Query Real-Time IoT Data in Azure Stream Analytics

Introduction

In this lab, we supply you with a robust sample of data and a series of real-world scenarios to help you practice some of the most practical and common IoT data query patterns.

Solution

1. Open an incognito or *InPrivate* browser window, and log in to the Azure portal using the credentials provided in the lab environment.
2. Navigate to the sample GitHub repository to download the data file you will use in the lab.
    - From the the upper right of the file viewer, click **Raw**.
    - Right-click, or CTRL-click depending on your OS and mouse configuration, and select the **Save As** option.
    - Before saving, add a `.JSON` extension to the file name.

Set Up Sample Data and Write a Query

1. Start on the resource group **Overview** page of the Azure portal. On the bottom, under *Resources*, select the **ASASampleJob** Stream Analytics job.
2. From the left navigation menu, under *Job topology*, select **Query**.
3. Under the *Input preview* tab, click **Upload sample input**.
4. Navigate through your file system to find and upload the `HelloWorldASA-InputStream.json` file you previously downloaded from GitHub.
5. Click **OK**. You will see a preview of the data displayed in the *Input preview* tab, on the bottom.
6. Click **Test query** to run the sample query that is provided in the portal. The query should return 1860 rows with the following four column headers: `time`, `dspl`, `temp`, `hmdt`.
7. Try creating your own query. This query should alias three of the four column names for better readability and return data from `sensorA`. Reasonable column names could be: `time` (no alias), `SensorName`, `Temperature`, and `Humidity`.

Test Your Sample Query

1. Your query should look similar to the example below. Note that the solution you create might not be an exact match. As long as it returns the expected results noted in the next steps, then you successfully completed the objective:

```
SELECT
time,
dspl as SensorName,
temp as Temperature,
hmdt as Humidity
INTO
Output
FROM InputStream
WHERE dspl='sensorA'
```

2. Above the query pane, click **Test query**.

3. View the results in the *Test results* tab, under the query pane. Correctly written, your query should return 389 rows.

Expand the First Query: Use a `CASE` Statement

For the next query, use the query from the last objective as a starting point. Your new query should meet the following criteria:

- Includes a new column called `Status` that returns the string `Alert` if the temperature is above 100 **and** the humidity is below 40.
- If the temperature/humidity is not in this range, then it should return `OK` in the `Status` column.

1. To implement this query, use a `CASE` statement that defines the additional criteria and adds on to what you created in the first objective:

```
    CASE
      WHEN temp > 100 AND hmdt < 40 THEN 'Alert'
      ELSE 'OK'
      END as Status
```

> **Note:** When you use the `CASE` statement, you should invoke the actual column names rather than their aliases.

Test Your Query

1. Your completed query should look similar to the below example. Note that the solution you create might not be an exact match. As long as your query returns the expected number of rows noted below, the your query is correct:

```
SELECT
time,
dspl as SensorName,
temp as Temperature,
hmdt as Humidity,
    CASE
      WHEN temp > 100 AND hmdt < 40 THEN 'Alert'
      ELSE 'OK'
      END as Status
INTO
Output
FROM InputStream
WHERE dspl='sensorA'
```

2. Above the query pane, click **Test query**.

3. View the results in the *Test results* tab, under the query pane. Correctly written, the query should return the same 389 rows as the first query challenge — but with an additional column (i.e., `Status`) that includes an alert when the previously described temperature and humidity conditions are present.

Query for Average Temperature

For Query #3, suppose you want to get the average temperature, every one minute, over all sensors. You need to display two columns: the timestamp representing the minute and the average temperature for that minute.

1. To begin, select `System.Timestamp()`, which is a timestamp of each window of time, aliased as `EndOfMinute`. Then, use the `AVG` function on the `temp` column, aliased as `AverageTemp`:

```
SELECT
System.Timestamp() AS EndOfMinute,
AVG(temp) as AverageTemp
INTO
Output
FROM InputStream
```

2. Next, use the `TIMESTAMP BY time` clause to use the `time` field from the sample data as the event timestamp:

```
TIMESTAMP BY time
```

3. Finally, use the `TumblingWindow` function to group by and report every `1 minute`:

```
GROUP BY TumblingWindow(minute,1)
```

Test Your Query

1. Your completed query should look similar to the below example. Note that the solution you create might not be an exact match. As long as it returns the expected results noted in the next steps, then that is fine:

```
SELECT
System.Timestamp() AS EndOfMinute,
AVG(temp) as AverageTemp
INTO
Output
FROM InputStream
TIMESTAMP BY time
GROUP BY TumblingWindow(minute,1)
```

2. Above the query pane, click **Test query**.

3. View the results in the *Test results* tab, under the query pane. Correctly written, this query should return 32 rows with average temperatures between 101 and 114 (rounded).

Query for `sensorE` Data

For your final task, create a query that meets the following criteria:

- Checks every one minute for the count of messages sent over the last five minutes.
- Limits the query to data from `sensorE`.
- The query should return two columns: the timestamp representing the interval/event timestamp and a column with the count of events over the last five minutes.

1. Select `System.Timestamp()`, which is a timestamp of each window of time, aliased as `window_end`. Then, select the total count of events, `COUNT(*)`, aliased as `messages`:

```
SELECT
    System.Timestamp() as window_end,
    COUNT(*) AS messages
INTO
Output
FROM InputStream
```

2. Add the `TIMESTAMP BY time` clause to use the `time` field from the sample data as the event timestamp:

```
TIMESTAMP BY time
```

3. Use the `WHERE` clause to limit the query to selections where the `dspl` column is equal to `sensorE`:

```
WHERE dspl='sensorE'
```

4. Finally, group by a hopping window function to hop each minute, with a window size of five minutes:

```
GROUP BY
    HOPPINGWINDOW(minute, 5, 1)
```

Test Your Query

1. Your completed query should look similar to the below example. Note that the solution you create might not be an exact match. As long as it returns the expected results noted in the next steps, then that is fine:

```
SELECT
    System.Timestamp() as window_end,
    COUNT(*) AS messages
INTO
Output
FROM InputStream
TIMESTAMP BY time
WHERE dspl='sensorE'
GROUP BY
    HOPPINGWINDOW(minute, 5, 1)
```

2. Above the query pane, click **Test query**.

3. View the results in the *Test results* tab, under the query pane. Correctly written, the query should return 36 rows, with timestamps one minute apart, and the total count of messages sent in the previous five minutes.

Conclusion

Congratulations — you've completed this hands-on lab! Feel free to continue practicing with creating queries based on the data and see what observations you can make from the results.