



Modern Data Management  
&  
Business Intelligence

Assignment 3

Jan 2022

**Student**

Maria Skoli (p2822131)

**Class**

Part Time

## Table of Contents

Introduction .....	2
Part 1 – Steps to Set up An Event Hub .....	3
Create a Resource Group.....	3
Create a Namespace.....	3
Create an Event Hub.....	4
Create Sending and Recording Policy .....	5
Event Hub Signature Generator.....	6
Create a Storage Account .....	7
Create a Container into Storage Account .....	9
Set up a Stream Analytics Job .....	10
Part 2 – Azure Analytics Tasks and Solutions .....	11
Create Inputs.....	11
Feed queries with a data sample .....	14
Azure Analytics Tasks and Solutions.....	17

## Introduction

In this report we are going to use Azure Stream Analytics to process a data stream of ATM transactions and answer possible tasks we would like to be executed in real time.

The first part of this report is about setting up an Event Hub and a Storage account where our results are going to be stored. Moreover, we are going to feed our stream with the Reference data we already have. Specifically, we have as reference data 3 json files. The first one “Area.json” contains geographical information, the second one “ATM.json” contains information about the ATM machines (e.g., the Atm code) and the last one, “Customer.json” contains information about each customer (e.g., his card number).

In the second part of this report, we are going to create an input and an output for our data and to execute queries that answers business questions. We will present the relative outputs as well.

## Part 1 – Steps to Set up An Event Hub

### Create a Resource Group

First, we should create a resource group which is a logical collection of Azure resources. We named the resource group as: “**MyEH**” and as region we selected the “(Europe) UK South”.

**Create a resource group** ...

Basics Tags Review + create

Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. [Learn more](#) ⓘ

**Project details**

Subscription \* ⓘ Azure for Students ✓

Resource group \* ⓘ MyEH ✓

**Resource details**

Region \* ⓘ (Europe) UK South ✓

**Resource groups** ⓘ ...

auelb.gr (auelb.gr.onmicrosoft.com)

+ Create Manage view Refresh Export to CSV Open query Assign tags Feedback

Filter for any field... Subscription == all Location == all Add filter

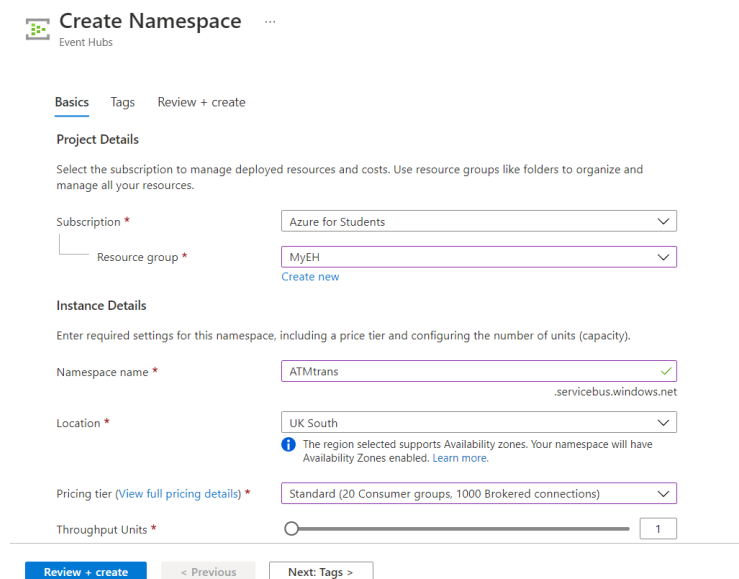
Showing 1 to 1 of 1 records.

<input type="checkbox"/>	Name ↑
<input checked="" type="checkbox"/>	MyEH

Figure 1: Creating a resource group

## Create a Namespace

Before we create an event hub, we should first create a Namespace. An Event Hubs namespace provides a unique scoping container in which we will create the Event Hub. As inputs in the requested fields, we inserted the name of the resource group that we have created before, we enter the name “*ATMtrans*” for the namespace, and we select the location of the namespace as “UK South”. As for the pricing tier we selected the Standard one.



The screenshot shows the 'Create Namespace' form in the Azure portal. The form is titled 'Create Namespace' with a sub-header 'Event Hubs'. It has three tabs: 'Basics', 'Tags', and 'Review + create'. The 'Basics' tab is selected. Under 'Project Details', there is a description: 'Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.' Below this, there are two dropdown menus: 'Subscription' (set to 'Azure for Students') and 'Resource group' (set to 'MyEH'). There is a 'Create new' link below the 'Resource group' dropdown. Under 'Instance Details', there is a description: 'Enter required settings for this namespace, including a price tier and configuring the number of units (capacity).' Below this, there are three fields: 'Namespace name' (set to 'ATMtrans'), 'Location' (set to 'UK South'), and 'Pricing tier' (set to 'Standard (20 Consumer groups, 1000 Brokered connections)'). There is a 'Throughput Units' slider set to '1'. At the bottom, there are three buttons: 'Review + create', '< Previous', and 'Next: Tags >'.

Figure 2: Creating a Namespace

We can see below that the deployment of the namespace “*ATMtrans*” we have created was successful.

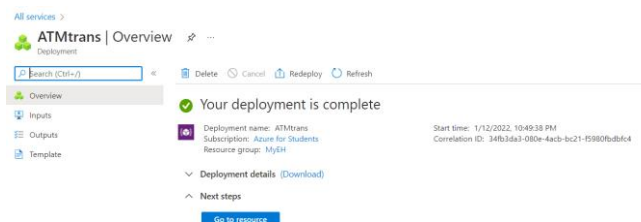
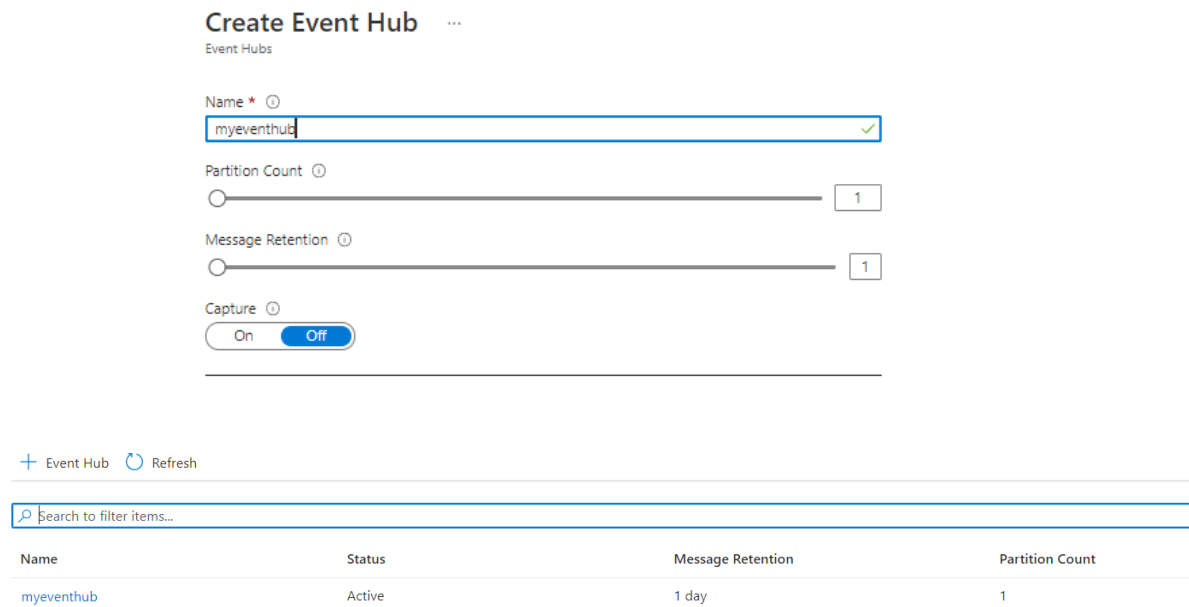


Figure 3: Successful Deployment of the Namespace

## Create an Event Hub

The next step was to enter the Namespace we have created and create a new Event Hub. We named the event hub as: myeventhub.



**Create Event Hub** ...

Event Hubs

Name \* ⓘ  
myeventhub ✓

Partition Count ⓘ  
1

Message Retention ⓘ  
1

Capture ⓘ  
On Off

+ Event Hub Refresh

Search to filter items...

Name	Status	Message Retention	Partition Count
myeventhub	Active	1 day	1

Figure 4: Creation of an Event Hub

## Create Sending and Recording Policy

Then we enter the Event hub, and we press the “Shared access policies” to create a Sending and a Recording Policy.

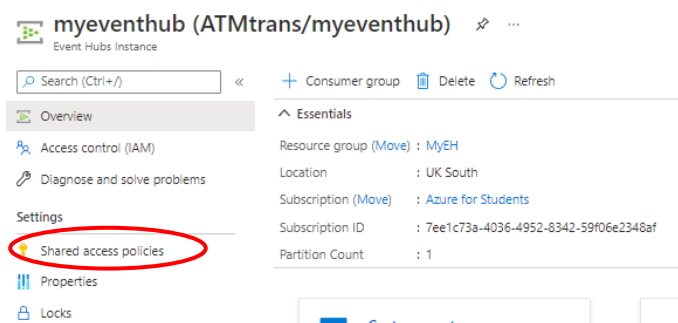


Figure 5: Creation of Sending and Recording Policy

MySendPolicy is used to send data into the Azure stream and the MyRecPolicy is used to listen the data that was send.

Policy	Claims
MyRecPolicy	Listen
MySendPolicy	Send

### SAS Policy: MySendPolicy

Save Discard Delete ...

☐ Manage

☒ Send

☐ Listen

Primary key  
KA2ejspzTnlt/o7MRRyMxeLQeWYGmnWUOwkjo...

Secondary key  
vw6cF1Jx/n8efB4TzW7yU8knaawt8sxZ63AGJfpFZ...

Connection string–primary key  
Endpoint=sb://atmtrans.servicebus.windows.net/...

Connection string–secondary key  
Endpoint=sb://atmtrans.servicebus.windows.net/...

Figure 6: Creation of Sending Policy

### SAS Policy: MyRecPolicy

Save Discard Delete ...

☐ Manage

☐ Send

☒ Listen

Primary key  
n4Y6D93/ElcDdClnMRsmqvDWdKcr2uy5vFTm...

Secondary key  
7P1IU1AM3XTHa4sDHnNFmugAuJmoB1QjX/...

Connection string–primary key  
Endpoint=sb://atmtrans.servicebus.windows.n...

Connection string–secondary key  
Endpoint=sb://atmtrans.servicebus.windows.n...

Figure 7: Creation of Recording Policy

## Event Hub Signature Generator

Then we are going to generate a signature from the signature generator to update our Html file. “Generator.html”. We fill the blank fields with the names of Namespace, Hub name and Sender key we created before. Then we press Generate and then the Signature link will appear.

Event Hubs - Signature Generator

Hub

Namespace: MyEH

Hub Name: myeventhub

Publisher: Laptop

Mode: Http

Credentials

Sender Key Name: MySendPolicy

Sender Key: t/o7MRRyMxeLQeWYGmnWUOwkjoHVBkUs=

Token TTL (minutes): 7200

Signature

SharedAccessSignature sr=https%3a%2f%2fmyeh.servicebus.windows.net%2fmyeventhub%2fpublishers%2flaptop%2fmessages&sig=DFY7C%2fFr%2fhMMce6qzuSadx4UDnD8%2fPCQCtIQBhhM2Ck%3d&se=1642750189&sk=MySendPolicy+

Generate

Figure 8: Signature Generator inputs

We opened the Generator.html file (with Notepad++) and we did the following transformations. First, we changed the link in the “sas variable” with the link that was generated from the Generator app above (Table 5). Then we changed the “service Namespace” to “ATMtrans” and the Hub Name to “myeventhub” to corresponds with the resources we have created in Azure.

```
<html>
<head>
  <script src="js/lodash.js"></script>
</head>
<body>
  <input type="button" value="Send Data" onclick="sendDummyData()" />
  <div id="status" style="display: inline-block;"></div>
<script type="text/javascript">
function sendDummyData() {

  /**** CONFIG ****/
  //Use the signature generator: https://github.com/sandrinodimattia/RedDog/releases
  var sas = "SharedAccessSignature sr=https%3a%2f%2fatmtrans.servicebus.windows.net%2fmyeventhub%2fpublishers%2flaptop%2fmessages%2f";
  var serviceNamespace = "ATMtrans";
  var hubName = "myeventhub";
  var deviceName = "Laptop";
}
```

Figure 9: Making the appropriate changes in Generator html file

Then we open the above file “Generator.html” with Chrome and and we press Send Data. We can see the system started to send data to the event hub.

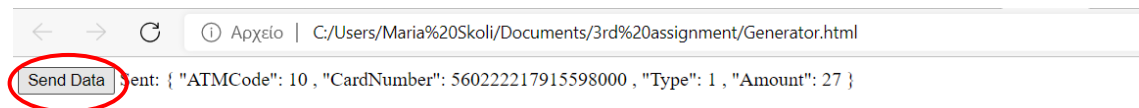


Figure 10: Send data to the source

## Create a Storage Account

Then we created a storage account where the results of our analysis will be saved.

Create a storage account

Basics

Advanced

Networking

Data protection

Encryption

Tags

Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more about Azure storage accounts](#)

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription \*

Azure for Students

Resource group \*

MyEH

Create new

Instance details

If you need to create a legacy storage account type, please click [here](#).

Storage account name \*

storagenameac

Region \*

(US) East US

Performance \*

☒ Standard: Recommended for most scenarios (general-purpose v2 account)

☐ Premium: Recommended for scenarios that require low latency.

Redundancy \*

Geo-redundant storage (GRS)

☒ Make read access to data available in the event of regional unavailability.

Review + create

< Previous

Next : Advanced >

Storage accounts

Filter for any field...

Subscription == all

Resource group == all

Location == all

Add filter

Showing 1 to 1 of 1 records.

☐

Name ↑↓

☐

storagenameacc

Type ↑↓

Storage account

Figure 11: Creating a Storage Account



## Create a Container into Storage Account

Then we enter the Storage account we created, and we add a new container. We named the container as “mycontainer”. This will be the place where the outputs of our queries will be saved. As public access level we select the private one.

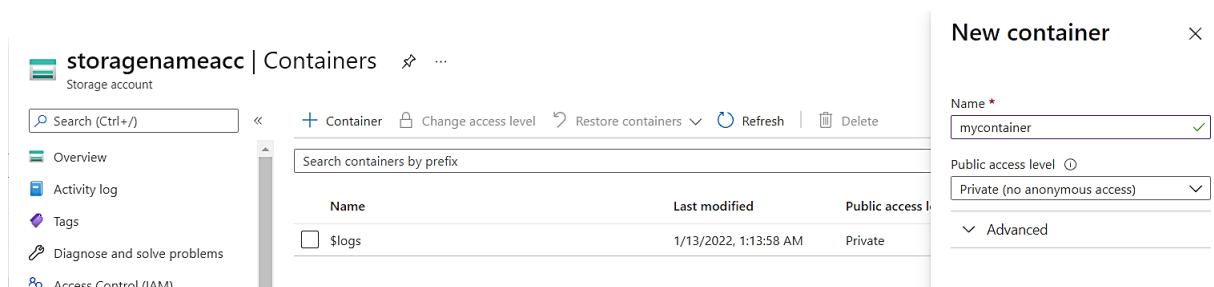


Figure 12: Creating a Container

The next step is to upload the Reference data into the container.

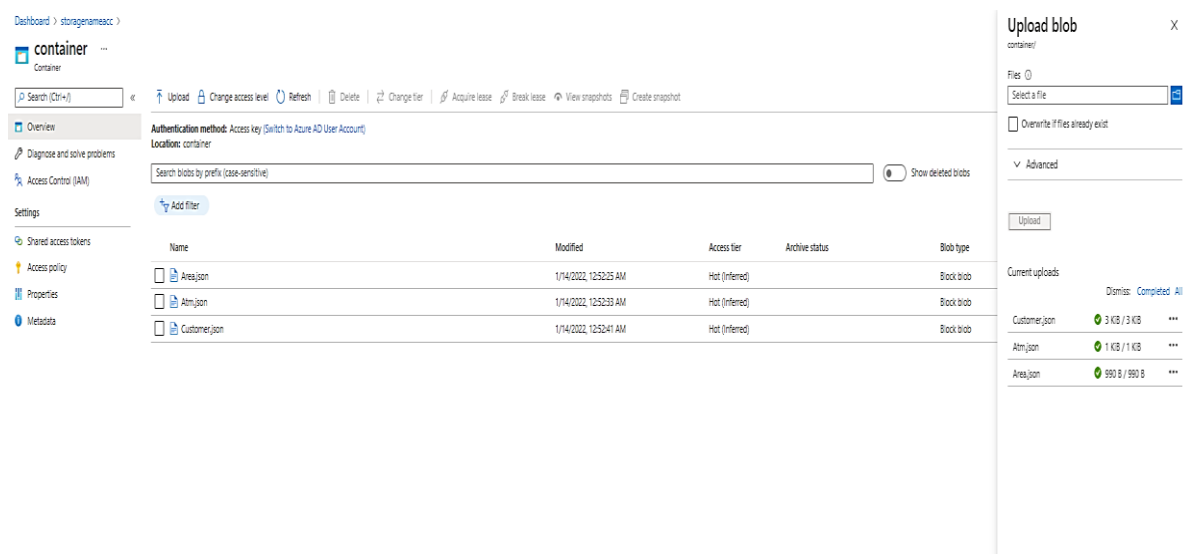
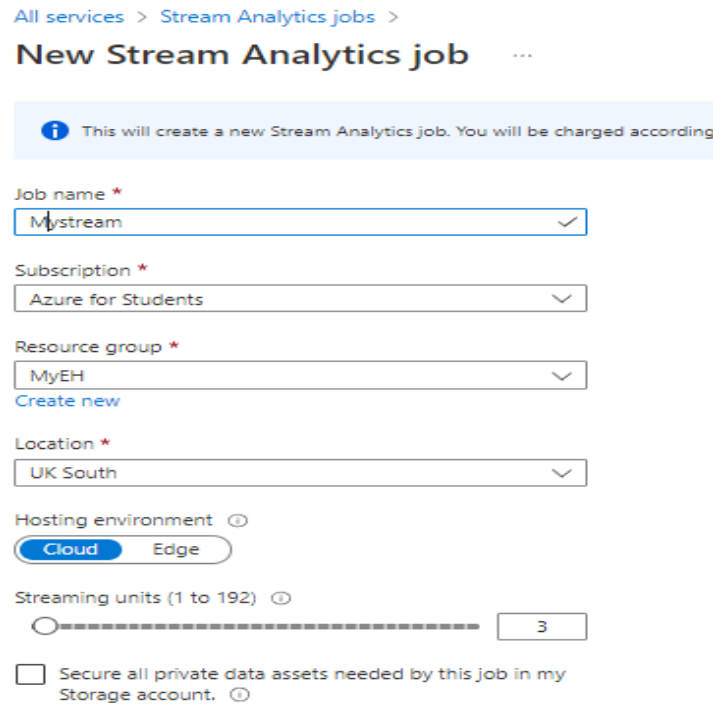


Figure 13: Insert Reference Data into the container

## Set up a Stream Analytics Job

To run live stream task in Azure Analytics we should first create a Stream Analytics Job.



The screenshot shows the 'New Stream Analytics job' form in the Azure portal. At the top, there are navigation links: 'All services' and 'Stream Analytics jobs'. The title is 'New Stream Analytics job'. Below the title is a blue information bar stating: 'This will create a new Stream Analytics job. You will be charged according'. The form contains several fields: 'Job name' with a dropdown menu showing 'Mystream'; 'Subscription' with a dropdown menu showing 'Azure for Students'; 'Resource group' with a dropdown menu showing 'MyEH' and a 'Create new' link; 'Location' with a dropdown menu showing 'UK South'; 'Hosting environment' with two radio buttons, 'Cloud' (selected) and 'Edge'; 'Streaming units (1 to 192)' with a slider and a box showing '3'; and a checkbox labeled 'Secure all private data assets needed by this job in my Storage account.' which is currently unchecked.

Figure 14: Creation of a Stream Analytics Job

We can see below that the deployment of the new steam Analytics job was successful. Then we press “Go to resource”.



The screenshot shows the 'Your deployment is complete' message in the Azure portal. At the top, there are four icons: 'Delete', 'Cancel', 'Redeploy', and 'Refresh'. Below the message, there is a green checkmark icon and the text 'Your deployment is complete'. To the left of the message, there is a purple icon with a white 'i' and the text 'Deployment name: StreamAnalyticsJob', 'Subscription: Azure for Students', and 'Resource group: MyEH'. To the right of the message, there is the text 'Start time: 1/13/2022, 1:05:32 AM' and 'Correlation ID: 57314978-bb15-4861-abd9-0dc727ca2e6b'. Below the message, there is a link 'Deployment details (Download)' and a link 'Next steps'. At the bottom, there is a blue button labeled 'Go to resource'.

Figure 15: Successful Deployment of a Stream Analytics Job

## Part 2 – Azure Analytics Tasks and Solutions

After creating the Stream Analytic Job “Mystream”, we are going to create inputs, outputs for the different queries we are going to examine.

### Create Inputs

We created 4 Inputs. The main input that we named as “Input” is a Stream – Event Hub Input and we will use it to download a data sample to test the validity of our queries. The extra inputs that we named as “InputArea”, “InputATM” and “InputCustomer”, are Reference – Blob storage inputs one for each reference data (Area, ATM and Customer). These inputs will be useful in joining the tables together in queries.

Inputs ...

+ Add stream input + Add reference input Refresh

Inputs can't be added or edited while a job is running. You can stop the job to add or edit inputs.

Name	Source type	Source
Input	Stream	Event Hub
InputArea	Reference	Blob storage
InputATM	Reference	Blob storage
InputCustomer	Reference	Blob storage

Figure 16: The appropriate inputs for the queries

In the outputs below we can see how we created these inputs.

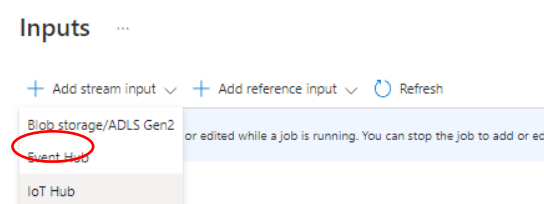


Figure 17: Creation of the Stream Event Hub input

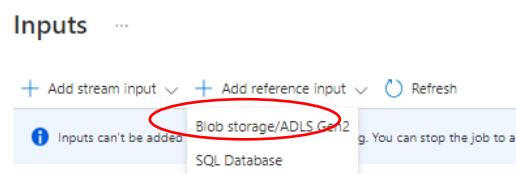


Figure 18: Creation of the Reference – blob storage input

**Input details**  
Input

Test Delete

Input alias  
Input

☒ Provide Event Hub settings manually  
☐ Select Event Hub from your subscriptions

Subscription  
Subscription information not needed

Event Hub namespace \* ⓘ  
ATMtrans

Event Hub name ⓘ  
☐ Create new ☒ Use existing  
myeventhub

Event Hub consumer group ⓘ  
☐ Create new ☒ Use existing  
\$Default

Authentication mode  
Connection string

Event Hub policy name ⓘ  
☐ Create new ☒ Use existing  
mystream\_input\_policy

Event Hub policy key  
.....

Partition key ⓘ

Event serialization format \* ⓘ  
JSON

Figure 19: Creation of the Stream Event Hub Input

**Input details**  
InputArea

Test Delete

Input alias  
InputArea

☒ Provide Blob storage/ADLS Gen2 settings manually  
☐ Select Blob storage/ADLS Gen2 from your subscriptions

Subscription  
Subscription information not needed

Storage account \* ⓘ  
storagenameacc

Container  
☐ Create new ☒ Use existing  
container

Authentication mode  
Connection string

Storage account key  
.....

Path pattern \* ⓘ  
Area.json

Date format  
YYYY/MM/DD

Time format  
HH

Event serialization format \* ⓘ  
JSON

Encoding ⓘ  
UTF-8

Figure 20: Creation of the Reference input

## Create Outputs

In this section we are going to create 9 Blob storage outputs. One main output that we can use to check the validity of our queries in the Test environment and 8 extra queries to create different folders into the container, one for each query.

**Outputs** ...

+ Add ▾ Refresh



Name	Sink
Output	Blob storage
OutputQ1	Blob storage
OutputQ2	Blob storage
OutputQ3	Blob storage
OutputQ4	Blob storage
OutputQ5	Blob storage
OutputQ6	Blob storage
OutputQ7	Blob storage
OutputQ8	Blob storage

Figure 21: Presentation of the outputs we have created

In the output below we can see how we created these Outputs. We should note here that for each Output Qi we declare the path pattern as Query i.

## Output details ✕

OutputQ1

 Test  Delete

---

Output alias

OutputQ1

☒ Provide Blob storage/ADLS Gen2 settings manually  
☐ Select Blob storage/ADLS Gen2 from your subscriptions

Subscription

Subscription information not needed

Storage account \* ⓘ

storagenameacc

Container \* ⓘ

container

Authentication mode

Connection string

Storage account key

.....

Path pattern ⓘ

Query 1

Date format

YYYY/MM/DD

Time format

HH

Event serialization format \* ⓘ

JSON

Figure 22: Creation of the Outputs

After running the queries, we will see the following classification in the container.

container

Container

Search (Ctrl+/)

«

Upload

Change access level

Refresh

Delete

Change tier

Acquire lease

Break lease

View snapshots

Overview

Diagnose and solve problems

Access Control (IAM)

Settings

Shared access tokens

Access policy

Properties

Metadata

Authentication method: Access key (Switch to Azure AD User Account)

Location: container

Search blobs by prefix (case-sensitive)

Add filter

Name	Modified
<input type="checkbox"/> Query 1	
<input type="checkbox"/> Query 2	
<input type="checkbox"/> Query 3	
<input type="checkbox"/> Query 4	
<input type="checkbox"/> Query 5	
<input type="checkbox"/> Query 6	
<input type="checkbox"/> Query 7	
<input type="checkbox"/> Query 8	
<input type="checkbox"/> Area.json	1/14/2022, 12:52:25 AM
<input type="checkbox"/> Atm.json	1/14/2022, 12:52:33 AM
<input type="checkbox"/> Customer.json	1/14/2022, 12:52:41 AM

Figure 23: Folder classification of the outputs

## Feed queries with a data sample

The process we are going to follow to execute a Stream analytic job query is the following. We first test the validity of the query using a small data sample from the “Input” input and using the reference data in the test environment. Once a query is well written we Save the query, and we execute the stream pressing “Start” in the main page of “mystream”. Once the stream finishes, we will be able to view the results in the container page. Below there are presented some outputs from the described process.

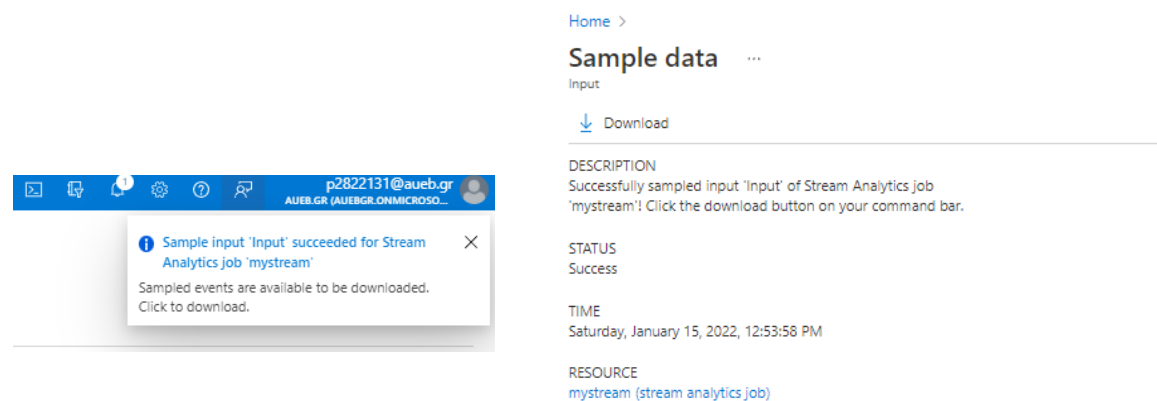


Figure 24: Downloading a Data Sample

Below we can see a part of the data sample we have downloaded.

```

[{"ATMCode":19,
"CardNumber":5200253312538103,
"Type":1,
"Amount":32,
"EventProcessedUtcTime":"2022-01-15T23:59:31.5647107Z",
"PartitionId":0,
"EventEnqueuedUtcTime":"2022-01-15T23:59:18.6430000Z"},
{"ATMCode":12,
"CardNumber":5893112367133403000,
"Type":1,
"Amount":10,
"EventProcessedUtcTime":"2022-01-15T23:59:31.5647107Z",
"PartitionId":0,
"EventEnqueuedUtcTime":"2022-01-15T23:59:18.6590000Z"},
{"ATMCode":19,
"CardNumber":50384191807294800,
"Type":0,
"Amount":19,
"EventProcessedUtcTime":"2022-01-15T23:59:31.5647107Z",
"PartitionId":0,
"EventEnqueuedUtcTime":"2022-01-15T23:59:19.3620000Z"},
{"ATMCode":18,
"CardNumber":56022176913710210,
"Type":0,
"Amount":11,
"EventProcessedUtcTime":"2022-01-15T23:59:31.5647107Z",
"PartitionId":0,
"EventEnqueuedUtcTime":"2022-01-15T23:59:20.6270000Z"},
{"ATMCode":13,
"CardNumber":50383945269330136,
"Type":0,
"Amount":36,
"EventProcessedUtcTime":"2022-01-15T23:59:31.5647107Z",
"PartitionId":0,
"EventEnqueuedUtcTime":"2022-01-15T23:59:21.6430000Z"},
{"ATMCode":13,
"CardNumber":50383945269330136,
"Type":0,
"Amount":10,
"EventProcessedUtcTime":"2022-01-15T23:59:31.5647107Z",
"PartitionId":0,
"EventEnqueuedUtcTime":"2022-01-15T23:59:22.6120000Z"},
{"ATMCode":15,
"CardNumber":30487898026193,
"Type":1,
"Amount":24,
"EventProcessedUtcTime":"2022-01-15T23:59:31.5647107Z",
"PartitionId":0,
"EventEnqueuedUtcTime":"2022-01-15T23:59:23.6590000Z"},
{"ATMCode":19,
"CardNumber":5602246755688900,
"Type":1,
"Amount":12,

```

Figure 25: The data sample that has been downloaded

We upload the data sample in the “Input” and we upload as well the Reference data files “Area.json”, “Atm.json” and “Customer.json” in the “inputArea”, “inputATM” and “InputCustomer” respectively.

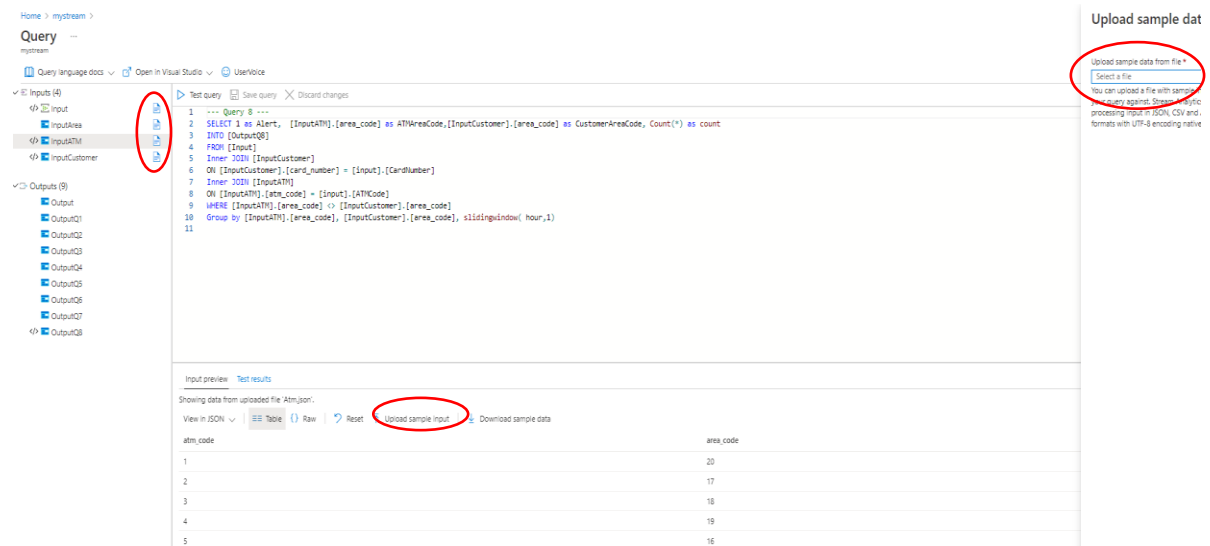


Figure 26: Upload the sample data and test the query

Last step is to press the start button.

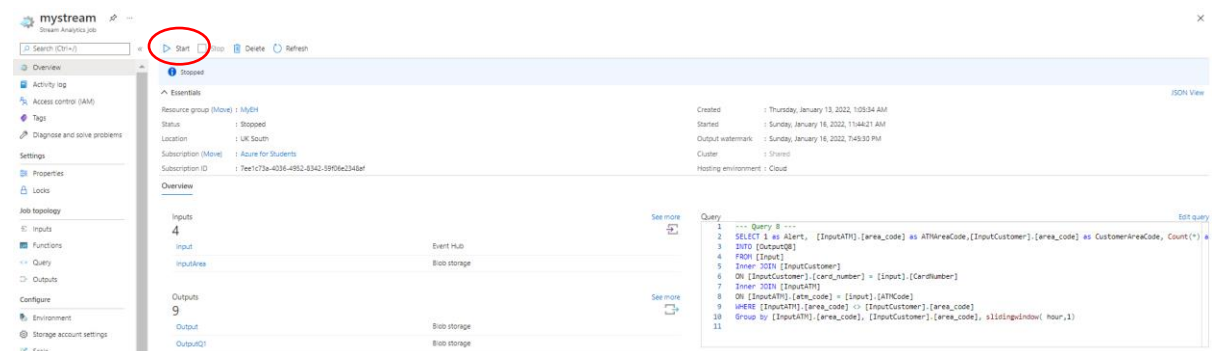


Figure 27: Starting the process



## Azure Analytics Tasks and Solutions

Now we are ready to create queries that can execute important tasks at any time. The most frequently needed task that we are going to analyze are the following:

**Task1:** Show the total “Amount” of “Type=0” transactions at “ATM Code=21” of the last 10 minutes. Repeat as new events keep flowing in (use a sliding window).

First, we created the appropriate query, and we did a run test. The test run returns no outputs because our input sample does not include the ATMCode 21.

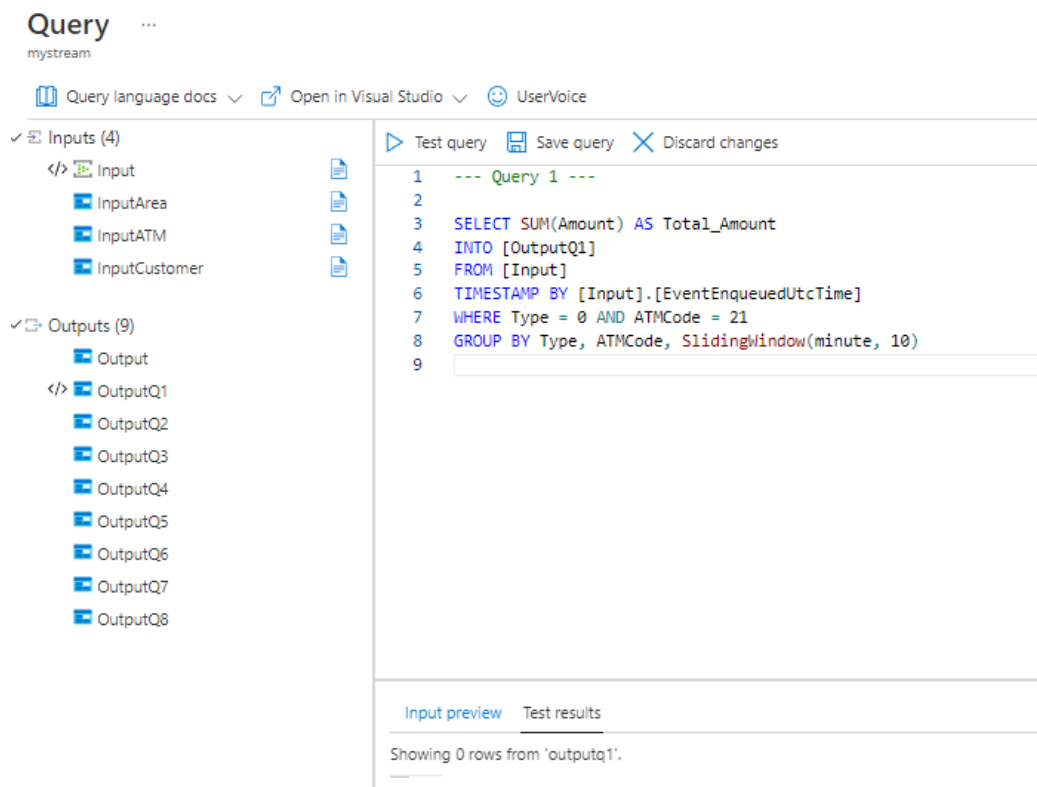


Figure 28: Testing the query

After starting the analytic stream for an hour, the output presented in the container folder “Query 1 “was the following

```

1  {"Total_Amount":18.0}
2  {"Total_Amount":45.0}
3  {"Total_Amount":71.0}
4  {"Total_Amount":116.0}
5  {"Total_Amount":162.0}
6  {"Total_Amount":208.0}
7  {"Total_Amount":247.0}
8  {"Total_Amount":229.0}
9  {"Total_Amount":202.0}
10 {"Total_Amount":176.0}
11 {"Total_Amount":218.0}
12 {"Total_Amount":173.0}
13 {"Total_Amount":127.0}
14 {"Total_Amount":81.0}
15 {"Total_Amount":42.0}

```

Figure 29:Output of the Stream

**Task2:** Show the total “Amount” of “Type=1” transactions at “ATMCode=21” of the last hour. Repeat once every hour (use a tumbling window).

First, we created the appropriate query, and we did a run test. The test run returns no outputs because our input sample does not include the ATMCode 21.

The screenshot shows the Azure Databricks Query Editor interface. On the left, there is a sidebar with 'Inputs (4)' and 'Outputs (9)'. The 'Inputs' section includes 'Input', 'InputArea', 'InputATM', and 'InputCustomer'. The 'Outputs' section includes 'Output', 'OutputQ1', 'OutputQ2', 'OutputQ3', 'OutputQ4', 'OutputQ5', 'OutputQ6', 'OutputQ7', and 'OutputQ8'. The main area displays a SQL query in a text editor. The query is as follows:

```

--- Query 2 ---
SELECT SUM(Amount) AS Total_Amount
INTO [OutputQ2]
FROM [Input]
TIMESTAMP BY [Input].[EventEnqueuedUtcTime]
WHERE Type = 1 AND ATMCode = 21
GROUP BY Type, ATMCode, TumblingWindow(hour, 1)

```

Below the query editor, there are tabs for 'Input preview' and 'Test results'. The 'Test results' tab is active, showing the message: 'Showing 0 rows from 'outputq2'.'

Figure 30: Testing the query

After starting the analytic stream for an hour, the output presented in the container folder “Query 2” was the following:

```
1 {"Total_Amount":472.0}
```

Figure 31: Output of the stream

**Task3:** Show the total “Amount” of “Type =1” transactions at “ATM Code =21” of the last hour. Repeat once every 30 minutes (use hopping window)

First, we created the appropriate query, and we did a run test. The test run returns no outputs because our input sample does not include the ATMCode 21.

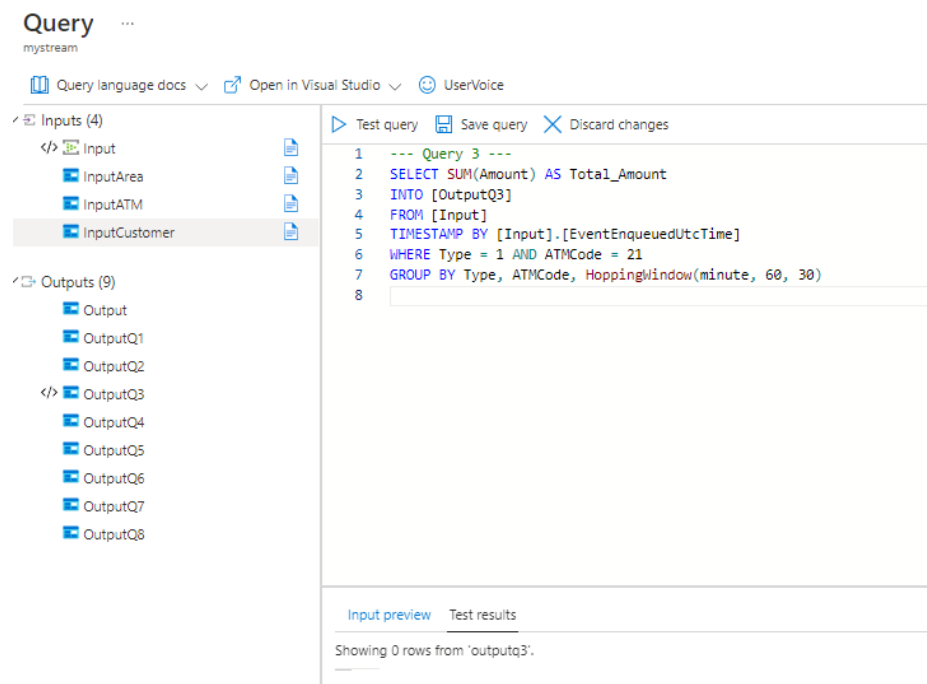


Figure 32: Testing the query

After starting the analytic stream for an hour, the output presented in the container folder “Query 3” was the following

```

1 [{"Total_Amount":42.0}]
2 [{"Total_Amount":137.0}]

```

Figure 33: Output of the stream

**Task 4:** Show the total “Amount” of “Type = 1” transactions per “ATM Code” of the last one hour (use a sliding window).

First, we created the appropriate query, and we did a run test. The test run returns the output above.

The screenshot shows the Query Editor interface with a SQL query and its test results. The query is as follows:

```

1 --- Query 4 ---
2 SELECT ATMCode, SUM(Amount) AS Total_Amount
3 INTO [Output4]
4 FROM [Input]
5 Timestamp BY [Input].[EventInQueueUtcTime]
6 WHERE Type = 1
7 GROUP BY Type, ATMCode, SlidingWindow(hour, 1)
8
9

```

The test results show the following data:

ATMCode	Total_Amount
19	32
12	10
15	24
19	44
19	12

Figure 34: Testing the query

After starting the analytic stream for several hours, the output presented in the container folder “Query 4 “was the following

```

1 {"ATMCode":15,"Total_Amount":275.0}
2 {"ATMCode":18,"Total_Amount":204.0}
3 {"ATMCode":18,"Total_Amount":179.0}
4 {"ATMCode":19,"Total_Amount":80.0}
5 {"ATMCode":20,"Total_Amount":113.0}
6 {"ATMCode":18,"Total_Amount":167.0}
7 {"ATMCode":15,"Total_Amount":302.0}
8 {"ATMCode":20,"Total_Amount":100.0}
9 {"ATMCode":20,"Total_Amount":76.0}
10 {"ATMCode":19,"Total_Amount":118.0}
11 {"ATMCode":21,"Total_Amount":42.0}
12 {"ATMCode":15,"Total_Amount":264.0}
13 {"ATMCode":15,"Total_Amount":239.0}
14 {"ATMCode":13,"Total_Amount":89.0}
15 {"ATMCode":15,"Total_Amount":208.0}
16 {"ATMCode":13,"Total_Amount":73.0}
17 {"ATMCode":18,"Total_Amount":119.0}
18 {"ATMCode":15,"Total_Amount":163.0}
19 {"ATMCode":10,"Total_Amount":195.0}
20 {"ATMCode":15,"Total_Amount":149.0}
21 {"ATMCode":10,"Total_Amount":158.0}
22 {"ATMCode":19,"Total_Amount":88.0}
23 {"ATMCode":18,"Total_Amount":82.0}
24 {"ATMCode":15,"Total_Amount":102.0}
25 {"ATMCode":18,"Total_Amount":40.0}
26 {"ATMCode":20,"Total_Amount":54.0}
27 {"ATMCode":13,"Total_Amount":27.0}
28 {"ATMCode":15,"Total_Amount":54.0}
29 {"ATMCode":20,"Total_Amount":30.0}
30 {"ATMCode":15,"Total_Amount":27.0}
31 {"ATMCode":10,"Total_Amount":116.0}
32 {"ATMCode":10,"Total_Amount":85.0}
33 {"ATMCode":10,"Total_Amount":58.0}
34 {"ATMCode":10,"Total_Amount":46.0}

```

Figure 35: Output of the stream

**Task 5:** Show the total “Amount” of “Type = 1” transactions per “Area Code” of the last hour. Repeat once every hour (use a tumbling window).

First, we created the appropriate query, and we did a run test. The test run returns the output above:

The screenshot shows the Azure Databricks Query Editor interface. On the left, there's a sidebar with 'Inputs (4)' (InputArea, InputATM, InputCustomer) and 'Outputs (9)' (Output, OutputQ1, OutputQ2, OutputQ3, OutputQ4, OutputQ5, OutputQ6, OutputQ7, OutputQ8). The main area displays a SQL query for 'Query 5'.

```

1 --- Query 5 ---
2
3 SELECT [InputATM].[area_code], SUM([Input].[Amount]) AS Total_Amount
4 INTO [OutputQ5]
5 FROM [Input]
6 TIMESTAMP BY [Input].[EventEnqueuedUtcTime]
7 left JOIN [InputATM]
8 ON [Input].[ATMCode]=[InputATM].[atm_code]
9 WHERE [Input].[Type] = 1
10 GROUP BY [Input].[Type], [InputATM].[area_code], TumblingWindow(hour, 1)
11

```

Below the query, the 'Test results' section shows a table with 3 rows from 'outputq5':

area_code	Total_Amount
2	44
9	10
5	24

Figure 36: Testing the query

After starting the analytic stream for several hours, the output presented in the container folder “Query 5 “was the following:

```

1 {"area_code":5,"Total_Amount":2654.0}
2 {"area_code":4,"Total_Amount":2070.0}
3 {"area_code":14,"Total_Amount":50.0}
4 {"area_code":11,"Total_Amount":1779.0}
5 {"area_code":19,"Total_Amount":83.0}
6 {"area_code":3,"Total_Amount":535.0}
7 {"area_code":6,"Total_Amount":51.0}
8 {"area_code":16,"Total_Amount":31.0}
9 {"area_code":9,"Total_Amount":637.0}
10 {"area_code":12,"Total_Amount":45.0}
11 {"area_code":13,"Total_Amount":36.0}
12 {"area_code":7,"Total_Amount":400.0}
13 {"area_code":10,"Total_Amount":737.0}
14 {"area_code":1,"Total_Amount":2042.0}
15 {"area_code":2,"Total_Amount":1285.0}
16 {"area_code":5,"Total_Amount":74.0}
17 {"area_code":4,"Total_Amount":70.0}
18 {"area_code":11,"Total_Amount":260.0}
19 {"area_code":3,"Total_Amount":55.0}
20 {"area_code":7,"Total_Amount":61.0}
21 {"area_code":10,"Total_Amount":18.0}
22 {"area_code":1,"Total_Amount":126.0}
23 {"area_code":2,"Total_Amount":141.0}

```

Figure 37: Output of the stream

**Task 6:** Show the total “Amount” per ATM’s “City” and Customer’s “Gender” of the last hour. Repeat once every hour (use a tumbling window).

First, we created the appropriate query, and we did a run test. The test run returns the output above:

The screenshot shows the Databricks Query Editor interface. On the left, there are sections for 'Inputs (4)' and 'Outputs (9)'. The main area displays a SQL query for 'Query 6'. Below the query, the 'Test results' tab is active, showing a table with 5 rows of data from 'output0'.

```

1 --- Query 6 ---
2 SELECT [InputArea].[area_city], [InputCustomer].[gender], SUM([Input].[Amount]) AS Total_Amount
3 INTO [Output0]
4 FROM [Input]
5
6 Timestamp BY [Input].[EventEnqueuedUtcTime]
7 LEFT JOIN [InputCustomer]
8 ON [InputCustomer].[card_number] = [Input].[CardNumber]
9 LEFT JOIN [InputATM]
10 ON [InputATM].[atm_code] = [Input].[ATMCode]
11 LEFT JOIN [InputArea]
12 ON [InputArea].[area_code] = [InputATM].[area_code]
13 GROUP BY [InputArea].[area_city], [InputCustomer].[gender], TumblingWindow(hour, 1)
14

```

area_city	gender	Total_Amount
"Omaha"	"Male"	63
"Schaumburg"	"Female"	24
"Memphis"	"Female"	11
"Baltimore"	"Male"	46
"Canton"	"Male"	10

Figure 38: Testing the query

After starting the analytic stream for several hours, the output presented in the container folder “Query 6 “was the following:

```

1 [{"area_city":"Vancouver","gender":"Female","Total_Amount":128.0}]
2 [{"area_city":"Dayton","gender":"Female","Total_Amount":49.0}]
3 [{"area_city":"Canton","gender":"Female","Total_Amount":49.0}]
4 [{"area_city":null,"gender":"Female","Total_Amount":2791.0}]
5 [{"area_city":"Schaumburg","gender":"Male","Total_Amount":1469.0}]
6 [{"area_city":"Baltimore","gender":"Male","Total_Amount":669.0}]
7 [{"area_city":"Baltimore","gender":"Female","Total_Amount":744.0}]
8 [{"area_city":"Memphis","gender":"Male","Total_Amount":2750.0}]
9 [{"area_city":"Springfield","gender":"Female","Total_Amount":1281.0}]
10 [{"area_city":null,"gender":"Male","Total_Amount":1369.0}]
11 [{"area_city":"Memphis","gender":"Female","Total_Amount":1108.0}]
12 [{"area_city":"Schaumburg","gender":"Female","Total_Amount":3348.0}]
13 [{"area_city":"Omaha","gender":"Male","Total_Amount":2084.0}]
14 [{"area_city":"Vancouver","gender":"Male","Total_Amount":928.0}]
15 [{"area_city":"Springfield","gender":"Male","Total_Amount":2719.0}]
16 [{"area_city":"Dayton","gender":"Male","Total_Amount":15.0}]
17 [{"area_city":"Tacoma","gender":"Female","Total_Amount":960.0}]
18 [{"area_city":"Omaha","gender":"Female","Total_Amount":437.0}]
19 [{"area_city":"Canton","gender":"Male","Total_Amount":1329.0}]
20 [{"area_city":null,"gender":"Female","Total_Amount":389.0}]
21 [{"area_city":"Schaumburg","gender":"Male","Total_Amount":126.0}]
22 [{"area_city":"Baltimore","gender":"Male","Total_Amount":103.0}]
23 [{"area_city":"Memphis","gender":"Male","Total_Amount":173.0}]
24 [{"area_city":"Baltimore","gender":"Female","Total_Amount":18.0}]
25 [{"area_city":"Springfield","gender":"Female","Total_Amount":140.0}]
26 [{"area_city":null,"gender":"Male","Total_Amount":137.0}]
27 [{"area_city":"Memphis","gender":"Female","Total_Amount":13.0}]
28 [{"area_city":"Schaumburg","gender":"Female","Total_Amount":133.0}]
29 [{"area_city":"Omaha","gender":"Male","Total_Amount":206.0}]
30 [{"area_city":"Vancouver","gender":"Male","Total_Amount":55.0}]
31 [{"area_city":"Springfield","gender":"Male","Total_Amount":97.0}]
32 [{"area_city":"Tacoma","gender":"Female","Total_Amount":61.0}]
33 [{"area_city":"Canton","gender":"Male","Total_Amount":53.0}]

```

Figure 39: Output of the stream

**Task 7:** Alert (Do a simple *SELECT* “1”) if a customer has performed two transactions of “Type = 1” in a window of an hour (use a sliding window).

First, we created the appropriate query, and we did a run test. The test run returns the output above:

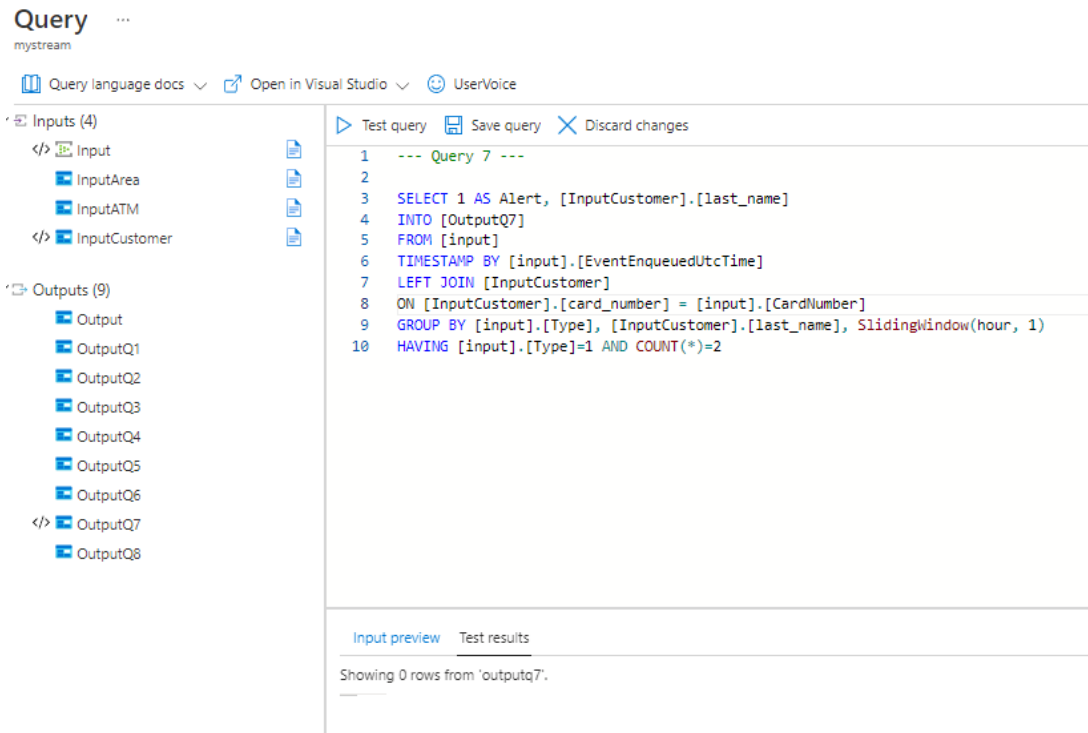


Figure 40: Testing the query

After starting the analytic stream for several hours, the output presented in the container folder “Query 7 “was the following:



```
{ "Alert": 1, "last_name": "Fuller" }
{ "Alert": 1, "last_name": "Mitchell" }
{ "Alert": 1, "last_name": "Perry" }
{ "Alert": 1, "last_name": "Perez" }
{ "Alert": 1, "last_name": "Simpson" }
{ "Alert": 1, "last_name": "Sims" }
{ "Alert": 1, "last_name": "Mason" }
{ "Alert": 1, "last_name": "Snyder" }
{ "Alert": 1, "last_name": "Young" }
{ "Alert": 1, "last_name": "Stone" }
{ "Alert": 1, "last_name": "Day" }
{ "Alert": 1, "last_name": "Russell" }
{ "Alert": 1, "last_name": "Jordan" }
{ "Alert": 1, "last_name": "Moreno" }
{ "Alert": 1, "last_name": "Morrison" }
{ "Alert": 1, "last_name": "Lee" }
{ "Alert": 1, "last_name": "Hansen" }
{ "Alert": 1, "last_name": "Cooper" }
{ "Alert": 1, "last_name": "Bradley" }
{ "Alert": 1, "last_name": "Carroll" }
{ "Alert": 1, "last_name": "Stone" }
{ "Alert": 1, "last_name": "Young" }
{ "Alert": 1, "last_name": "Simpson" }
{ "Alert": 1, "last_name": "Perry" }
{ "Alert": 1, "last_name": "Mitchell" }
{ "Alert": 1, "last_name": "Hansen" }
{ "Alert": 1, "last_name": "Jordan" }
{ "Alert": 1, "last_name": "Cooper" }
{ "Alert": 1, "last_name": "Moreno" }
{ "Alert": 1, "last_name": "Perez" }
{ "Alert": 1, "last_name": "Lee" }
{ "Alert": 1, "last_name": "Mason" }
{ "Alert": 1, "last_name": "Morrison" }
{ "Alert": 1, "last_name": "Fuller" }
{ "Alert": 1, "last_name": "Russell" }
{ "Alert": 1, "last_name": "Day" }
{ "Alert": 1, "last_name": "Snyder" }
{ "Alert": 1, "last_name": "Bradley" }
{ "Alert": 1, "last_name": "Sims" }
{ "Alert": 1, "last_name": "Russell" }
{ "Alert": 1, "last_name": "Bradley" }
{ "Alert": 1, "last_name": "Fuller" }
{ "Alert": 1, "last_name": "Hansen" }
{ "Alert": 1, "last_name": "Mitchell" }
{ "Alert": 1, "last_name": "Lee" }
{ "Alert": 1, "last_name": "Young" }
```

Figure 41: Output of the stream

**Task 8:** Alert (Do a simple *SELECT "1"*) if the “Area Code” of the ATM of the transaction is not the same as the “Area Code” of the “Card Number” (Customer’s Area Code) - (use a sliding window)

First, we created the appropriate query, and we did a run test. The test run returns the output above:

The screenshot shows a query editor with a SQL query and its test results. The query is as follows:

```

1 --- Query 8 ---
2 SELECT 1 as Alert, [InputATM].[area_code] as ATMAreaCode, [InputCustomer].[area_code] as CustomerAreaCode, Count(*) as count
3 INTO [OutputQ8]
4 FROM [Input]
5 INNER JOIN [InputCustomer]
6 ON [InputCustomer].[card_number] = [Input].[CardNumber]
7 INNER JOIN [InputATM]
8 ON [InputATM].[atm_code] = [Input].[ATMCode]
9 WHERE [InputATM].[area_code] <> [InputCustomer].[area_code]
10 Group by [InputATM].[area_code], [InputCustomer].[area_code], slidingwindow( hour,1)
11

```

The test results show 7 rows from outputQ8:

Alert	ATMAreaCode	CustomerAreaCode	count
1	2	1	1
1	9	10	1
1	2	1	3
1	4	2	1
1	10	6	2
1	5	7	1
1	2	1	2

Figure 42: Testing the query

After starting the analytic stream for several hours, the output presented in the container folder “Query 8 “was the following:

```

1 [{"Alert":1,"ATMAreaCode":10,"CustomerAreaCode":6,"count":4}]
2 [{"Alert":1,"ATMAreaCode":2,"CustomerAreaCode":1,"count":8}]
3 [{"Alert":1,"ATMAreaCode":2,"CustomerAreaCode":1,"count":7}]
4 [{"Alert":1,"ATMAreaCode":1,"CustomerAreaCode":6,"count":3}]
5 [{"Alert":1,"ATMAreaCode":5,"CustomerAreaCode":7,"count":5}]
6 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":10}]
7 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":9}]
8 [{"Alert":1,"ATMAreaCode":10,"CustomerAreaCode":6,"count":3}]
9 [{"Alert":1,"ATMAreaCode":10,"CustomerAreaCode":6,"count":2}]
10 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":8}]
11 [{"Alert":1,"ATMAreaCode":5,"CustomerAreaCode":7,"count":4}]
12 [{"Alert":1,"ATMAreaCode":10,"CustomerAreaCode":6,"count":1}]
13 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":7}]
14 [{"Alert":1,"ATMAreaCode":1,"CustomerAreaCode":6,"count":2}]
15 [{"Alert":1,"ATMAreaCode":1,"CustomerAreaCode":6,"count":1}]
16 [{"Alert":1,"ATMAreaCode":2,"CustomerAreaCode":1,"count":6}]
17 [{"Alert":1,"ATMAreaCode":2,"CustomerAreaCode":1,"count":5}]
18 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":6}]
19 [{"Alert":1,"ATMAreaCode":4,"CustomerAreaCode":2,"count":7}]
20 [{"Alert":1,"ATMAreaCode":4,"CustomerAreaCode":2,"count":6}]
21 [{"Alert":1,"ATMAreaCode":4,"CustomerAreaCode":2,"count":5}]
22 [{"Alert":1,"ATMAreaCode":4,"CustomerAreaCode":2,"count":4}]
23 [{"Alert":1,"ATMAreaCode":5,"CustomerAreaCode":7,"count":3}]
24 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":5}]
25 [{"Alert":1,"ATMAreaCode":5,"CustomerAreaCode":7,"count":2}]
26 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":4}]
27 [{"Alert":1,"ATMAreaCode":9,"CustomerAreaCode":10,"count":1}]
28 [{"Alert":1,"ATMAreaCode":5,"CustomerAreaCode":7,"count":1}]
29 [{"Alert":1,"ATMAreaCode":2,"CustomerAreaCode":1,"count":4}]
30 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":3}]
31 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":2}]
32 [{"Alert":1,"ATMAreaCode":2,"CustomerAreaCode":1,"count":3}]
33 [{"Alert":1,"ATMAreaCode":11,"CustomerAreaCode":8,"count":1}]
34 [{"Alert":1,"ATMAreaCode":4,"CustomerAreaCode":2,"count":3}]
35 [{"Alert":1,"ATMAreaCode":2,"CustomerAreaCode":1,"count":2}]
36 [{"Alert":1,"ATMAreaCode":2,"CustomerAreaCode":1,"count":1}]
37 [{"Alert":1,"ATMAreaCode":4,"CustomerAreaCode":2,"count":2}]

```

Figure 43: The output of the stream