Kusto Query Language (KQL): Kusto Query Language (KQL) is a read-only query language designed for analysing large datasets efficiently. It is widely used in Microsoft Defender, Azure Sentinel, and Azure Data Explorer. KQL enables analysts to fetch logs, telemetry data, and operational events from these platforms, providing fast insights, anomaly detection, and incident investigation. KQL is declarative and works with a tabular mindset, meaning all queries return tables that can be filtered, aggregated, or joined for deeper analysis. In practice, KQL is used to:

- Detect security anomalies, such as unusual login patterns or suspicious network connections
- Correlate events across multiple data sources to investigate incidents
- Monitor system performance and user activities in real time
- Build dashboards and reports for operational insights
- Query large-scale telemetry data efficiently, without impacting system performance

For example, to count the number of successful logins per user in the last 24 hours:

```
SecurityEvent

| where TimeGenerated > ago(24h)

| where EventID == 4624

| summarize login_count = count() by TargetUserName

| where login_count > 100
```

This query aggregates login events, filters out benign activity, and highlights potential security risks. KQL's simplicity and power make it an essential tool for data exploration, security monitoring, and operational analytics.

1. KQL Syntax: The Noun-Verb Structure

KQL is verb oriented. Think of it as:

```
<Table> | <Operator/Verb> <Parameters>
```

- Noun: The table or dataset you are querying (e.g., SecurityEvent, DeviceProcessEvents)
- Verb: The action you perform (e.g., where, summarize, join, project)

Example:

```
SecurityEvent

| where EventID == 4624
```

Meaning: From the SecurityEvent table, show rows where EventID equals 4624.

2. Core Nouns (Tables & Columns)

- **Tables:** SecurityEvent, DeviceProcessEvents, DeviceNetworkEvents, EmailEvents, UserLoginEvents
- **Columns:** TimeGenerated, UserName, Computer, EventID, FileName

3. Scalar Operators (Comparing Values)

Operator	Meaning	Example
==	Equals	where EventID == 4624
!=	Not equals	where FileName != "notepad.exe"
> / < / >= / <=	Greater/Less than	where RemotePort > 1024
=~	Case-insensitive equals	where AccountName =~ "alice"
has	Contains substring	where FileName has "powershell"
!has	Does not contain	where FileName !has "notepad"
in	Is in a list	where UserName in ("Alice","Bob")
!in	Is not in a list	where UserName !in ("SYSTEM","Admin")
contains / !contains	Substring match	where FileName contains "mimikatz"
startswith / endswith	Matches beginning/end	where FileName startswith "powershell"
matches regex	Regex pattern	where FolderPath matches regex ".*TEMP.*"

Example with multiple operators:

```
DeviceProcessEvents

| where FileName contains "powershell"

| where InitiatingProcessFileName !contains "explorer"

| where Timestamp > ago(1d)
```

4. Tabular Operators (Verbs)

4.1 Filtering Data

```
SecurityEvent

| where EventID == 4624

| where TargetUserName !endswith '$'
```

4.2 Selecting Columns

```
SecurityEvent

| project TimeGenerated, TargetUserName, Computer
```

4.3 Sorting

```
SecurityEvent

| where EventID == 4624

| sort by TimeGenerated desc
```

4.4 Aggregating

```
SecurityEvent

| where EventID == 4624

| summarize login_count = count() by TargetUserName

| where login_count > 100
```

Other aggregation functions: sum(Column), avg(Column), min(Column), max(Column), dcount(Column)

5. Let Statements (Variables for Reusability)

```
let suspiciousIPs = dynamic(["192.168.1.1","10.10.0.1"]);

let startTime = ago(24h);

DeviceNetworkEvents

| where LocalIP in (suspiciousIPs)

| where Timestamp > startTime
```

- **Scalars:** let threshold = 1024
- **Tables:** let NTLMEvents = SecurityEvent | where EventID == 4624
- **Expressions:** let avgLogins = toscalar(...)

6. Handling Nulls

```
DeviceNetworkEvents

| where RemotePort < 1024 or isnull(RemotePort)
```

7. Combining Tables

7.1 Union

```
union (

    DeviceProcessEvents | where Timestamp > ago(1d)

),

(

    DeviceEvents | where Timestamp > ago(1d)

)
```

7.2 Joins

Inner Join:

```
UserLoginEvents
| join kind=inner UserProfiles on UserId
| where Location !~ UserLocation
Left Outer Join:
UserProfiles
| join kind=leftouter UserLoginEvents on UserId
| project UserId, UserName, LastLogin=TimeGenerated
```

8. Real-World Security Use Cases

8.1 Detect Anomalous Logins

```
let whitelisted_users = dynamic(['SYSTEM','svc-sccm']);
UserLoginEvents
| join kind=inner UserProfiles on UserId
| where Location !~ UserLocation
| where TargetUserName !in (whitelisted_users)
| summarize anomaly_count = count() by TargetUserName
| where anomaly_count > 1
```

8.2 Detect Malware Execution

```
DeviceProcessEvents
| where FileName contains "mimikatz"
| summarize count() by Computer, InitiatingProcessFileName
```

8.3 Detect High-Volume Email Sending

```
EmailEvents
| where TimeGenerated > ago(24h)
| summarize emails_sent = count() by Sender
| where emails_sent > 100
```

8.4 Detect Rare Events

```
SecurityEvent
| summarize event_count=dcount(TargetUserName) by EventID
| where event_count < 3
```

9. Additional KQL Syntax for Mastery

9.1 Top-N Records

```
DeviceProcessEvents
| summarize count() by FileName
| top 5 by count_
```

9.2 Time-Based Analysis

```
SecurityEvent
| where TimeGenerated > ago(7d)
| summarize login_count = count() by bin(TimeGenerated, 1h)
```

9.3 Regex Filtering

```
DeviceProcessEvents
| where FileName matches regex "(?i)^(powershell|cmd)\.exe$"
9.4 Detecting Unusual Ports
DeviceNetworkEvents
| summarize port_count = dcount(RemotePort) by Computer
| where port_count > 50
```

10. KQL Best Practices

- Use let statements for reusable queries and lists
- Filter early before unions or joins for performance
- Handle nulls with isnull() to prevent false negatives
- Use whitelisting for benign accounts or hosts
- Pre-aggregate before threshold-based anomaly detection
- Join tables for context, not just combination