MUST LEARN

KQL

THE SERIES

ROD TRENT
SENIOR MICROSOFT SECURITY ADVOCATE
MICROSOFT

This is part of an ongoing series to educate about the simplicity and power of the Kusto Query Language (KQL). If you'd like the 90 second-post commercial recap that seems to be a standard part of every TV show these days…

The full series index (including code and queries) is located here:

https://aka.ms/MustLearnKQL

Contents

Must Learn KQL Part 1: Tools and Resources

After hearing that our customers' largest barrier to using things like Defender, Microsoft Sentinel and even reporting for Intune is KQL, the query language, that was a wake-up call for me. And, of course, (if you know me) I want to do something about it. KQL is a beautifully simple query language to learn. And believe me – if I can learn it, there's no question that you can learn it. I feel bad that there's just not enough knowledge around it because I've taken for granted that everyone already had the proper resources to become proficient. But that's not the case.

Internally, plans are being developed now to make KQL learning a bigger focus and you'll see new education around this query language start to take shape in various areas on the Microsoft properties and elsewhere. So, that's good news for everyone.

There's bits and pieces already scattered about the Internet, but they are seemingly now difficult to identify and locate.

So, as a first step in a series that I'll be writing called "**Must Learn KQL** ", I want to supply some good resources that can be used to accomplish the other things I'll talk about going forward. Some of these I use every day. Some I use only when the need arises, but they're valuable, nonetheless. This is a working document, so expect updates over time. This is not a definitive list by any means, so if you have other resources not listed here that you find valuable and believe others would benefit, let me know and I'll add them in.

Stay tuned as I map out this series. Of course, since my area of forte at Microsoft is security, the series will be security focused. So, the knowledge you gain will help you with our security platforms but also anything data centric that utilizes KQL.

One last tidbit of a tip… I use Microsoft Edge's Collections feature quite a bit. This is an extremely useful tool for capturing and grouping topics. If you find any of the links below valuable, I suggest using Edge Collections so you can always come back to them later.

Reference

The code repository for this series (GitHub)

Kusto Query Language Reference Guide

Azure Monitor Logs table reference

Marcus Bakker's Kusto Query Language (KQL) – cheat sheet

SQL to Kusto cheat sheet

Splunk to Kusto Query Language map

Write your first query with Kusto Query Language (Learn module)

Practice Environments

KQL Playground – only need a valid Microsoft account to access.

Data Explorer – not security focused. Contains things like geographical data and weather patterns. Exercises for this can be found in the *Learn Azure Sentinel* book below.

Actual Books

Learn Azure Sentinel: Integrate Azure security with artificial intelligence to build secure cloud systems – this book uses Data Explorer (see above) for hands-on exercises.

Azure Sentinel in Action: Architect, design, implement, and operate Azure Sentinel as the core of your security solutions – this book is the next edition of the one just above and also used Data Explorer for hands-on examples.

Tools

Kusto.Explorer – a rich desktop application that enables you to explore your data using the Kusto Query Language in an easy-to-use user interface.

Kusto CLI – a command-line utility that is used to send requests to Kusto and display the results.

Visual Studio Code with the Kusto extensions pack

Real-Time KQL – eliminates the need to ingest data first before querying by processing event streams with KQL queries as events arrive, in real-time

getschema operator – As I noted in Part 5 of this series: this is *the Rosetta stone of KQL operators*. When used, getschema displays the Column Name, Column Ordinal, Data Type, and Column Type for a table. This is important information for filtering data. Part 5 talks about this.

Blogs, Websites, and Social

#MustLearnKQL – the official hashtag of this series

The #KQL hashtag on Twitter

The #365daysofkql hashtag on Twitter

Kusto King

Video

TeachJing's KQL Tutorial Series

Recon your Azure resources with Kusto Query Language (KQL)

How to start with KQL?

Azure Sentinel webinar: KQL part 1 of 3 – Learn the KQL you need for Azure Sentinel

Azure Sentinel webinar: KQL part 2 of 3 – KQL hands-on lab exercises

Azure Sentinel webinar: KQL part 3 of 3 – Optimizing Azure Sentinel KQL queries performance

Querying Azure Log Analytics (with KQL)

GitHub Query Examples

My GitHub repo for Microsoft Sentinel KQL

The official Microsoft Sentinel repo

Wortell's KQL queries

Clive Watson's KQL queries and workbooks

Matt Zorich's (the originator of the #365daysofkql Twitter hashtag) KQL queries

Must Learn KQL Part 2: Just Above Sea Level

To start the journey learning KQL in this *Must Learn KQL* series, it's helpful to understand where the name KQL came from and why the reference makes so much sense. Once you understand the idea behind the query language, a lightbulb should go off and prepare you for the rest of the series through an expanded scope of learning capability.

Plus, not everyone knows about this, so you'll be the cool kid. And, if you ever play *Trivial Pursuit* and this question comes up, you'll win the pie piece and possibly the entire game. How can that not be good knowledge?

The question?

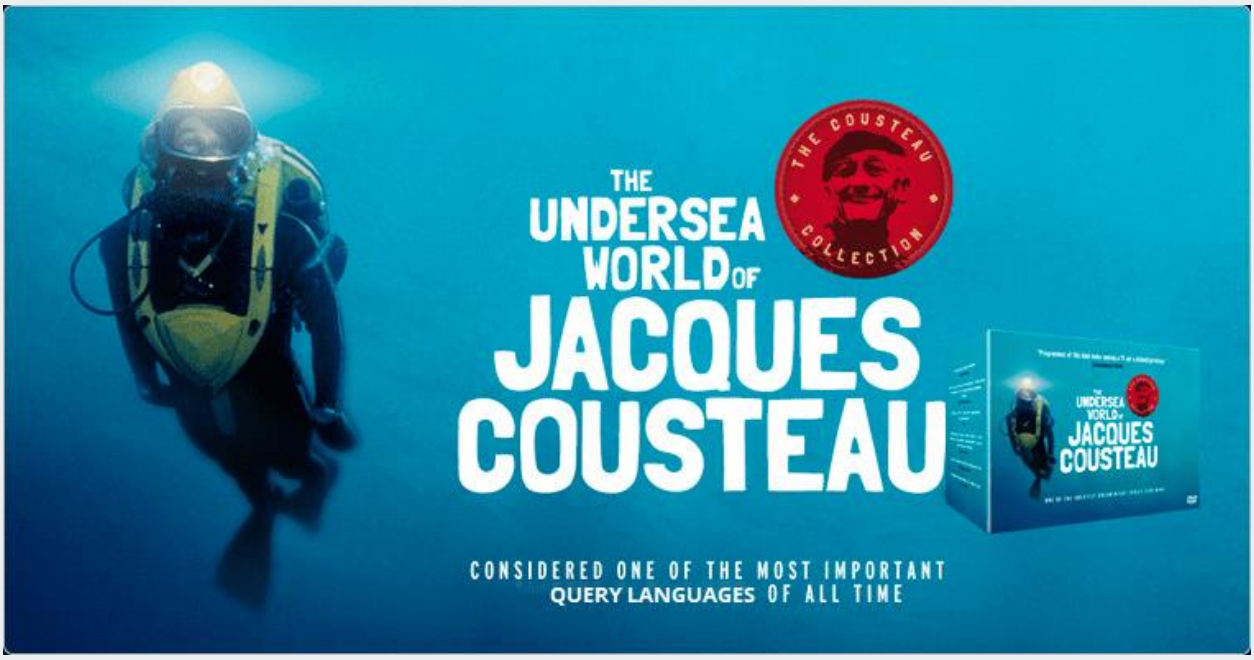
Where does the name *Kusto* come from? (from **K**usto **Q**uery **L**anguage)

To help explain this, I harken back to my childhood. Bear with me for a minute…

Growing up, my family was *one-of-those-families* that attended church anytime the church doors were open. As such, the majority of my parents' friends were at church. This meant that they would spend time before and after church services catching up with their friends, sometimes in a local restaurant where they'd all gather to have pie and coffee. Of course, Facebook didn't exist then, so in-person connections were even more important. Well…and there was pie. My mom, in particular, wanted to catch up with everyone she hadn't seen in a few days so this meant that our round-trip from home to church and back could take 3-4 hours.

On Sunday nights this was particularly problematic for me in that I wanted to rush home to catch TV shows like the Six Million Dollar Man, The Magical World of Disney, Mutual of Omaha's Wild Kingdom, and the TV show that's the topic of our discussion here: The Undersea World of Jacques Cousteau…

That's right. KQL is named after the undersea pioneer, Jacques Cousteau.



I loved this TV series. It was absolutely enthralling to me to understand that an entire world existed beneath the ocean waves and this unknown world was being brought to me by this wonderful, thick-accented explorer each week who dedicated his life to discovering what existed beneath the surface depths.

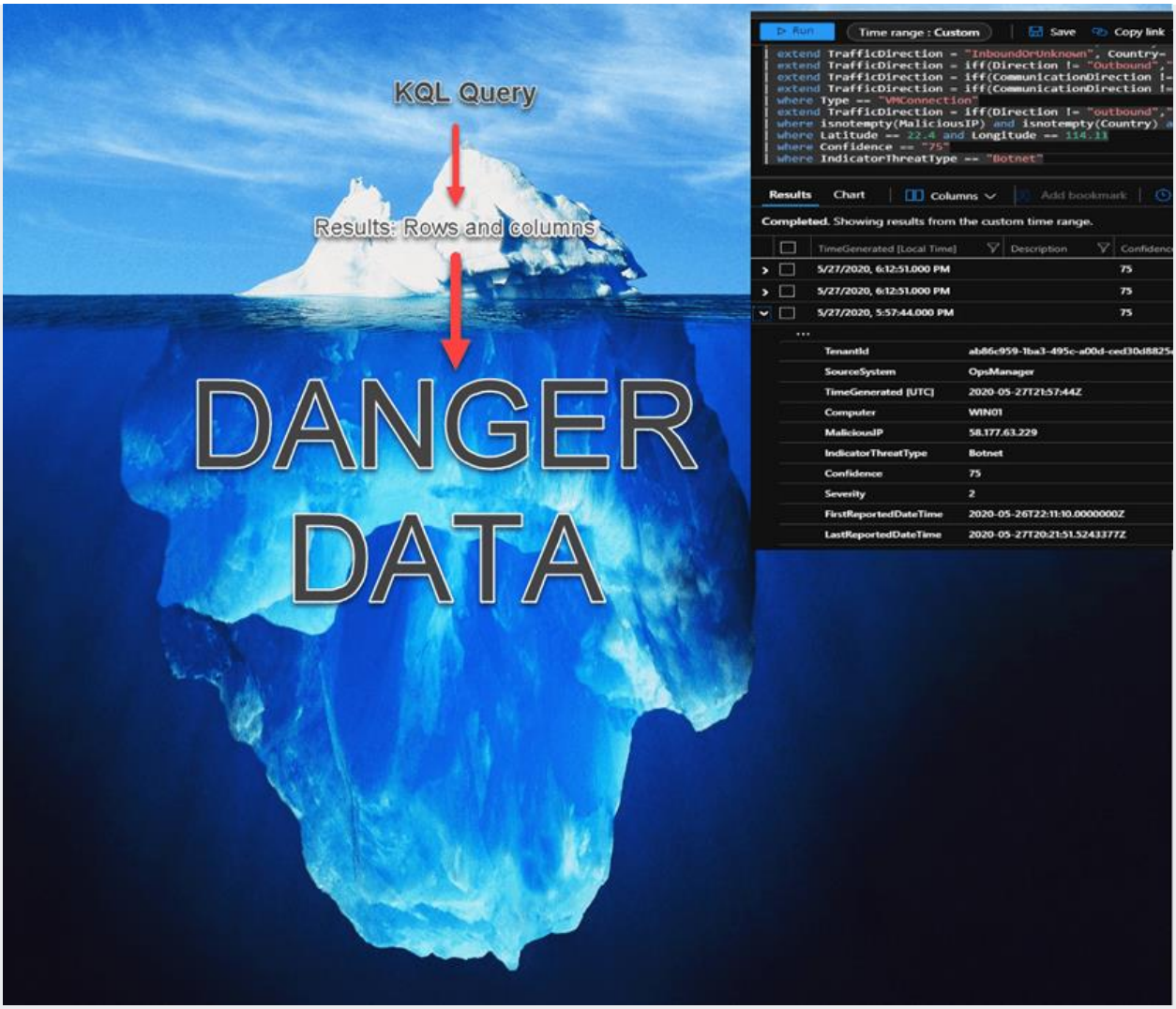
So, as you can imagine, I tried my dead-level best every Sunday night to rush my mom along. It didn't always work and was mostly just annoying, and you can bet I caught a few groundings from my insistence. But, still, this topic of discovering the undiscoverable drove me to concoct every type of machination imaginable to get home sooner on Sunday nights. I can't tell you the number of times I faked illness on Sunday afternoon in attempt to stay home Sunday night. And, as you can imagine my mom quickly caught on and instituted a policy that if I stayed home on Sunday nights, I couldn't go to school on Monday. Which…at the time…I truly loved school, so that halted that plan. Give me a few years, and that wouldn't have worked. Timing is everything.

So, KQL is named after Jacques Cousteau. Even today, you can find evidence of this in our own Azure Monitor Docs. If you go to the datatable operator page right now, you'll still find a reference to Mr. Cousteau in an example that lists his date of birth, the date he entered the naval academy, when he published his first book entitled "The Silent World: A Story of Undersea Discovery and Adventure," and the date of his passing.

So, I hope you're catching on to this. If not, what is it that we are trying to accomplish when we query data tables for security purposes? What is that we're trying to accomplish though Hunting exercises and operations?

The answer? We are exploring the depths of our data. We are attempting to *surface* the critical and necessary security information that will tell us about potential exposure through simple, powerful queries.

Much like the story of the failed voyage of the Titanic. It wasn't the beautiful, pristine, easy-to-see and avoid iceberg mass that existed above the surface of the ocean that sunk the unsinkable ship and sent over 1,500 people to their grave. No, it's was the huge mass under the surface that the captain and crew couldn't see and couldn't swerve to avoid that doomed the luxury passenger liner.



And, like that, it's the information that exists underneath the viewable rows and columns of data in our tables that we need to expose to identify threats and compromise and use to guard the gates. Just the initial rows and columns of exposed data isn't enough. We must delve into the depths of the data to find actionable information. And we need to do it quickly.

I hope all this makes sense.

It's as important to know *why* we do things, sometimes, as *how* to do them. Like Jacque Cousteau, security folks are explorers. We are mining the depths of the data no one sees to protect the environment against ever-growing and constantly evolving threats. We are discovering the undiscoverable.

KQL is an amazing and important piece of this capability. KQL was developed to take advantage of the power of the cloud through clustering and compute. Using this capability, KQL is designed as a well-performing tool to help surface critical data quickly. This a big part of why it works so well and outshines many other query languages like it. KQL was built for the cloud and to be used against large data sets.

As a security person, you know that if a threat exists in the environment, you are on the clock to discover it, report it, investigate it, and remediate. A poorly performing query language can be the biggest barrier to that and become a security flaw. I've sat with customers who use other query languages and other SIEM-like tools that thought it was status quo that query results would take hours or sometimes days. When I showed that KQL produced those same results in seconds, they were astonished. So, the technology and infrastructure behind the query language is also critically important.

In the next post, I'll talk about the actual structure of a query. Even though the structure can deviate, understanding a common workflow of a KQL query can have powerful results and help you develop the logic needed to build your own workflows when it's time to create your own queries. In addition to being well-performing to enhance efficiency, the query language itself is simple to use and learn which, in turn, makes for more efficiency.

So, while we're *Just Above Sea Level* in this post (I hope you now appreciate the reference), we'll be using KQL as the sonar and diving bell to search the depths of our data.

Must Learn KQL Part 3: Workflow

As I noted in Part 2 of this *Must Learn KQL* series…

Even though the structure can deviate, understanding a common workflow of a KQL query can have powerful results and help you develop the logic needed to build your own workflows when it's time to create your own queries.

Rod Trent, November 18, 2021

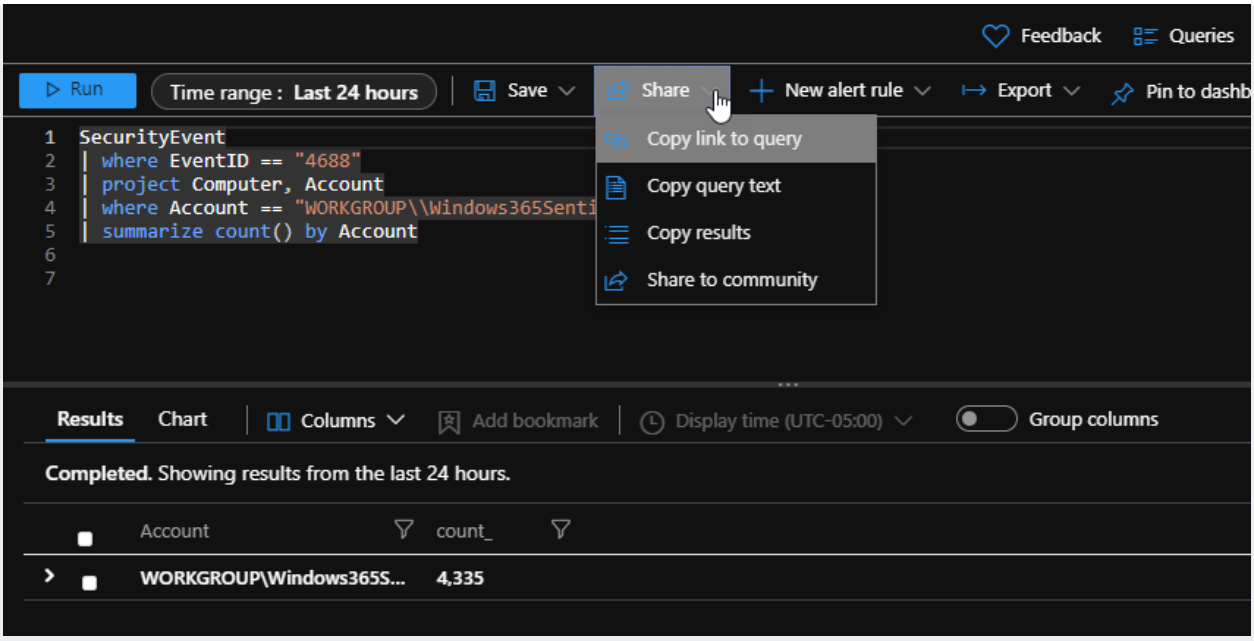
The workflow (some folks call it *logic*, others call it *anatomy*, even others call is something else) is a big step into wrapping your mind around how to produce a KQL query. Just like a developer, assigning uniform, repeatable steps ensure you're not missing something and that your query results will produce the information you are looking to capture.

I tell customers all the time that it's not necessary to be a pro at creating KQL queries. It's OK not to be a pro on day 1 and still be able to use tools like Microsoft Sentinel to monitor security for the environment. If you understand the workflow of the query and can comprehend it line-by-line, you'll be fine. Because ultimately, the query is unimportant. Seriously. What's important for our efforts as security folks is the *results* of the query. The results contain the critical information we need to understand if a threat exists and then – if it does exist – how that threat occurred from compromise to impact.

Now, those that go on to develop their own queries and own Sentinel Analytics Rules after becoming a KQL pro will be much more capable. And that should be your goal, too. BUT don't get hung up on that. Again, it's about the results.

We've made it so crazily easy to share KQL queries that it's quite possible you may never have to create your own KQL query (*aside: I highly doubt it but COULD BE possible*).

In a future post in this series, I'll go over the actual interface you use to write and run the KQL queries in-depth but suffice to say that almost every service in Azure has a Logs *blade* (option in the Azure portal interface/menu) to accommodate querying that service's logs. This area provides for saving your queries, but also to *share* your queries.



Share your queries

Because of this built-in capability, many of our customers regularly share their creations with each other, other colleagues, to their own blogs and GitHub repos, and even to the official Microsoft Sentinel GitHub repository (https://aka.ms/ASGitHub). In Part 1 of this series, I supplied links to these and more. So, to prove my point…yes, it's absolutely possible you might not have to write your own KQL query for a long time.

So, because of that, it becomes even more critical that you at least understand the workflow. Again, if you can read a query line-by-line and determine that the results will produce what you are looking for, you're golden. If, through your newfound understanding, the query can't produce your requirements, you can modify it by line instead of a wholesale adaption. This should be your first KQL goal: read queries.

Through this series, I'll provide queries for you to use and get hands-on experience because I believe in learning by doing. We'll be using the links in the **Practice Environments** section in Part 1 for the hands-on. But focus initially more on the structure and logical workflow.

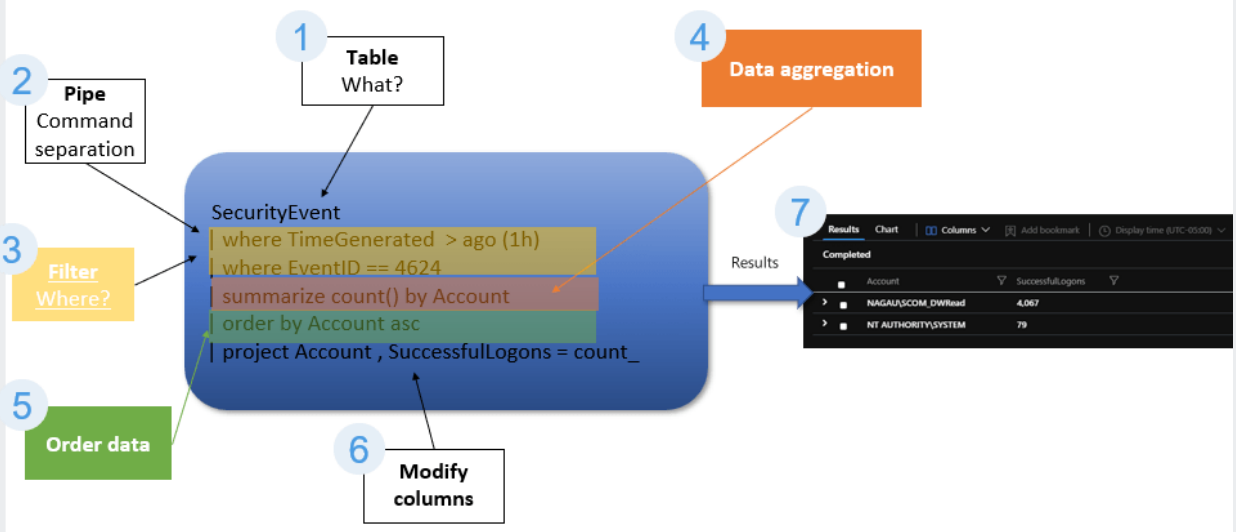
And, with that…

A Common KQL Workflow

To get started on the journey to learning KQL, let's look at the standard workflow of a common search query. Not the search *operator* (I'll talk about in the next post), but the search query. This is the query structure we use to search, locate information, and produce results.

The following represents the common workflow of a KQL search query.

P.S. I've enabled image linking in this post so you can click or tap to open the image in a larger view. So, you can open the image in a new window or new tab to better follow along.



Let's break this query down by the steps.

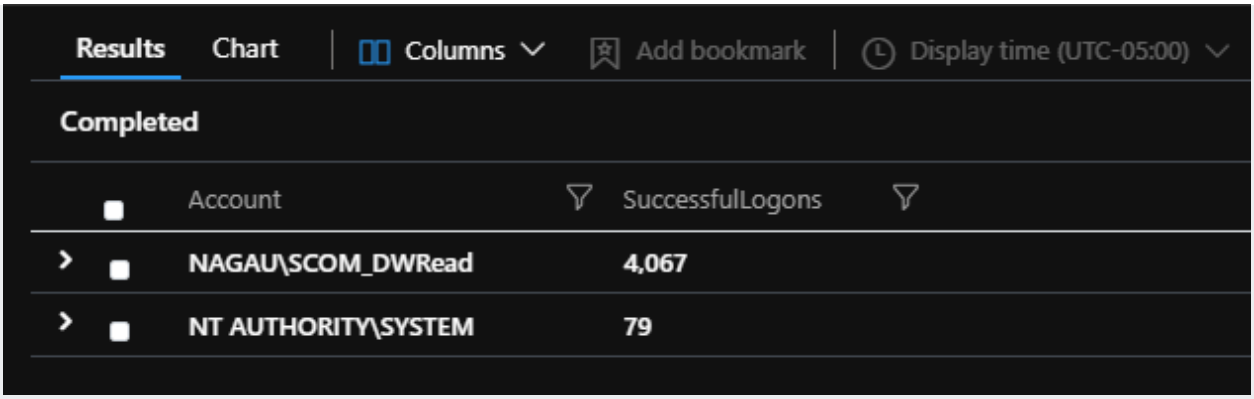
1. The first step is to identify the table we want to query against. This table will contain the information that we're looking for. In our example here, we're querying the SecurityEvent table. The SecurityEvent table contains security events collected from windows machines by Microsoft Defender for Cloud or Microsoft Sentinel. For a full list of all services tables, see the Azure Monitor Logs table reference (also available in Part 1).
2. The **pipe** (|) character (the shifted key above the Enter key on most keyboards) is used to separate commands issued to the query engine. You can see here that each command is on its own line. It doesn't have to be this way. A KQL query can be all one single line. For our efforts, and as a recommendation, I prefer each command on its own line. For me, it's just neater and more organized which makes it easier to troubleshoot when a query fails or when I need to adjust the query to produce different results.

3. Next, we want to filter the data in some way. If I simply entered the table and ran that as its own, single query, it will run just fine. Doing that returns all rows and columns (up to a limit – which I believe is now 50,000 rows) of the data stored in the table. But our goal is get exact data back. As an analyst looking for threats, we don't want to have to sift through 50,000 rows of data. No, we want to look for specific things. The **Where operator** is one of the best ways to accomplish this. You can see here in the example that I'm filtering first by when the occurrence happened (*TimeGenerated*) and then (remember the pipe character – *another line, another command*) by a common Windows Event ID (4624 – successful login).
4. The next step in our workflow is to provide data aggregation. What do we want to do with this filtered data? In our case in the example, we want to create a count of the Accounts (*usernames*) that produced a successful login (*EventID 4624*) in the last 24 hours (*TimeGenerated*).
5. Next let's tell the query engine how we want to order the results. Using the Order operator, I'm telling the query engine that when the results are displayed, I want it shown in alphabetical order by the Account column. The 'asc' in the query in the Order Data step is what produces this ordering. If we wanted descending order we'd use 'desc'. Don't worry, we'll dig deeper into each of these operators as we go along in the series.
6. Generally, the last thing that I'll do with this search query is tell the query engine exactly what data I want displayed. The Project operator is a powerful command. We'll dig deeper into this operator later in this series, but for our step here, I'm telling the query engine that after all my filtering, data aggregation, and ordering, I only want to display two columns in my results: Account and SuccessfulLogins

So, let's recap what this query accomplished…

It searched our stored security events in the *SecurityEvent* table for all Accounts that had a successful login in the last hour and chose to display only the Account and number of successful logins per Account in alphabetical order.

7. Our search query output is exactly that:



Search query output

See that? The Account column is in alphabetical order *ascending* and the SuccessfulLogons column shows how many times each Account successfully logged in.

If you need to, jump back through each step above until you get a good understanding of the workflow. Again, this is very common, and you'll see this structure many times working with Microsoft Sentinel and Defender products. Remember, it's about the results. If you can look at this example and get a good feel that you understand how the results were accomplished, line-by-line, you're on your way.

I invite you, though, to take this example and copy/paste it into a Logs environment to test. You can have this query to play with it in your own Microsoft Sentinel environment, or using the KQL Playground I provided as a resource in Part 1.

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

This query is also available from the GitHub repository for this blog series: https://cda.ms/3fS

I'd like to share one extra tidbit with you that you might find helpful as you start testing this KQL query example in your own, or our, environment.

Every language (scripting, coding, querying) has the capability to add comments or comment-out code through special characters. When the query, scripting, or development engine locates these characters, it just skips them. KQL has this same type of character. The character for KQL is the double forwardslash, or **//**

When you start testing this post's KQL query example, comment-out a line or two (put the double forwardslash at the beginning of the line) and rerun the query just to see how eliminating a single line can alter the results. You'll find that this is an important technique as you start developing your own KQL queries. I'll talk about this more later, too.

In the next post (Part 4) I'll talk through another, yet just as powerful, way to search for information using KQL that is a top pocket tool for Threat Hunters.

And, then I'll come back for Part 5 and show how to tie together both search methods to create the full operation of hunting to Analytics Rule. But don't worry, that's not the end. I have no clue how many parts this series be. A lot of it depends on you.

Must Learn KQL Part 4: Search for Fun and Profit

Now that we have some understanding of the workflow (from Part 3) under our belts, I'm going to deviate from that for a brief minute in this post and then I'll bring it back together in Part 5 and combine Parts 4 and 5 to provide something extra meaningful to show you how it all fits together like an unsolved Hardy Boys mystery novel. Hopefully, you're starting to see that my efforts here are logical and designed to accumulate enough knowledge that is necessary to move to the next plane of understanding.

What I want to do in this post, is give you something you can use today. When I'm done here, you should be able to take the knowledge and the query snippets to do your own hunting – or, rather, look inside your own environment to get an understanding of what is happening that's worth exposing and investigating.

One of the easiest ways to get started with KQL is the **search operator**. In Part 3, I talked through the structure and workflow of a *search query*. In this post, I'll talk about the search *operator* (or command) and how it could be the most powerful KQL operator in the universe but will always be the best tool in the toolbelt to start any search operation.

Search is the first operator I reach for when trying to verify if something exists within the environment. In fact, our whole goal for using KQL as a security tool is to answer the following questions:

1. Does it exist?
2. Where does it exist?
3. Why does it exist?
4. *BONUS: There's a final question to this that's not part of* this *KQL series, but one that's important to the total equation and one that should be part of your SOC processes. That question is: How do we respond?*

If you click or tap the image to open it in a larger view, you'll see how the power of the search operator enables you to answer these questions.

Its starts with an idea or theory that "something" exists in the environment. You may have gotten this idea from a dream or nightmare that someone in your organization is performing nefarious activities. But, most likely, the idea came from a news report or a post on social media from a trusted source about a nation-state actor being active with a new kind of ransomware.

Once these reports are available, someone (like Microsoft) will supply the Indicators of Compromise (IOCs) so you can search your environment to see if they exist. IOCs could be a number of things including filenames, file hashes, IP addresses, domain names, and more.

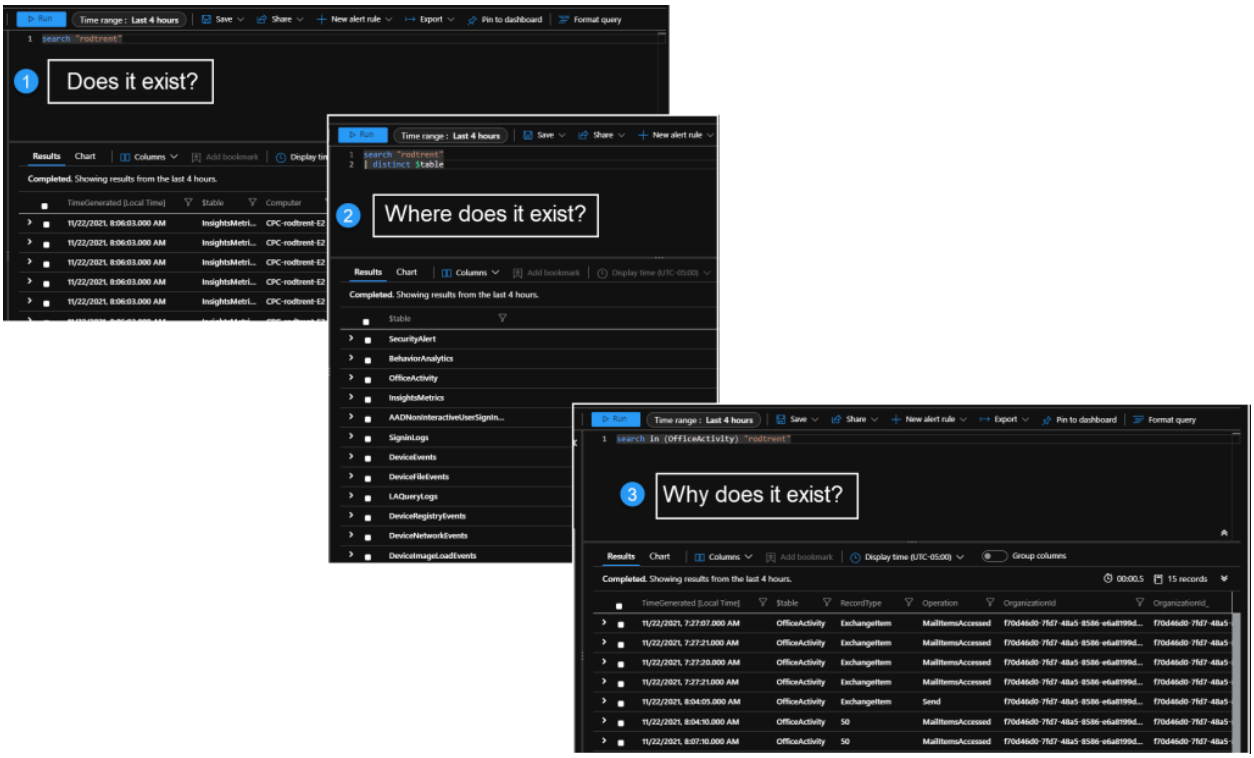
If they don't exist, you move on. If any of them do exist, you start to dig deeper to figure out where they exist, so you can, for example, quarantine systems or users, or block IP addresses or domains.

And, then you need to determine why they exist. Did a specific user click on something they shouldn't have clicked on in an email? Or did a threat actor successfully compromise a Domain Controller through control over a service or

elevate user account? Could it be that there is more impact on your environment than you originally thought?

All of this can be exposed through the simple process of search using the search operator.

Let's walk through this together with a few simple queries that you can take and use to test your own environment. (*click or tap the image to open the larger version in a new browser tab to following along*)



Who, What, When, Where?

In *step 1* in the image, I'm performing a simple search for a username. In this case, it's an ego search – I'm searching in my own environment for my own activity. This could be an IOC that you want to search for. Just replace my name with the string of text you want to expose in the results.

```
search "rodtrent"
```

As you can see in the image, my search produced results, telling me that this *thing* I searched for *does* exist in my environment.

Since it does exist, I want to understand where it exists. I do this by making a simple adjustment to my original query by adding a line that tells the query engine to just show me the specific tables that my IOC exists in. This will give me a good indication of what type of activity it was. *Step 2* shows...

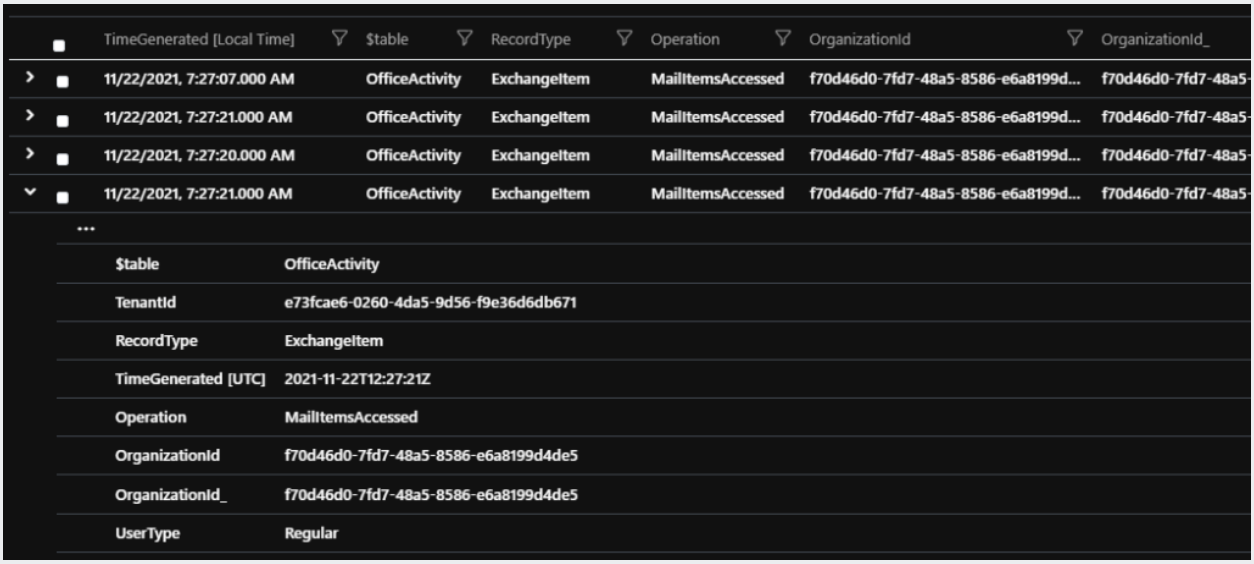
```
search "rodtrent"
| distinct $table
```

Let's assume that I'm looking for user activity because the reported threat is malware. I know that user activity is most generally recorded and contained in a few places including Microsoft Office and Defender for Endpoint.

In my example in *Step 3* in the image, I've adjusted my search operator query to focus only on the OfficeActivity table. Here what that looks like:

```
search in (OfficeActivity) "rodtrent"
```

Now that I have my results of rodtrent's activity in the OfficeActivity table, I can begin sifting through the rows and columns of data to learn more about the occurrence and to start to tune my query even more.



TimeGenerated [Local Time]	$table	RecordType	Operation	OrganizationId	OrganizationId_
11/22/2021, 7:27:07.000 AM	OfficeActivity	ExchangeItem	MailItemsAccessed	f70d46d0-7fd7-48a5-8586-e6a8199d...	f70d46d0-7fd7-48a5-
11/22/2021, 7:27:21.000 AM	OfficeActivity	ExchangeItem	MailItemsAccessed	f70d46d0-7fd7-48a5-8586-e6a8199d...	f70d46d0-7fd7-48a5-
11/22/2021, 7:27:20.000 AM	OfficeActivity	ExchangeItem	MailItemsAccessed	f70d46d0-7fd7-48a5-8586-e6a8199d...	f70d46d0-7fd7-48a5-
11/22/2021, 7:27:21.000 AM	OfficeActivity	ExchangeItem	MailItemsAccessed	f70d46d0-7fd7-48a5-8586-e6a8199d...	f70d46d0-7fd7-48a5-

$table	OfficeActivity
TenantId	e73fcae6-0260-4da5-9d56-f9e36d6db671
RecordType	ExchangeItem
TimeGenerated [UTC]	2021-11-22T12:27:21Z
Operation	MailItemsAccessed
OrganizationId	f70d46d0-7fd7-48a5-8586-e6a8199d4de5
OrganizationId_	f70d46d0-7fd7-48a5-8586-e6a8199d4de5
UserType	Regular

Results from the OfficeActivity table

When we come back for Part 5, I'll show you how to turn your search query into a workflow like I talked about in Part 3.

One last thing for this post. I mentioned that user activity is generally reported from the Microsoft Office and Defender for Endpoint tables. I've given you examples for searching the OfficeActivity table. But Defender for Endpoint is more than one table. In fact, Defender for Endpoint consists of the following 10 tables: DeviceEvents, DeviceFileCertificateInfo, DeviceFileEvents, DeviceImageLoadEvents, DeviceInfo, DeviceLogonEvents, DeviceNetworkEvents, DeviceNetworkInfo, DeviceProcessEvents, and DeviceRegistryEvents.

Fortunately, the KQL search operator supports the wildcard character. So, you can search for those IOCs across the entire Defender for Endpoint solution by doing the following:

```
search in (Device*) "rodtrent"
```

I've given you examples for searching the OfficeActivity table. But Defender for Endpoint is more than one table. In fact, Defender for Endpoint consists of the following 10 tables: DeviceEvents, DeviceFileCertificateInfo, DeviceFileEvents, DeviceImageLoadEvents, DeviceInfo, DeviceLogonEvents, DeviceNetworkEvents, DeviceNetworkInfo, DeviceProcessEvents, and DeviceRegistryEvents.

And, incidentally, if you have the Defender for 365 Data Connector enabled for Microsoft Sentinel and you enable the Microsoft Defender for Office 365 logs, the OfficeActivity table isn't the only Microsoft Office data you can query. Enabling these logs gives you access to EmailEvents, EmailUrlInfo, EmailAttachmentInfo, and EmailPostDeliveryEvents tables which means you can take advantage of the search operator's wildcard capability here, too.

All the query code in this post is contained in the series' GitHub repo here: https://cda.ms/3gG

P.S. Enjoying this series? Share it with someone!

Must Learn KQL Part 5: Turn Search into Workflow

Now, that we've talked about using the Search operator in Part 4 to answer those three basic SOC analyst questions of: 1) Does it exist? 2) Where does it exist? and, 3)

Why does it exist?, we can take that learning and the results of that type of query and meld it with the standard search query structure I talked about in Part 3.

In part 4, I ended with a query to locate activity by a user called "*rodtrent*". I found that this *rodtrent* person had performed potentially strange activity in the OfficeActivity table (the table for Office 365 activity) that needs to be checked out. As shown, the search operator is a powerful tool to find things of interest. The results of the search operator query were thousands of rows of data. That's inefficient.

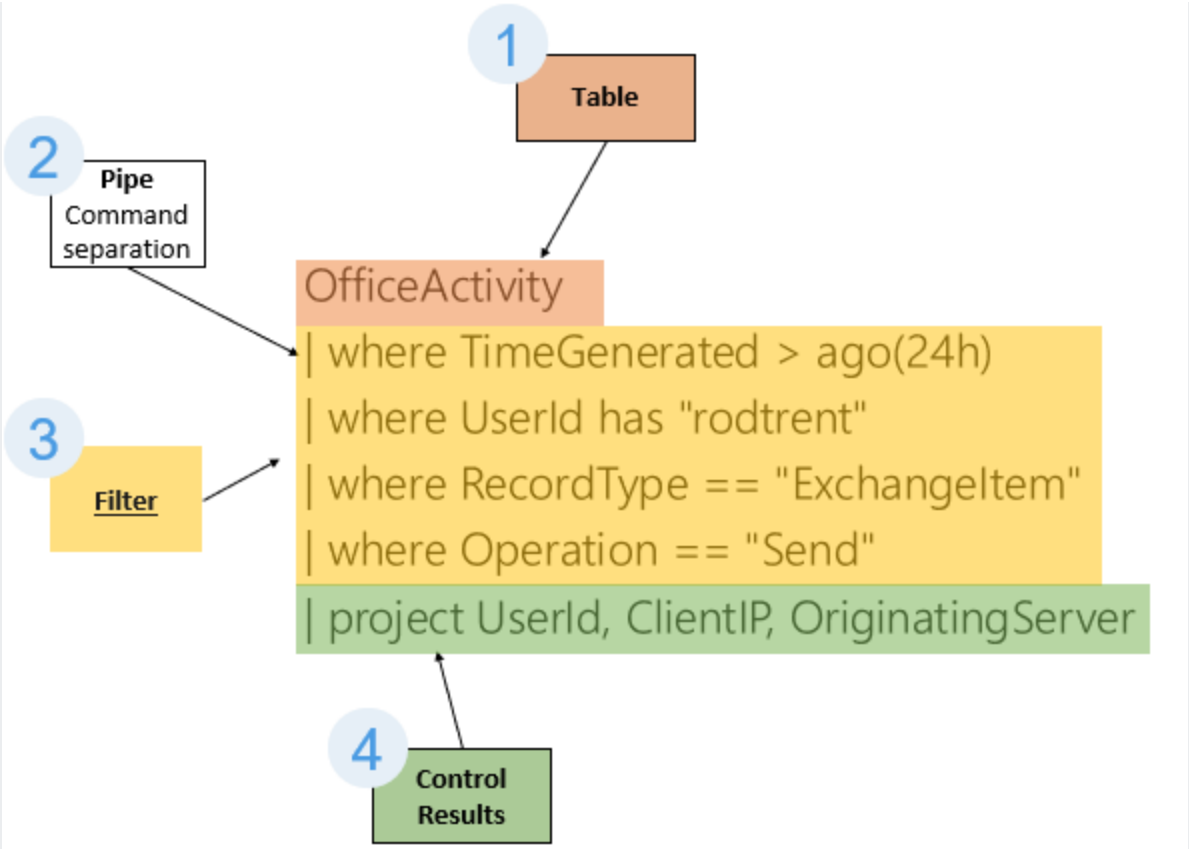
So, now that we've found something interesting, we want to use the structure of the Search Query to pare down the results to minimize the effort and workload to identify that that *something interesting* is something notable and worth investigating.

If you need to, open up Part 3 in a new Window or browser Tab to review the Search Query Workflow as I walk through the next section.

In the following example, note that this is a non-issue situation, but I want to start with a basic Search query before we start building toward more complex queries in future posts to get a fully rounded understanding of the "why" behind why we do this. The one below is even simpler than the one discussed in Part 3 where I also talk about aggregating and ordering data. I'll come back to those concepts later, particularly when I get into creating your own in-query visualizations like pie and bar charts. No, for our efforts in this post, I want to focus on how easy it is to filter the data. Again, KQL isn't hard, and some of your most powerful queries may only be a few lines of code.

Turning your hunting operations into more formal Search structure queries is the building blocks for creating your own Analytics Rules in Microsoft Sentinel. Analytics Rules should be precise logic to enable your operations to focus exactly where it needs to focus; and because, capturing data outside of what was intended is both inefficient and problematic for isolating actual security events.

The example (*available from the series' GitHub repo at: https://cda.ms/3jd*):



New search query

Let's break this new Search query down together like was done in Part 3. This one, again, is even a tad bit simpler than when describing the Search workflow, but as you'll see, it's the where operator that is sometimes our biggest, most powerful, and best workhorse and pal for tuning efficient results.

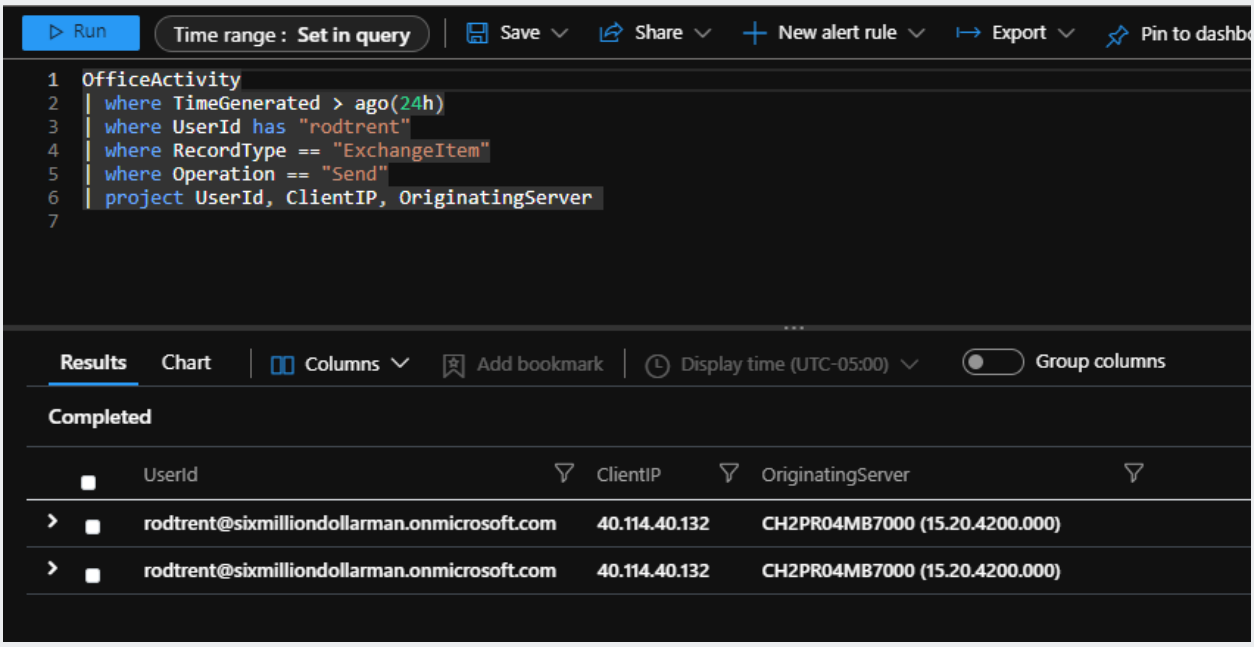
1. The first step in our workflow is to query the OfficeActivity table. If you remember, from our time together in Part 4, we're looking for user activity (in our case the user "*rodtrent*") in Microsoft Office.
2. As per the discussion in Part 3 on workflow, I want to highlight the importance of the pipe command once again. I don't rehash the importance here. If you missed it, jump to Part 3 to catch up.
3. In step 3 of the new Search query, I'm filtering how the query engine searches. I'm first telling to only look at data in the last 24 hours (TimeGenerated), then only looking through a column called UserId for the string "rodtrent", then telling the query engine to only capture Exchange activity from the RecordType data column, and finally

pinpointing the search to only Send operations. So, essentially, I'm looking for any emails that rodtrent sent in the last 24 hours.

- Filtering the data is the key to everything. <= Read that again. Filtering the data that is returned produces exact, actionable data. It also improves the results performance of our queries. Where the search operator may return thousands of rows of data in 15 seconds (or less), by properly filtering the data to return exactly what is necessary returns just the number of rows of data we asked for which greatly improves the processing time. Where the search operator may have taken 15 seconds, our new Search structure query will take 5 seconds or less. The Where operator is the key to this operation. Learn it. Know it. Keep the Where operator reference page handy: https://cda.ms/3jh.

4. Finally, I'm using the project operator to control exactly what is show in the results window. In this case, I only want to show the user, the user's IP address, and the server where the email originated from.

The results?



Search query results

As you can plainly see in the query results, this matches exactly what my query proposed.

EXTRA: We saw in Part 4 with our Search operator, how results from our queries are in named rows and columns of data. And, you see here in this post, how I'm constantly filtering against known column names in the tables. Some might wonder how I come up with those schema names. Of course, it helps that I work with these tables constantly, but I do have a couple secrets to share. First off, as noted in Part 1, I use the Azure Monitor Logs table reference quite a bit. However, there's also the Rosetta stone of KQL operators: getschema

Running a simple...

```
OfficeActivity
| getshema
```

...will produce a list of all the named columns of a specific table. The example above displays all the named columns of the OfficeActivity table. Each of these columns can be used in your where operator filtering efforts.

...

In this post, I've given you a simple query to practice with. In Part 6, I'll come back and dig into the actual interface for developing your own queries (instead of just running the ones I've given).