🔍 Kustonaut's KQL Cheatsheet

A quick reference to Kuto Query Language for Eventhouse in Real-Time Intelligence, Azure Data Explorer (ADX), Azure Monitor, and Microsoft Security solutions.

📝 Compiled by: Akshay Dixit (kustonaut) | 💻 Source Code: http://aka.ms/kustonautKQLcheatsheet

📖 Introduction

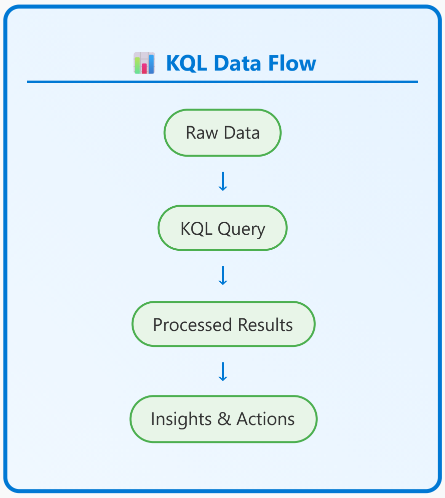
Kusto Query Language (KQL) is Microsoft's powerful open-source query language designed for analyzing large volumes of structured, semi-structured, and unstructured data. Originally developed for internal Microsoft use and released publically with Azure Data Explorer, KQL is now the standard query language across Microsoft's analytics, security and monitoring ecosystem.

🎯 Where KQL is used?

- **Eventhouse in Real-time Analytics** - Lightening fast analytics on streaming data
- **Azure Data Explorer** - Big data log analytics platform
- **Azure Monitor** - Application and infrastructure monitoring
- **Microsoft Sentinel** - Security information and event management
- **Azure Resource Graph** - Azure resource management
- **Application Insights** - Application performance monitoring
- **Log Analytics** - Centralized log data analysis

🚀 Why KQL in the era of Artificial Intelligence

- **Scale** - Handle petabytes of data with sub-second response times
- **Flexibility** - Works with structured, semi-structured, and unstructured data
- **Integration** - Native integration with Microsoft's AI and ML services
- **Visualization** - Built-in charting, graph analytics and dashboard capabilities



📊 KQL Data Flow

Raw Data
↓
KQL Query
↓
Processed Results
↓
Insights & Actions

💡 **Learning Path:** Start with basic filtering and aggregation, then progress to joins and advanced analytics functions.

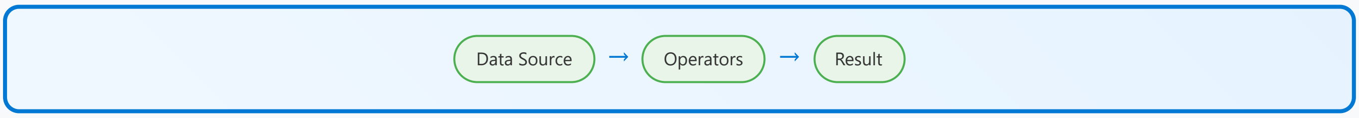
📚 Official Documentation

Primary Resource: https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/

KQL Tutorial: https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/tutorial

Best Practices: https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/best-practices

📋 Basic Structure



Data Source → Operators → Result

```
TableName | where condition | project columns | take 10
```

💡 Always start with a table name, then chain operators with " | " (pipe operator)

🔧 Essential Operators

🔍 where

Filters rows based on specified conditions. Most important operator for performance.

```
TableName | where Column == "Value"
```

📖 Official Docs

📋 project

Selects specific columns and can create computed columns. Reduces data volume.

```
| project Name, Age, City
```

📖 Official Docs

🎯 take

Limits the number of rows returned. Useful for sampling and testing queries.

```
| take 100
```

📖 Official Docs

🔢 count

Returns the number of rows in the input table. Essential for metrics.

```
| count
```

📖 Official Docs

📊 summarize

Groups rows and applies aggregation functions. Core operator for analytics.

```
| summarize count() by Category
```

📖 Official Docs

🔄 sort

Orders rows by one or more columns. Alias: order by.

```
| sort by Timestamp desc
```

📖 Official Docs

➕ extend

Adds calculated columns to the result set. Preserves original columns.

```
| extend NewColumn = Column1 + Column2
```

📖 Official Docs

🔗 join

Combines rows from two tables based on matching values. Multiple join types available.

```
| join (Table2) on CommonColumn
```

📖 Official Docs

🎭 distinct

Returns unique combinations of specified columns. Removes duplicates.

```
| distinct Column1, Column2
```

📖 Official Docs

🏆 top

Returns the top N rows sorted by specified columns. Combines sort and take.

```
| top 10 by Count desc
```

📖 Official Docs

🔀 union

Combines rows from multiple tables. Tables must have compatible schemas.

```
union Table1, Table2
```

📖 Official Docs

🎨 render

Visualizes query results as charts. Multiple chart types supported.

```
| render timechart
```

📖 Official Docs

⏰ Time Functions

```
// Time ranges where TimeGenerated >
ago(1h) where TimeGenerated >
now(-1d) where TimeGenerated
between(datetime(2024-01-01) ..
datetime(2024-01-02))
```

```
// Time formatting extend Hour =
bin(TimeGenerated, 1h) extend Date =
format_datetime(TimeGenerated, 'yyyy-
MM-dd')
```

📖 ago() | now() | bin() | format_datetime()

🔍 Filtering Patterns

String Operations

```
where Name contains "error" where
Name startswith "App" where Name
endswith ".exe" where Name matches
regex @"[0-9]+"
```

Numeric & Lists

```
where Status in ("Success",
"Warning") where Count between (10 ..
100) where isnotnull(Field)
```

📖 contains | startswith | in operator | between

📊 Aggregations

```
// Basic aggregations | summarize
count(), sum(Amount), avg(Duration),
max(Timestamp), min(Size) by Category
```

```
// Advanced aggregations | summarize
percentile(Duration, 95),
dcount(UserId), countif(Status ==
"Error") by bin(TimeGenerated, 1h)
```

🔗 Joins

📊 Table A 🔗 Join 📊 Table B

```
// Inner join Table1 | join (Table2)
on CommonField // Left join Table1 |
join kind=leftouter (Table2) on
CommonField // Join with conditions
Table1 | join (Table2 | where Active
== true) on $left.ID == $right.UserID
```

📝 String Functions

```
extend Upper = toupper(Name), Lower =
tolower(Name), Length =
strlen(Message), Split = split(Path,
"/"), Extract = extract(@"(\d+)", 1,
Text), Replace = replace(@"\s+", " ",
Text), Trim = trim(" ", Text)
```

📖 toupper() | strlen() | split() | extract()

🚀 Advanced Operators

🔀 union & distinct

```
union Table1, Table2 | distinct
Column1, Column2
```

📖 union docs | distinct docs

🪟 Window Functions

```
| extend RowNumber = row_number() |
extend RunningSum =
row_cumsum(Amount)
```

📖 row_number docs | row_cumsum docs

🔀 Conditional Logic

```
| extend Category = case( Value >
100, "High", Value > 50, "Medium",
"Low")
```

📖 case function docs

🔄 mv-expand

```
| mv-expand ArrayColumn
```

📖 mv-expand docs

🎯 evaluate

```
| evaluate pivot(Status, count(),
Category)
```

📖 pivot plugin docs

⚡ Performance Optimization

🎯 Strategy	📝 Implementation	💡 Impact
Early Filtering	Put where clauses first	Reduces data processed
Time Range Limits	Use ago() for recent data	Leverages time indexing
Column Selection	Use project early	Reduces memory usage
Efficient Joins	Join smaller tables first	Minimizes computation
Avoid Wildcards	Use contains instead of *	Better index usage

🎯 Common Patterns

📈 Top N Analysis

```
| summarize Total = count() by Category | top 10 by Total desc
```

📊 Time Series Analysis

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

🔍 Error Rate Calculation

```
| summarize Total = count(), Errors = countif(Status == "Error") | extend ErrorRate =
todouble(Errors) / Total * 100
```

🔄 Pivot Operations

```
| evaluate pivot(Status, count(), Category)
```

🔐 Security Queries

```
// Failed logins SigninLogs | where
ResultType != "0" | summarize count()
by UserPrincipalName | top 10 by
count_
```

```
// Suspicious activities
SecurityEvent | where EventID in
(4625, 4648, 4719) | summarize
count() by Account, Computer
```

📊 Visualization

```
// Chart types | render timechart | render piechart | render barchart | render columnchart | render scatterchart
```

> Example: Daily user activity trend
>
> ```
> Users | summarize count() by bin(TimeGenerated, 1d) | render timechart
> ```

🚀 Quick Reference

🔤 Data Types

```
bool, int, long, real string,
datetime timespan, guid
dynamic (JSON)
```

📅 Time Literals

```
1d = 1 day 1h = 1 hour 1m = 1
minute 1s = 1 second 1ms = 1
millisecond
```

🧮 Math Functions

```
abs(), ceil(), floor()
round(), sqrt(), pow() log(),
exp(), pi()
```

🔄 Array Functions

```
array_length() array_slice()
array_concat() mv-expand
```

🎯 Pro Tips for AI Product Managers

💡 **Data-Driven Decisions:** Use KQL to analyze user behavior patterns, feature adoption rates, and performance metrics to inform product roadmap decisions.

💡 **Incident Response:** Master time-based queries to quickly identify and analyze system anomalies, user impact, and service degradation patterns.

💡 **User Insights:** Combine multiple data sources with joins to create comprehensive user journey analytics and identify optimization opportunities.