**MODULE-6**

**WORK WITH REAL TIME DATA IN MS FABRIC EVENTHOUSE**

Eventhouse 🡪 provides a datastore for large volumes of data.

* Optimized for data that represents time-based events
* Designed to handle real-time data streams (query & analyze near real time)

MS Fabric 🡪 use eventstream 🡪 to load stream of real time data 🡪 eventhouse

Then:

* Query using KQ/ SQL
* Use realtime dashboards to visualize the data
* Use Fabric Activator to automate actions based on the data

A screenshot of a computer

AI-generated content may be incorrect.

Eventhouse 🡪 fabric data store at center of all of these capabilities

Create one or more eventhouses for your data

Event house contains one or more KQL databases

After creating eventhouse🡪 use default KQL database/ create new one

To get your data into KQL database in an eventhouse 🡪 import from static loc (local file/ onelake/ AZ storage/ sample dataset) or real time src (event hubs/ fabric eventstream)

Querying tables using KQL database:

**SQL:** Select \* from automotive

**KQL:** Automotive

**SQL:** select top 100 \* from automotive

**KQL:** automotive

| take 100

|  |  |  |
| --- | --- | --- |
| **Query** | **SQL** | **KQL** |
| Select top 10 | Select top 10 from a | a  | take 10 |
| Select all | Select \* from a | a |
| Specific cols | Select x,y,x from a | a  |x,y,z |
| Filter rows | Select x,y,z from a where x>20 | a  | where x>20  | project x,y,z |
| Sort results | Select x,y,z  From a  Where x>20  Order by y desc | a  | where x>29  | project x,y,z  | sort by y desc |
| Group and aggregate | Select p,count(\*) as q  From a  Group by p | a  | summarize q = count(\*) by p  | project p,q |

KQL VS SQL:

* **Simplicity**: KQL is a simpler language than SQL, making it easier to learn and use.
* **Performance**: KQL is optimized for performance and can handle large amounts of data more efficiently than SQL.
* **Flexibility**: KQL is more flexible than SQL, allowing users to perform complex queries with ease.
* **Integration**: KQL is integrated with other Microsoft products, such as Azure Monitor and Azure Sentinel.

Materialized views and stored functions:

* **Materialized views:** Summary of data from source table or another materialized view. The view encapsulates a summarize statement.

**.create materialized-view TripsByVendor on table Automotive**

**{**

**Automotive**

**| summarize trips = count() by vendor\_id, pickup\_date = format\_datetime(pickup\_datetime,”yyyy-MM-dd”)**

**}**

This view is populated as new data is ingested into the source table.

To ingest existing data, you can use an asynchronous operation to create the materialized view with the backfill option:

**.create async materialized-view with (backfill = true)**

**TripsByVendor on table Automative**

**{**

**Automotive**

**| summarize trips = count() by vendor\_id, pickup\_date = format\_datetime(pickup\_datetime,”yyyy-MM-dd”)**

**}**

After creating and populating the materialized vview, you can query it just like a table.

**TripsByVendor**

**| project pickup\_date, vendor\_id, trips**

**| sort by pickup\_date desc**

Materialized views 🡪 listed in materialized views folder for the KQL database where you defined them in the eventhouse page.

* **Stored Functions:** To encapsulate a query as a function, to make it easier to repeat common queries.

You can also specify parameters for a function, so you can repeat the same query with variable values.

To create a function, use the **“.create-or-alter function”** KQL command.

* Defining function:

**.create-or-alter function trips\_by\_min\_passenger\_count(num\_passengers: long)**

**{**

**Automative**

**| where passenger\_count >= num\_passengers**

**| project trip\_id, pickup\_datetime**

**}**

* Calling function:

**trips\_by\_min\_passenger\_count(3)**

**| take 10**

A screenshot of a computer

AI-generated content may be incorrect.