Lab 9 - Support Hybrid Transactional Analytical Processing (HTAP) with Azure Synapse Link

In this lab, you will learn how Azure Synapse Link enables seamless connectivity of an Azure Cosmos DB account to a Synapse workspace. You will learn how to enable and configure Synapse link, then how to query the Azure Cosmos DB analytical store using Apache Spark and SQL Serverless.

After completing this lab, you will be able to:

- Configure Azure Synapse Link with Azure Cosmos DB
- Query Azure Cosmos DB with Apache Spark for Synapse Analytics
- Query Azure Cosmos DB with serverless SQL pool for Azure Synapse Analytics

Lab setup and pre-requisites

Before starting this lab, you should complete **Lab 6: *Transform data with Azure Data Factory or Azure Synapse Pipelines***.

> **Note**: If you have ***not*** completed lab 6, but you have completed the lab setup for this course, you can complete these steps to create the required linked service and dataset.
>
> 1. In Synapse Studio, on the **Manage** hub, add a new **Linked service** for **Azure Cosmos DB (SQL API)** with the following settings:
>   - **Name**: asacosmosdb01
>   - **Cosmos DB account name**: asacosmosdb*xxxxxxx*
>   - **Database name**: CustomerProfile
> 2. On the **Data** hub, create the following **Integration dataset**:
>   - **Source**: Azure Cosmos DB (SQL API)
>   - **Name**: asal400_customerprofile_cosmosdb
>   - **Linked service**: asacosmosdb01
>   - **Collection**: OnlineUserProfile01
>   - **Import schema**: From connection/store

Exercise 1 - Configuring Azure Synapse Link with Azure Cosmos DB

Tailwind Traders uses Azure Cosmos DB to store user profile data from their eCommerce site. The NoSQL document store provided by the Azure Cosmos DB SQL API provides the familiarity of managing their data using SQL syntax, while being able to read and write the files at a massive, global scale.

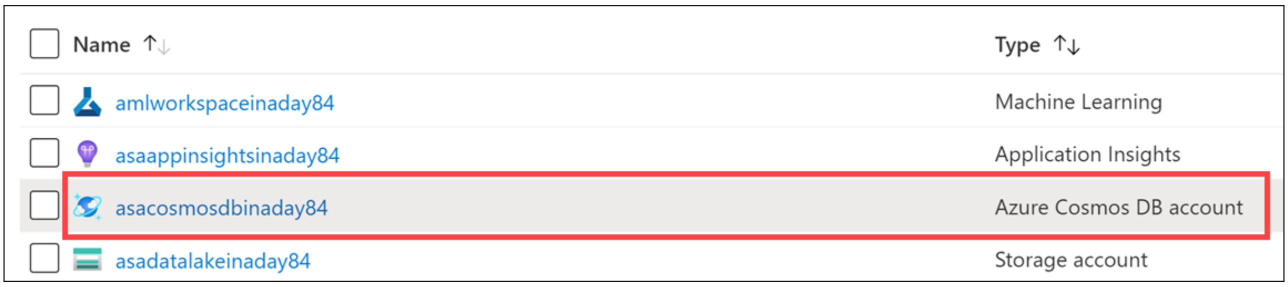
While Tailwind Traders is happy with the capabilities and performance of Azure Cosmos DB, they are concerned about the cost of executing a large volume of analytical queries over multiple partitions (cross-partition queries) from their data warehouse. They want to efficiently access all the data without needing to increase the Azure Cosmos DB request units (RUs). They have looked at options for extracting data from their containers to the data lake as it changes, through the Azure Cosmos DB change feed mechanism. The problem with this approach is the extra service and code dependencies and long-term maintenance of the solution. They could perform bulk exports from a Synapse Pipeline, but then they won't have the most up-to-date information at any given moment.

You decide to enable Azure Synapse Link for Cosmos DB and enable the analytical store on their Azure Cosmos DB containers. With this configuration, all transactional data is automatically stored in a fully isolated column store. This store enables large-scale analytics against the operational data in Azure Cosmos DB, without impacting the transactional workloads or incurring resource unit (RU) costs. Azure Synapse Link for Cosmos DB creates a tight integration between Azure Cosmos DB and Azure Synapse Analytics, which enables Tailwind Traders to run near real-time analytics over their operational data with no-ETL and full performance isolation from their transactional workloads.

By combining the distributed scale of Cosmos DB's transactional processing with the built-in analytical store and the computing power of Azure Synapse Analytics, Azure Synapse Link enables a Hybrid Transactional/Analytical Processing (HTAP) architecture for optimizing Tailwind Trader's business processes. This integration eliminates ETL processes, enabling business analysts, data engineers & data scientists to self-serve and run near real-time BI, analytics, and Machine Learning pipelines over operational data.

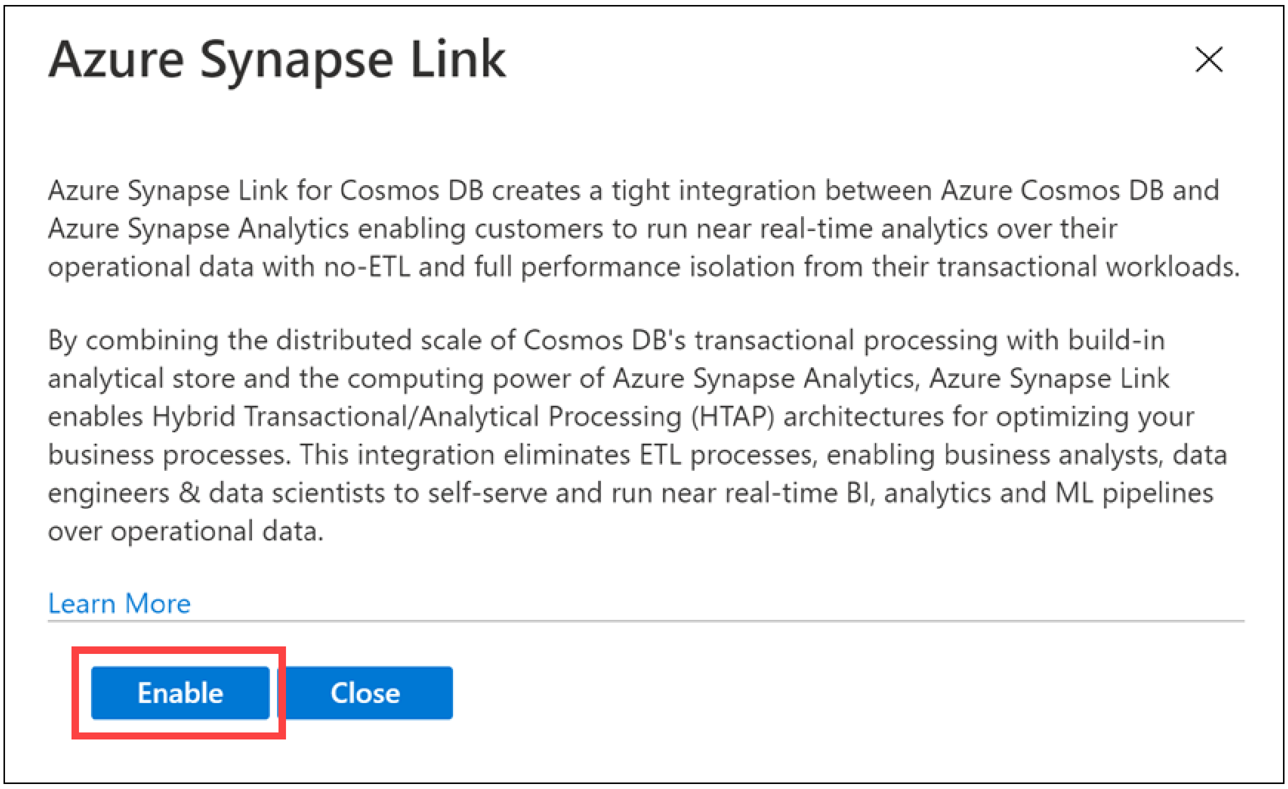
Task 1: Enable Azure Synapse Link

1. In the Azure portal (https://portal.azure.com), open the resource group for your lab environment.
2. Select the **Azure Cosmos DB account**.



Name	Type
amlworkspaceinaday84	Machine Learning
asaappinsightsinaday84	Application Insights
asacosmosdbinaday84	Azure Cosmos DB account
asadatalakeinaday84	Storage account

3. In the left-hand menu, select **Azure Synapse Link**.
4. Select **Enable**.



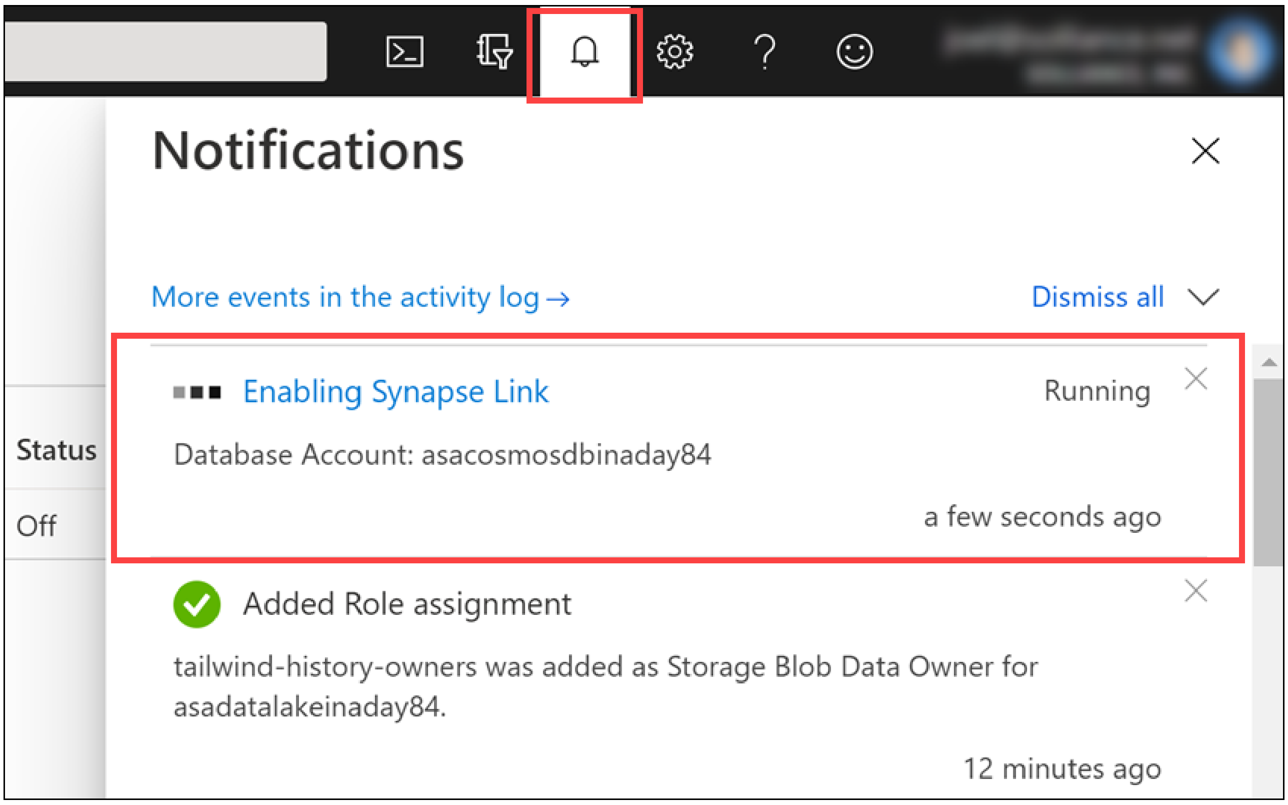
Azure Synapse Link

Azure Synapse Link for Cosmos DB creates a tight integration between Azure Cosmos DB and Azure Synapse Analytics enabling customers to run near real-time analytics over their operational data with no-ETL and full performance isolation from their transactional workloads.

By combining the distributed scale of Cosmos DB's transactional processing with build-in analytical store and the computing power of Azure Synapse Analytics, Azure Synapse Link enables Hybrid Transactional/Analytical Processing (HTAP) architectures for optimizing your business processes. This integration eliminates ETL processes, enabling business analysts, data engineers & data scientists to self-serve and run near real-time BI, analytics and ML pipelines over operational data.

Before we can create an Azure Cosmos DB container with an analytical store, we must first enable Azure Synapse Link.

5. You must wait for this operation to complete before continuing, which should take about a minute. Check the status by selecting the Azure **Notifications** icon.



Notifications

Enabling Synapse Link Running

Database Account: asacosmosdbinaday84

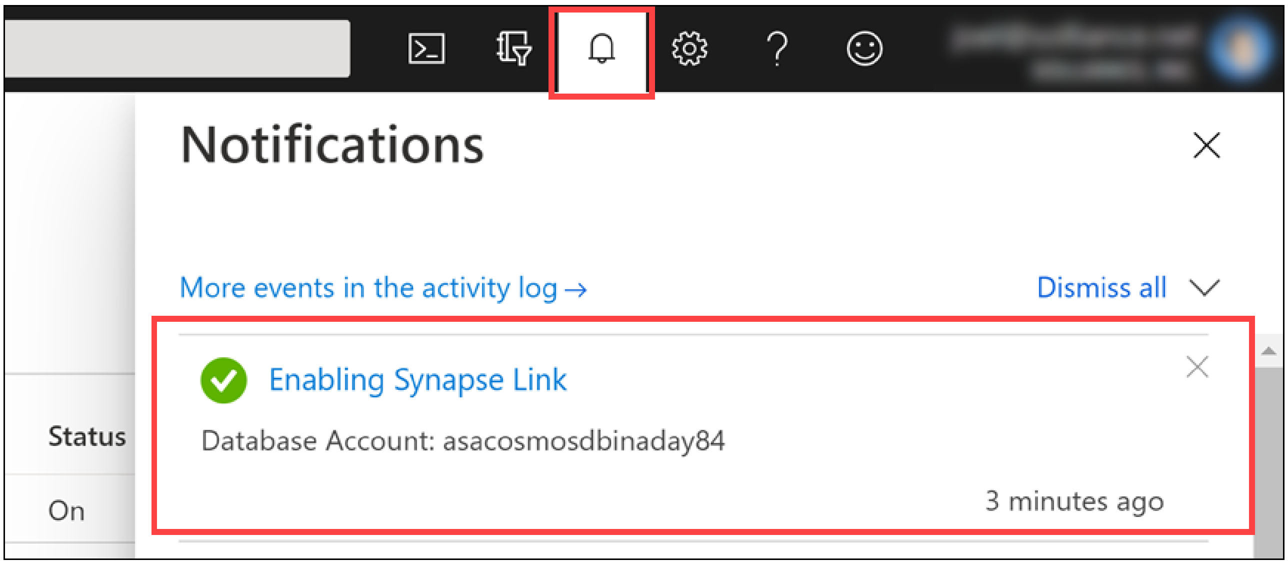
a few seconds ago

Added Role assignment

tailwind-history-owners was added as Storage Blob Data Owner for asadatalakeinaday84.

12 minutes ago

You will see a green checkmark next to "Enabling Synapse Link" when it successfully completes.


Notifications

More events in the activity log →

Dismiss all

Enabling Synapse Link

Database Account: asacosmosdbinaday84

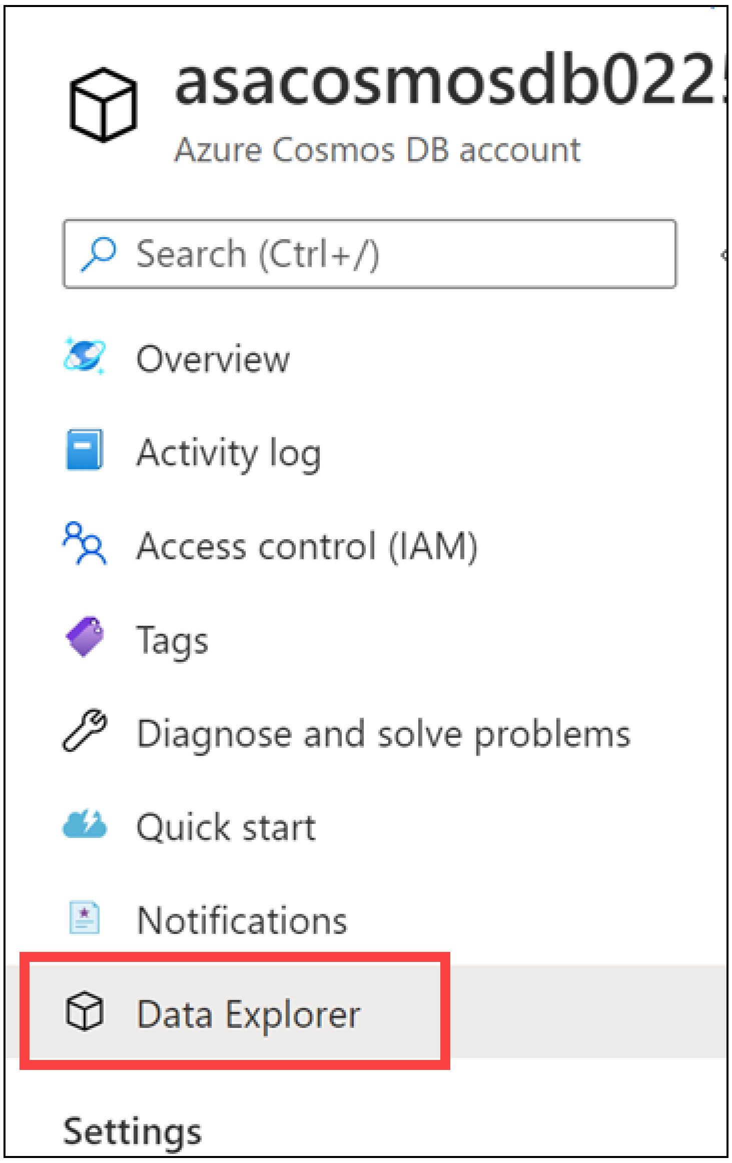
3 minutes ago

Task 2: Create a new Azure Cosmos DB container

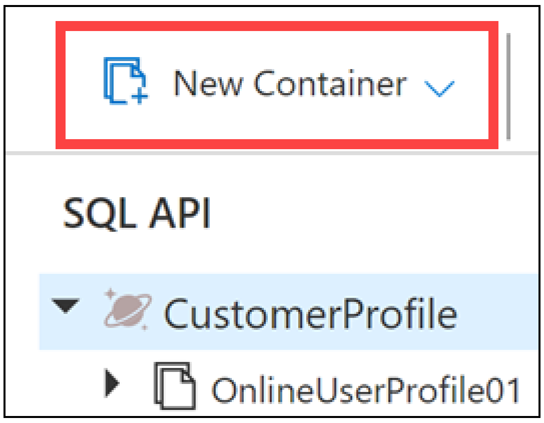
Tailwind Traders has an Azure Cosmos DB container named **OnlineUserProfile01**. Since we enabled the Azure Synapse Link feature *after* the container was already created, we cannot enable the analytical store on the container. We will create a new container that has the same partition key and enable the analytical store.

After creating the container, we will create a new Synapse Pipeline to copy data from the **OnlineUserProfile01** container to the new one.

1. Select **Data Explorer** on the left-hand menu.

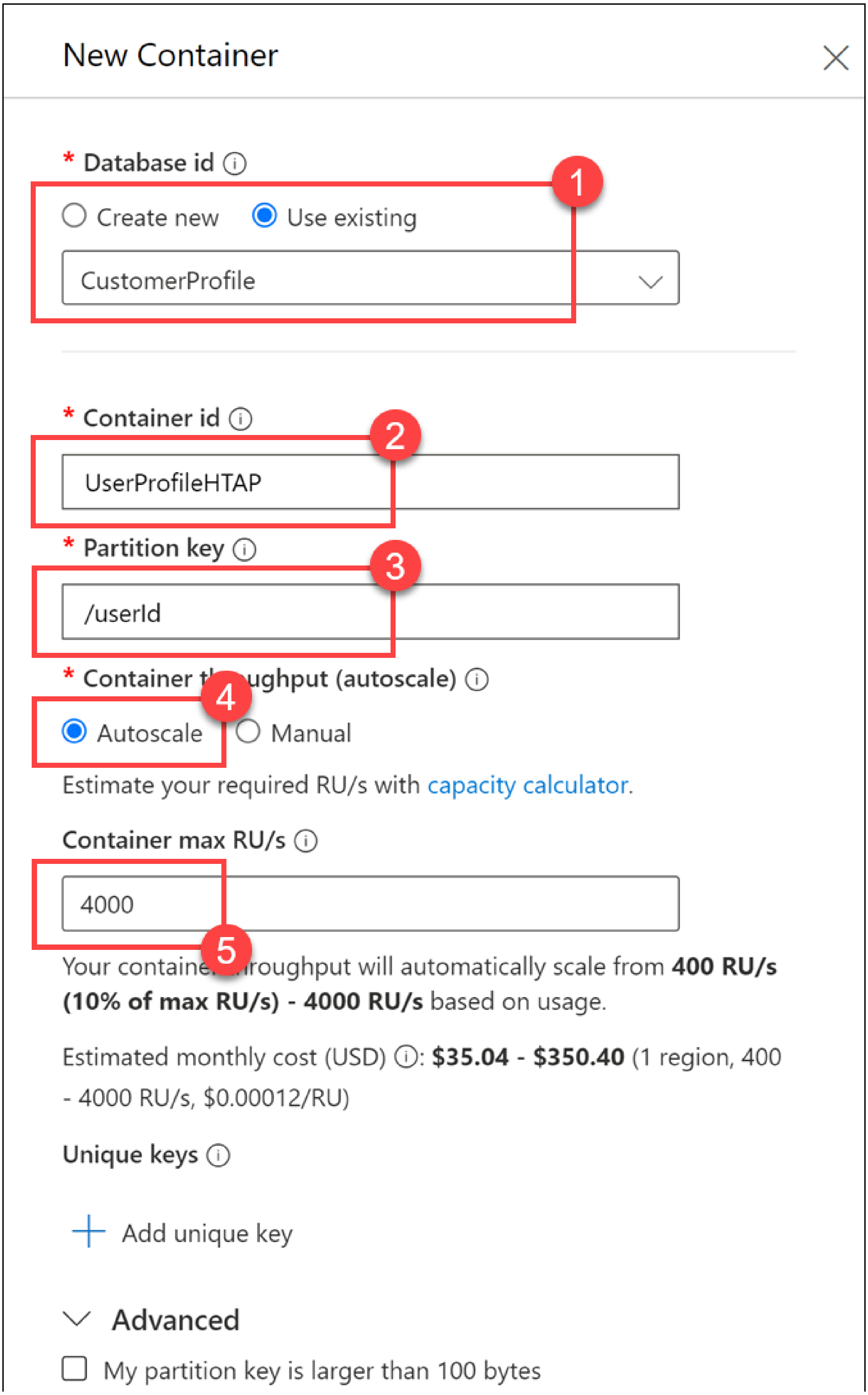


2. Select **New Container**.



3. Create a new container with the following settings:

   - **Database id**: Use the existing **CustomerProfile** database.
   - **Container id**: Enter `UserProfileHTAP`
   - **Partition key**: Enter `/userId`
   - **Throughput**: Select **Autoscale**
   - **Container max RU/s**: Enter `4000`
   - **Analytical store**: On


New Container

* Database id

( ) Create new [✓ Use existing]

CustomerProfile

* Container id

UserProfileHTAP

* Partition key

/userId

* Container throughput (autoscale)

[✓ Autoscale] ( ) Manual

Estimate your required RU/s with capacity calculator.

Container max RU/s

4000

Your container throughput will automatically scale from **400 RU/s (10% of max RU/s) - 4000 RU/s** based on usage.

Estimated monthly cost (USD): **$35.04 - $350.40** (1 region, 400 - 4000 RU/s, $0.00012/RU)

Unique keys

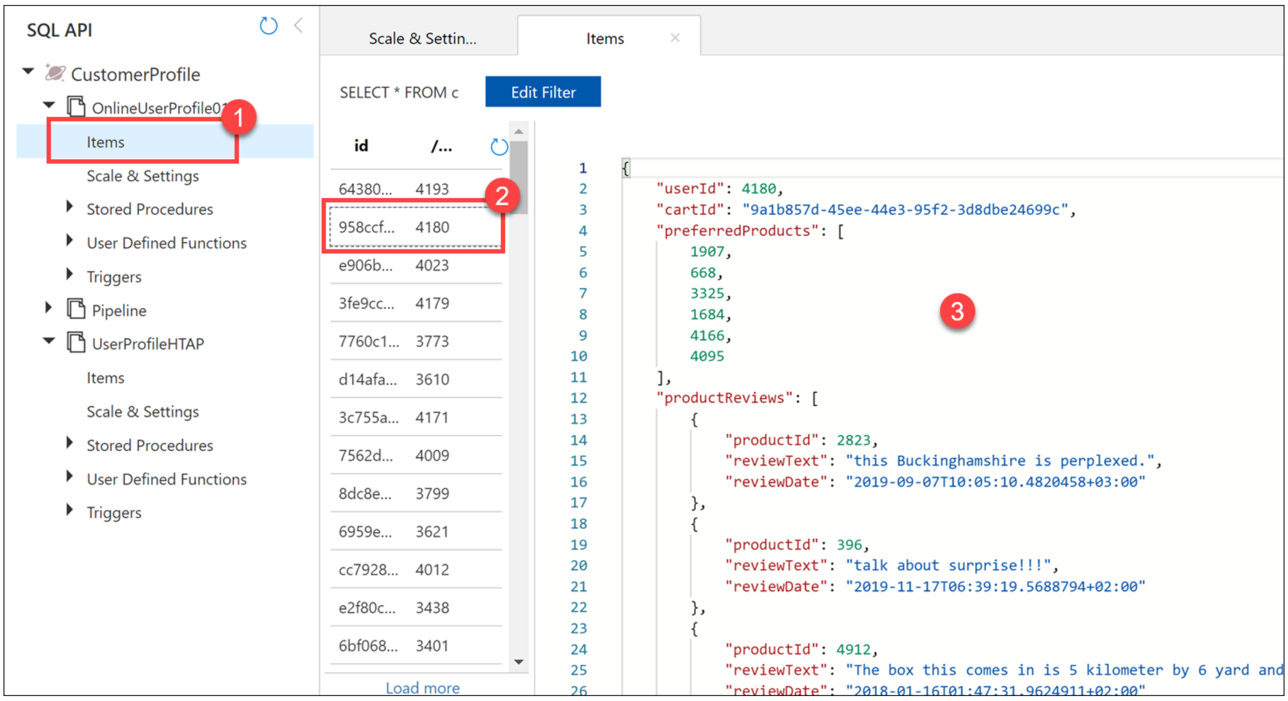
☐ My partition key is larger than 100 bytes

6
Analytical store
[✓ On] Off

Here we set the **partition key** value to **userId**, because it is a field we use most often in queries and contains a relatively high cardinality (number of unique values) for good partitioning performance. We set the throughput to Autoscale with a maximum value of 4,000 request units (RUs). This means that the container will have a minimum of 400 RUs allocated (10% of the maximum number), and will scale up to a maximum of 4,000 when the scale engine detects a high enough demand to warrant increasing the throughput. Finally, we enable the **analytical store** on the container, which allows us to take full advantage of the Hybrid Transactional/Analytical Processing (HTAP) architecture from within Synapse Analytics.

Let's take a quick look at the data we will copy over to the new container.

4. Expand the **OnlineUserProfile01** container underneath the **CustomerProfile** database, then select **Items**. Select one of the documents and view its contents. The documents are stored in JSON format.

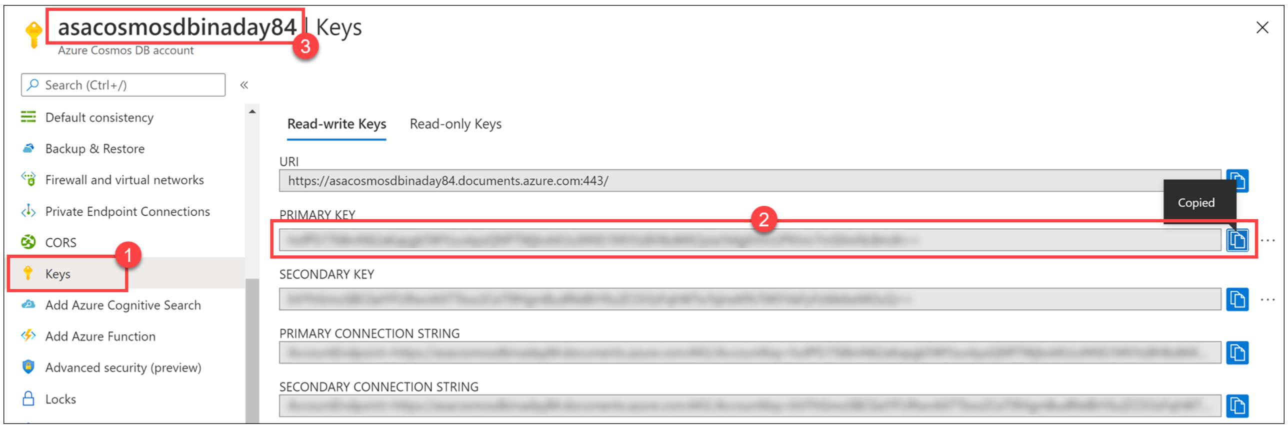


SELECT * FROM c

id	/...
64380...	4193
958ccf...	4180
e906b...	4023
3fe9cc...	4179
7760c1...	3773
d14afa...	3610
3c755a...	4171
7562d...	4009
8dc8e...	3799
6959e...	3621
cc7928...	4012
e2f80c...	3438
6bf068...	3401

```
{
    "userId": 4180,
    "cartId": "9a1b857d-45ee-44e3-95f2-3d8dbe24699c",
    "preferredProducts": [
        1907,
        668,
        3325,
        1684,
        4166,
        4095
    ],
    "productReviews": [
        {
            "productId": 2823,
            "reviewText": "this Buckinghamshire is perplexed.",
            "reviewDate": "2019-09-07T10:05:10.4820458+03:00"
        },
        {
            "productId": 396,
            "reviewText": "talk about surprise!!!",
            "reviewDate": "2019-11-17T06:39:19.5688794+02:00"
        },
        {
            "productId": 4912,
            "reviewText": "The box this comes in is 5 kilometer by 6 yard and
            "reviewDate": "2018-01-16T01:47:31.9624911+02:00"
```

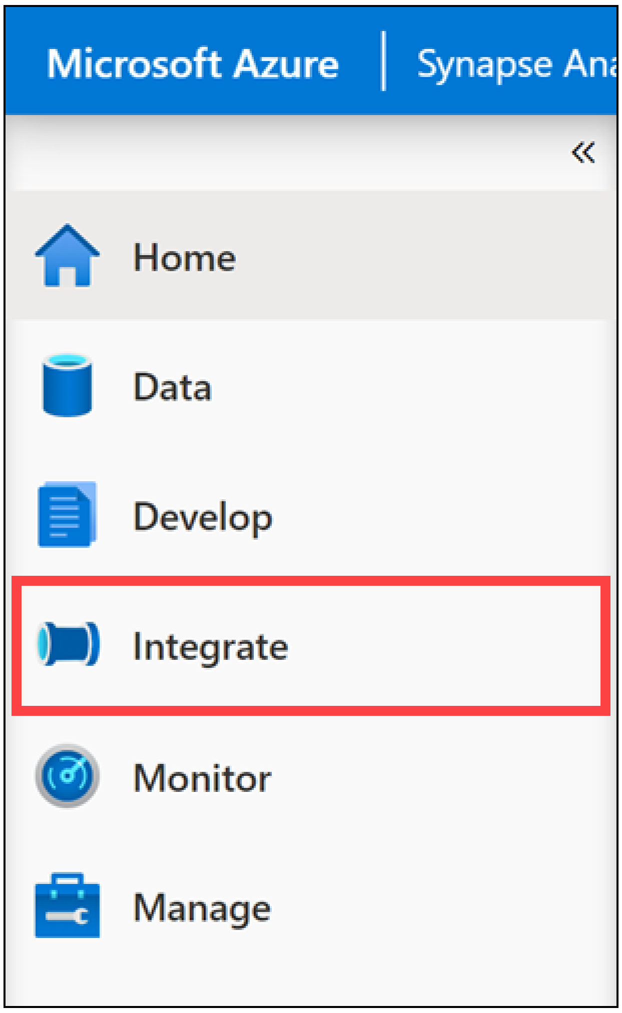
5. Select **Keys** in the left-hand menu. You will need the **Primary Key** and the Cosmos DB account name (in the upper-left corner) later, so keep this tab open.


asacosmosdbinaday84 | Keys
Azure Cosmos DB account
Search (Ctrl+/)
Default consistency
Backup & Restore
Firewall and virtual networks
Private Endpoint Connections
CORS
Keys
Add Azure Cognitive Search
Add Azure Function
Advanced security (preview)
Locks
Read-write Keys
Read-only Keys
URI
https://asacosmosdbinaday84.documents.azure.com:443/
PRIMARY KEY
Copied
SECONDARY KEY
PRIMARY CONNECTION STRING
SECONDARY CONNECTION STRING
1
2
3

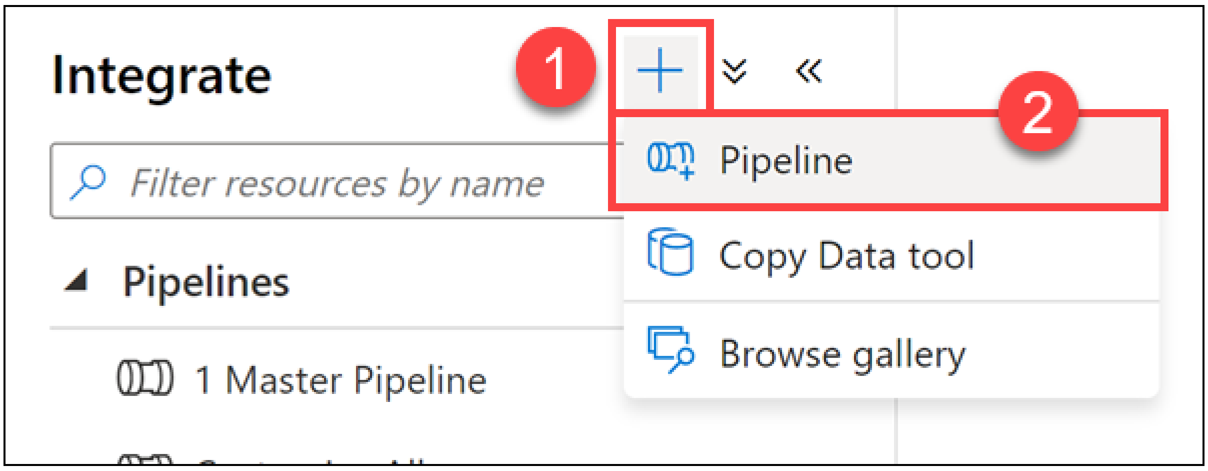
Task 3: Create and run a copy pipeline

Now that we have the new Azure Cosmos DB container with the analytical store enabled, we need to copy the contents of the existing container by using a Synapse Pipeline.

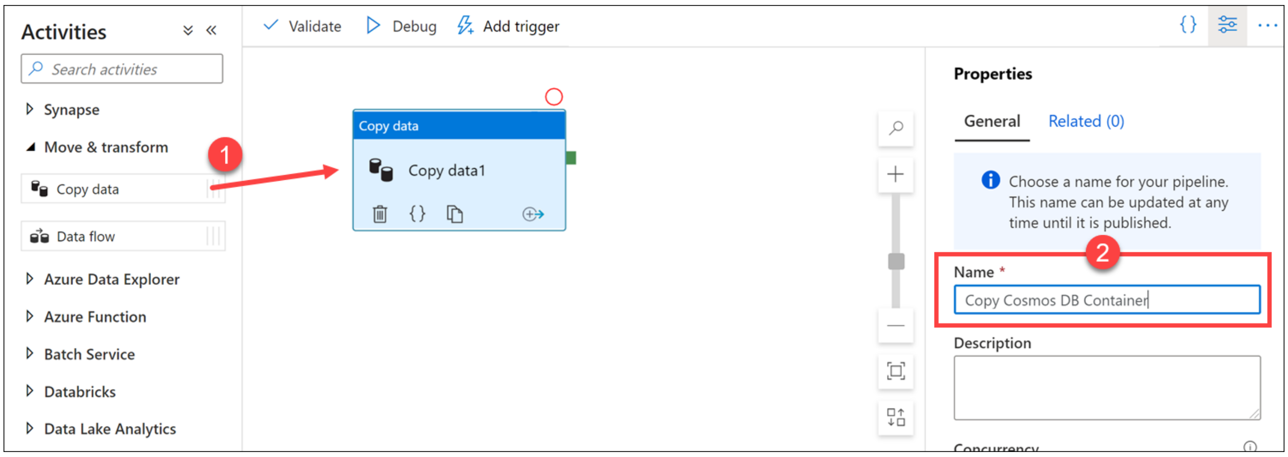
1. Open Synapse Studio (https://web.azuresynapse.net/) in a different tab, and then navigate to the **Integrate** hub.



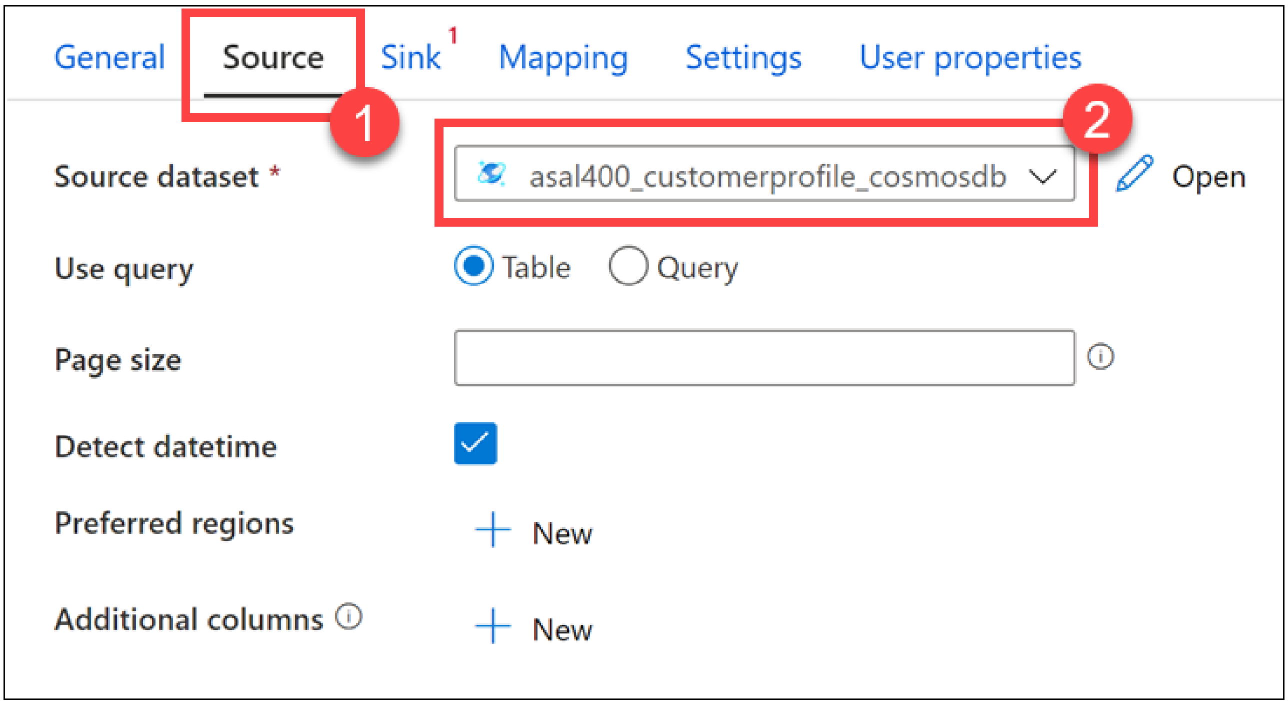
2. In the **+** menu, select **Pipeline**.



3. Under **Activities**, expand the **Move & transform** group, then drag the **Copy data** activity onto the canvas. Set the **Name** to `Copy Cosmos DB Container` in the **Properties** blade.

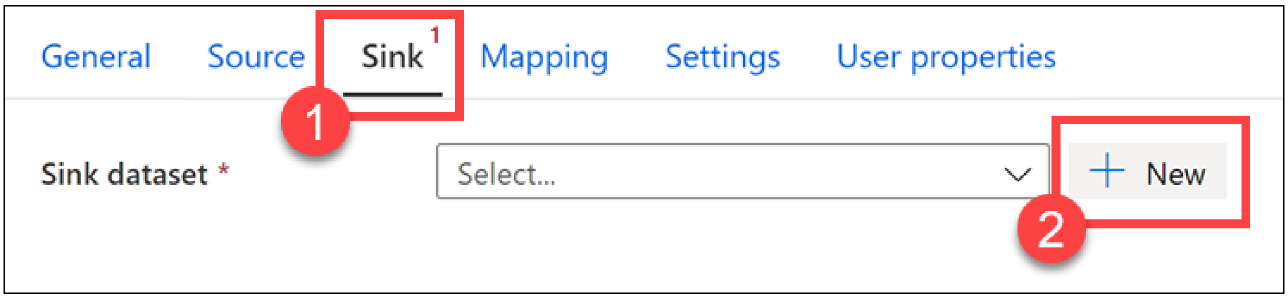


4. Select the new **Copy data** activity that you added to the canvas; and on the **Source** tab beneath the canvas, select the **asal400_customerprofile_cosmosdb** source dataset.

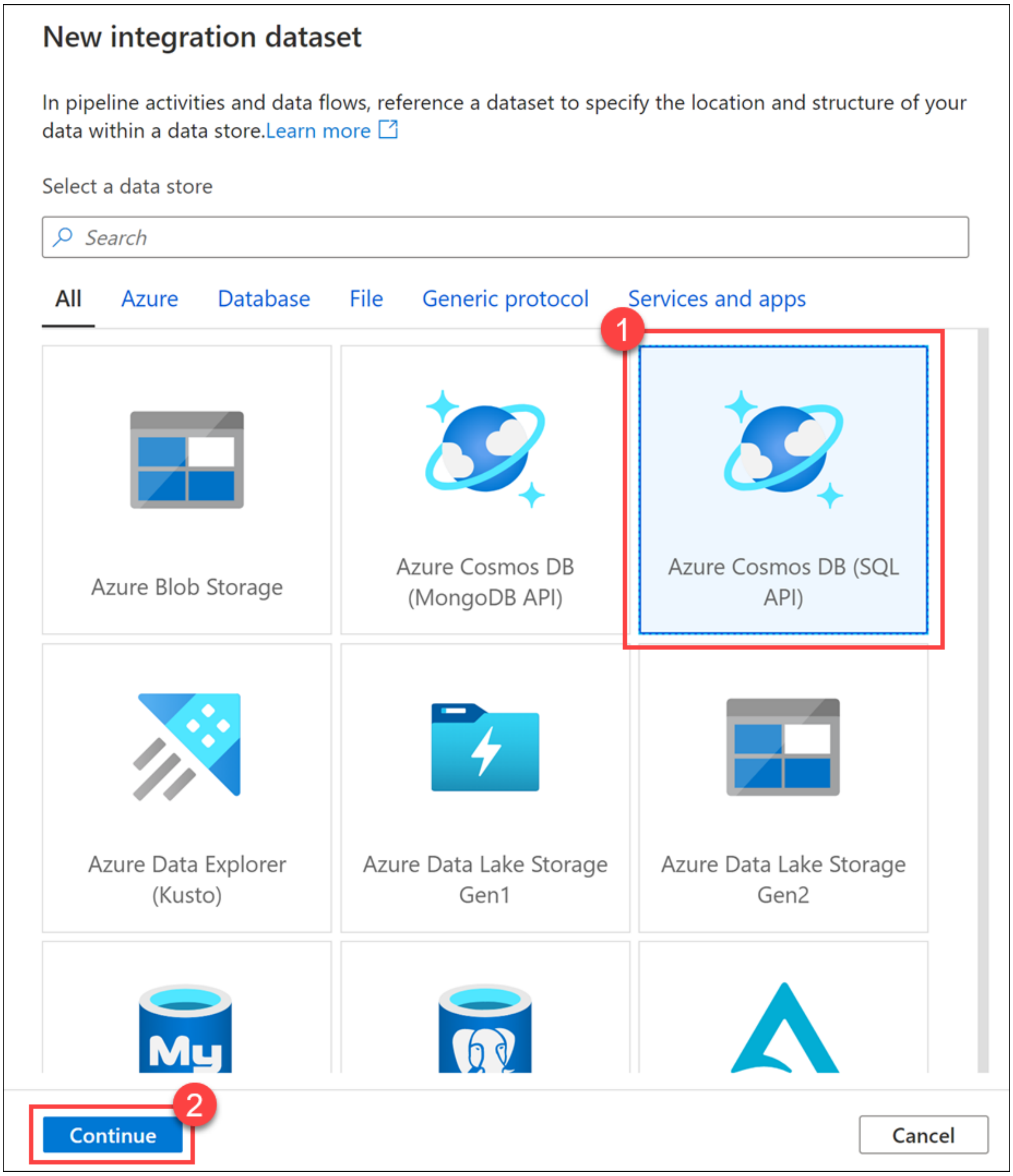


Source dataset * asal400_customerprofile_cosmosdb
Use query [✓ Table] Query
Page size
Detect datetime ✓
Preferred regions + New
Additional columns + New

5. Select the **Sink** tab, then select **+ New**.



6. Select the **Azure Cosmos DB (SQL API)** dataset type, then select **Continue**.


New integration dataset

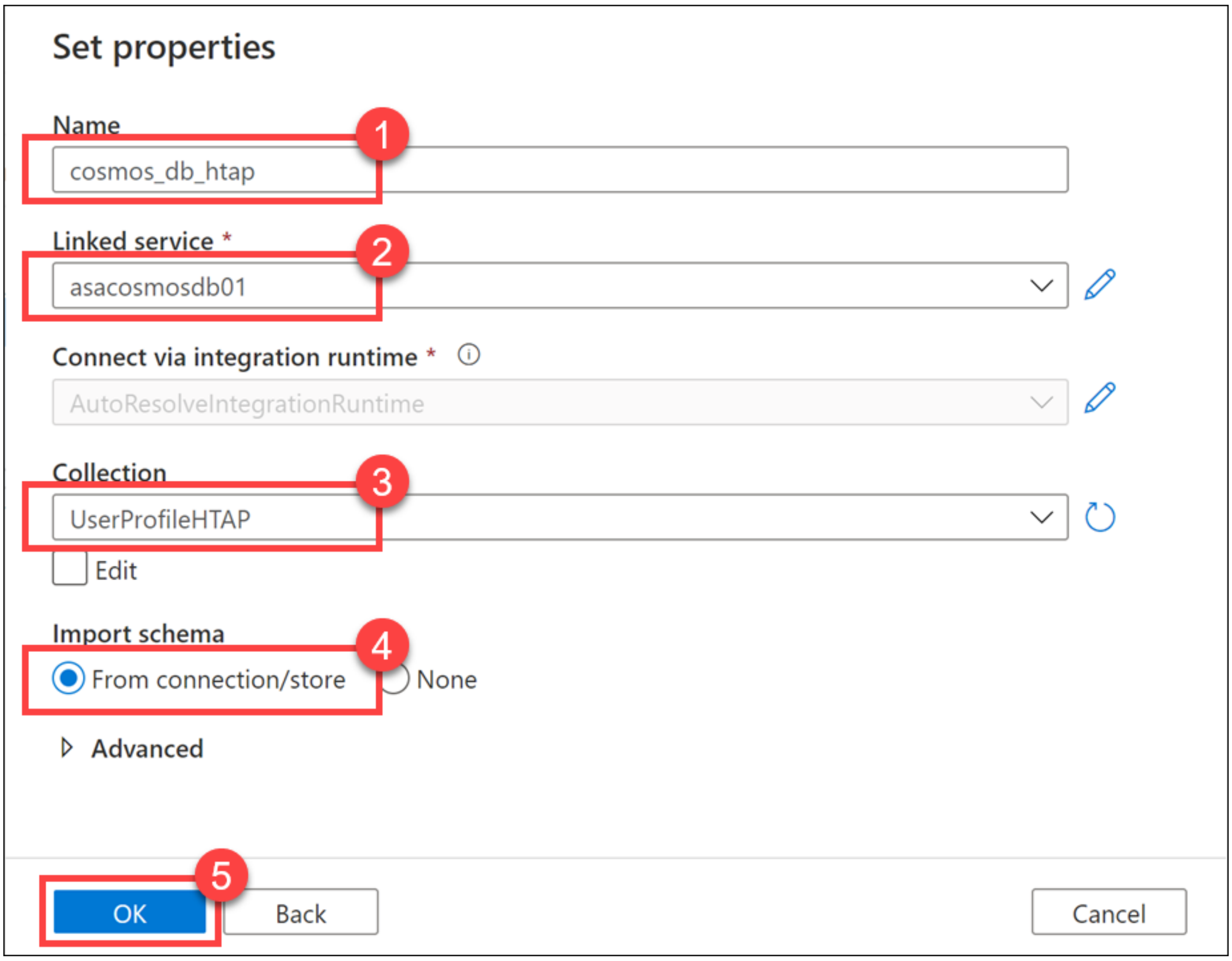
In pipeline activities and data flows, reference a dataset to specify the location and structure of your data within a data store.Learn more

Select a data store
Search
All
Azure
Database
File
Generic protocol
Services and apps
Azure Blob Storage
Azure Cosmos DB (MongoDB API)
Azure Cosmos DB (SQL API)
Azure Data Explorer (Kusto)
Azure Data Lake Storage Gen1
Azure Data Lake Storage Gen2
Continue
Cancel
1
2

7. Set the following properties, then click **OK**:

   - **Name**: Enter `cosmos_db_htap`.

- **Linked service**: Select **asacosmosdb01**.
- **Collection**: Select **UserProfileHTAP**/
- **Import schema**: Select **From connection/store** under **Import schema)**.



Set properties

Name: cosmos_db_htap

Linked service *: asacosmosdb01

Connect via integration runtime *: AutoResolveIntegrationRuntime

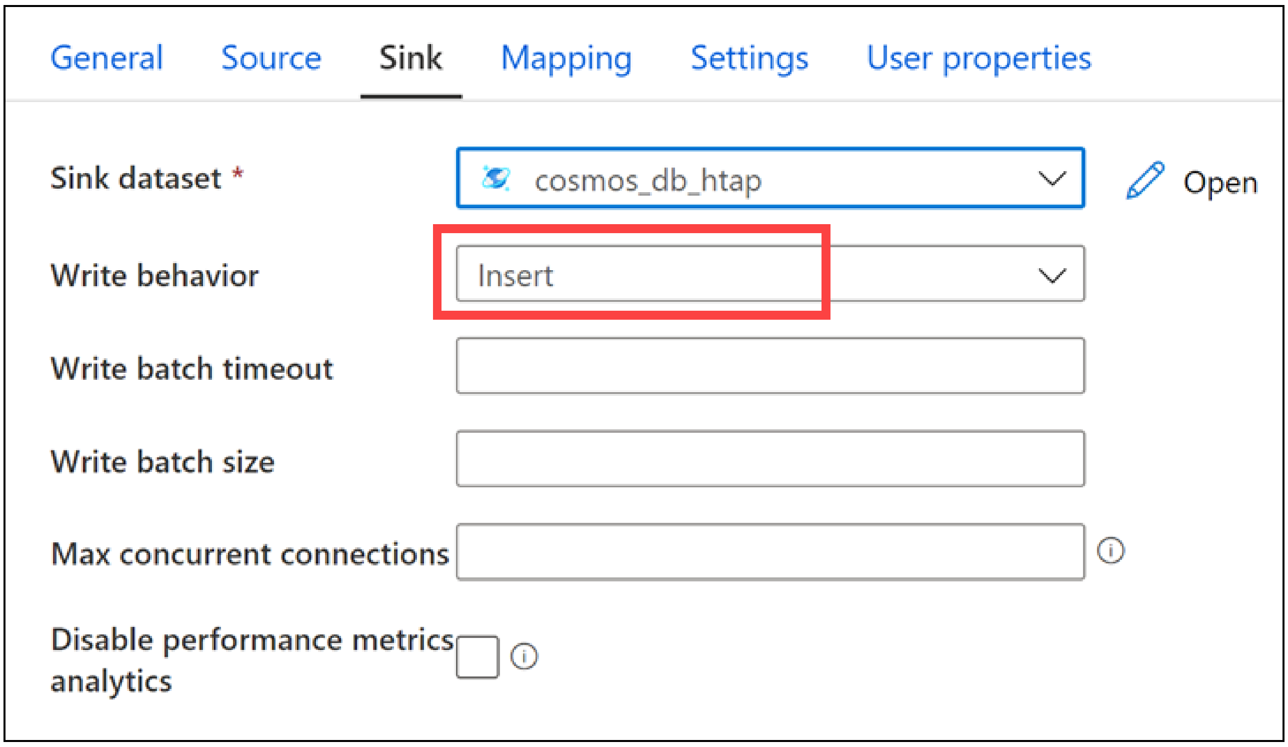
Collection: UserProfileHTAP

Edit

Import schema: [✓ From connection/store] None

Advanced

8. Underneath the new sink dataset you just added, ensure that the **Insert** write behavior is selected.



Sink dataset *: cosmos_db_htap

Write behavior: Insert

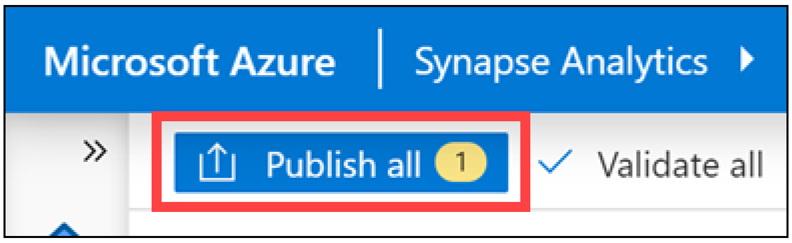
Write batch timeout

Write batch size

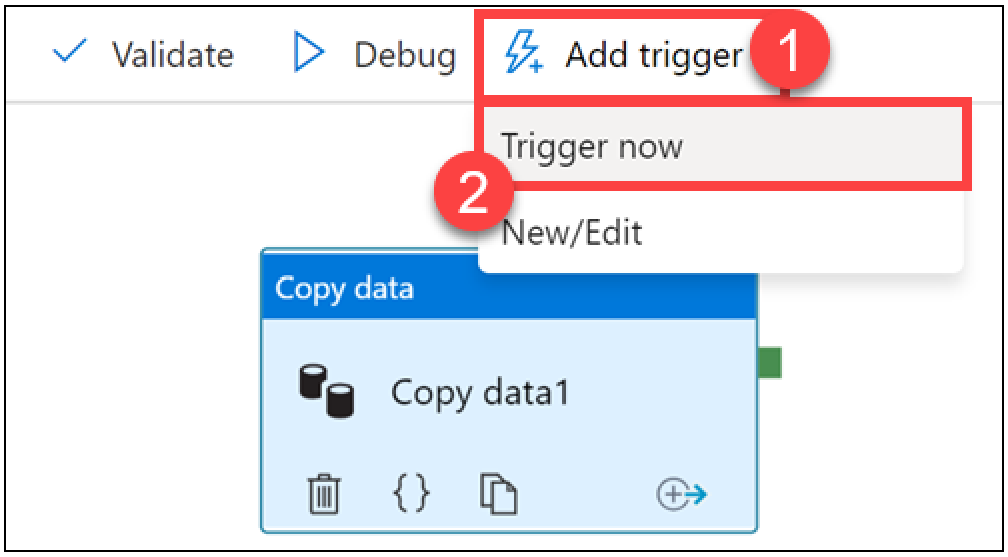
Max concurrent connections

Disable performance metrics analytics

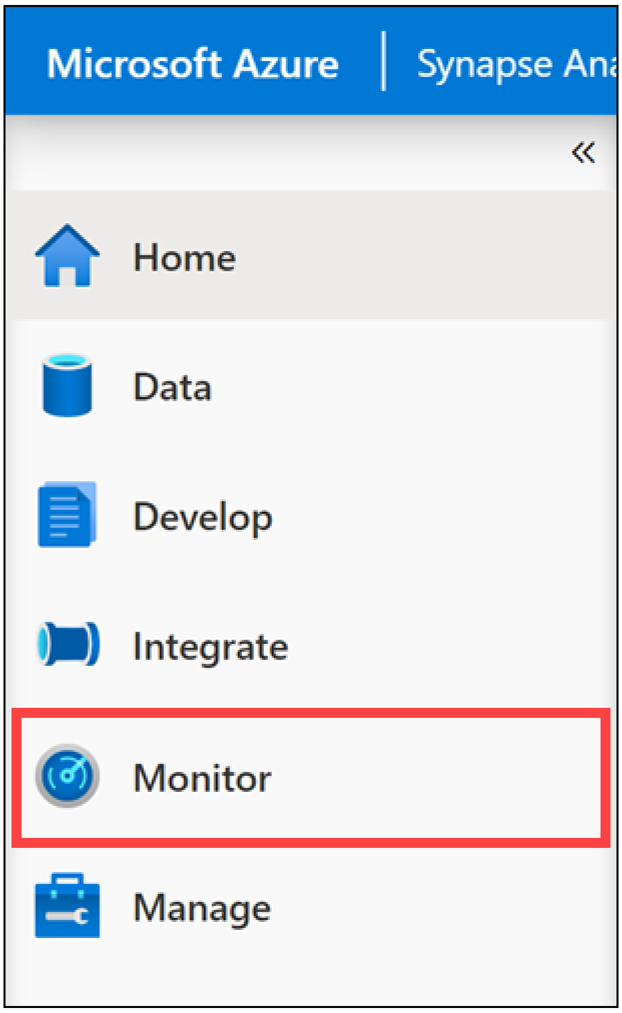
9. Select **Publish all**, then **Publish** to save the new pipeline.



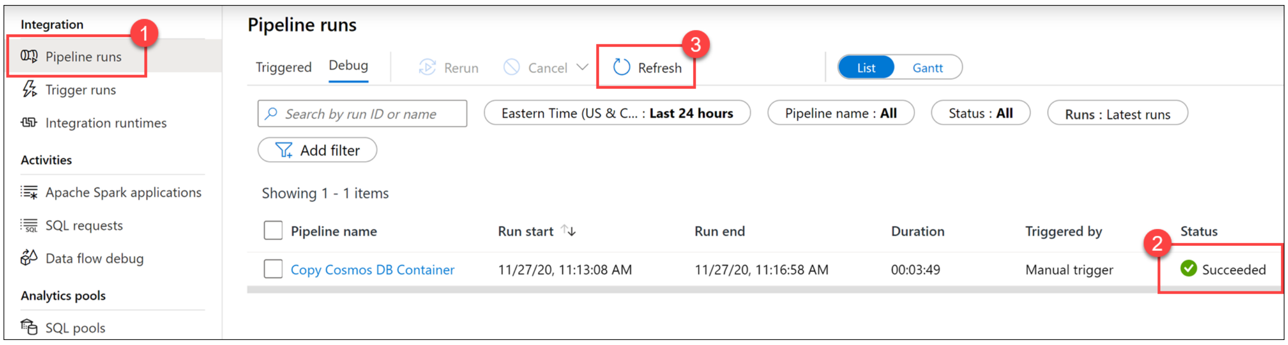
10. Above the pipeline canvas, select **Add trigger**, then **Trigger now**. Select **OK** to trigger the run.



11. Navigate to the **Monitor** hub.



12. Select **Pipeline runs** and wait until the pipeline run has successfully completed. You may have to select **Refresh** a few times.



Pipeline name	Run start	Run end	Duration	Triggered by	Status
Copy Cosmos DB Container	11/27/20, 11:13:08 AM	11/27/20, 11:16:58 AM	00:03:49	Manual trigger	Succeeded

> This may take **around 4 minutes** to complete.

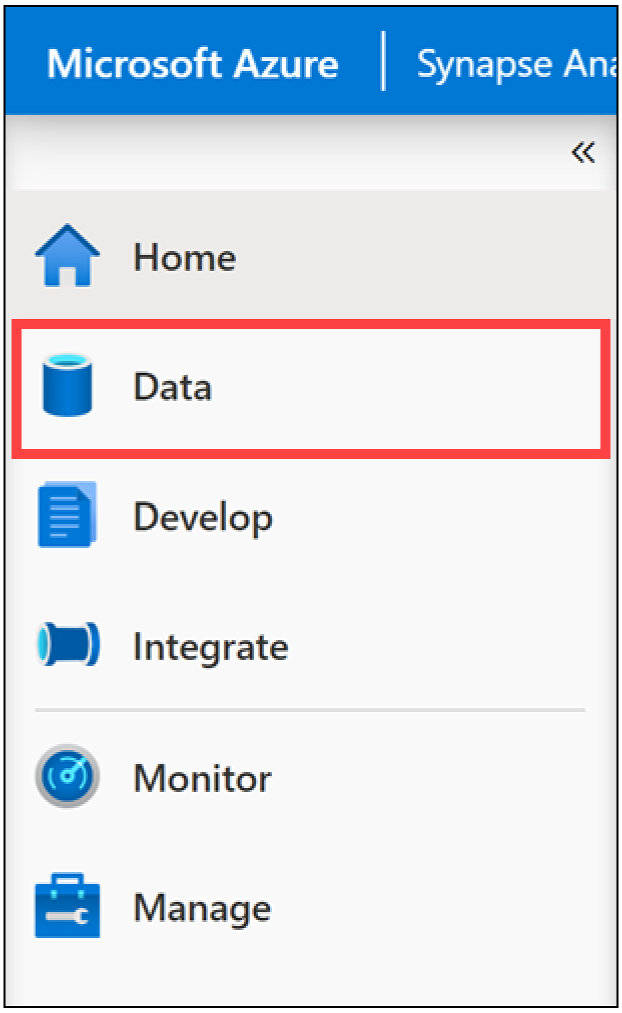
Exercise 2 - Querying Azure Cosmos DB with Apache Spark for Synapse Analytics

Tailwind Traders wants to use Apache Spark to run analytical queries against the new Azure Cosmos DB container. In this segment, we will use built-in gestures in Synapse Studio to quickly create a Synapse Notebook that loads data from the analytical store of the HTAP-enabled container, without impacting the transactional store.

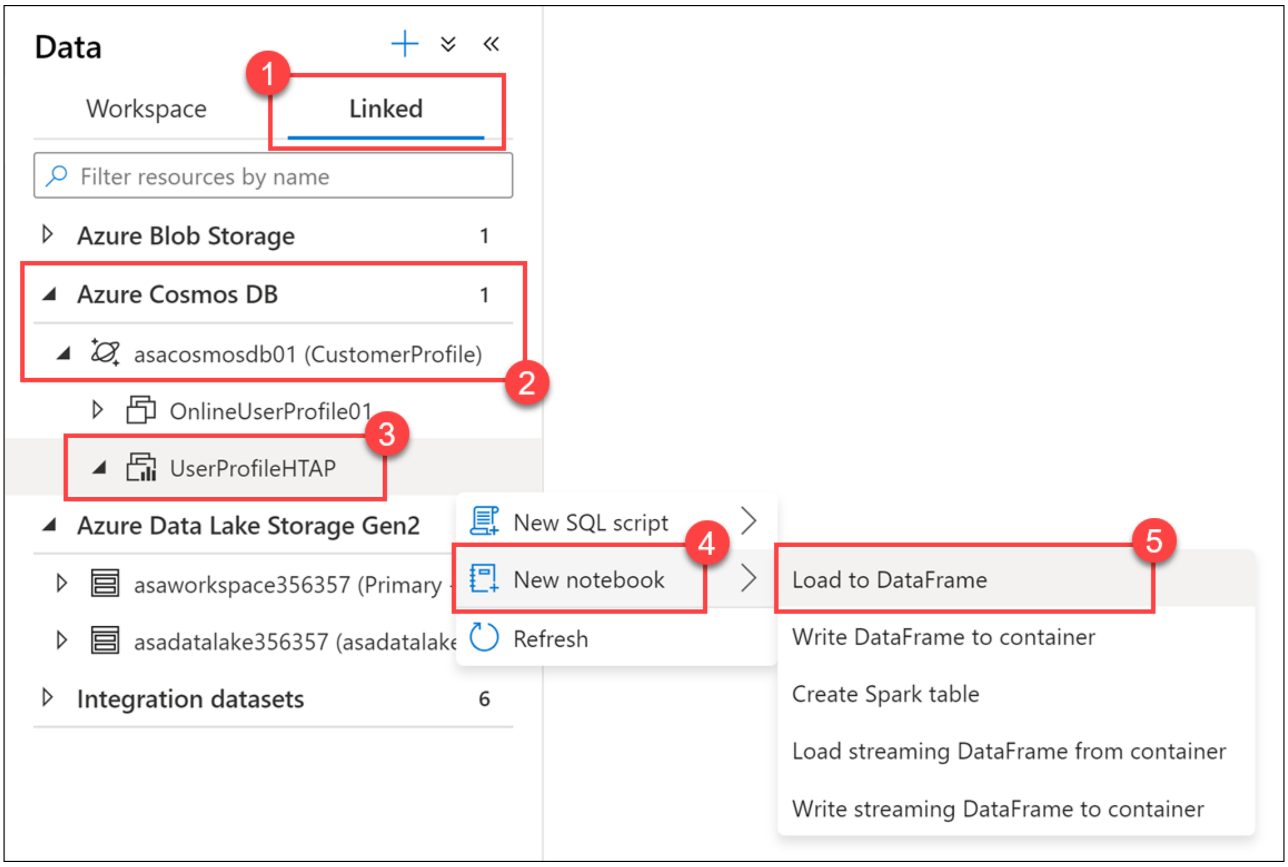
Tailwind Traders is trying to solve how they can use the list of preferred products identified with each user, coupled with any matching product IDs in their review history, to show a list of all preferred product reviews.

Task 1: Create a notebook

1. Navigate to the **Data** hub.

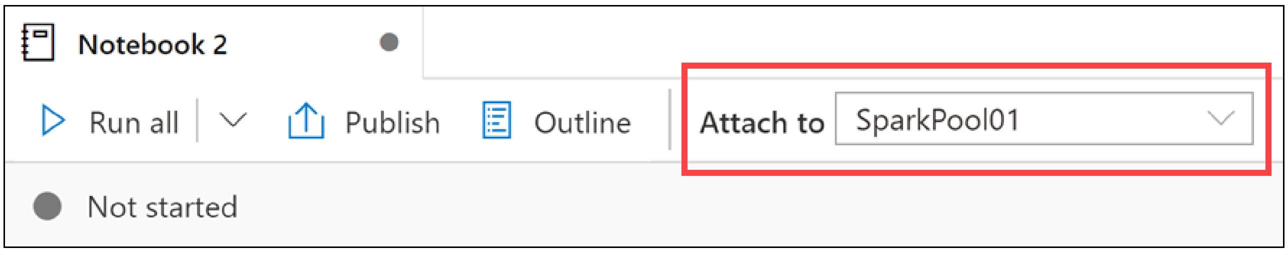


2. Select the **Linked** tab and expand the **Azure Cosmos DB** section (if this is not visible, use the ↻ button at the top right to refresh Synapse Studio), then expand the **asacosmosdb01 (CustomerProfile)** linked service. Right-click the **UserProfileHTAP** container, select **New notebook**, and then select **Load to DataFrame**.

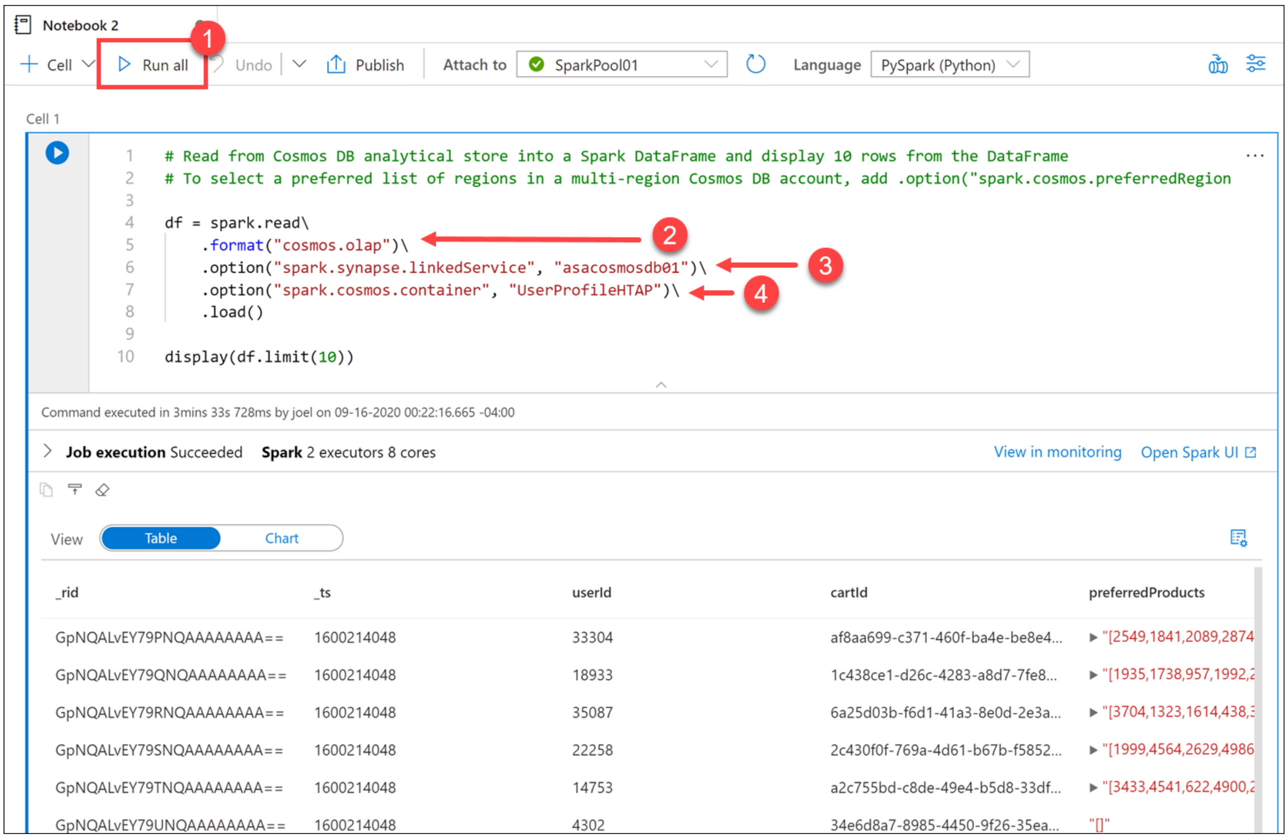

Data
Workspace
Linked
Filter resources by name
Azure Blob Storage 1
Azure Cosmos DB 1
asacosmosdb01 (CustomerProfile)
OnlineUserProfile01
UserProfileHTAP
Azure Data Lake Storage Gen2
asaworkspace356357 (Primary
asadatalake356357 (asadatalake
Integration datasets 6
New SQL script
New notebook
Refresh
Load to DataFrame
Write DataFrame to container
Create Spark table
Load streaming DataFrame from container
Write streaming DataFrame to container
1
2
3
4
5

Notice that the **UserProfileHTAP** container that we created has a slightly different icon than the other container. This indicates that the analytical store is enabled.

3. In the new notebook, select the **SparkPool01** Spark pool in the **Attach to** dropdown list.



4. Select **Run all**.



```
# Read from Cosmos DB analytical store into a Spark DataFrame and display 10 rows from the DataFrame
# To select a preferred list of regions in a multi-region Cosmos DB account, add .option("spark.cosmos.preferredRegion

df = spark.read\
    .format("cosmos.olap")\
    .option("spark.synapse.linkedService", "asacosmosdb01")\
    .option("spark.cosmos.container", "UserProfileHTAP")\
    .load()

display(df.limit(10))
```

Command executed in 3mins 33s 728ms by joel on 09-16-2020 00:22:16.665 -04:00

Job execution Succeeded **Spark** 2 executors 8 cores View in monitoring Open Spark UI

_rid	_ts	userId	cartId	preferredProducts
GpNQALvEY79PNQAAAAAAAA==	1600214048	33304	af8aa699-c371-460f-ba4e-be8e4...	▸ "[2549,1841,2089,2874
GpNQALvEY79QNQAAAAAAAA==	1600214048	18933	1c438ce1-d26c-4283-a8d7-7fe8...	▸ "[1935,1738,957,1992,2
GpNQALvEY79RNQAAAAAAAA==	1600214048	35087	6a25d03b-f6d1-41a3-8e0d-2e3a...	▸ "[3704,1323,1614,438,3
GpNQALvEY79SNQAAAAAAAA==	1600214048	22258	2c430f0f-769a-4d61-b67b-f5852...	▸ "[1999,4564,2629,4986
GpNQALvEY79TNQAAAAAAAA==	1600214048	14753	a2c755bd-c8de-49e4-b5d8-33df...	▸ "[3433,4541,622,4900,2
GpNQALvEY79UNQAAAAAAAA==	1600214048	4302	34e6d8a7-8985-4450-9f26-35ea...	"[]"

It will take a few minutes to start the Spark session the first time.

In the generated code within Cell 1, notice that the **spark.read** format is set to **cosmos.olap**. This instructs Synapse Link to use the container's analytical store. If we wanted to connect to the transactional store instead, like to read from the change feed or write to the container, we'd use **cosmos.oltp** instead.

> **Note:** You cannot write to the analytical store, only read from it. If you want to load data into the container, you need to connect to the transactional store.

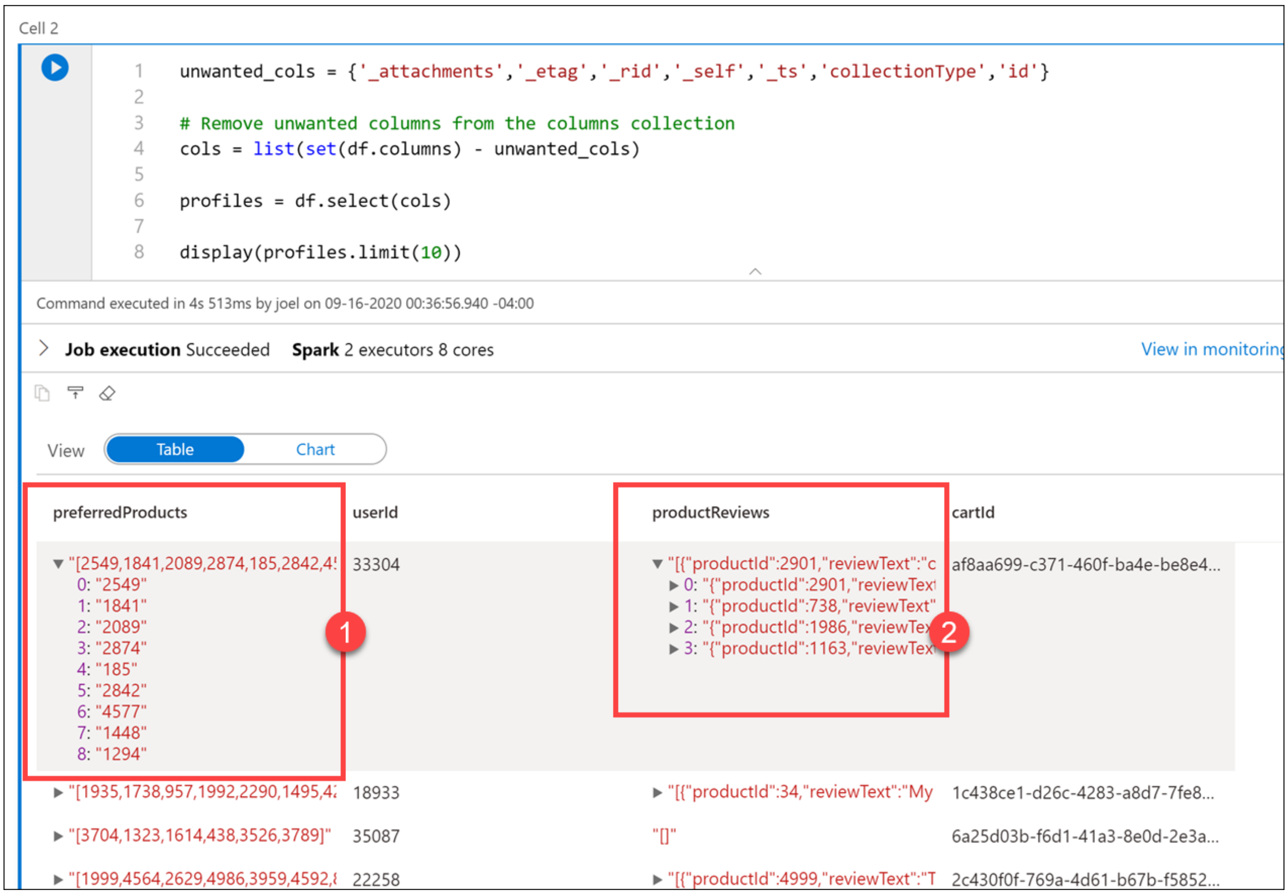
The first option configures the name of the Azure Cosmos DB linked service. The second `option` defines the Azure Cosmos DB container from which we want to read.

5. Select the **+ Code** button underneath the cell you ran. This adds a new code cell beneath the first one.

6. The DataFrame contains extra columns that we don't need. Let's remove the unwanted columns and create a clean version of the DataFrame. To do this, enter the following in the new code cell and run it:

```
unwanted_cols =
{'_attachments','_etag','_rid','_self','_ts','collectionType','id'}

# Remove unwanted columns from the columns collection
cols = list(set(df.columns) - unwanted_cols)

profiles = df.select(cols)

display(profiles.limit(10))
```

The output now only contains the columns that we want. Notice that the **preferredProducts** and **productReviews** columns contain child elements. Expand the values on a row to view them. You may recall seeing the raw JSON format in the **UserProfiles01** container within the Azure Cosmos DB Data Explorer.



Cell 2

```python
unwanted_cols = {'_attachments','_etag','_rid','_self','_ts','collectionType','id'}

# Remove unwanted columns from the columns collection
cols = list(set(df.columns) - unwanted_cols)

profiles = df.select(cols)

display(profiles.limit(10))
```

Command executed in 4s 513ms by joel on 09-16-2020 00:36:56.940 -04:00

Job execution Succeeded **Spark** 2 executors 8 cores View in monitoring

View Table Chart

preferredProducts	userId	productReviews	cartId
▼ "[2549,1841,2089,2874,185,2842,4! 0: "2549" 1: "1841" 2: "2089" 3: "2874" 4: "185" 5: "2842" 6: "4577" 7: "1448" 8: "1294"	33304	▼ "[{"productId":2901,"reviewText":"c ▶ 0: "{"productId":2901,"reviewText ▶ 1: "{"productId":738,"reviewText" ▶ 2: "{"productId":1986,"reviewTex ▶ 3: "{"productId":1163,"reviewTex	af8aa699-c371-460f-ba4e-be8e4...
▶ "[1935,1738,957,1992,2290,1495,4;	18933	▶ "[{"productId":34,"reviewText":"My	1c438ce1-d26c-4283-a8d7-7fe8...
▶ "[3704,1323,1614,438,3526,3789]"	35087	"[]"	6a25d03b-f6d1-41a3-8e0d-2e3a...
▶ "[1999,4564,2629,4986,3959,4592,{	22258	▶ "[{"productId":4999,"reviewText":"T	2c430f0f-769a-4d61-b67b-f5852...

7. We should know how many records we're dealing with. To do this, enter the following in a new code cell and run it:

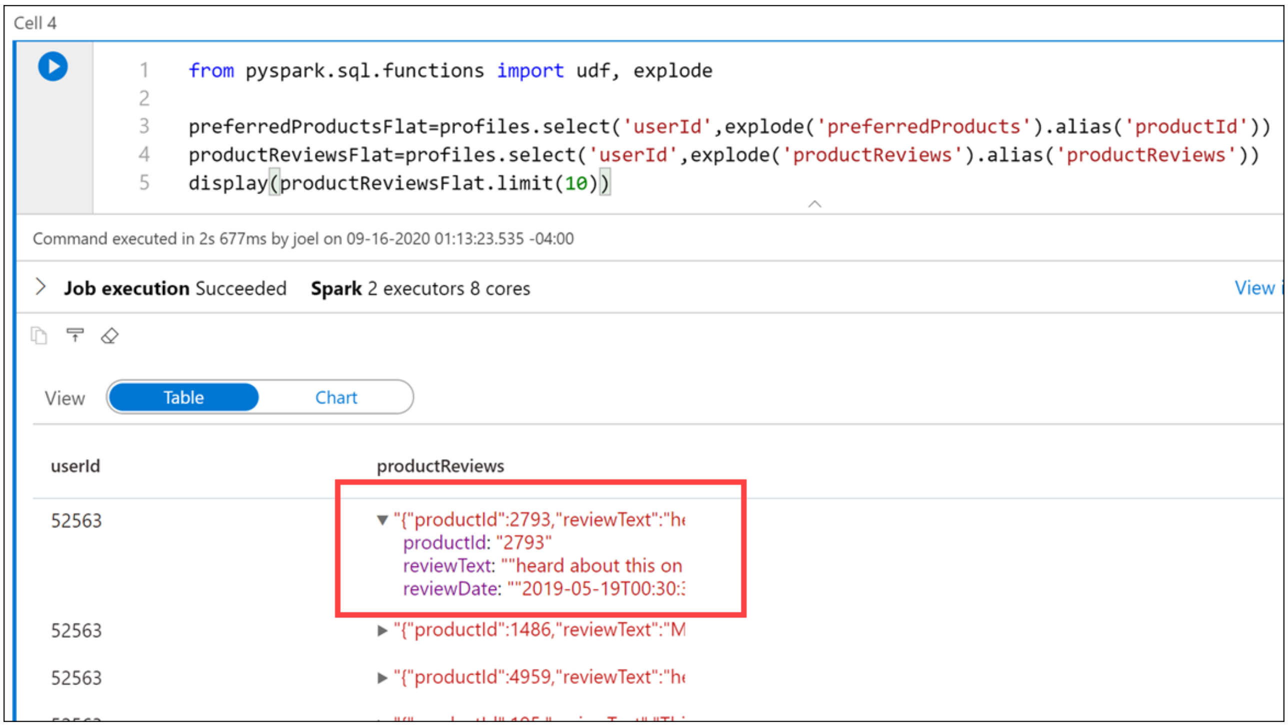
```python
profiles.count()
```

You should see a count result of 99,999.

8. We want to use the **preferredProducts** column array and **productReviews** column array for each user and create a graph of products that are from their preferred list that match with products that they have reviewed. To do this, we need to create two new DataFrames that contain flattened values from those two columns so we can join them in a later step. Enter the following in a new code cell and run it:

```python
from pyspark.sql.functions import udf, explode

preferredProductsFlat=profiles.select('userId',explode('preferredProducts').
alias('productId'))
productReviewsFlat=profiles.select('userId',explode('productReviews').alias(
'productReviews'))
display(productReviewsFlat.limit(10))
```

In this cell, we imported the special PySpark explode function, which returns a new row for each element of the array. This function helps flatten the **preferredProducts** and **productReviews** columns for better readability or for easier querying.


Cell 4

```
from pyspark.sql.functions import udf, explode

preferredProductsFlat=profiles.select('userId',explode('preferredProducts').alias('productId'))
productReviewsFlat=profiles.select('userId',explode('productReviews').alias('productReviews'))
display(productReviewsFlat.limit(10))
```

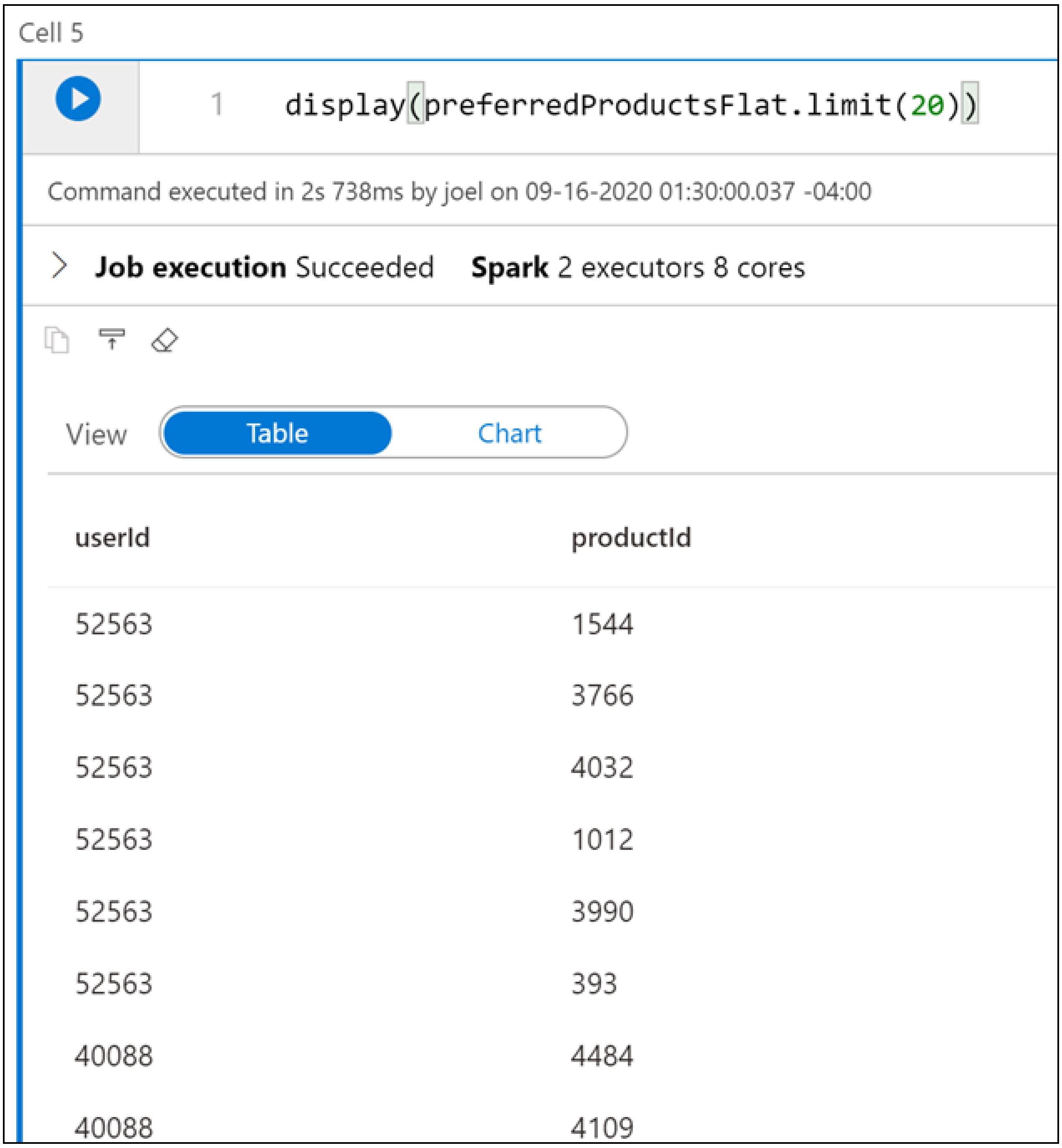
```
Command executed in 2s 677ms by joel on 09-16-2020 01:13:23.535 -04:00
```

userId	productReviews
52563	▼ "{"productId":2793,"reviewText":"he productId: "2793" reviewText: ""heard about this on reviewDate: ""2019-05-19T00:30:3
52563	► "{"productId":1486,"reviewText":"M
52563	► "{"productId":4959,"reviewText":"he

Observe the cell output where we display the **productReviewFlat** DataFrame contents. We see a new **productReviews** column that contains the **productId** we want to match to the preferred products list for the user, as well as the **reviewText** that we want to display or save.

9. Let's look at the **preferredProductsFlat** DataFrame contents. To do this, enter the following in a new cell and **run** it:

```
display(preferredProductsFlat.limit(20))
```



Cell 5

```
display(preferredProductsFlat.limit(20))
```

Command executed in 2s 738ms by joel on 09-16-2020 01:30:00.037 -04:00

Job execution Succeeded Spark 2 executors 8 cores

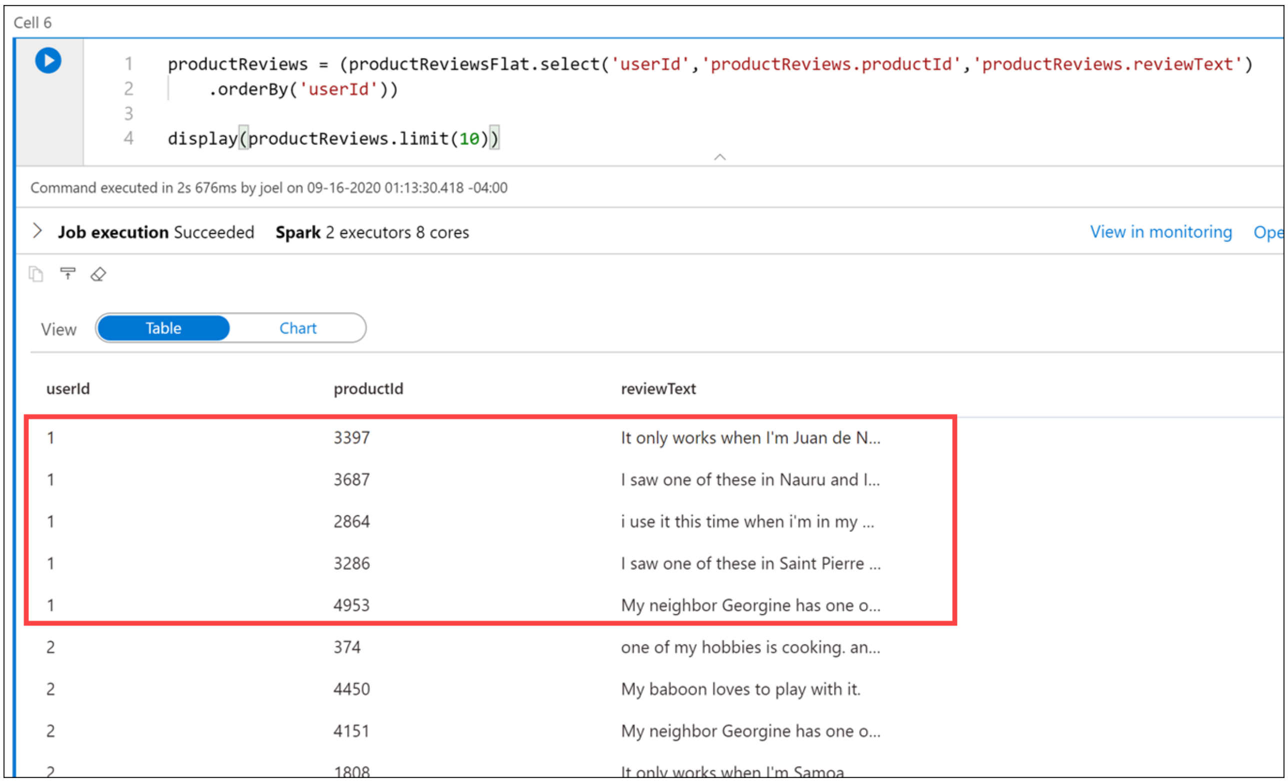
userId	productId
52563	1544
52563	3766
52563	4032
52563	1012
52563	3990
52563	393
40088	4484
40088	4109

Since we used the **explode** function on the preferred products array, we have flattened the column values to **userId** and **productId** rows, ordered by user.

10. Now we need to further flatten the **productReviewFlat** DataFrame contents to extract the **productReviews.productId** and **productReviews.reviewText** fields and create new rows for each data combination. To do this, enter the following in a new code cell and run it:

```
productReviews =
(productReviewsFlat.select('userId','productReviews.productId','productRevie
ws.reviewText')
    .orderBy('userId'))

display(productReviews.limit(10))
```

In the output, notice that we now have multiple rows for each `userId`.



```
productReviews = (productReviewsFlat.select('userId','productReviews.productId','productReviews.reviewText')
    .orderBy('userId'))

display(productReviews.limit(10))
```

Command executed in 2s 676ms by joel on 09-16-2020 01:13:30.418 -04:00

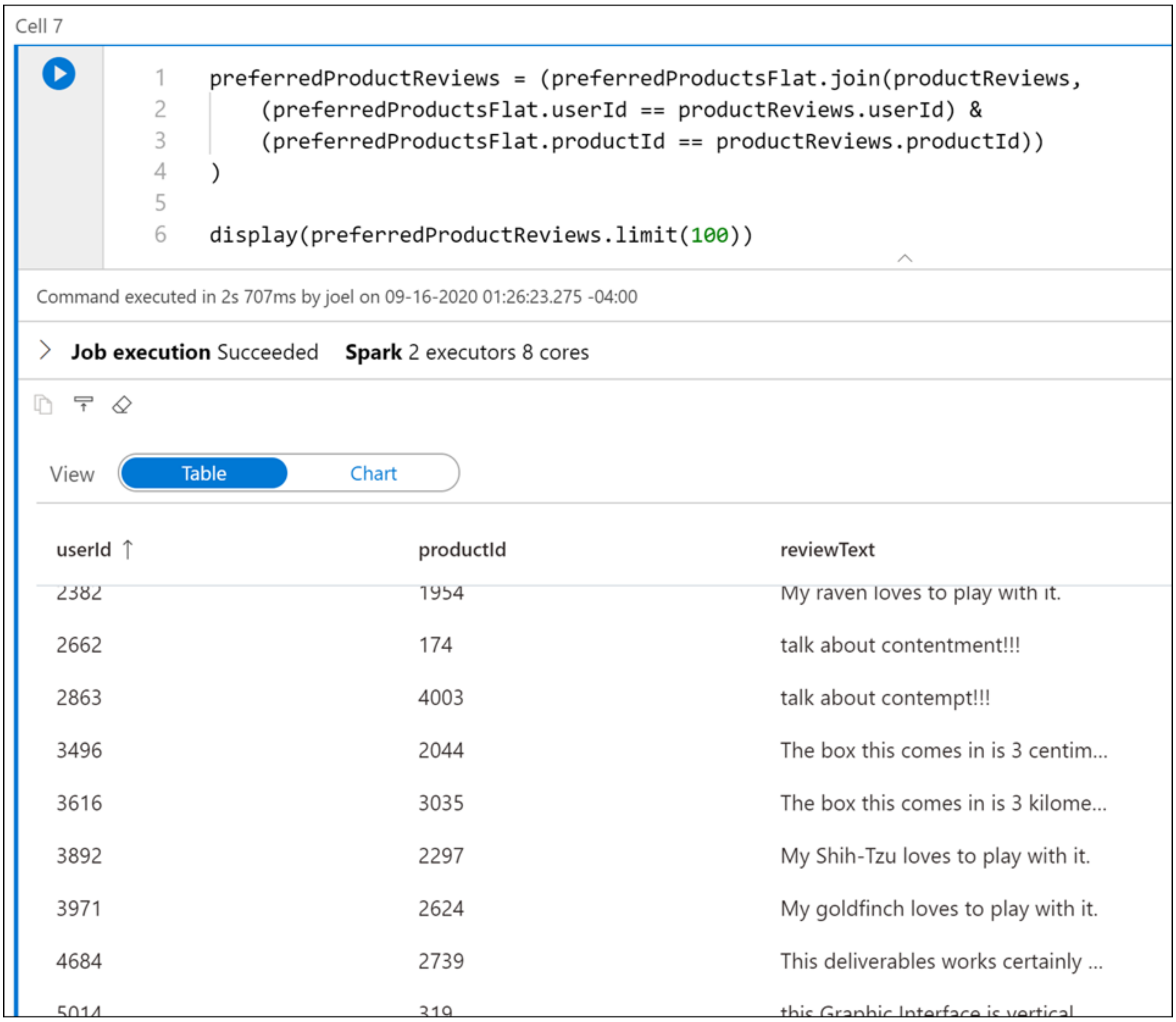
Job execution Succeeded Spark 2 executors 8 cores

userId	productId	reviewText
1	3397	It only works when I'm Juan de N...
1	3687	I saw one of these in Nauru and I...
1	2864	i use it this time when i'm in my ...
1	3286	I saw one of these in Saint Pierre ...
1	4953	My neighbor Georgine has one o...
2	374	one of my hobbies is cooking. an...
2	4450	My baboon loves to play with it.
2	4151	My neighbor Georgine has one o...
2	1808	It only works when I'm Samoa

11. The final step is to join the **preferredProductsFlat** and **productReviews** DataFrames on the **userId** and **productId** values to build our graph of preferred product reviews. To do this, enter the following in a new code cell and run it:

```
preferredProductReviews = (preferredProductsFlat.join(productReviews,
    (preferredProductsFlat.userId == productReviews.userId) &
    (preferredProductsFlat.productId == productReviews.productId))
)

display(preferredProductReviews.limit(100))
```

> **Note**: You can click on the column headers in the Table view to sort the result set.


Cell 7

```
preferredProductReviews = (preferredProductsFlat.join(productReviews,
    (preferredProductsFlat.userId == productReviews.userId) &
    (preferredProductsFlat.productId == productReviews.productId))
)

display(preferredProductReviews.limit(100))
```

Command executed in 2s 707ms by joel on 09-16-2020 01:26:23.275 -04:00

Job execution Succeeded **Spark** 2 executors 8 cores

View Table Chart

userId	productId	reviewText
2382	1954	My raven loves to play with it.
2662	174	talk about contentment!!!
2863	4003	talk about contempt!!!
3496	2044	The box this comes in is 3 centim...
3616	3035	The box this comes in is 3 kilome...
3892	2297	My Shih-Tzu loves to play with it.
3971	2624	My goldfinch loves to play with it.
4684	2739	This deliverables works certainly ...
5014	319	this Graphic Interface is vertical

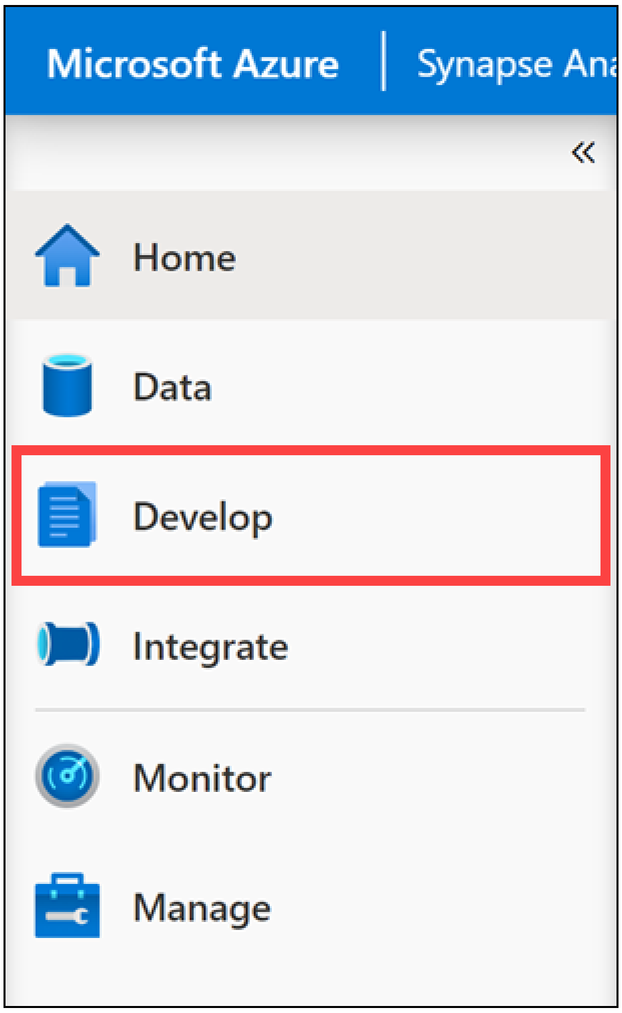
12. At the top right of the notebook, use the **Stop Session** button to stop the notebook session. Then close the notebook, discarding the changes.

Exercise 3 - Querying Azure Cosmos DB with serverless SQL pool for Azure Synapse Analytics

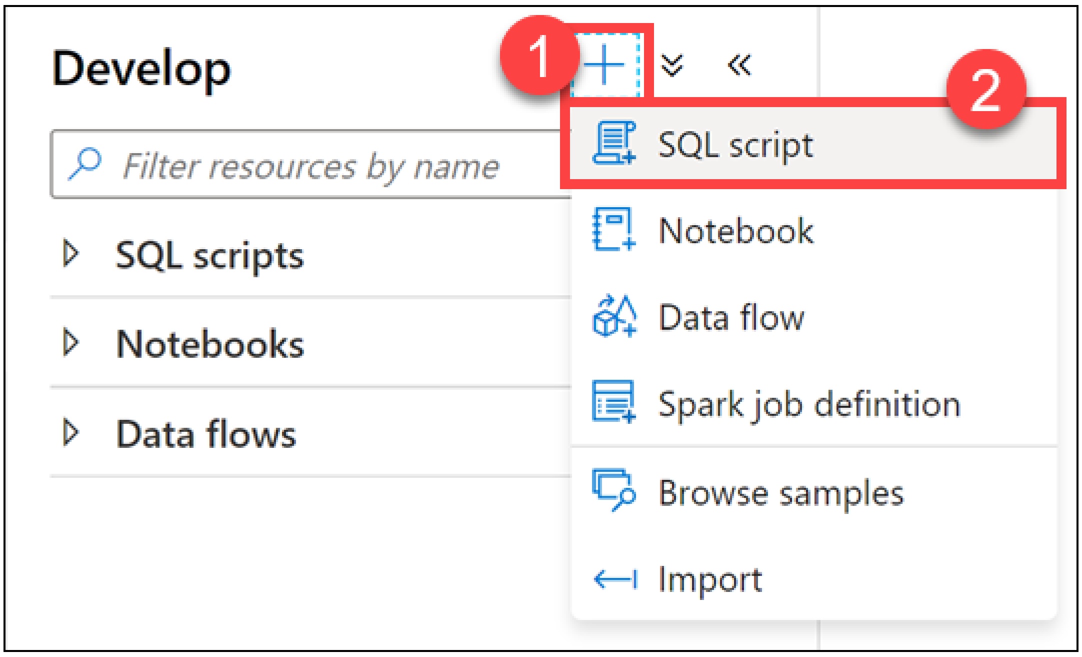
Tailwind Traders wants to explore the Azure Cosmos DB analytical store with T-SQL. Ideally, they can create views that can then be used for joins with other analytical store containers, files from the data lake, or accessed by external tools, like Power BI.

Task 1: Create a new SQL script

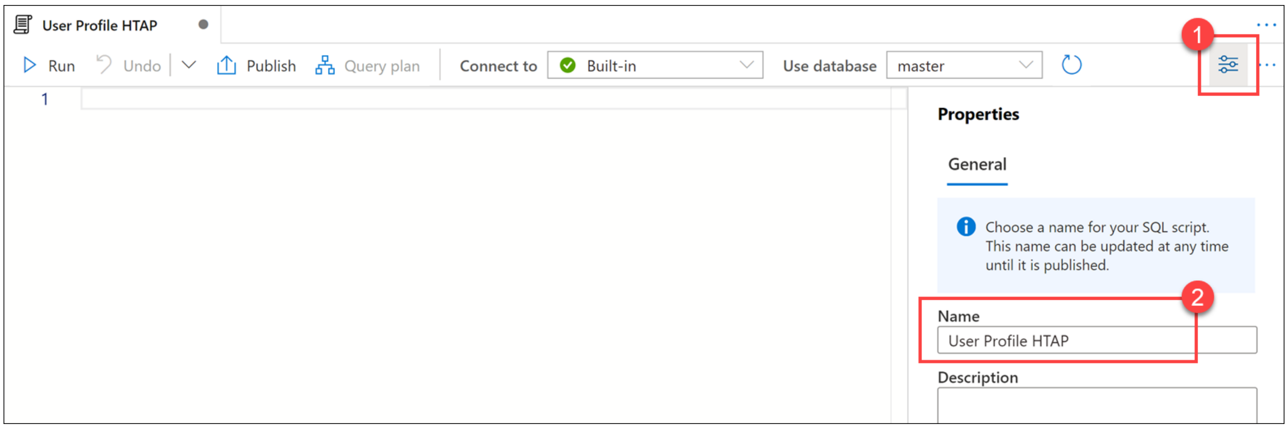
1. Navigate to the **Develop** hub.



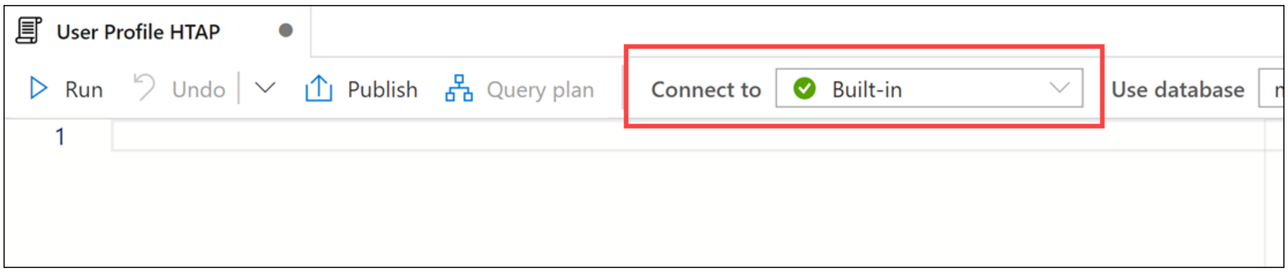
2. In the **+** menu, select **SQL script**.



3. When the script opens, in the **Properties** pane to the right, change the **Name** to `User Profile HTAP`. Then use the **Properties** button to close the pane.



4. Verify that the serverless SQL pool (**Built-in**) is selected.



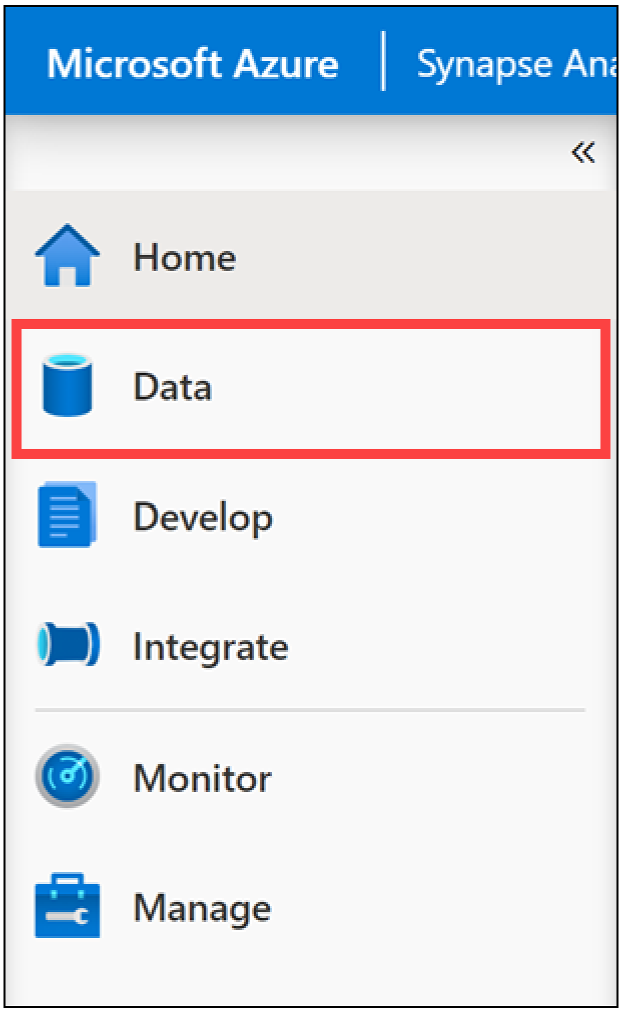
5. Paste the following SQL query. In the OPENROWSET statement, replace **YOUR_ACCOUNT_NAME** with the Azure Cosmos DB account name and **YOUR_ACCOUNT_KEY** with the Azure Cosmos DB Primary Key from the **Keys** page in the Azure portal (which should still be open in another tab).

```
USE master
GO

IF DB_ID (N'Profiles') IS NULL
BEGIN
    CREATE DATABASE Profiles;
END
GO

USE Profiles
GO

DROP VIEW IF EXISTS UserProfileHTAP;
GO

CREATE VIEW UserProfileHTAP
AS
SELECT
    *
FROM OPENROWSET(
    'CosmosDB',

N'account=YOUR_ACCOUNT_NAME;database=CustomerProfile;key=YOUR_ACCOUNT_KEY',
    UserProfileHTAP
)
WITH (
    userId bigint,
    cartId varchar(50),
    preferredProducts varchar(max),
    productReviews varchar(max)
) AS profiles
CROSS APPLY OPENJSON (productReviews)
WITH (
    productId bigint,
    reviewText varchar(1000)
) AS reviews
GO
```

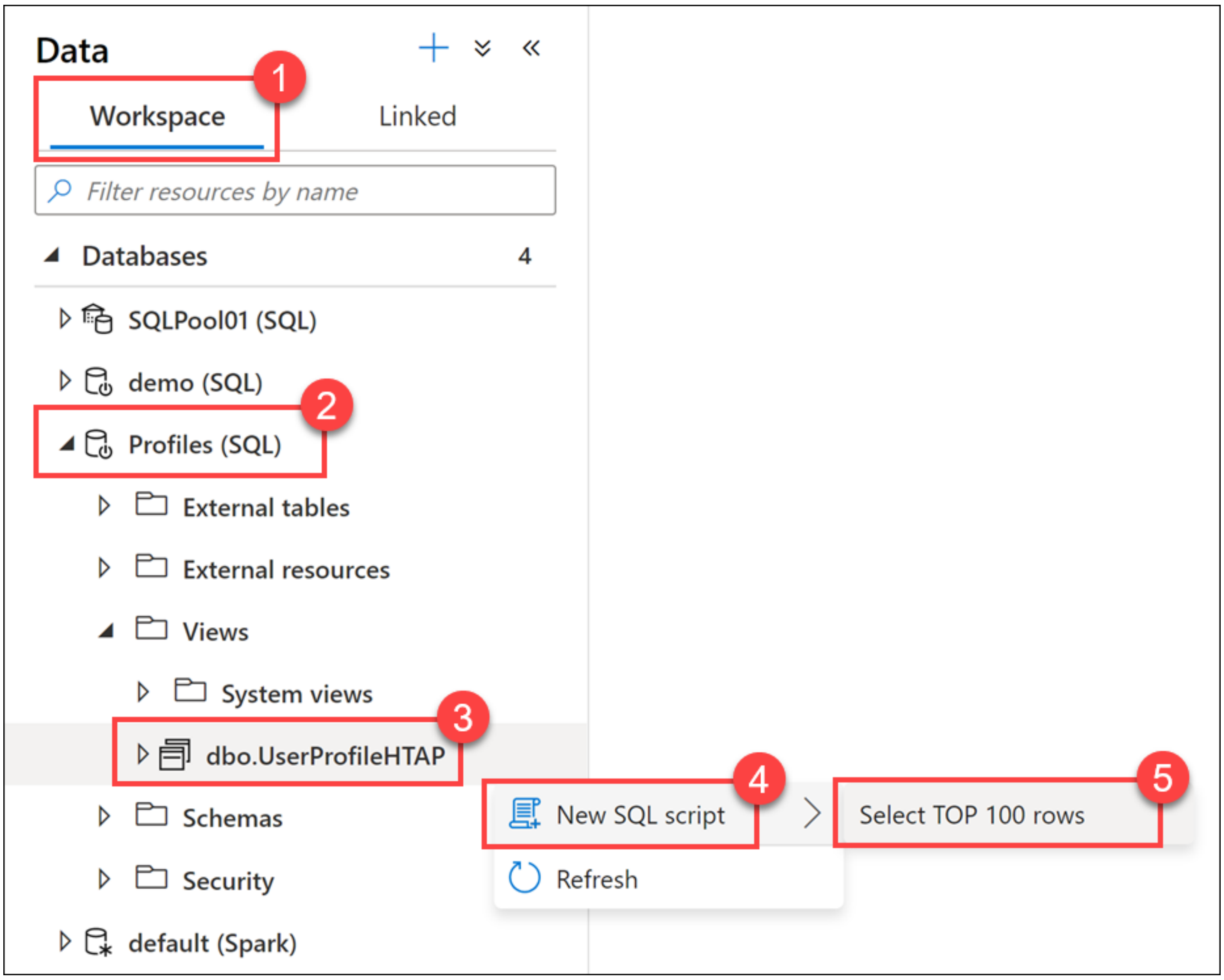
6. Use the **Run** button to run the query, which:

    - Creates a new serverless SQL pool database named **Profiles** if it does not exist
    - Changes the database context to the **Profiles** database.
    - Drops the **UserProfileHTAP** view if it exists.
    - Creates a SQL view named **UserProfileHTAP**.
    - Uses the OPENROWSET statement to set the data source type to **CosmosDB**, sets the account details, and specifies that we want to create the view over the Azure Cosmos DB analytical store container named **UserProfileHTAP**.
    - Matches the property names in the JSON documents and applies the appropriate SQL data types. Notice that we set the **preferredProducts** and **productReviews** fields to **varchar(max)**. This is because both of these properties contain JSON-formatted data within.
    - Since the **productReviews** property in the JSON documents contain nested subarrays, the script needs to "join" the properties from the document with all elements of the array. Synapse SQL enables us to flatten the nested structure by applying the OPENJSON function on the nested array. We flatten the values within **productReviews** like we did using the Python **explode** function earlier in the Synapse Notebook.

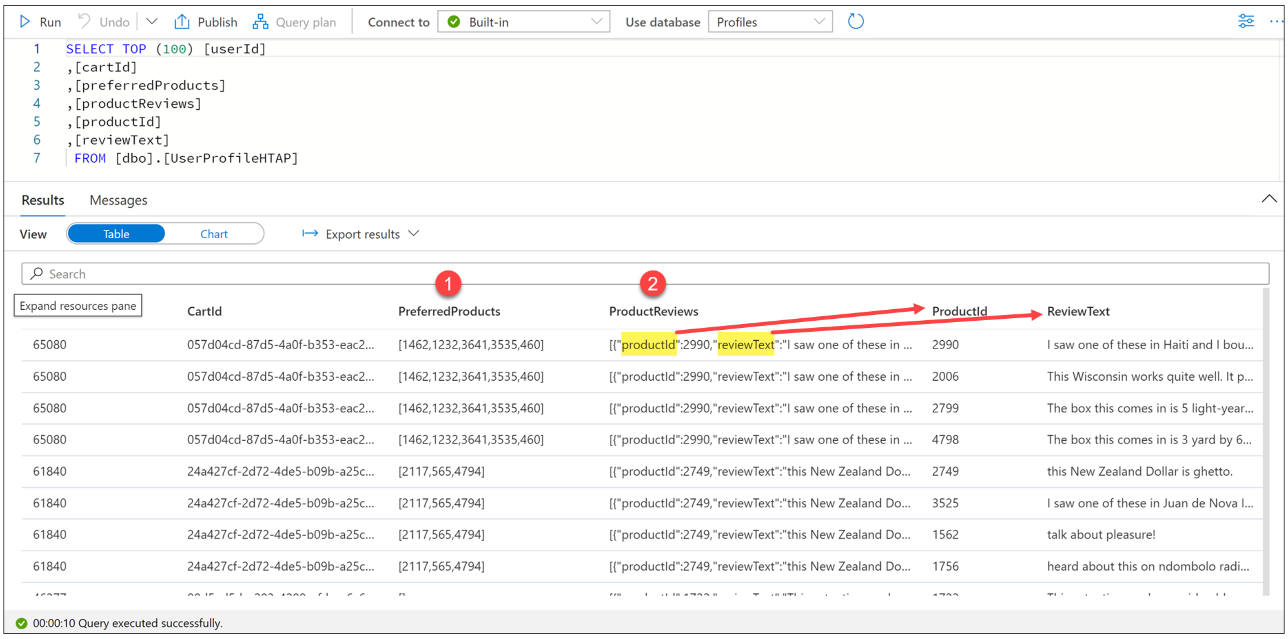
7. Navigate to the **Data** hub.



8. Select the **Workspace** tab and expand the **SQL database** group. Expand the **Profiles** SQL on-demand database (if you do not see this on the list, refresh the **Databases** list). Expand **Views**, then right-click the **UserProfileHTA** view, select **New SQL script**, and then **Select TOP 100 rows**.



9. Ensure the script is connected to the **Built-in** SQL pool, then run the query and view the results.



```
SELECT TOP (100) [userId]
,[cartId]
,[preferredProducts]
,[productReviews]
,[productId]
,[reviewText]
 FROM [dbo].[UserProfileHTAP]
```

	CartId	PreferredProducts	ProductReviews	ProductId	ReviewText
65080	057d04cd-87d5-4a0f-b353-eac2...	[1462,1232,3641,3535,460]	[{"productId":2990,"reviewText":"I saw one of these in ...	2990	I saw one of these in Haiti and I bou...
65080	057d04cd-87d5-4a0f-b353-eac2...	[1462,1232,3641,3535,460]	[{"productId":2990,"reviewText":"I saw one of these in ...	2006	This Wisconsin works quite well. It p...
65080	057d04cd-87d5-4a0f-b353-eac2...	[1462,1232,3641,3535,460]	[{"productId":2990,"reviewText":"I saw one of these in ...	2799	The box this comes in is 5 light-year...
65080	057d04cd-87d5-4a0f-b353-eac2...	[1462,1232,3641,3535,460]	[{"productId":2990,"reviewText":"I saw one of these in ...	4798	The box this comes in is 3 yard by 6...
61840	24a427cf-2d72-4de5-b09b-a25c...	[2117,565,4794]	[{"productId":2749,"reviewText":"this New Zealand Do...	2749	this New Zealand Dollar is ghetto.
61840	24a427cf-2d72-4de5-b09b-a25c...	[2117,565,4794]	[{"productId":2749,"reviewText":"this New Zealand Do...	3525	I saw one of these in Juan de Nova I...
61840	24a427cf-2d72-4de5-b09b-a25c...	[2117,565,4794]	[{"productId":2749,"reviewText":"this New Zealand Do...	1562	talk about pleasure!
61840	24a427cf-2d72-4de5-b09b-a25c...	[2117,565,4794]	[{"productId":2749,"reviewText":"this New Zealand Do...	1756	heard about this on ndombolo radi...

00:00:10 Query executed successfully.

The **preferredProducts** and **productReviews** fields are included in the view, which both contain JSON-formatted values. Notice how the CROSS APPLY OPENJSON statement in the view successfully flattened the nested subarray values in the **productReviews** field by extracting the **productId** and **reviewText** values into new fields.