed."]
> DC00.na.contosohotels.com	["4672 - Special privileges assigned to new logon.", "4670 - Permissions on an object were changed."]
> DC01.na.contosohotels.com	["4672 - Special privileges assigned to new logon.", "4670 - Permissions on an object were changed."]
> JBOX00	["4670 - Permissions on an object were changed.", "4672 - Special privileges assigned to new logon."]
> JBOX10	["4672 - Special privileges assigned to new logon.", "4670 - Permissions on an object were changed."]
> SQL01.na.contosohotels.com	["4672 - Special privileges assigned to new logon.", "4670 - Permissions on an object were changed."]
> SQL10.na.contosohotels.com	["4670 - Permissions on an object were changed.", "4672 - Special privileges assigned to new logon."]
> SQL00.na.contosohotels.com	["4670 - Permissions on an object were changed.", "4672 - Special privileges assigned to new logon."]
> AppBE01.na.contosohotels.com	["4672 - Special privileges assigned to new logon.", "4670 - Permissions on an object were changed."]

Taken from Ten Minute KQL YouTube Channel

Homework

Environment: LADemo
Table: VMBoundPort

1. Make a variable for ports (22, 53, 80, 389, 445, 1433)
2. Make second variable for 8h
3. From the last 3 days
4. Identify volume of use of each port type by computer
5. At 1h intervals
6. Timezone of UTC -8h

Follow On: Rewrite query to show devices that have 'only' used all 6 ports every 1d.

Solution:

Run | Time range : Set in query | Save | Share | New alert rule | Export

```

1 let _checkPort = pack_array(22,53,80,389,445,1433);
2 let _timePrecision = (8h);
3 VMBoundPort
4 | where TimeGenerated >= ago(3d)
5     and tostring(Port) has_any(_checkPort)
6 | extend EST = TimeGenerated - _timePrecision
7 | summarize count() by bin(EST, 1h), Computer, Port
8

```

Results Chart

EST [UTC]	Computer	Port	count_
> 1/10/2025, 11:00:00.000 PM	JBOX00	445	60
> 1/10/2025, 11:00:00.000 PM	DC11.na.contosohotels.com	389	97
> 1/10/2025, 11:00:00.000 PM	DC11.na.contosohotels.com	53	240

Follow On: Rewrite query to show devices that have 'only' used all 6 ports every 1d.

Solution:

```
Run Time range : Custom | Save | Share | New alert rule | Export | Pin to | Format query  
1 let _checkPort = pack_array(22,53,80,389,445,1433);  
2 let _timePrecision = (8h);  
3 VMBoundPort  
4 | where TimeGenerated >= ago(3d)  
5 ... and tostring(Port) has_all(_checkPort)  
6 | extend EST = TimeGenerated - _timePrecision  
7 | summarize count() by bin(EST, 1d), Computer, Port  
8
```

externaldata operator: The externaldata operator returns a table whose schema is defined in the query itself, and whose data is read from an external storage artifact, such as a blob in Azure Blob Storage or a file in Azure Data Lake Storage.

How to export data csv, excel or PowerBI file?

You can use the export feature.

Run Time range : Last 24 hours | Save | Share | New alert rule | Export | Pin to | Format query

1 SecurityEvent
2 | take 10

Results Chart

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
> 1/11/2025, 9:28:04.001 AM	NA\AppBE01\$	Machine	AppBEC1.na.contosohotels.com	Microsoft-Windows-Security-Auditing	Security
> 1/11/2025, 9:28:04.011 AM	NA\AppBE01\$	Machine	AppBEC1.na.contosohotels.com	Microsoft-Windows-Security-Auditing	Security
> 1/11/2025, 9:28:04.006 AM	NT AUTHORITY\SYSTEM	User	AppBEC1.na.contosohotels.com	Microsoft-Windows-AppLocker	Microso
> 1/11/2025, 9:28:04.016 AM	NT AUTHORITY\SYSTEM	User	AppBEC1.na.contosohotels.com	Microsoft-Windows-AppLocker	Microso
> 1/11/2025, 9:32:04.928 AM	NA\DC11\$	Machine	DC11.na.contosohotels.com	Microsoft-Windows-Security-Auditing	Security
> 1/11/2025, 9:32:04.929 AM	NA\timadmin	User	DC11.na.contosohotels.com	Microsoft-Windows-Security-Auditing	Security

CSV (all columns)
CSV (displayed columns)
Open in Excel
Power BI (as an M query)
Power BI (new Dataset)

How to share your Query?

You can use the share feature.

Run Time range : Last 24 hours | Save | Share | New alert rule | Export | Pin to | Format query

1 SecurityEvent
2 | take 10

Share | Copy link to query | Copy query text | Copy results

Results Chart

TimeGenerated [UTC]	Account	AccountType	Computer	EventSourceName	Channel
> 1/11/2025, 9:28:04.001 AM	NA\AppBE01\$	Machine	AppBEC1.na.contosohotels.com	Microsoft-Windows-Security-Auditing	Security
> 1/11/2025, 9:28:04.011 AM	NA\AppBE01\$	Machine	AppBEC1.na.contosohotels.com	Microsoft-Windows-Security-Auditing	Security
> 1/11/2025, 9:28:04.006 AM	NT AUTHORITY\SYSTEM	User	AppBEC1.na.contosohotels.com	Microsoft-Windows-AppLocker	Microso
> 1/11/2025, 9:28:04.016 AM	NT AUTHORITY\SYSTEM	User	AppBEC1.na.contosohotels.com	Microsoft-Windows-AppLocker	Microso
> 1/11/2025, 9:32:04.928 AM	NA\DC11\$	Machine	DC11.na.contosohotels.com	Microsoft-Windows-Security-Auditing	Security
> 1/11/2025, 9:32:04.929 AM	NA\timadmin	User	DC11.na.contosohotels.com	Microsoft-Windows-Security-Auditing	Security

Let's assume IPS.txt file includes risky IP Addresses

```
▶ Run Time range : Last 24 hours | Save Share + New alert rule Export Pin to Format query
1 // let's assume IPS.txt file includes risky IP Addresses
2 let OSINT = externaldata (IP:string) [h@https://raw.githubusercontent.com/aytuncbakir/KQL-Intermediate-Trainig/refs/heads/main/IPs.txt] with(format='txt');
3 OSINT
4 | take 100
5 ...
Results Chart
IP
> 168.63.129.16
> 167.63.139.16
> 10.1.5.10
> 20.50.80.214
> 195.47.238.93
> 185.40.4.150
```

Check if it was connected to those IPs by your VMs.

```
▶ Run Time range : Set in query | Save Share + New alert rule Export Pin to Format query
1 // let's assume IPS.txt file includes risky IP Addresses
2 let OSINT = externaldata (IP:string)
3 [h@https://raw.githubusercontent.com/aytuncbakir/KQL-Intermediate-Trainig/refs/heads/main/IPs.txt]
4 with(format='txt');
5 VMConnection
6 | where TimeGenerated >= ago(365d)
7 and DestinationIp in (OSINT)
8 | summarize count() by DestinationIp, Computer
9 ...
Results Chart
DestinationIp Computer count_
> 168.63.129.16 DC11.na.contosohotels.com 1733525
> 168.63.129.16 SQL10.na.contosohotels.com 1494972
> 10.1.5.10 SQL10.na.contosohotels.com 667258
> 168.63.129.16 JBOX00 1028296
> 168.63.129.16 DC00.na.contosohotels.com 1732867
> 10.1.5.10 DC00.na.contosohotels.com 99439
> 168.63.129.16 AppBE01.na.contosohotels.com 426836
> 168.63.129.16 JBOX10 1494751
```

Let's read malicious hashes from bazar.abuse

```
▶ Run Time range : Last 24 hours | Save Share + New alert rule Export Pin to Format query
1 let _malwareHashes = (externaldata(sha256_hash:string)[@"https://bazaar.abuse.ch/export/txt/sha256/recent/"]);
2 _malwareHashes
...
Results Chart
sha256_hash
> #####
> # MalwareBazaar recent malware samples (SHA256 hashes) #
> # Last updated: 2025-01-11 10:06:25 UTC #
> #
> # Terms Of Use: https://bazaar.abuse.ch/faq/#tos #
> # For questions please contact bazaar [at] abuse.ch #
> #####
> #
> # sha256_hash
> c9543effa107eb8ce3fc12b92395e248180c3cc664330677a3ae8c0f9564c3ce
> 92b22f9837aee9110802ab2442ec6de682046a78959f277a530b48cb68628e0
```

Let's drop the lines start with “#”

The screenshot shows a Splunk search interface with the following KQL query:

```
1 let _malwareHashes = (externaldata(sha256_hash:string)[@"https://bazaar.abuse.ch/export/txt/sha256/recent/"])
2 | where sha256_hash !startswith "#";
3 _malwareHashes
```

The results table has "Results" selected and displays a list of SHA-256 hashes. The first few entries are:

- > c9543effa107eb8ce3fc12b92395e248180c3cc664330677a3ae8c0f9564c3ce
- > 92b22f9837aee9110802ab2442ec6de682046a78959f277a530b48cb68628e0c
- > 490e793d8e2eee8ee51c9c601ff6de3272006d9c8b9ee57bb16001e91b77ada
- > 3c6f7c6f6583f8a632a15cd6f2c833a87693eb46201029fa912180c7cd53ac43
- > 80e9d07dd1222b4908748d20dc9864d316a194b62bb541b130b4f054b8b504df
- > f96542e015d336b20eecf5ce8b7add831791eaea5994ef1d91ba5c230ed2aa73
- > 09b423cd950d74da88818aad56e52b7278bc0d14e9a96e50b6ccfd54c79c326
- > 6e6b6dd3955df2ffd78be1b1b2379aedd3e617b22000af59815019048f83568c
- > bd5a5856a3a4d7865fd7f1c2fa789206a17e32ad2da3680ef7dc9c5e0158da35
- > 30327d430e0fb6205a73d9a9bfd3c7eb971484a737e42a615a7624fbfd1e83

Homework

Environment: LA Demo

SecurityEvent

You have malicious FileHases at:

<https://raw.githubusercontent.com/aytuncbakir/KQL-Intermediate-Trainig/main/FileHashes.txt>

Hunt those given FileHashes by using KQL.

The screenshot shows a Splunk search interface with the following KQL query:

```
1 let _malwareHashes = (externaldata(sha256_hash:string)[@"https://raw.githubusercontent.com/aytuncbakir/KQL-Intermediate-Trainig/main/FileHashes.txt"])
2 | where sha256_hash !startswith "#";
3 SecurityEvent
4 | where FileHash in (_malwareHashes)
5 | project Computer, FileHash
```

The results table has "Results" selected and displays a list of computers and their corresponding file hashes. The entries are:

Computer	FileHash
JBOX00	7DEF5DD11D9F1CA52686EE2F2F8DB2C3A104C640917F36EDE485B8684A47C64A
JBOX00	184BE906B710D5B7442F7E0F457E75C7F58920297CCD1D8A076A76EEFCC8A78D
JBOX10	7DEF5DD11D9F1CA52686EE2F2F8DB2C3A104C640917F36EDE485B8684A47C64A
JBOX10	184BE906B710D5B7442F7E0F457E75C7F58920297CCD1D8A076A76EEFCC8A78D
SQL00.na.contosohotels.com	7DEF5DD11D9F1CA52686EE2F2F8DB2C3A104C640917F36EDE485B8684A47C64A
SQL00.na.contosohotels.com	184BE906B710D5B7442F7E0F457E75C7F58920297CCD1D8A076A76EEFCC8A78D
AppBE01.na.contosohotels.com	7DEF5DD11D9F1CA52686EE2F2F8DB2C3A104C640917F36EDE485B8684A47C64A

round(), arg_max(), union

round(): Returns the rounded number to the specified precision. The rounded number to the specified precision. Round is different from the `bin()` function in that the `round()` function rounds a number to a specific number of digits while the `bin()` function rounds the value to an integer multiple of a given bin size. For example, `round(2.15, 1)` returns 2.2 while `bin(2.15, 1)` returns 2.

```
round(2.98765, 3)      // 2.988
round(2.15, 1)          // 2.2
round(2.15)              // 2 // equivalent to round(2.15, 0)
round(-50.55, -2)       // -100
round(21.5, -1)         // 20
```

Run Recall KQL tools MyFreeCluster/MyDatabase

```
1
2   cluster('help').database('SampleIoTData').TransformedSensorsData
3   | project SensorName, Value
4   | take 100
```

Table 1 Add visual Stats

SensorName	Value
sensor-82	0,05129636427656748
sensor-82	0,8168483546548001
sensor-130	0,016883112467987793
sensor-130	0,01584247191121413

Run Recall KQL tools MyFreeCluster/MyDatabase

```
1
2   cluster('help').database('SampleIoTData').TransformedSensorsData
3   | project SensorName, round(Value,2)
4   | take 100
```

Table 1 Add visual Stats

SensorName	Value
sensor-82	0,05
sensor-82	0,82
sensor-130	0,02
sensor-130	0,02
sensor-56	0,59

▷ Run

Recall

KQL tools

MyFreeCluster/MyDatabase

```
1
2   cluster('help').database('SampleIoTData').TransformedSensorsData
3   | project SensorName, RoundedSensorValue = round(Value,2)
4   | take 100
```

Table 1 Add visual Stats

SensorName	RoundedSensorValue
sensor-82	0,05
sensor-82	0,82
sensor-130	0,02

max():

▷ Run

Recall

KQL tools

MyFreeCluster/MyDatabase

```
1
2   cluster('help').database('SampleIoTData').TransformedSensorsData
3   | summarize max(Value) by SensorName
4
```

Table 1 Add visual Stats

SensorName	max_Value
sensor-9	0,9999494369169925
sensor-4	0,9999999527575304
sensor-0	0,999985231584243
sensor-18	0,9999396198816732
sensor-12	0,999984562159392
sensor-16	0,9999900584046968
sensor-3	0,9999744453929984
sensor-13	0,999980045916803

The **arg_max()** function differs from the **max()** function. The **arg_max()** function allows you to return additional columns along with the maximum value, and **max()** only returns the maximum value itself.

Run Recall KQL tools MyFreeCluster/MyDatabase

```
1
2   cluster('help').database('SampleIoTData').TransformedSensorsData
3   | summarize arg_max(Value, *) by SensorName
4
```

Table 1 Add visual Stats

SensorName	Value	Timestamp	PublisherId	MachineId
> sensor-89	0,9999223243864502	2022-04-13 00:57:53.3020	31ee7981-9a4e-4573-885f-fb1e23a44422	M100
> sensor-27	0,999606616270891	2022-04-13 00:57:58.2030	46344283-4868-482b-92d8-fbf91fd53425	M100
> sensor-103	0,999805667518576	2022-04-13 00:58:10.2870	ba264b51-362f-485d-ae43-b052440a5464	M100
> sensor-100	0,999896718137476	2022-04-13 00:58:12.4870	6ccb0b2c-5b44-4fa2-9d2b-ffc0de7ff62f	M100
> sensor-179	0,9999164514638924	2022-04-13 00:58:13.7970	f07b910d-b33f-407c-bee3-58c236b624e8	M100
> sensor-193	0,9995683859919664	2022-04-13 00:58:14.3400	0906fb21-c344-4e16-a282-0b03d97b8d16	M100
> sensor-142	0,9975463762315112	2022-04-13 00:58:15.3830	89f91978-a970-4a1d-9511-44ea9c402fcc	M100
> sensor-154	0,9995068600564726	2022-04-13 00:58:15.4790	35cdf71f-7f63-48d5-8dc6-f5e935c674cf	M100
> sensor-16	0,9999900584046968	2022-04-13 00:58:17.9140	3efc4394-4c9e-481e-a363-53fe89e63011	M100

Run Recall KQL tools MyFreeCluster/MyDatabase

```
1
2   cluster('help').database('SampleIoTData').TransformedSensorsData
3   | summarize arg_max(Value, Timestamp, MachineId) by SensorName
4
```

Table 1 Add visual Stats

SensorName	Value	Timestamp	MachineId
> sensor-89	0,9999223243864502	2022-04-13 00:57:53.3020	M100
> sensor-27	0,999606616270891	2022-04-13 00:57:58.2030	M100
> sensor-103	0,999805667518576	2022-04-13 00:58:10.2870	M100
> sensor-100	0,999896718137476	2022-04-13 00:58:12.4870	M100
> sensor-179	0,9999164514638924	2022-04-13 00:58:13.7970	M100
> sensor-193	0,9995683859919664	2022-04-13 00:58:14.3400	M100

```

1
2   cluster('help').database('ContosoSales').SalesTable
3   | summarize arg_max(TotalCost, State) by FirstName, LastName
4   | sort by FirstName asc, LastName asc
5   | take 100
6
7

```

Table 1

FirstName	LastName	TotalCost	State
Aaron	Adams	915,08	California
Aaron	Alexander	915,08	Washing...
Aaron	Allen	509,78	British C...
Aaron	Baker	915,08	Washing...
Aaron	Bryant	914,67	California
Aaron	Butler	915,08	Oregon
Aaron	Campbell	914,67	California

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

```

1
2   let _sales = cluster('help').database('ContosoSales').SalesTable;
3   let _customers = cluster('help').database('ContosoSales').Customers;
4   union _sales, _customers
5   | sort by FirstName, LastName, City
6   | take 100

```

Taken from Ten Minute KQL YouTube Channel.

Homework

Environment: ADX Free Environment

Cluster: help

Database: ContosoSales

1. Write a query that unions the SalesFact and Customer Tables.
2. Display FirstName, LastName, CustomerKey, and StateorProvinceName.
3. Sample 100 records.
4. Post query in comments.

Solution:

Run Recall KQL tools MyFreeCluster/MyDatabase

```
1
2 let _salesFact = cluster('help').database('ContosoSales').SalesFact;
3 let _customers = cluster('help').database('ContosoSales').Customers;
4 union _salesFact, _customers
5 | summarize count() by FirstName, LastName, CustomerKey, CityName, StateProvinceName
6 | take 100
7
```

Table 1 + Add visual ⚡ Stats

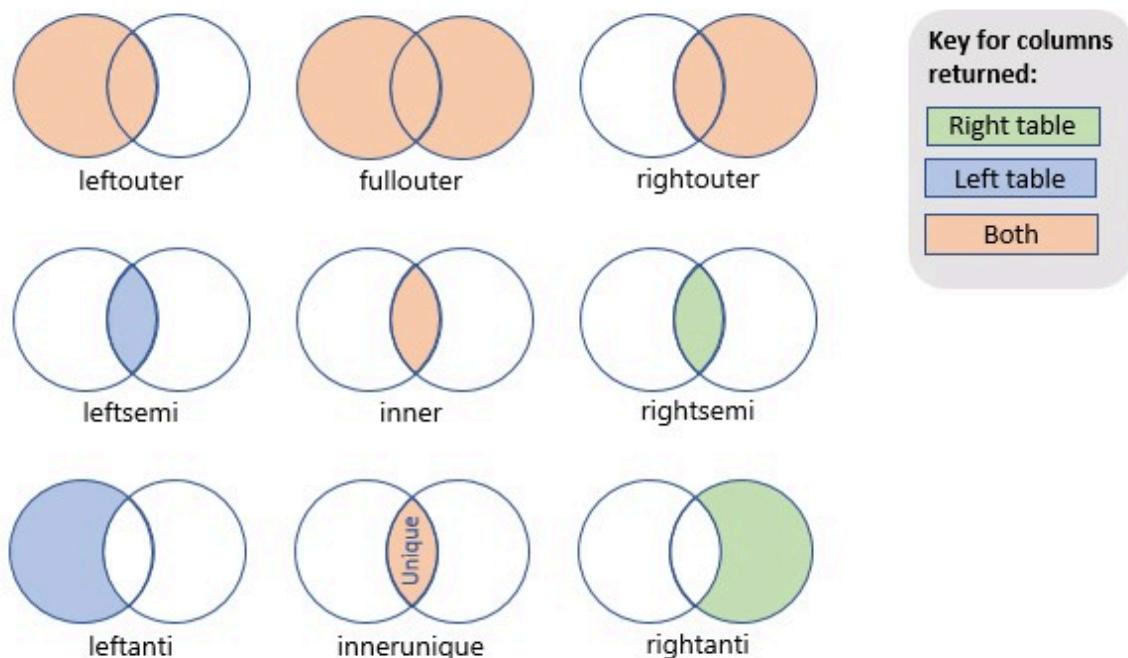
FirstName	LastName	CustomerKey	CityName	StateProvinceName	count_
Edward	Hernandez	145	Ballard	Washington	1
Angel	Stewart	236	Ballard	Washington	1
Cameron	Rodriguez	302	Ballard	Washington	1
Peter	Nara	771	Ballard	Washington	1
Theodore	Diaz	786	Ballard	Washington	1
Christine	Nara	879	Ballard	Washington	1
Latoya	Shen	885	Ballard	Washington	1
Stephanie	Cox	1 367	Ballard	Washington	1
Joshua	Lee	2 275	Ballard	Washington	1
Dalton	Wood	2 433	Ballard	Washington	1

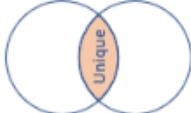
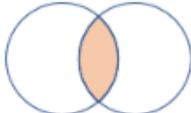
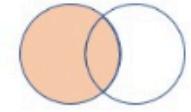
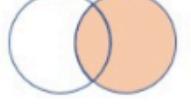
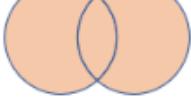
Join

Merge the rows of two tables to form a new table by matching values of the specified columns from each table.

Kusto Query Language (KQL) offers many kinds of joins that each affect the schema and rows in the resultant table in different ways. For example, if you use an `inner` join, the table has the same columns as the left table, plus the columns from the right table. For best performance, if one table is always smaller than the other, use it as the left side of the `join` operator.

The following image provides a visual representation of the operation performed by each join. The color of the shading represents the columns returned, and the areas shaded represent the rows returned.



Join flavor	Returns	Illustration
innerunique (default)	Inner join with left side deduplication Schema: All columns from both tables, including the matching keys Rows: All deduplicated rows from the left table that match rows from the right table	
inner	Standard inner join Schema: All columns from both tables, including the matching keys Rows: Only matching rows from both tables	
leftouter	Left outer join Schema: All columns from both tables, including the matching keys Rows: All records from the left table and only matching rows from the right table	
rightouter	Right outer join Schema: All columns from both tables, including the matching keys Rows: All records from the right table and only matching rows from the left table	
fullouter	Full outer join Schema: All columns from both tables, including the matching keys Rows: All records from both tables with unmatched cells populated with null	
leftsemi	Left semi join Schema: All columns from the left table Rows: All records from the left table that match records from the right table	
leftanti, anti, leftantisemi	Left anti join and semi variant Schema: All columns from the left table Rows: All records from the left table that don't match records from the right table	
rightsemi	Right semi join Schema: All columns from the right table Rows: All records from the right table that match records from the left table	
rightanti, rightantisemi	Right anti join and semi variant Schema: All columns from the right table Rows: All records from the right table that don't match records from the left table	

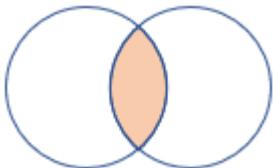
Let's create tables for practice:

```
let leftTable = datatable(id:int, value:string)
[
    0,"Hello",
    0,"Hola",
    1,"Salut",
    1,"Ciao",
    2,"Hei"
];
let rightTable = datatable(id:int, value:string)
[
    0,"World",
    0,"Mundo",
    1,"Monde",
    1,"Mondo",
    2,"Mailmaa"
];
```

	id ≡	value ≡
>	0	Hello
>	0	Hola
>	1	Salut
>	1	Ciao
>	2	Hei

	id ≡	value ≡
>	0	World
>	0	Mundo
>	1	Monde
>	1	Mondo
>	2	Mailmaa

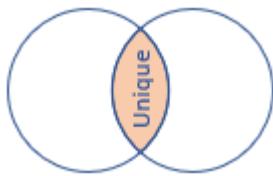
inner join: The inner join flavor is like the standard inner join from the SQL world. An output record is produced whenever a record on the left side has the same join key as the record on the right side.



```
leftTable
| join kind=inner rightTable on id
```

	<code>id ≡</code>	<code>value ≡</code>	<code>id1 ≡</code>	<code>value1 ≡</code>
>	0	Hola	0	World
>	0	Hello	0	World
>	0	Hola	0	Mundo
>	0	Hello	0	Mundo
>	1	Ciao	1	Monde
>	1	Salut	1	Monde
>	1	Ciao	1	Mondo
>	1	Salut	1	Mondo
>	2	Hei	2	Mailmaa

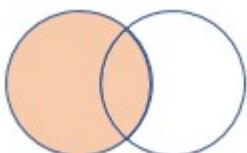
innerunique join: The `innerunique` join flavor removes duplicate keys from the left side. This behavior ensures that the output contains a row for every combination of unique left and right keys. By default, the `innerunique` join flavor is used if the `kind` parameter isn't specified. This default implementation is useful in log/trace analysis scenarios, where you aim to correlate two events based on a shared correlation ID. It allows you to retrieve all instances of the phenomenon while disregarding duplicate trace records that contribute to the correlation.



```
leftTable
| join kind=innerunique rightTable on id
```

	<code>id ≡</code>	<code>value ≡</code>	<code>id1 ≡</code>	<code>value1 ≡</code>
>	0	Hello	0	World
>	0	Hello	0	Mundo
>	1	Salut	1	Monde
>	1	Salut	1	Mondo
>	2	Hei	2	Mailmaa

leftouter join: The `leftouter` join flavor returns all the records from the left side table and only matching records from the right side table.



```

let leftTable = datatable(id:int, value:string)
[
    0,"Hello",
    0,"Hola",
    1,"Salut",
    1,"Ciao",
    2,"Hei",
    3,"Moi"
];
let rightTable = datatable(id:int, value:string)
[
    0,"World",
    0,"Mundo",
    1,"Monde",
    1,"Mondo",
    2,"Mailmaa",
    4,"Lume"
];

```

	id ≡	value ≡
>	0	Hello
>	0	Hola
>	1	Salut
>	1	Ciao
>	2	Hei
>	3	Moi

	id ≡	value ≡
>	0	World
>	0	Mundo
>	1	Monde
>	1	Mondo
>	2	Mailmaa
>	4	Lume

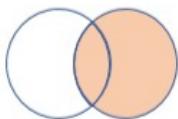
```

leftTable
| join kind=leftouter rightTable on id

```

	id ≡	value ≡	id1 ≡	value1 ≡
>	0	Hola	0	World
>	0	Hello	0	World
>	0	Hola	0	Mundo
>	0	Hello	0	Mundo
>	1	Ciao	1	Monde
>	1	Salut	1	Monde
>	1	Ciao	1	Mondo
>	1	Salut	1	Mondo
>	2	Hei	2	Mailmaa
>	3	Moi		

rightjoin: The `rightouter` join flavor returns all the records from the right side and only matching records from the left side. This join flavor resembles the [leftouter join flavor](#), but the treatment of the tables is reversed.



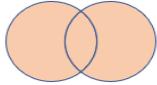
```
let leftTable = datatable(id:int, value:string)
[
    0,"Hello",
    0,"Hola",
    1,"Salut",
    1,"Ciao",
    2,"Hei",
    3,"Moi"
];
let rightTable = datatable(id:int, value:string)
[
    0,"World",
    0,"Mundo",
    1,"Monde",
    1,"Mondo",
    2,"Mailmaa",
    4,"Lume"
];
```

	id ≡	value ≡
>	0	Hello
>	0	Hola
>	1	Salut
>	1	Ciao
>	2	Hei
>	3	Moi

	id ≡	value ≡
>	0	World
>	0	Mundo
>	1	Monde
>	1	Mondo
>	2	Mailmaa
>	4	Lume

```
leftTable
| join kind=rightouter rightTable on id
|> +-----+-----+-----+-----+
|> | id ≡ | value ≡ | id1 ≡ | value1 ≡ |
|> +-----+-----+-----+-----+
|> | 0     | Hola      | 0     | World   |
|> | 0     | Hello     | 0     | World   |
|> | 0     | Hola      | 0     | Mundo   |
|> | 0     | Hello     | 0     | Mundo   |
|> | 1     | Ciao      | 1     | Monde   |
|> | 1     | Salut     | 1     | Monde   |
|> | 1     | Ciao      | 1     | Mondo   |
|> | 1     | Salut     | 1     | Mondo   |
|> | 2     | Hei       | 2     | Mailmaa |
|> |        |           | 4     | Lume    |
```

fullouter join: A fullouter join combines the effect of applying both left and right outer-joins. For columns of the table that lack a matching row, the result set contains `null` values. For those records that do match, a single row is produced in the result set containing fields populated from both tables.



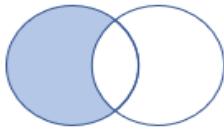
```
let leftTable = datatable(id:int, value:string)
[
    0,"Hello",
    0,"Hola",
    1,"Salut",
    1,"Ciao",
    2,"Hei",
    3,"Moi"
];
let rightTable = datatable(id:int, value:string)
[
    0,"World",
    0,"Mundo",
    1,"Monde",
    1,"Mondo",
    2,"Mailmaa",
    4,"Lume"
];
```

	id	value
>	0	Hello
>	0	Hola
>	1	Salut
>	1	Ciao
>	2	Hei
>	3	Moi

	id	value
>	0	World
>	0	Mundo
>	1	Monde
>	1	Mondo
>	2	Mailmaa
>	4	Lume

```
leftTable
| join kind=fullouter rightTable on id
|>
|+-----+-----+-----+-----+
| id   | value | id1  | value1 |
|-----+-----+-----+-----+
| 0    | Hola  | 0    | World |
| 0    | Hello | 0    | World |
| 0    | Hola  | 0    | Mundo |
| 0    | Hello | 0    | Mundo |
| 1    | Ciao  | 1    | Monde |
| 1    | Salut | 1    | Monde |
| 1    | Ciao  | 1    | Mondo |
| 1    | Salut | 1    | Mondo |
| 2    | Hei   | 2    | Mailmaa|
|      |        | 4    | Lume  |
| 3    | Moi   |      |        |
|-----+-----+-----+-----+
```

leftsemi join: The `leftsemi` join flavor returns all records from the left side that match a record from the right side. Only columns from the left side are returned.



```
let leftTable = datatable(id:int, value:string)
[
    0,"Hello",
    0,"Hola",
    1,"Salut",
    1,"Ciao",
    2,"Hei",
    3,"Moi"
];
let rightTable = datatable(id:int, value:string)
[
    0,"World",
    0,"Mundo",
    1,"Monde",
    1,"Mondo",
    2,"Mailmaa",
    4,"Lume"
];
```

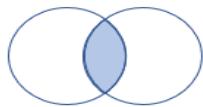
	id ≡	value ≡
>	0	Hello
>	0	Hola
>	1	Salut
>	1	Ciao
>	2	Hei
>	3	Moi

	id ≡	value ≡
>	0	World
>	0	Mundo
>	1	Monde
>	1	Mondo
>	2	Mailmaa
>	4	Lume

```
leftTable
| join kind=leftsemi rightTable on id
```

	id ≡	value ≡
>	2	Hei
>	1	Ciao
>	1	Salut
>	0	Hola
>	0	Hello

leftanti join: The `leftanti` join flavor returns all records from the left side that don't match any record from the right side. The anti join models the "NOT IN" query.



```
let leftTable = datatable(id:int, value:string)
[
    0,"Hello",
    0,"Hola",
    1,"Salut",
    1,"Ciao",
    2,"Hei",
    3,"Moi"
];
let rightTable = datatable(id:int, value:string)
[
    0,"World",
    0,"Mundo",
    1,"Monde",
    1,"Mondo",
    2,"Mailmaa",
    4,"Lume"
];
```

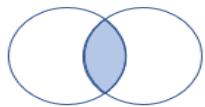
	id	value
>	0	Hello
>	0	Hola
>	1	Salut
>	1	Ciao
>	2	Hei
>	3	Moi

	id	value
>	0	World
>	0	Mundo
>	1	Monde
>	1	Mondo
>	2	Mailmaa
>	4	Lume

```
leftTable
| join kind=leftsemi rightTable on id
```

	id	value
>	3	Moi

rightsemi join: The rightsemi join flavor returns all records from the right side that match a record from the left side. Only columns from the right side are returned.



```
let leftTable = datatable(id:int, value:string)
[
    0,"Hello",
    0,"Hola",
    1,"Salut",
    1,"Ciao",
    2,"Hei",
    3,"Moi"
];
let rightTable = datatable(id:int, value:string)
[
    0,"World",
    0,"Mundo",
    1,"Monde",
    1,"Mondo",
    2,"Mailmaa",
    4,"Lume"
];
```

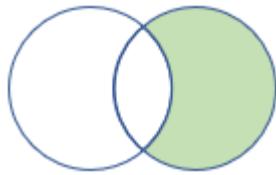
	id ≡ value ≡
>	0 Hello
>	0 Hola
>	1 Salut
>	1 Ciao
>	2 Hei
>	3 Moi

	id ≡ value ≡
>	0 World
>	0 Mundo
>	1 Monde
>	1 Mondo
>	2 Mailmaa
>	4 Lume

```
leftTable
| join kind=rightsemi rightTable on id
```

	id ≡ value ≡
>	0 World
>	0 Mundo
>	1 Monde
>	1 Mondo
>	2 Mailmaa

rightanti join: The `rightanti join` flavor returns all records from the right side that don't match any record from the left side. The anti join models the "NOT IN" query.



```
let leftTable = datatable(id:int, value:string)
[
    0,"Hello",
    0,"Hola",
    1,"Salut",
    1,"Ciao",
    2,"Hei",
    3,"Moi"
];
let rightTable = datatable(id:int, value:string)
[
    0,"World",
    0,"Mundo",
    1,"Monde",
    1,"Mondo",
    2,"Mailmaa",
    4,"Lume"
];
```

	id ≡	value ≡
>	0	Hello
>	0	Hola
>	1	Salut
>	1	Ciao
>	2	Hei
>	3	Moi

	id ≡	value ≡
>	0	World
>	0	Mundo
>	1	Monde
>	1	Mondo
>	2	Mailmaa
>	4	Lume

```
leftTable
| join kind=rightanti rightTable on id
```

	id ≡	value ≡
>	4	Lume