Microsoft

Kusto Query Language In A Day

Conditions and terms of use

For a more detailed description of confidentiality, see the slide at the end of the presentation.

Lessons 1: Introduction to Kusto Query Language

Learning Units covered in this Module

Entities and Data Types

Basic Aggregate Functions
Graph Output and Visualization

KQL Tools
- Kusto Explorer
- Kusto CLI

Objectives

After completing this Learning, you will be able to:

1

Understand….

- ✓ What is Kusto Query Language
- ✓ Anatomy of Kusto Query

2

Understand….

- ✓ Entities and Datatypes
- ✓ Basic Kusto Query

3

Understand….

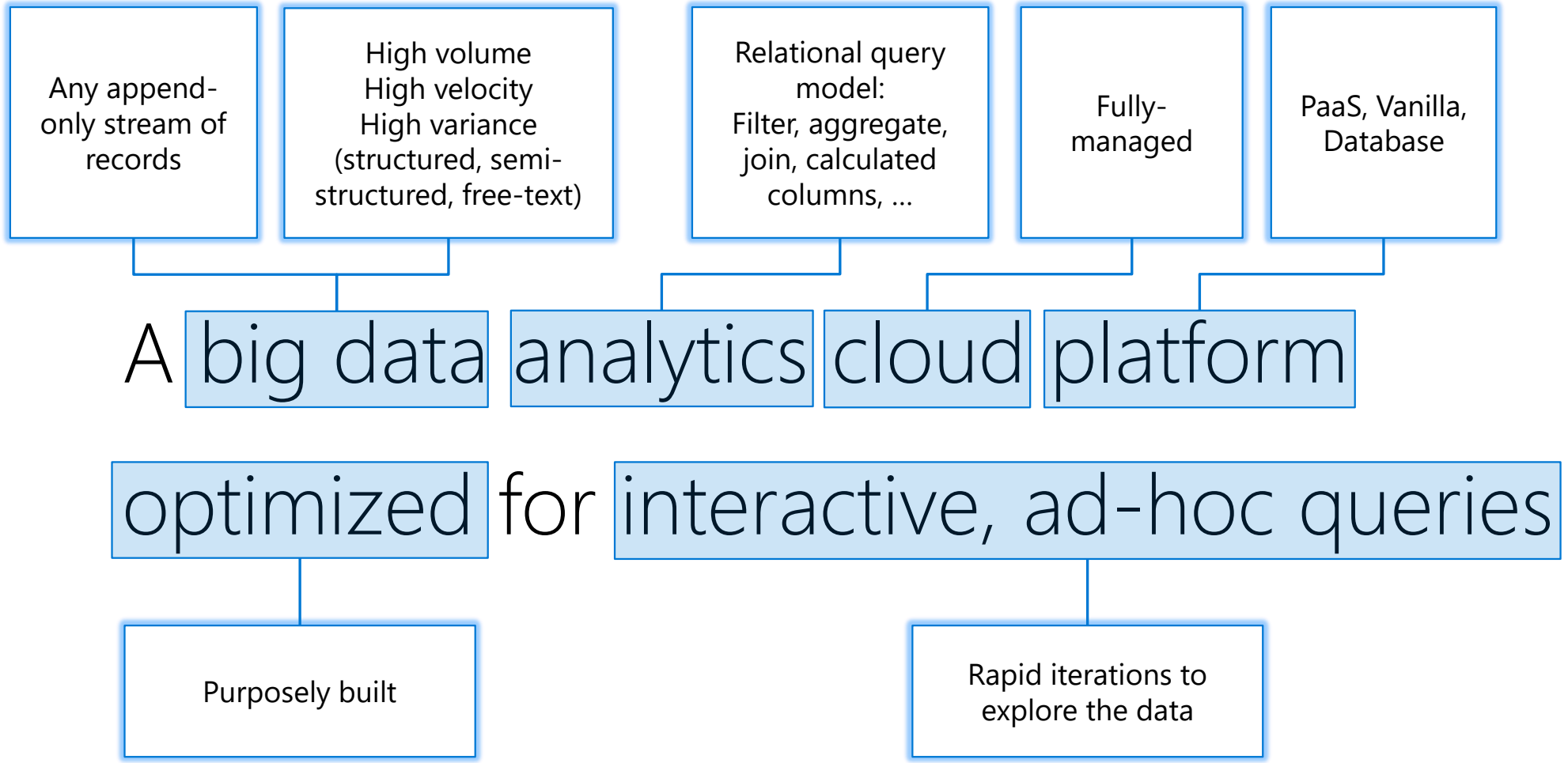
- ✓ Essential Elements like basic aggregate functions, sort, take and summarize operators
- ✓ Chart output using render operator
- ✓ KQL Tools
    1. Kusto Explorer
    2. Kusto CLI

Necessity is the mother of invention

- Text search capabilities
  - Inverted text index
- Performance & analytics
  - Column store DB
- Query agility
  - Ad-hoc query language: SQL-like (tables, columns)
  - Ability to “pipe” (flow) the data from one operator to the next

Jacques-Yves Cousteau -> Kusto

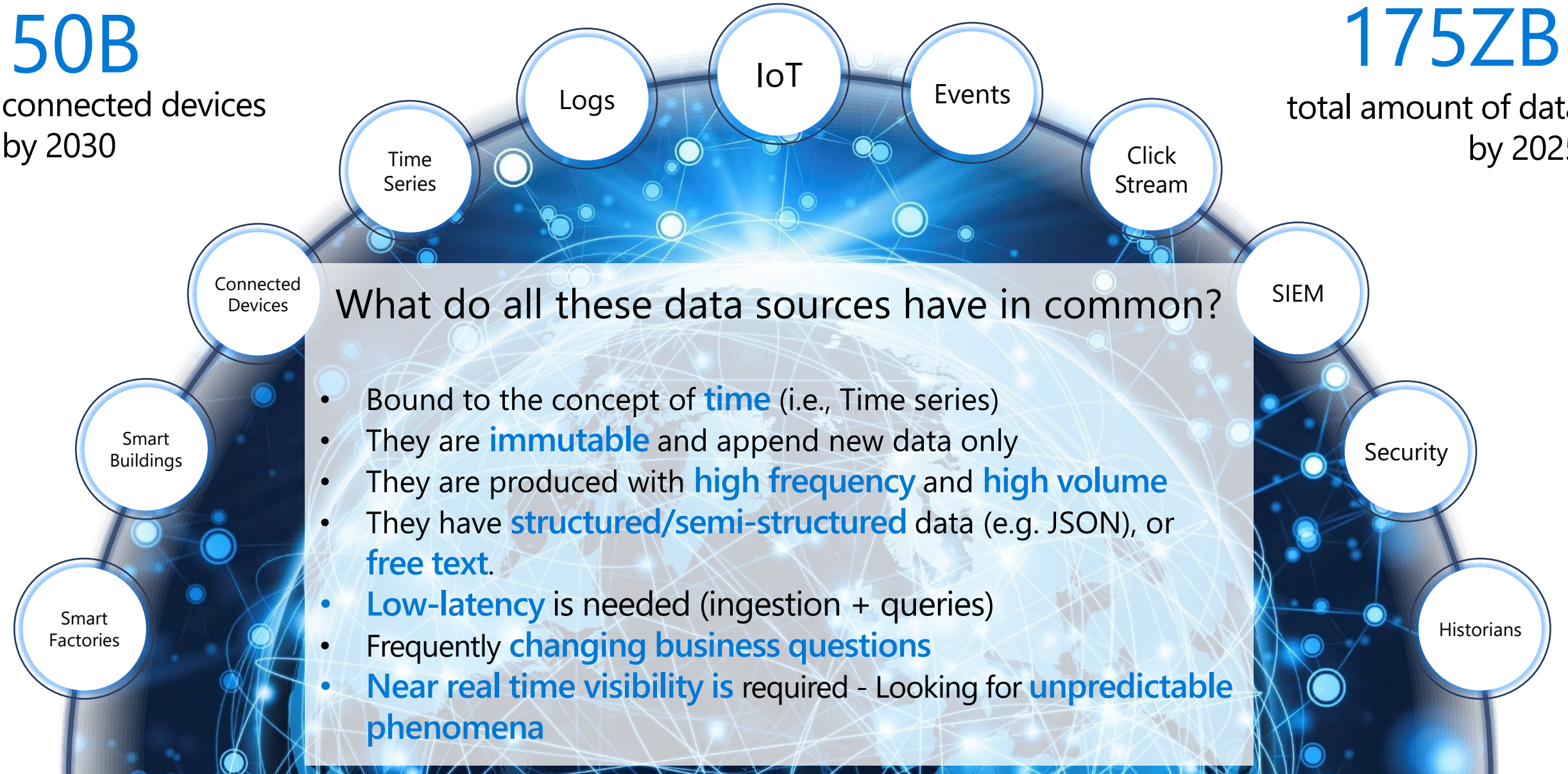
Introducing Azure Data Explorer!



Telemetry - A key data for Digital Transformation

50B
connected devices
by 2030

175ZB
total amount of data
by 2025



What do all these data sources have in common?

- Bound to the concept of **time** (i.e., Time series)
- They are **immutable** and append new data only
- They are produced with **high frequency** and **high volume**
- They have **structured/semi-structured** data (e.g. JSON), or **free text**.
- **Low-latency** is needed (ingestion + queries)
- Frequently **changing business questions**
- **Near real time visibility is** required - Looking for **unpredictable phenomena**

Proven Technology

In production since 2015 for internal Microsoft workload, GA since Feb 2019.

Battle tested for Microsoft internal workload

Windows
Skype
LinkedIn
Visual Studio
SQL
bing
Microsoft Azure
Xbox
Power BI
Office

The platform for analytical solutions (SaaS)

Azure Monitor
Log Analytics,
Application Insights

Security Products
(MS Defender for Endpoint (MDATP),
MS Sentinel, MS Defender for IoT,
MS Defender for Cloud Apps (MCAS))

Azure IoT
Central

PLAYFAB
Gaming
platform

Available as PaaS

esri
DocuSign
Check Point SOFTWARE TECHNOLOGIES LTD.
Taboola
ECOLAB
ZOOMD KNOW MORE, DO MORE.
BÜHLER
agl
BASF We create chemistry
Grab
UPTAKE Orchestrating a brighter world
BOSCH Invented for life
NEC

https://aka.ms/adx.customers

What is Kusto Query?

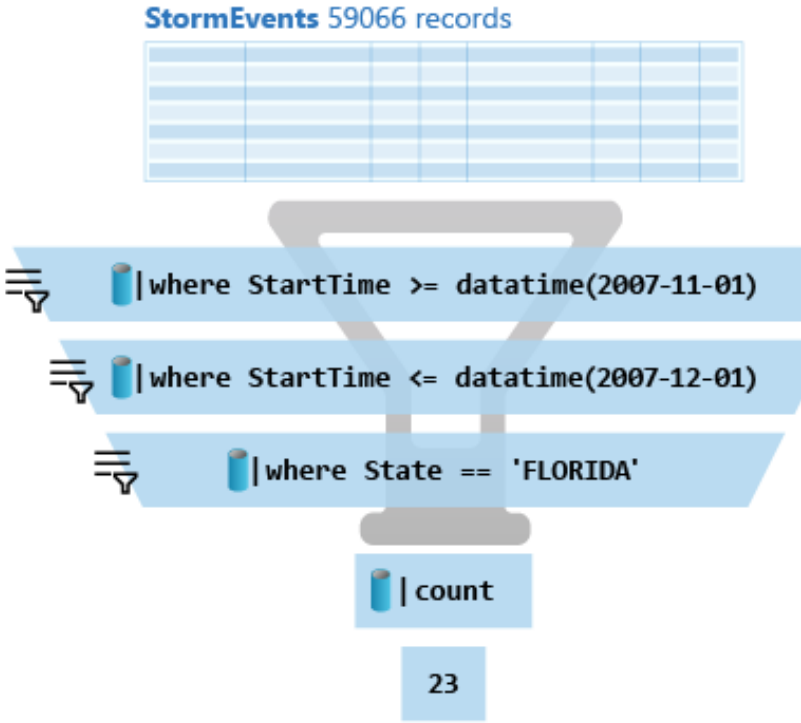
- A Kusto query is a read-only request to **process** data and **return** results.

  The request is stated in plain text, using a data-flow model designed to make the syntax easy to read, author, and automate.

- The query uses schema entities that are organized in a hierarchy similar to SQL's:
  - Databases
  - Tables
  - Columns

- Azure Data Explorer was first announced at Ignite 2018

Azure Data Explorer is a big data analytics cloud service optimized for interactive ad-hoc queries over structured, semi-structured, and unstructured data



Kusto query usage

KQL is the query language, and the Kusto Engine is the engine that receives the queries in KQL to execute them, and specifically the large datasets from Azure like

- **Azure Application Insights**
- **Azure Log Analytics**
- **Windows Defender Advanced Threat Protection**
- **Azure Security Center**
- **Azure Synapse Analytics**
- **Azure Sentinel**

Apart from these, the data can be ingested from external sources as well.

Anatomy of Kusto query

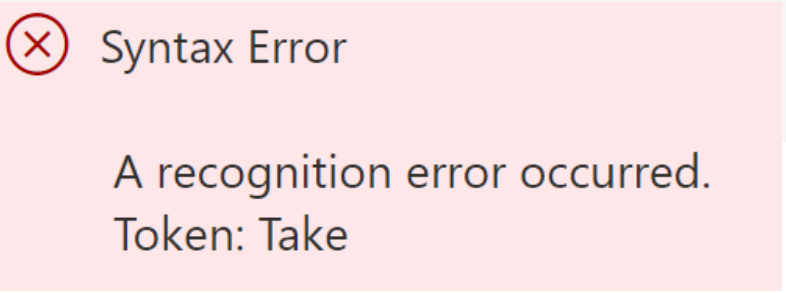
- The query consists of a sequence of query statements, delimited by a semicolon (;),
- At least one statement must be tabular expression statement which is a statement that produces data arranged in a table-like mesh of columns and rows.
- syntax of the tabular expression statement has tabular data flow from one tabular query operator to another,
  - starting with data source
  - flowing through a set of data transformation operators
- Bound together using the pipe (|) delimiter.

Anatomy of Kusto Query - continued

- KQL **commands are case-sensitive** but the operators could be both sensitive and insensitive
- Comments are noted with //

```
StormEvents  //let's list 10 sampling rows
| take 10

StormEvents  //same query taking wrong command name
| Take 10
```



Entity Types

Kusto queries execute in the context of some Kusto database that is attached to a Kusto cluster. Data in the database is arranged in tables, which the query may reference, and within the table it is organized as a rectangular grid of columns and rows

Entity types are :

- Clusters
- Databases
- Tables
- Columns
- Stored Functions
- Views
- External Tables

Entity Names

Entity names are **case-sensitive** for resolving purposes (so, for example, you can't refer to a table called ThisTable as thisTABLE) and is guaranteed to be **unique** in the scope of its container given its type

Naming rules :

- **They have between 1 and 1024 characters long.**
- **They may contain letters, digits, underscores (_), spaces, dots (.), and dashes (-).**
  - **Identifiers consisting only of letters, digits, and underscores do not require quoting when the identifier is being referenced.**
  - **Identifiers containing at last one of (spaces, dots, or dashes) do require quoting.**
- **They are case-sensitive.**

```
StormEvents  //no special chars

['Storm-Events']  //special chars must use [] and '' or ""

["where"]  //language keywords must use [] and '' or ""
```

Kusto Data Types

Type	Additional name(s)	Equivalent .NET type	gettype()	Link
bool	boolean	System.Boolean	int8	The bool data type - Azure Data Explorer | Microsoft Docs
datetime	date	System.DateTime	Datetime	The datetime data type - Azure Data Explorer | Microsoft Docs
dynamic		System.Object	array or dictionary or any of the other values	The dynamic data type - Azure Data Explorer | Microsoft Docs
guid		System.Guid	guid	The guid data type - Azure Data Explorer | Microsoft Docs
int		System.Int32	int	The int data type - Azure Data Explorer | Microsoft Docs
long		System.Int64	long	The long data type - Azure Data Explorer | Microsoft Docs
real	double	System.Double	real	The real data type - Azure Data Explorer | Microsoft Docs
string		System.String	string	The string data type - Azure Data Explorer | Microsoft Docs
timespan	time	System.TimeSpan	timespan	The timespan data type - Azure Data Explorer | Microsoft Docs
decimal		System.Data.SqlTypes.SqlDecimal	decimal	The decimal data type - Azure Data Explorer | Microsoft Docs

Dynamic Type

The dynamic scalar data type is special in that it can take on any value of other scalar data types from the list below, as well as arrays and property bags. Specifically, a dynamic value can be:

- Null.
- A value of any of the primitive scalar data types: bool, datetime, guid, int, long, real, string, and timespan.
- An array of dynamic values, holding zero or more values with zero-based indexing.
- A property bag that maps unique string values to dynamic values. The property bag has zero or more such mappings (called "slots"), indexed by the unique string values. The slots are unordered.

Basic Kusto Query

Operation	KQL Query
Count Rows : keyword **count** **Ref :** count() (aggregation function) - Azure Data Explorer | Microsoft Docs	StormEvents | count StormEvents | where StartTime >= datetime(2007-11-01) and StartTime < datetime(2007-12-01) | where State == "FLORIDA" | count // inline comment

Show n rows : Keyword **take**
 	
Ref: take operator - Azure Data Explorer | Microsoft Docs
	// take 5 rows
	StormEvents
	| take 5
	
	//Take n rows and select columns
	StormEvents
	| take 5
	| project StartTime, EndTime , State , EventType , EpisodeNarrative

Show Distinct values: **distinct** Operator Produces a table with the distinct combination of the provided columns of the input table. **Ref:** distinct operator - Azure Data Explorer | Microsoft Docs	// show distinct State in StormEvents StormEvents | distinct State StormEvents | where StartTime > datetime(2007-02-01) and StartTime < datetime(2007-03-01) | distinct State , EventType , EpisodeNarrative

Basic Kusto Query

Operation	KQL Query

Select a subset of columns: **project**
Use project to pick out only the columns you want.
 	
Ref : Project operator - Azure Data Explorer | Microsoft Docs
	`//Project specific columns`
	`StormEvents`
	`| where StartTime > datetime(2007-02-01) and StartTime < datetime(2007-03-01)`
	`| where EventType == 'Flood' and State == 'CALIFORNIA'`
	`| project StartTime, EndTime , State , EventType , EpisodeNarrative`
	
	`//Select a subset of columns, replace a column name and perform some calculation :`
	`T`
	`| project`
	`X=C, // Rename column C to X`
	`A=2*B, // Calculate a new column A from the old B`
	`C=strcat("-",tostring(C)), // Calculate a new column C from the old C`
	`B=2*B // Calculate a new column B from the old B`

Order results: ***sort, top***
 	
Show me the first *n* rows, ordered by a specific column:
 	
Ref :
sort operator - Azure Data Explorer | Microsoft Docs
top operator - Azure Data Explorer | Microsoft Docs
	`//Show me the first n rows, ordered by a specific column:`
	`StormEvents`
	`| top 5 by StartTime desc`
	`| project StartTime, EndTime, EventType, State, EventNarrative`
	
	`StormEvents`
	`| sort by StartTime desc`
	`| take 5`
	`| project StartTime, EndTime, EventType, State, EventNarrative`

Compute Derived Columns : **extend**
Create a new column by computing a value in every row
Ref : extend operator - Azure Data Explorer | Microsoft Docs
	`StormEvents`
	`| limit 5`
	`| extend Duration = EndTime - StartTime`
	`| project StartTime, EndTime, Duration, EventType, State`

Aggregate groups of rows : summarize

summarize groups together rows that have the same values in the by clause, and then uses an aggregation function (for example, count) to combine each group in a single row.

```
StormEvents
| summarize event_count = count() by State
```

```
StormEvents
| summarize StormCount = count(), TypeOfStorms = dcount(EventType) by State
| top 5 by StormCount desc
```

Summarize: list of aggregate functions

Statistical functions

Function Name	Description
avg()	Returns an average value across the group.
avgif()	Returns an average value across the group (with predicate).
count(), countif()	Returns a count of the group without/with a predicate.
dcount(), dcountif()	Returns an approximate distinct count of the group elements without/with a predicate.
hll()	Returns the hyper log log (hll) results of the group elements, an intermediate value of the dcount approximation.
hll_merge()	Returns a value for merged hll results.
max(), maxif()	Returns the maximum value across the group without/with a predicate.
merge_tdigest()	Returns the merged tdigest value across the group, is an alias of tdigest_merge.
min(), minif()	Returns the minimum value across the group without/with a predicate.
percentile()	Returns a percentile estimation of the group.
percentiles()	Returns percentile estimations of the group.
percentiles_array()	Returns the percentile approximates of the array.
percentilesw()	Returns the weighted percentile approximate of the group.
percentilesw_array()	Returns the weighted percentile approximate of the array.
stdev(), stdevif()	Returns the standard deviation across the group for a population that is considered a sample without/with a predicate.
stdevp()	Returns the standard deviation across the group for a population that is considered representative.
sum(), sumif()	Returns the sum of the elements within the group without/with a predicate.
tdigest()	Returns an intermediate result for the percentiles approximation, the weighted percentile approximate of the group.
tdigest_merge()	Returns the merged tdigest value across the group.
variance(), varianceif()	Returns the variance across the group without/with a predicate.
variancep()	Returns the variance across the group for a population that is considered representative.

Summarize: by scalar values

We can use scalar (numeric, time, or interval) values in the by clause, but you'll want to put the values into bins by using the bin() function:

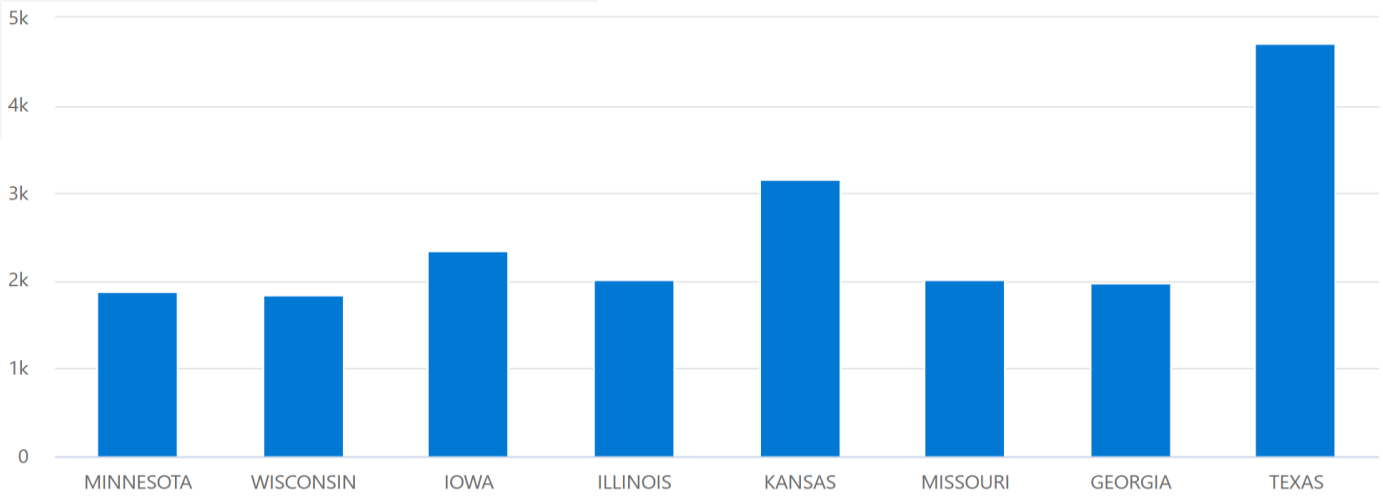
bin() function simply reduces every value to the nearest multiple of the modulus that you supply, so that summarize can assign the rows to groups.

```
StormEvents
| where StartTime > datetime(2007-02-14) and StartTime < datetime(2007-02-21)
| summarize event_count = count() by bin(StartTime, 1d)
```

Display a Chart or table : render

We can project two columns and use them as the x-axis and the y-axis of a chart:

```
StormEvents
| summarize event_count=count(), mid = avg(BeginLat) by State
| sort by mid
| where event_count > 1800
| project State, event_count
| render columnchart
```



Display a chart or table: Timecharts

We can project two columns and use them as the x-axis and the y-axis of a chart:

```
StormEvents
| summarize event_count=count() by bin(StartTime, 1d)
| render timechart
```

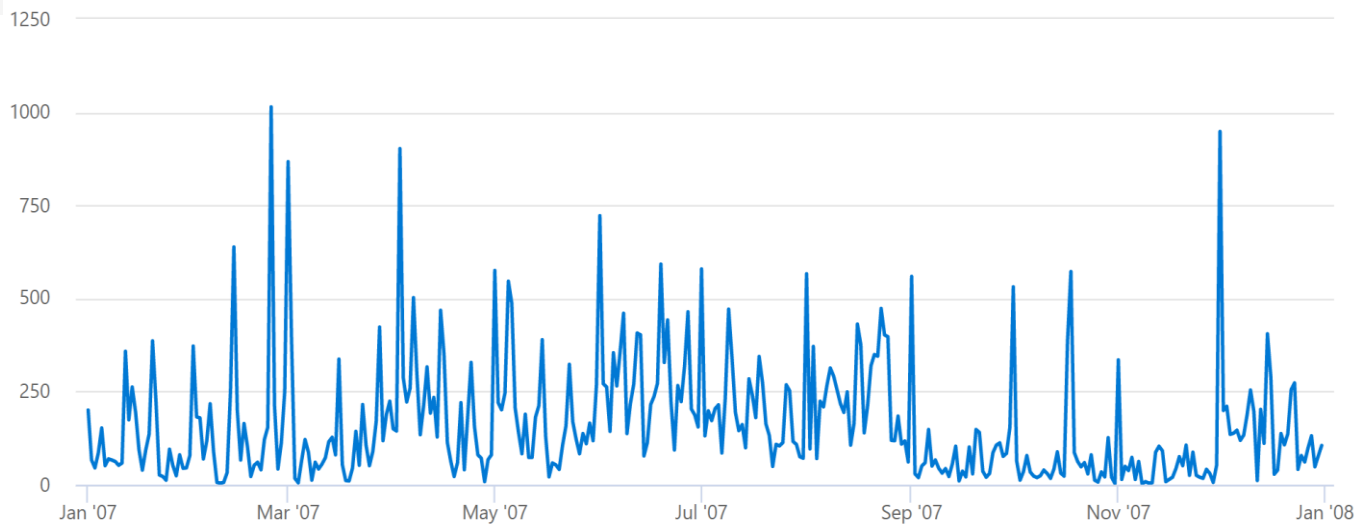
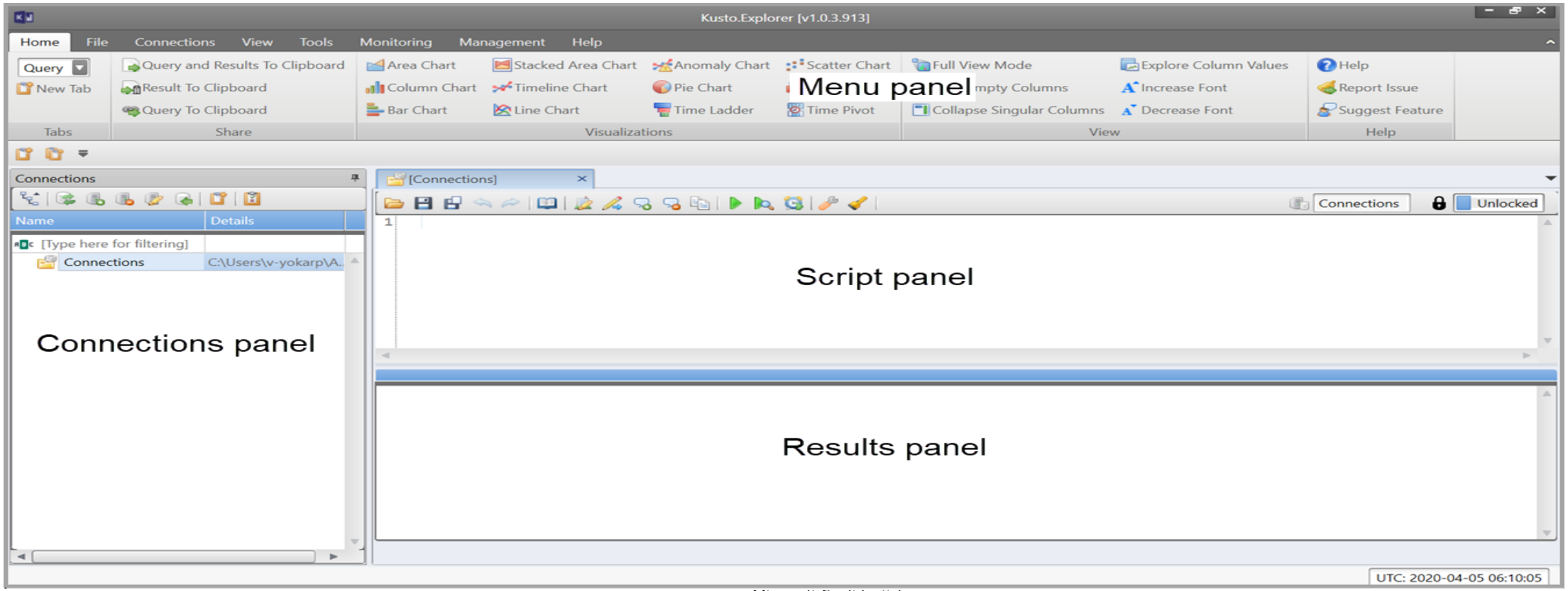


Chart Visualizations: render

Visualization	Description	Example
anomalychart	Similar to timechart, but highlights anomalies using series_decompose_anomalies function.	Click to run sample query
areachart	Area graph. First column is the x-axis and should be a numeric column. Other numeric columns are y-axes.	Click to run sample query
barchart	First column is the x-axis and can be text, datetime or numeric. Other columns are numeric, displayed as horizontal strips.	Click to run sample query
card	First result record is treated as set of scalar values and shows as a card.	Click to run sample query
columnchart	Like barchart with vertical strips instead of horizontal strips.	Click to run sample query
ladderchart	Last two columns are the x-axis, other columns are y-axis.	
linechart	Line graph. First column is x-axis, and should be a numeric column. Other numeric columns are y-axes.	Click to run sample query
piechart	First column is color-axis, second column is numeric.	Click to run sample query
pivotchart	Displays a pivot table and chart. User can interactively select data, columns, rows and various chart types.	
scatterchart	Points graph. First column is x-axis and should be a numeric column. Other numeric columns are y-axes.	Click to run sample query
stackedareachart	Stacked area graph. First column is x-axis, and should be a numeric column. Other numeric columns are y-axes.	Click to run sample query
table	Default - results are shown as a table.	Click to run sample query
timechart	Line graph. First column is x-axis, and must be datetime. Other (numeric) columns are y-axes. There is one string column whose values are used to "group" the numeric columns and create different lines in the chart (further string columns are ignored).	Click to run sample query
timepivot	Interactive navigation over the events time-line (pivoting on time axis)	

Kusto Tools: Kusto Explorer

Kusto Explorer : Kusto Explorer is a rich desktop application that enables you to explore your data using the KQL language in an easy to use Interface. Install from https://aka.ms/ke



Kusto Tools: Kusto CLI

Kusto CLI : Kusto CLI is a command-line utility that is used to send requests to kusto, and display the results. Kusto.Cli is primarily provided for automating tasks against a Kusto service that normally requires writing code. For example, a C# program or a PowerShell script.

It supports three modes:

1. **REPL mode** : The user enters queries and commands, and the tool displays the results, then awaits the next user query/command. ("REPL" stands for "read/eval/print/loop".)

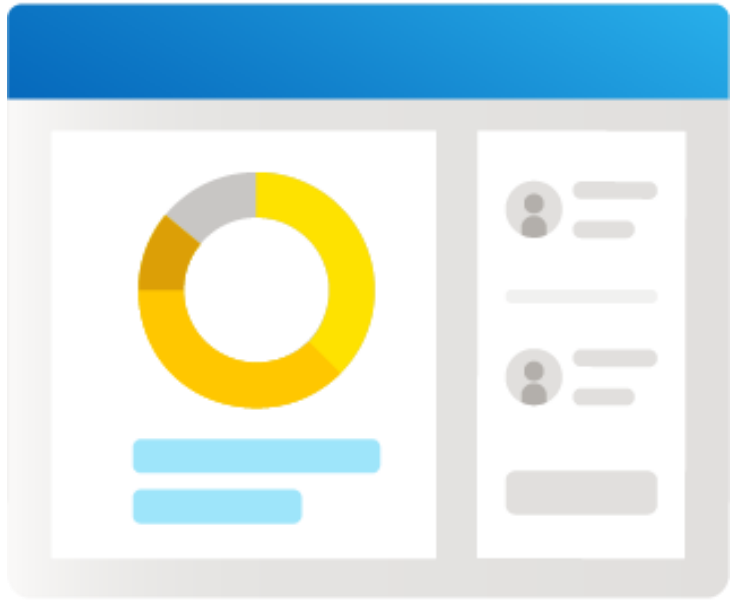
2. **Execute mode** : The user enters one or more queries and commands to run as command-line arguments. The arguments are automatically run in sequence, and their results output to the console. Optionally, after all the input queries and commands have run, the tool goes into REPL mode.

3. **Script mode** : Similar to execute mode, but with the queries and commands specified through a "script" file.

Ref: https://docs.microsoft.com/azure/data-explorer/kusto/tools/kusto-cli

Demonstration

Demo of Kusto Explorer and basic KQL queries



Questions?

Lesson 2 : Functions

Objectives

After completing this Learning, you will be able to:

Understand different types of built-in functions

1. Built-in Functions
2. Query Defined Functions
3. Usage of built-in functions in Kusto/Log Analytics

KQL Functions

Functions are reusable queries or query parts.
Kusto supports several kinds of functions:

- **Stored functions**,
  - which are user-defined functions that are stored and managed a one kind of a database's schema entities. See Stored functions.

- **Query-defined functions**
  - which are user-defined functions that are defined and used within the scope of a single query. The definition of such functions is done through a let statement. See User-defined functions.

- **Built-in functions**
  - which are hard-coded (defined by Kusto and cannot be modified by users).

Stored Functions

Functions are reusable queries or query parts. Functions can be stored as database entities (similar to, for example, tables) called **stored functions**.

Built-in KQL Functions

Functions are reusable queries or query parts. Kusto supports several kinds of functions:

- **Scalar functions**
  - Function whose result or output is a scalar quantity. The output is one-dimensional

- **Aggregation functions**
  - An aggregate function results in a single value expressing the significance of the accumulated data it is derived from.

- **Window functions**
  - Window functions operate on multiple rows (records) in a row set at a time. Unlike aggregation functions, window functions require that the rows in the row set be serialized (have a specific order to them). Window functions may depend on the order to determine the result.

- **Special Functions**

Scalar KQL Functions

Following table is a subset of the types of scalar functions, list available here :

Binary functions	Conditional functions
Conversion functions	Series element-wise functions
DateTime/timespan functions	Series processing functions
Dynamic/array functions	String functions
Window scalar functions	IPv4/IPv6 functions
Flow control functions	Type functions
Mathematical functions	Scalar aggregation functions
Metadata functions	Geospatial functions
Rounding functions	Hash functions

Frequently Used KQL Functions

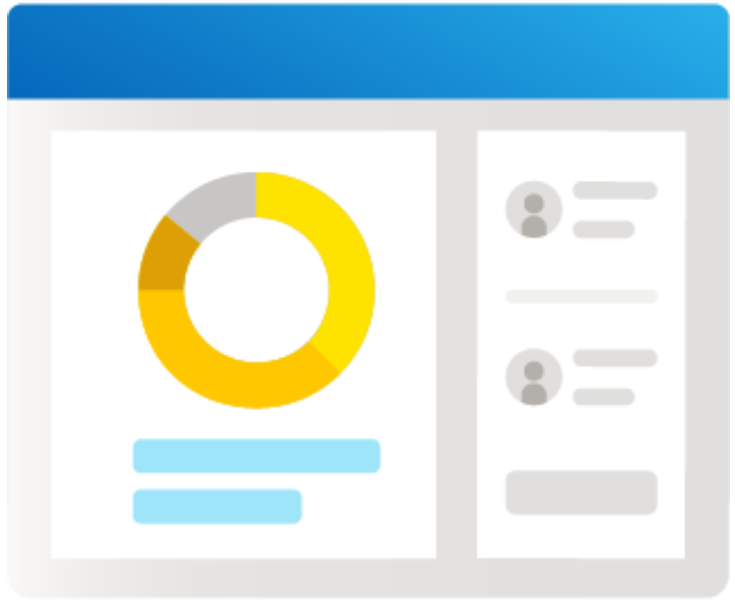
Following frequently used string and datetime scalar functions:

ago()	Subtracts the given timespan from the current UTC clock time.
datetime_add()	Calculates a new datetime from a specified datepart multiplied by a specified amount, added to a specified datetime.
now()	Returns the current UTC clock time, optionally offset by a given timespan.
case()	Evaluates a list of predicates and returns the first result expression whose predicate is satisfied.
coalesce()	Evaluates a list of expressions and returns the first non-null (or non-empty for string) expression.
iif()/iff()	Evaluates the first argument (the predicate), and returns the value of either the second or third arguments, depending on whether the predicate evaluated to true (second) or false (third).
max_of()	Returns the maximum value of several evaluated numeric expressions.
min_of()	Returns the minimum value of several evaluated numeric expressions.

countof()	Counts occurrences of a substring in a string. Plain string matches may overlap; regex matches don't.
extract()	Get a match for a regular expression from a text string.
extract_all()	Get all matches for a regular expression from a text string.
extractjson()	Get a specified element out of a JSON text using a path expression.
has_any_index()	Searches the string for items specified in the array and returns the position of the first item found in the string.
indexof()	Function reports the zero-based index of the first occurrence of a specified string within input string.
isempty()	Returns true if the argument is an empty string or is null.
isnotempty()	Returns true if the argument isn't an empty string or a null.
isnotnull()	Returns true if the argument is not null.
isnull()	Evaluates its sole argument and returns a bool value indicating if the argument evaluates to a null value.

Demonstration

Demo for String and Date Functions



Parsing Functions

Evaluates a string expression and parses its value into one or more calculated columns. The calculated columns will have nulls, for unsuccessfully parsed strings

```
AppExceptions
| parse Assembly with *
    "Version=" _version
    "Culture=" _myCulture
    "PublicKeyToken" _myPKT
| project Assembly, _version, _myCulture, _myPKT
| top 10 by _version desc
```

Parse Arguments

T | parse [kind=regex [flags=regex_flags] |simple|relaxed] *Expression* with *

- Kind
  - Simple (default)
  - Regex
  - Flags
  - relaxed
- Expression
- Column Name
- Column Type

Other Parse Operators

parse_json: Interprets a string as a JSON value and returns the value as dynamic.

```
let d='{"a":123, "b":"{\\"c\\":456}"}'; print
d_b_c=parse_json(tostring(parse_json(d).b)).c
```

parse_csv: Splits a given string representing a single record of comma-separated values and returns a string array with these values.

```
print result=parse_csv('aa,"b,b,b",cc,"Escaping quotes:
""Title""","line1\nline2"')
```

Conditional Functions

Following are some of the important conditional functions :

case()	Evaluates a list of predicates and returns the first result expression whose predicate is satisfied.
coalesce()	Evaluates a list of expressions and returns the first non-null (or non-empty for string) expression.
iif()/iff()	Evaluates the first argument (the predicate), and returns the value of either the second or third arguments, depending on whether the predicate evaluated to true (second) or false (third).
max_of()	Returns the maximum value of several evaluated numeric expressions.
min_of()	Returns the minimum value of several evaluated numeric expressions.

Conversion Functions

Following are some of the important conversion functions :

tobool()	Converts input to boolean (signed 8-bit) representation.
todatetime()	Converts input to datetime scalar.
todouble()/toreal()	Converts the input to a value of type real. (todouble() and toreal() are synonyms.)
tostring()	Converts input to a string representation.
totimespan()	Converts input to timespan scalar.

Metadata Functions

Following are some of the important metadata functions :

column_ifexists()	Takes a column name as a string and a default value. Returns a reference to the column if it exists, otherwise - returns the default value.
current_cluster_endpoint()	Returns the current cluster running the query.
current_database()	Returns the name of the database in scope.
current_principal()	Returns the current principal running this query.
current_principal_details()	Returns details of the principal running the query.
current_principal_is_member_of()	Checks group membership or principal identity of the current principal running the query.
cursor_after()	Used to access to the records that were ingested after the previous value of the cursor.
estimate_data_size()	Returns an estimated data size of the selected columns of the tabular expression.
extent_id()	Returns a unique identifier that identifies the data shard ("extent") that the current record resides in.
extent_tags()	Returns a dynamic array with the tags of the data shard ("extent") that the current record resides in.
ingestion_time()	Retrieves the record's $IngestionTime hidden datetime column, or null.

Window Scalar Functions

Following are some of the important window scalar functions :

next()	For the serialized row set, returns a value of a specified column from the later row according to the offset.
prev()	For the serialized row set, returns a value of a specified column from the earlier row according to the offset.
row_cumsum()	Calculates the cumulative sum of a column.
row_number()	Returns a row's number in the serialized row set - consecutive numbers starting from a given index or from 1 by default.
row_rank()	Returns a row's rank in the serialized row set.

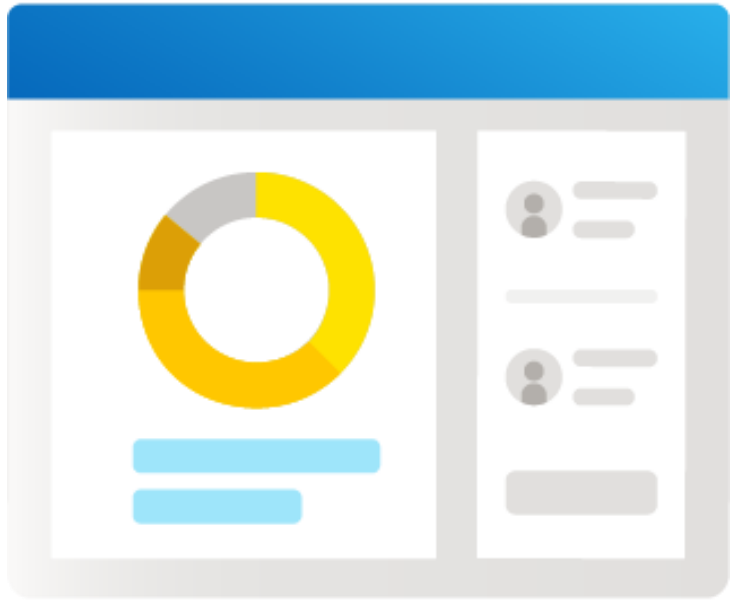
Mathematical Functions

Following are some of the important mathematical scalar functions :

abs()	Calculates the absolute value of the input.
isfinite()	Returns whether input is a finite value (isn't infinite or NaN).
isinf()	Returns whether input is an infinite (positive or negative) value.
isnan()	Returns whether input is Not-a-Number (NaN) value.
log()	Returns the natural logarithm function.
not()	Reverses the value of its bool argument.
pow()	Returns a result of raising to power.
rand()	Returns a random number.
range()	Generates a dynamic array holding a series of equally spaced values.
round()	Returns the rounded source to the specified precision.

Demonstration

Parsing Demo



Questions?

Lesson 3 : Datasets

Objectives

After completing this Learning, you will be able to:

Understand different types of operators related to variables and datasets

1. Datatable
2. Joins, Unions
3. Let, Set, Batch