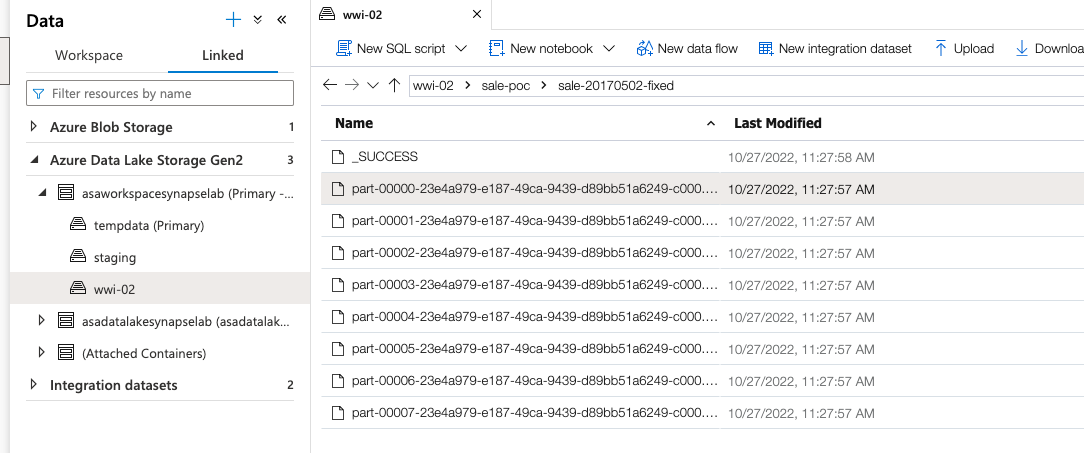
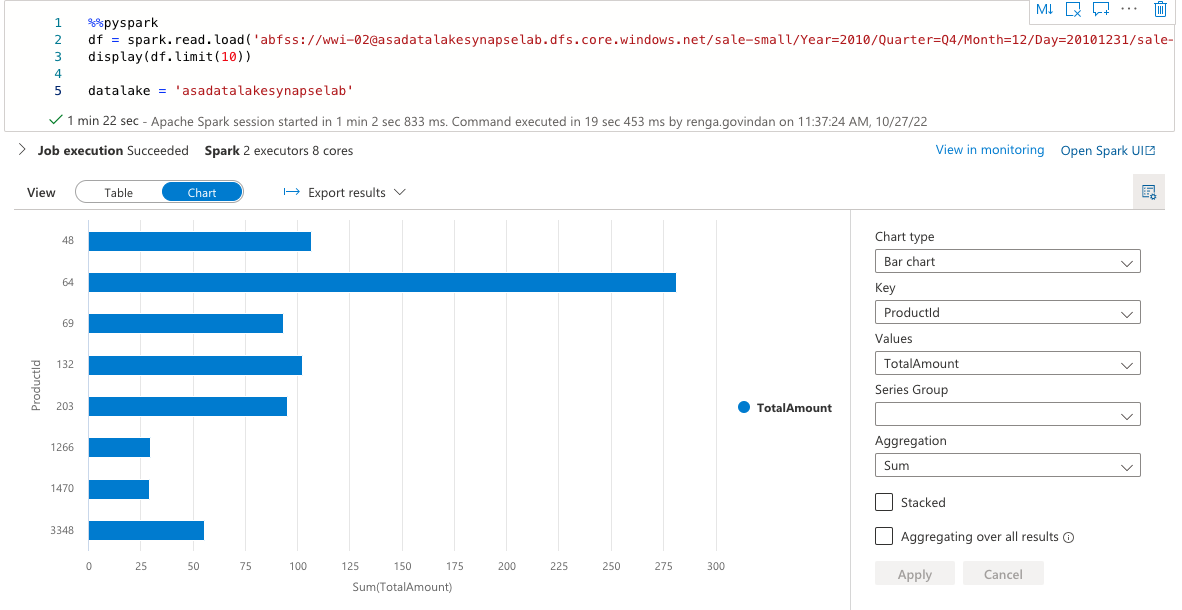
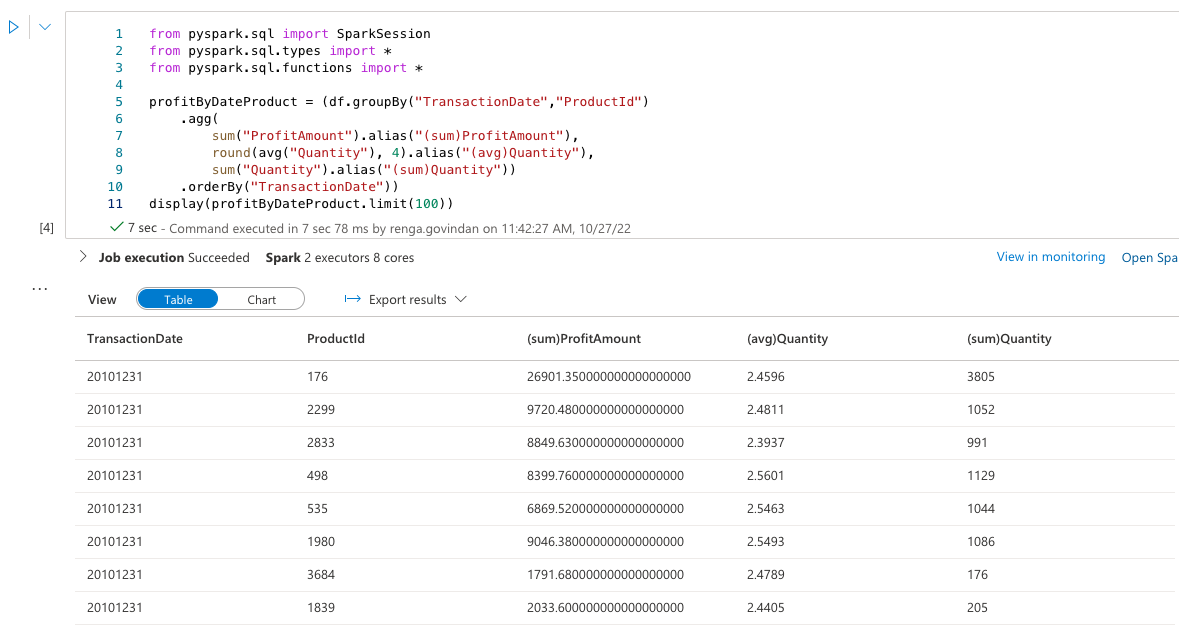
Lab 1: https://github.com/MicrosoftLearning/DP-203T00-Data-Engineering-on-Microsoft-Azure/blob/main/Instructions/Labs/05/README.md

Exercise 1 – fixing File

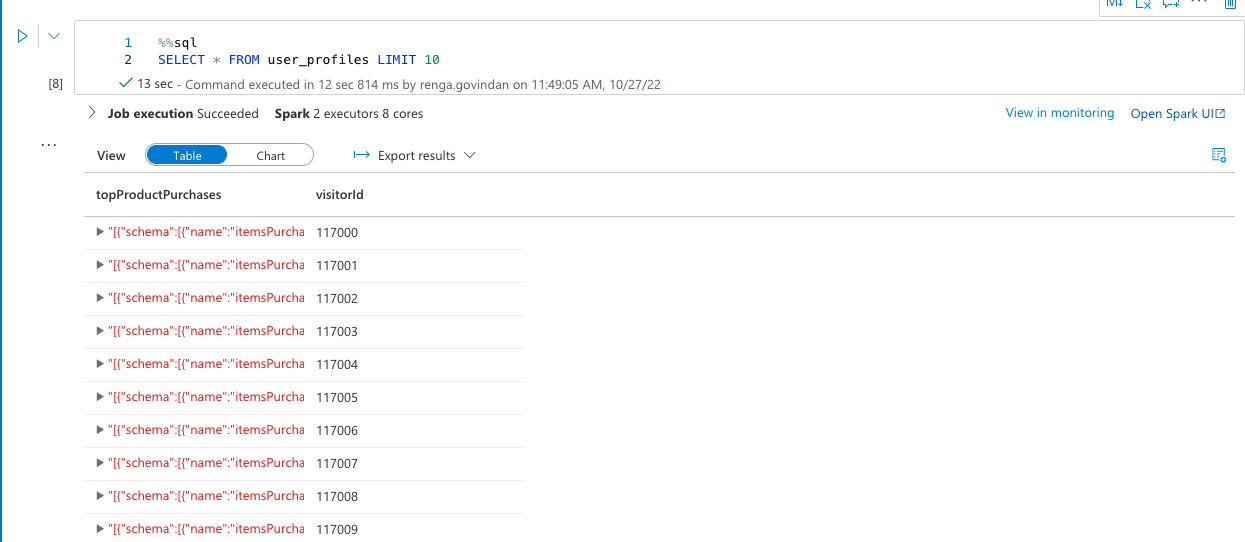


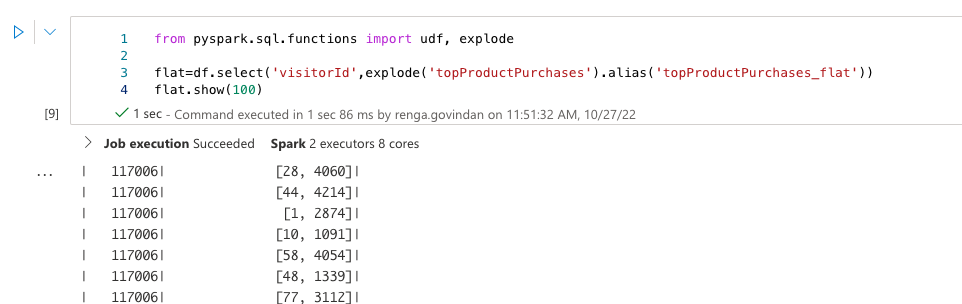
Exercise 2 – Ingest data from file and explore

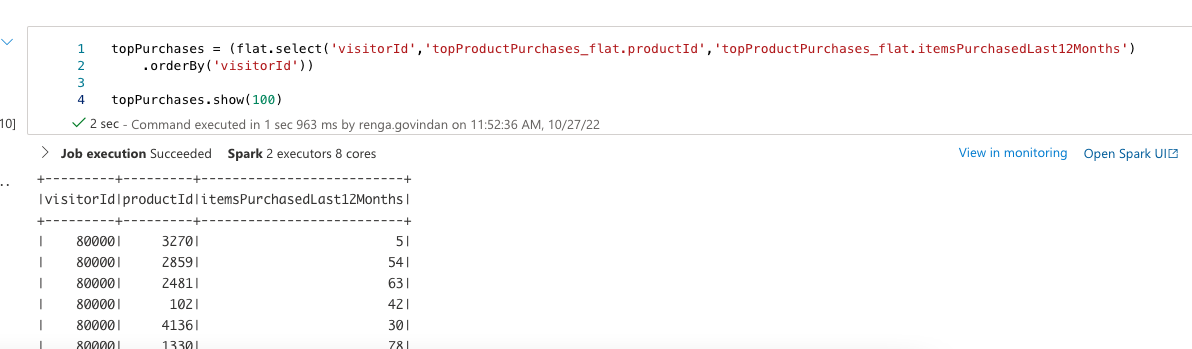


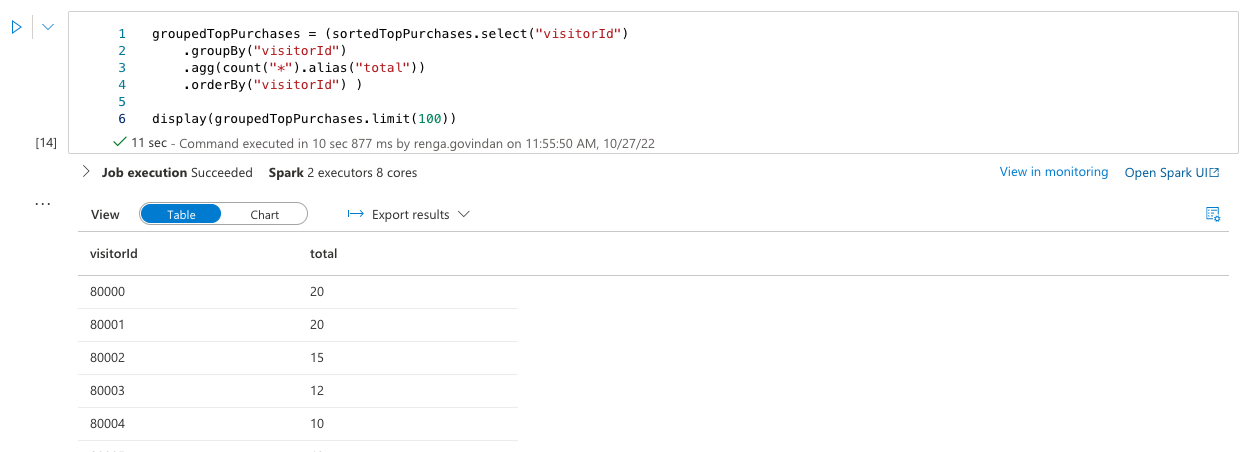


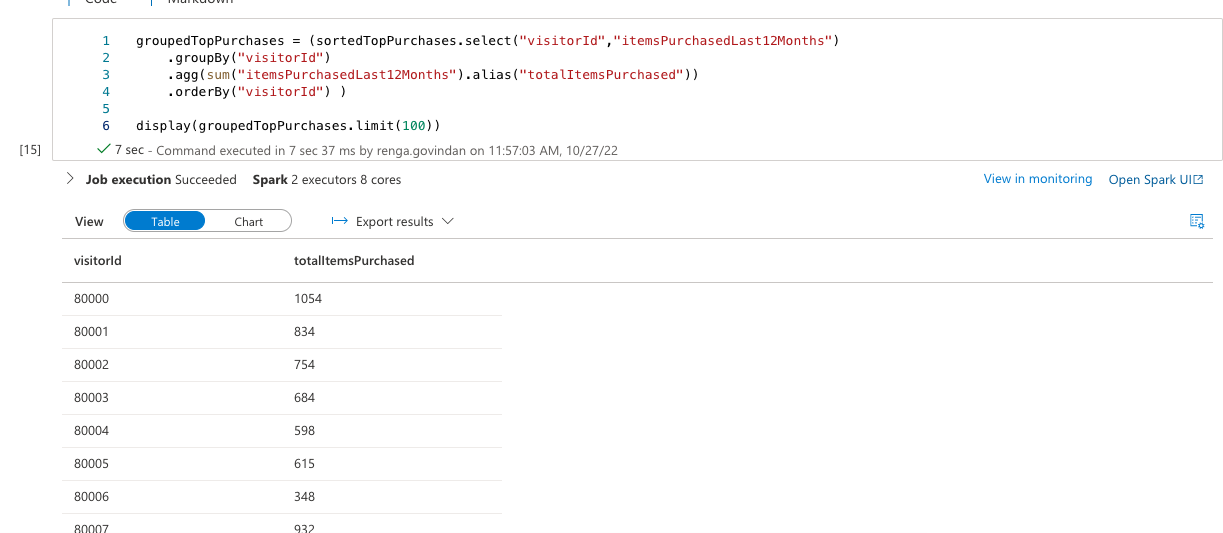
Exercise 3: Transforming data with DataFrames





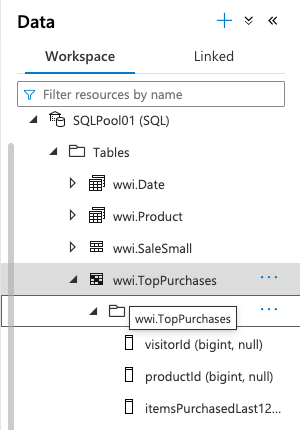


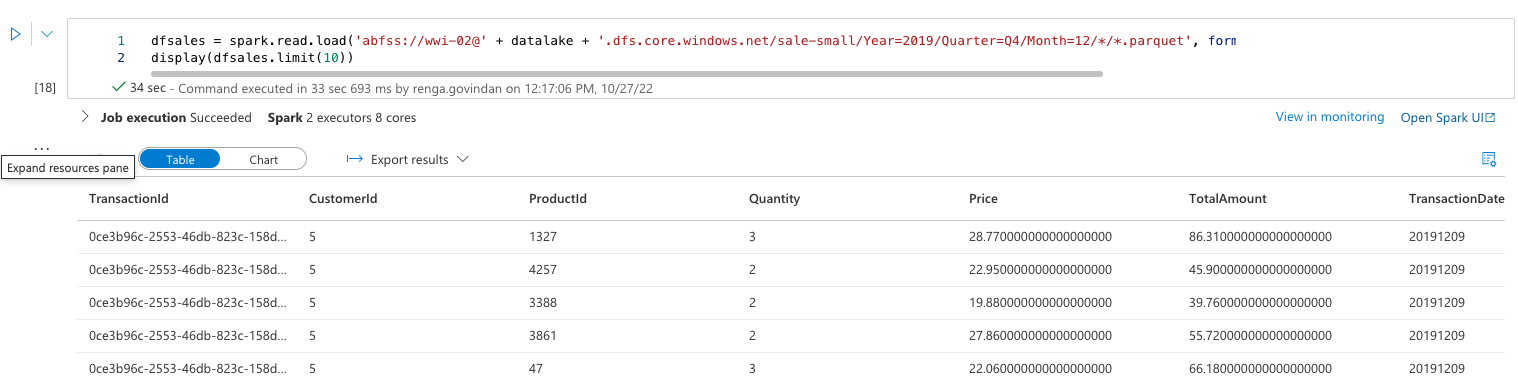


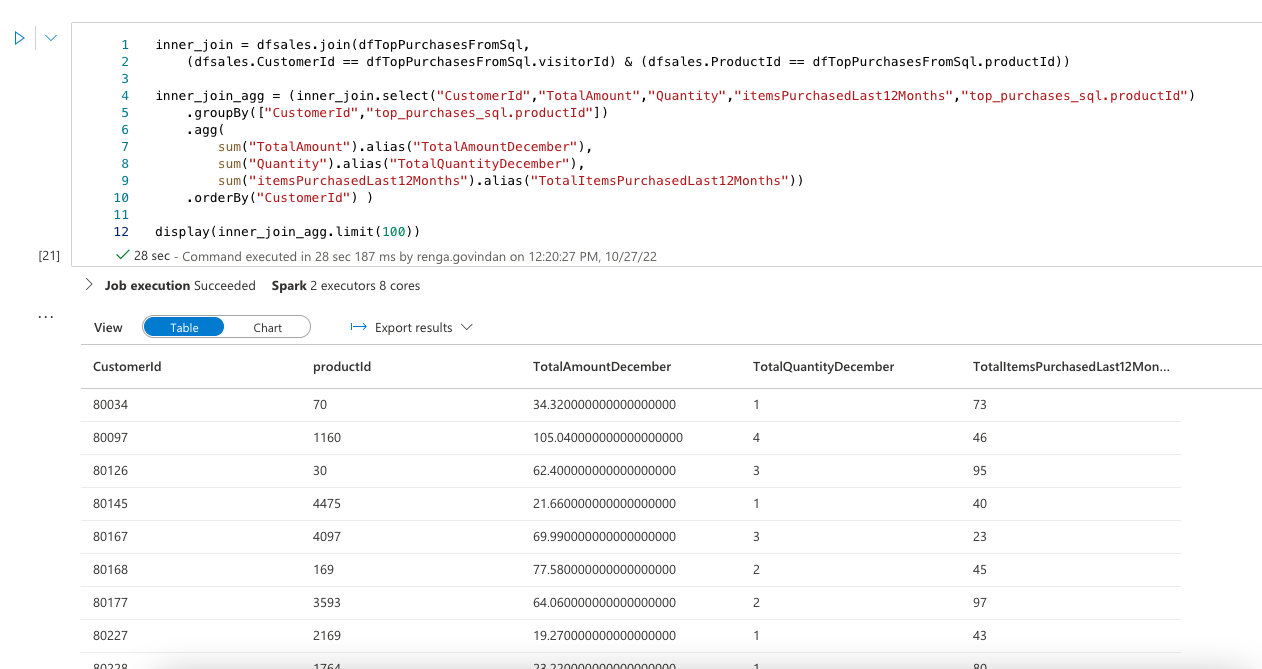


Exercise 4:





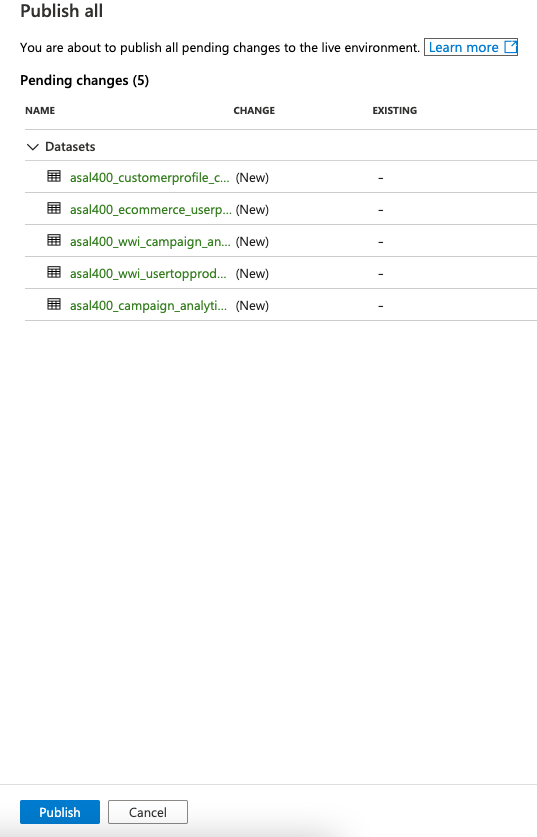


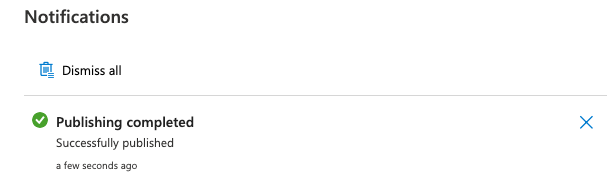


Lab2: https://github.com/MicrosoftLearning/DP-203T00-Data-Engineering-on-Microsoft-Azure/blob/main/Instructions/Labs/08/README.md

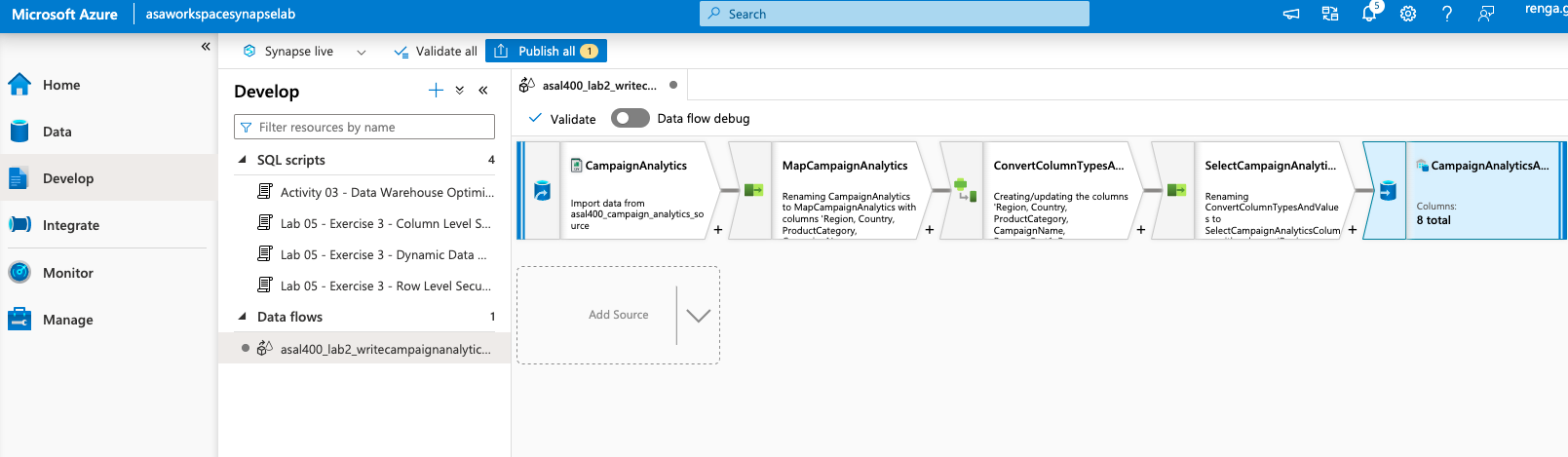
1 – Code free transformation at scale with Azure Synapse pipelines

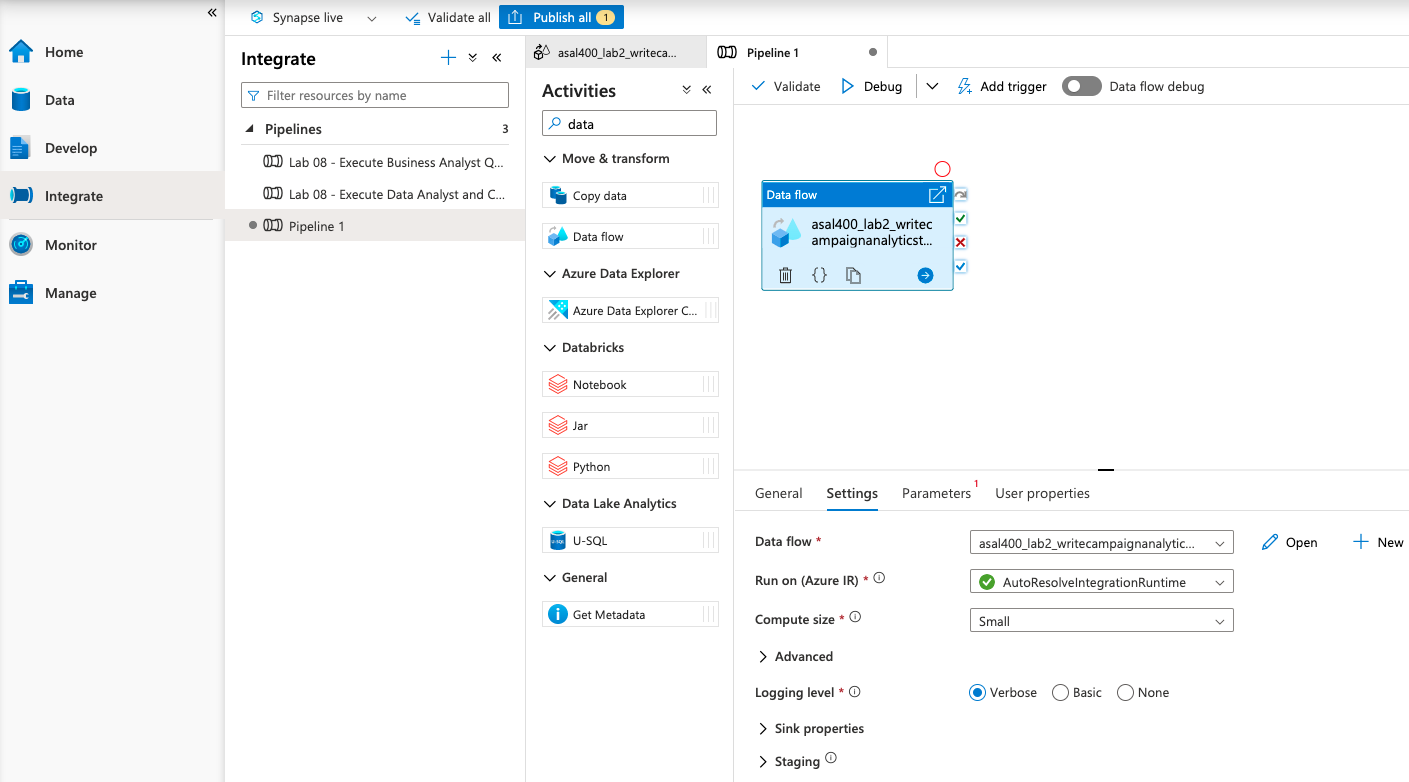
Exercise 1 – Create Artifacts

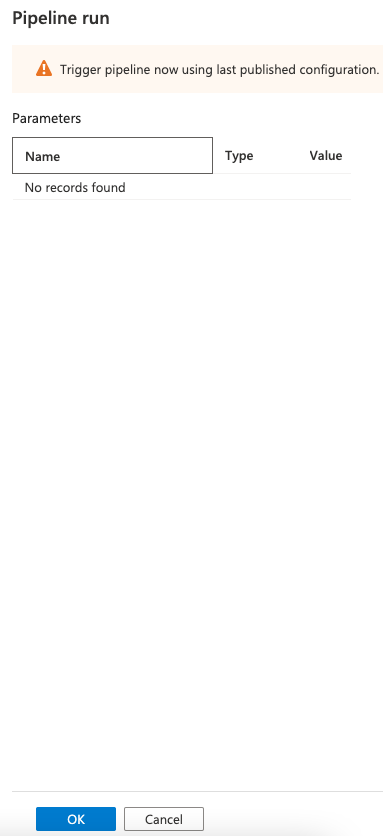


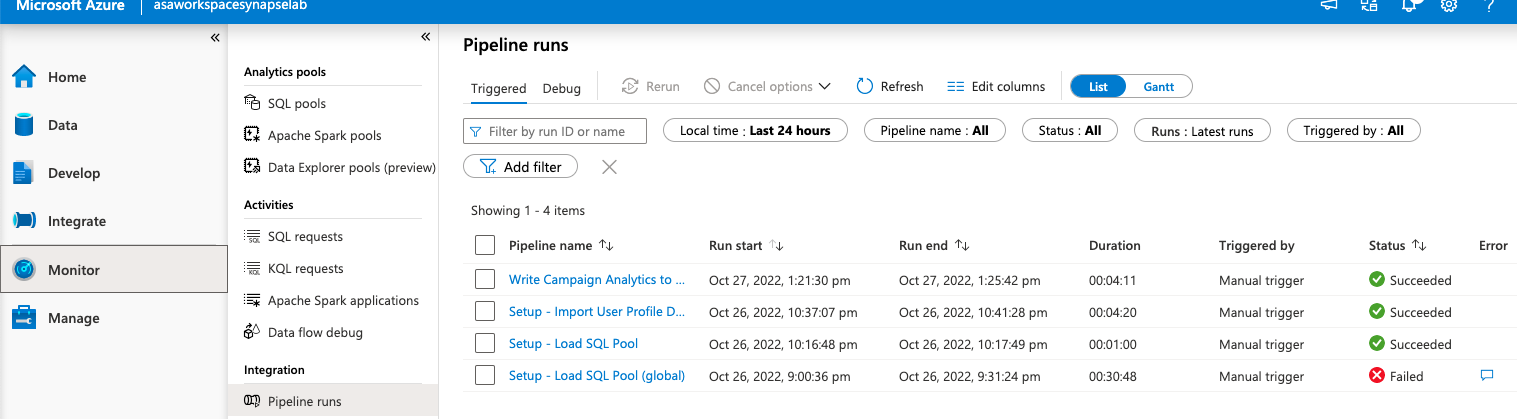


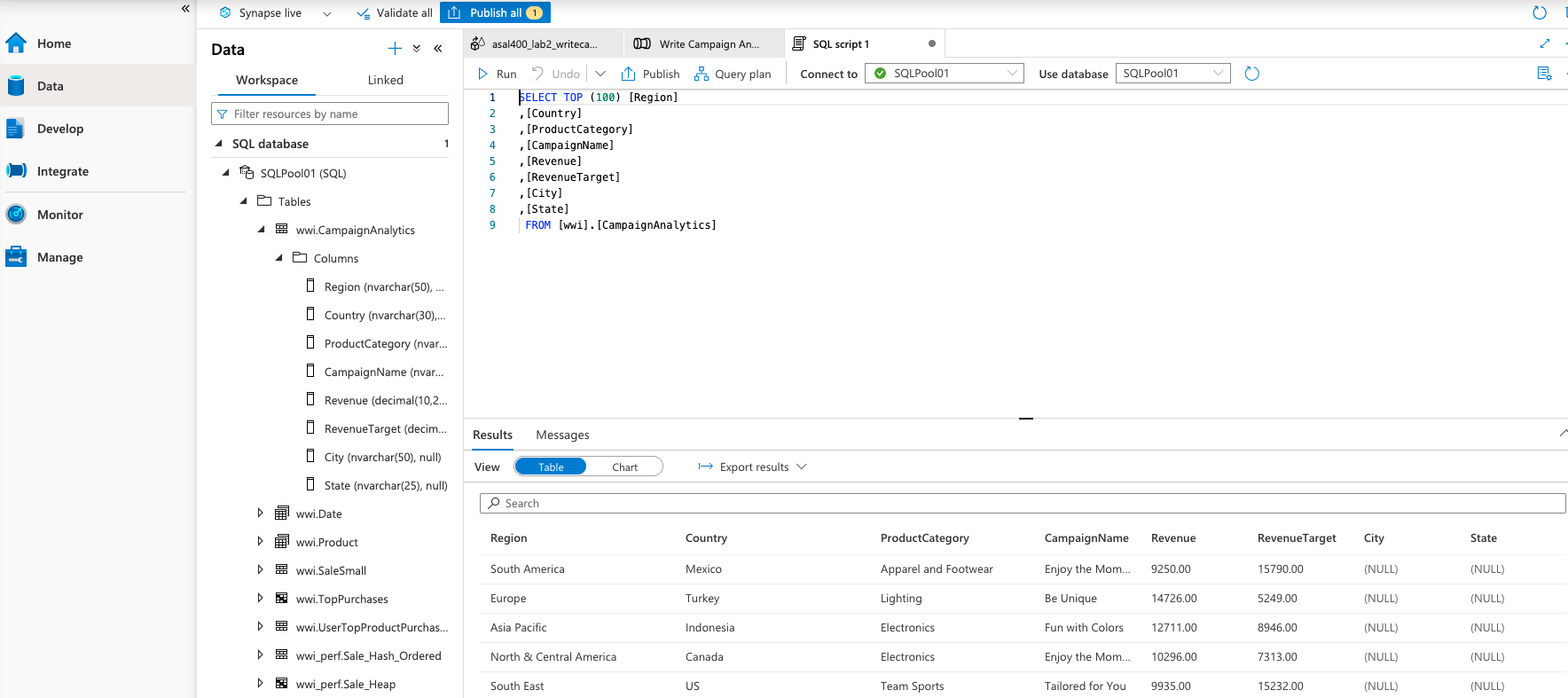
Exercise 2: Create Data pipeline to import poorly formatted CSV

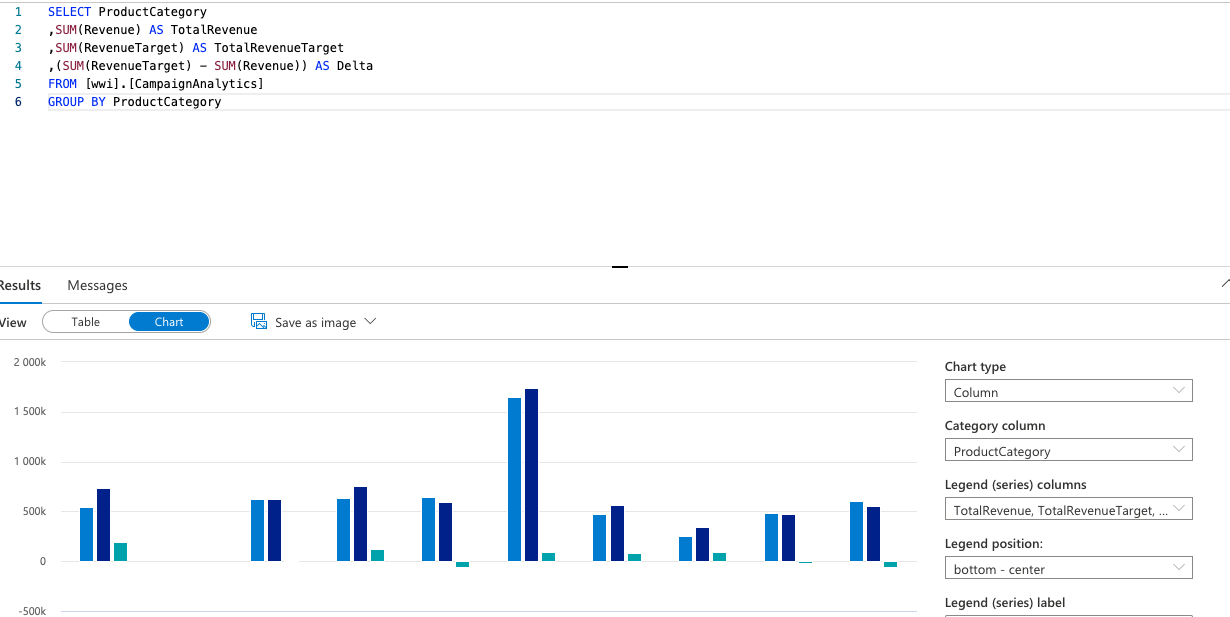




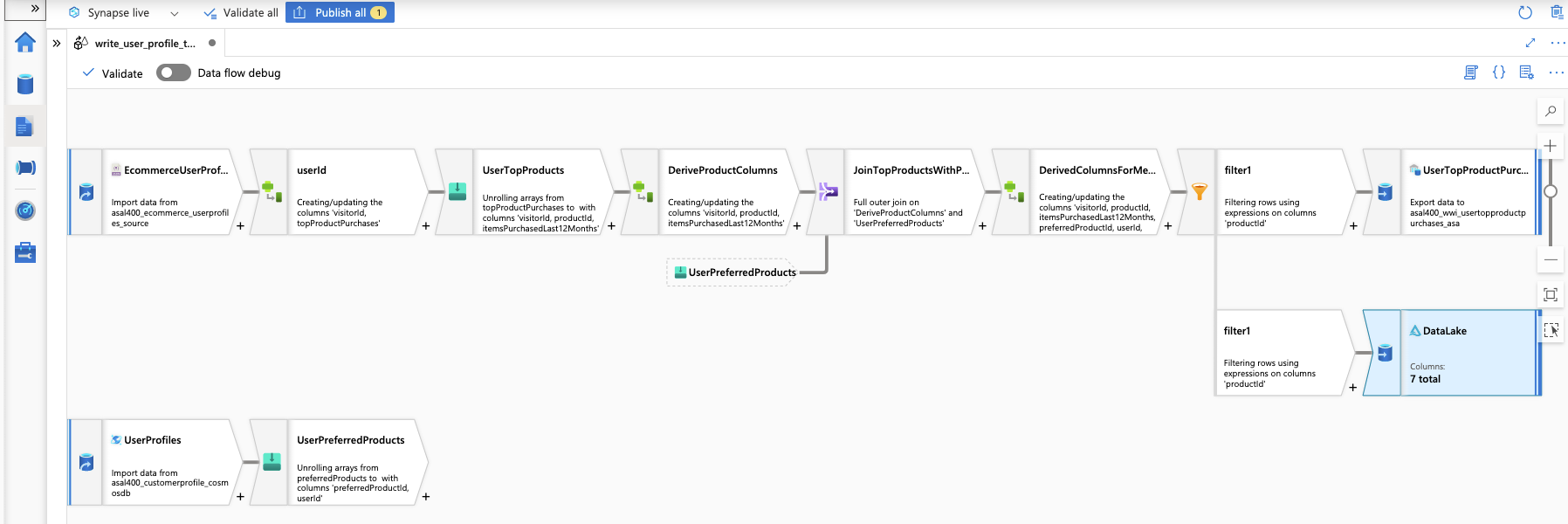






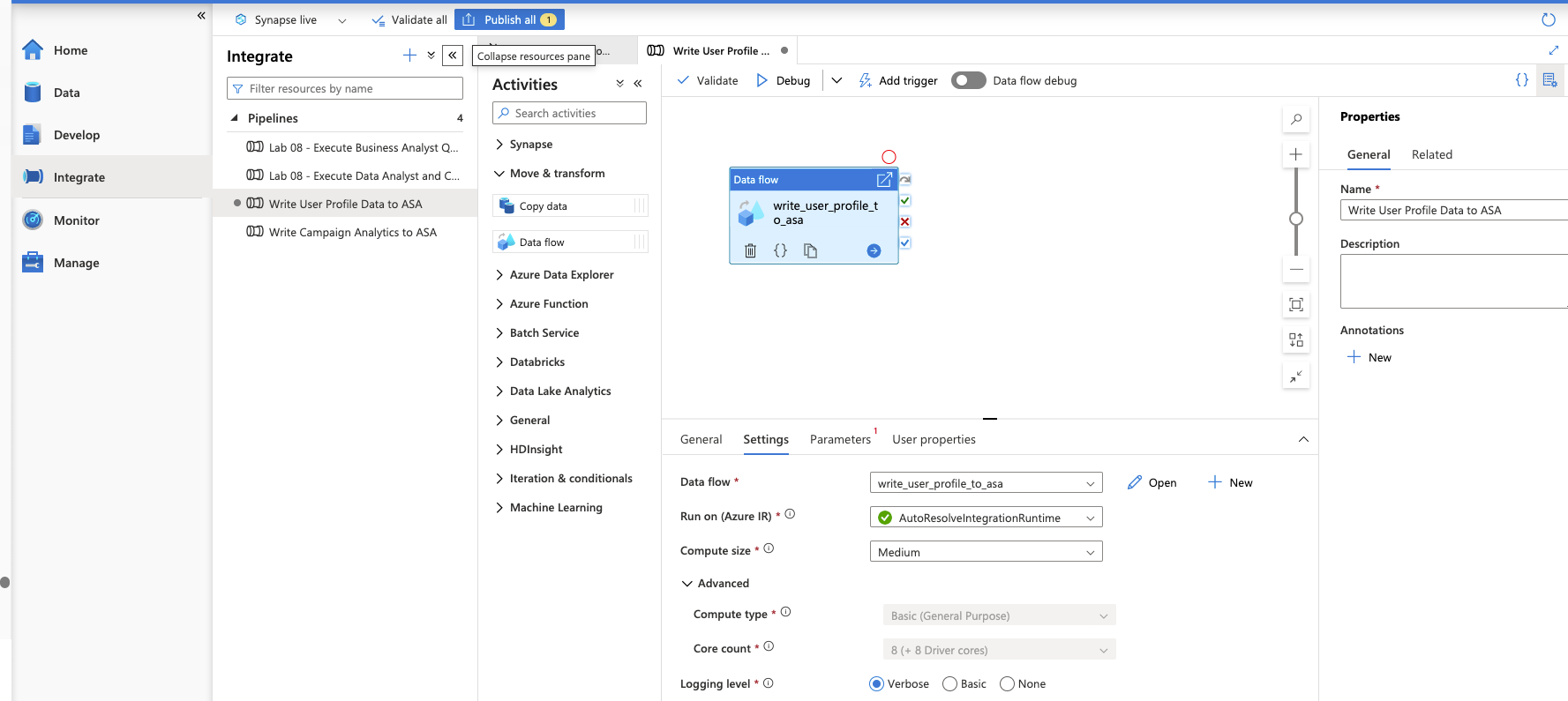


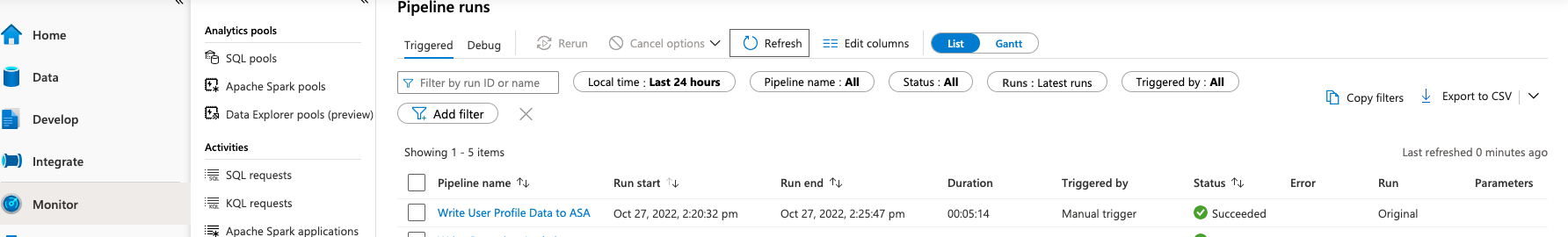
Exercise 3: Create Mapping Data Flow for top product purchases

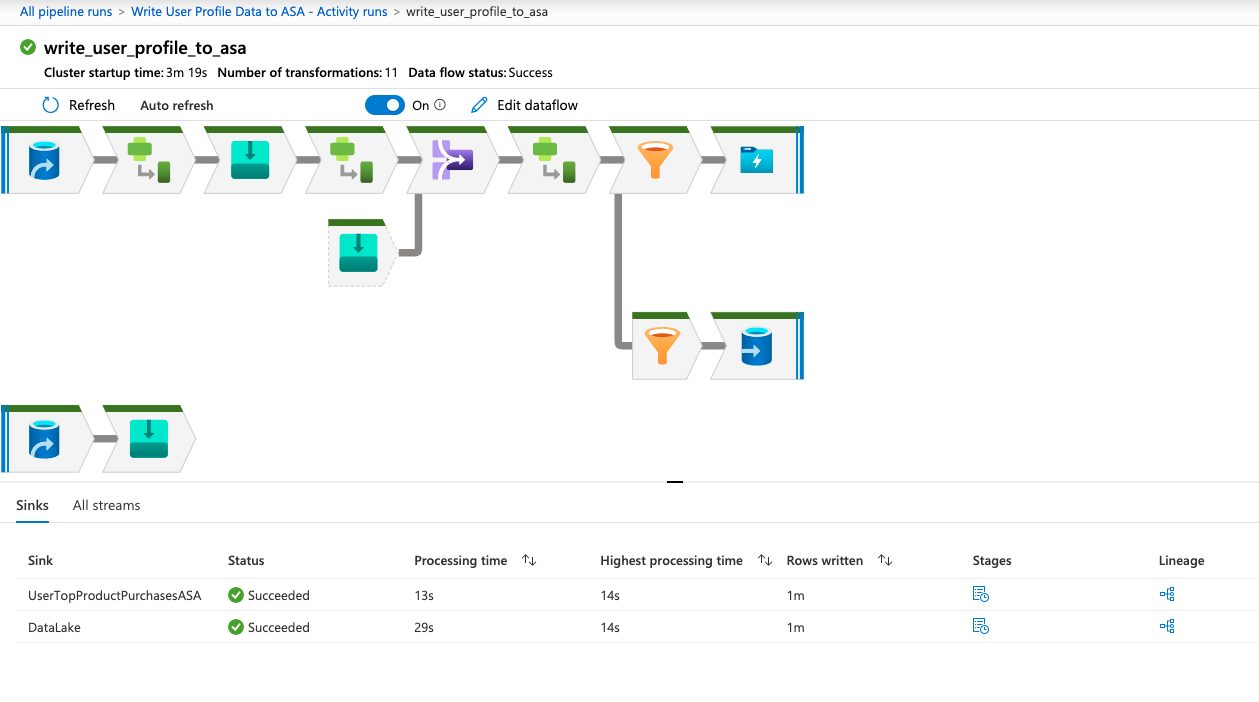


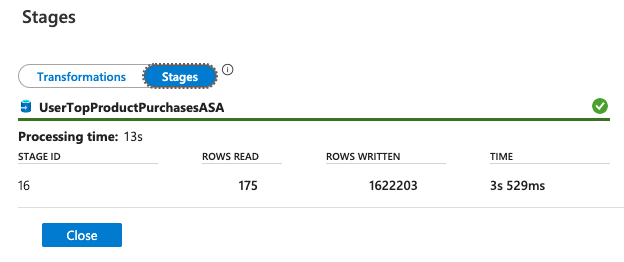
2: Orchestrate data movement and transformation in Azure Synapse Pipelines

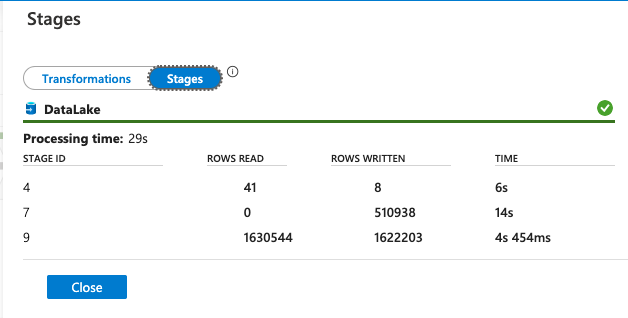
Exercise 1: Create, Trigger and Monitor Pipeline





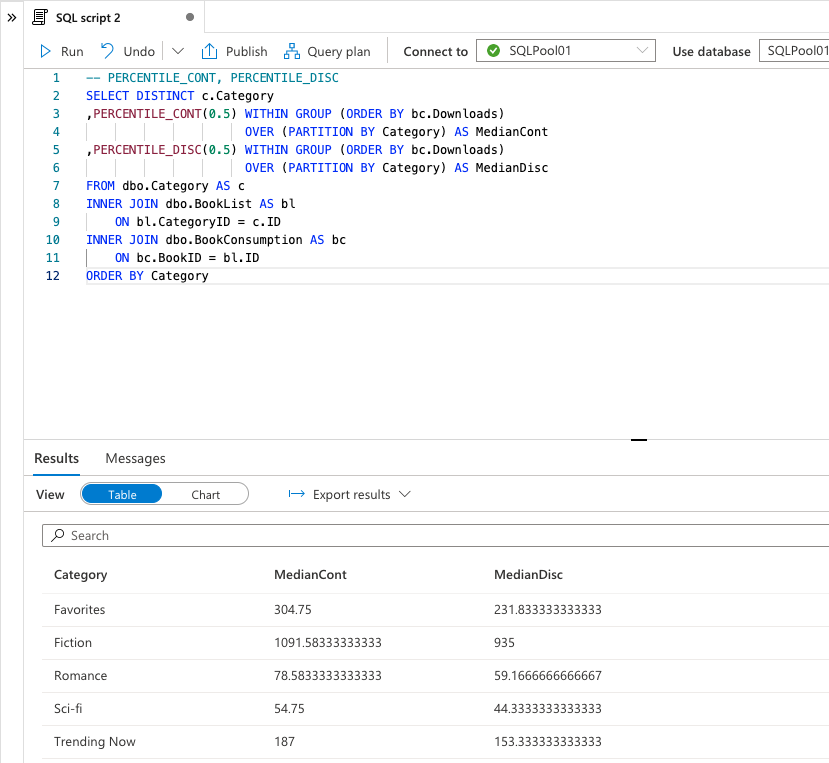


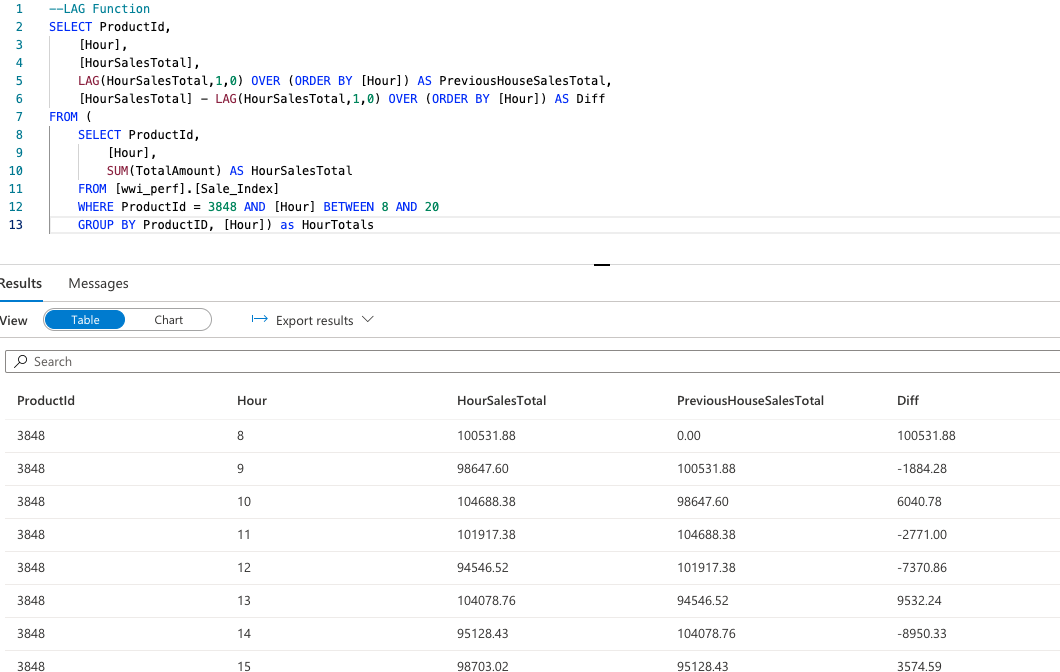




Lab 3: https://github.com/MicrosoftLearning/DP-203T00-Data-Engineering-on-Microsoft-Azure/blob/main/Instructions/Labs/10/README.md

Exercise 1: Understanding developer features of Azure Synapse Analytics

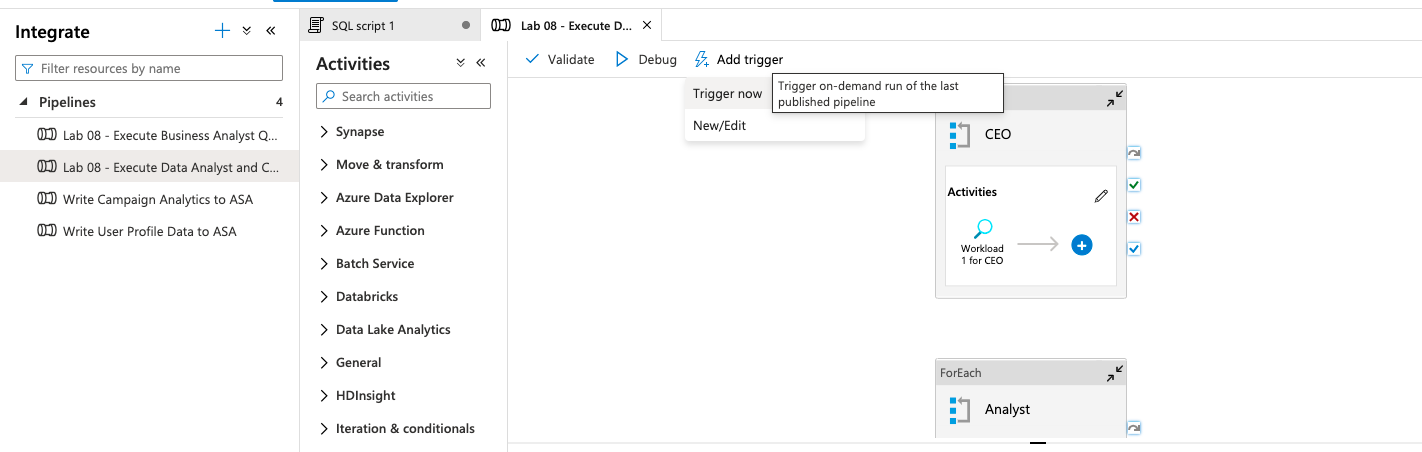


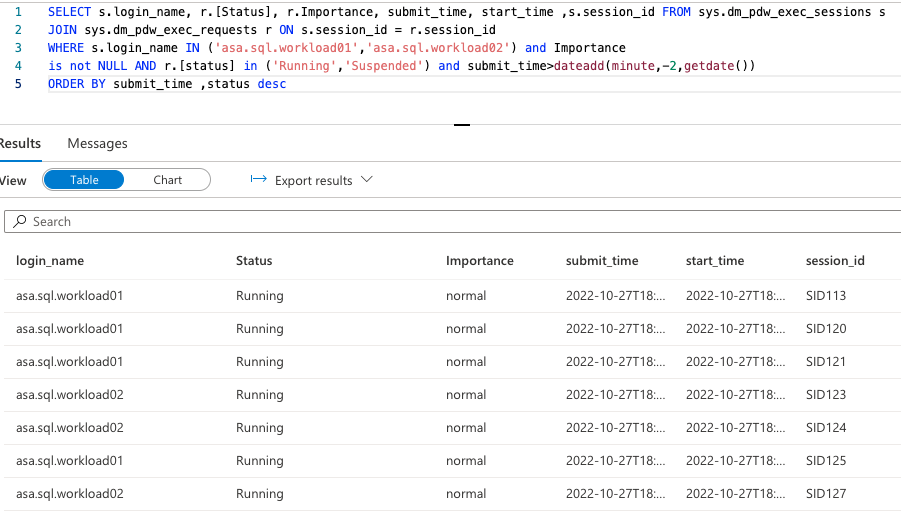


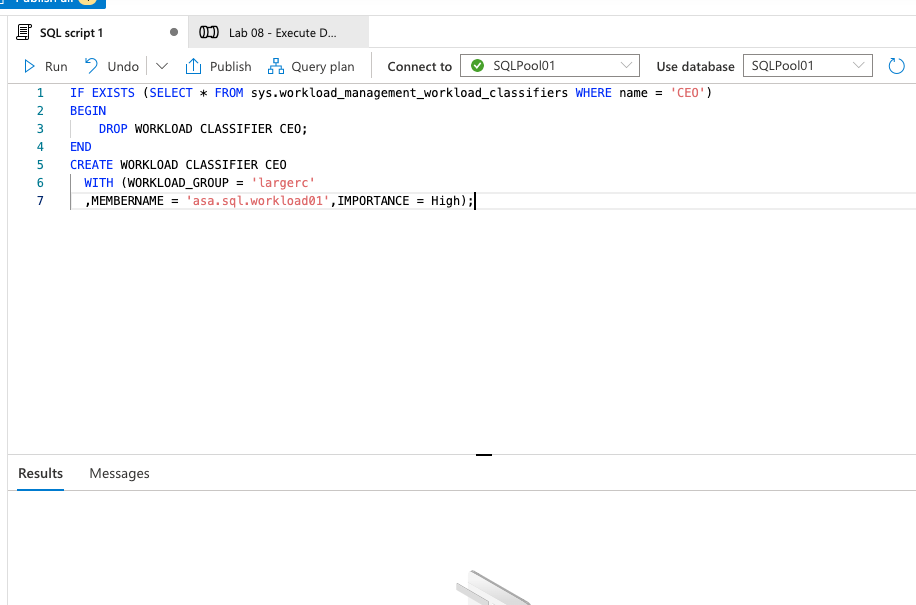


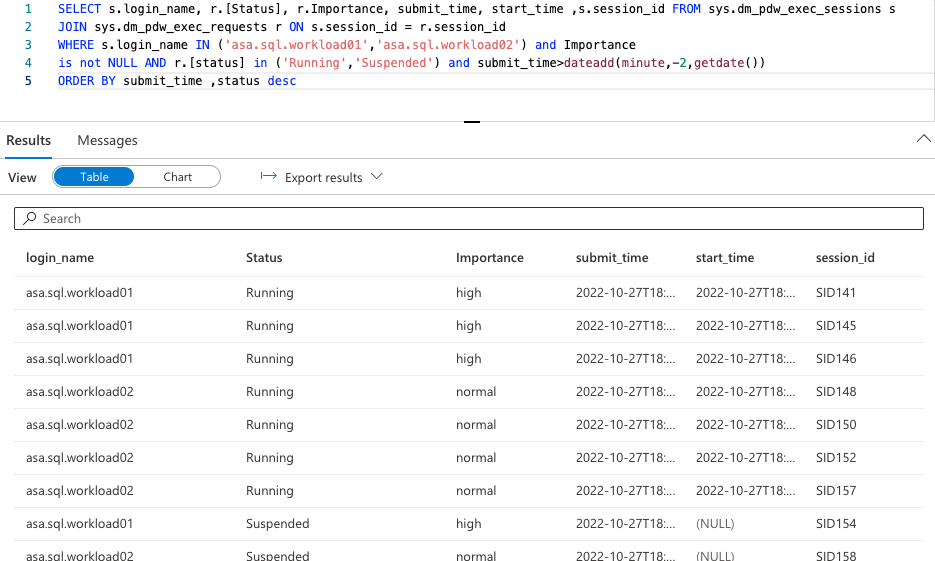
Exercise 2: Data Loading best practices

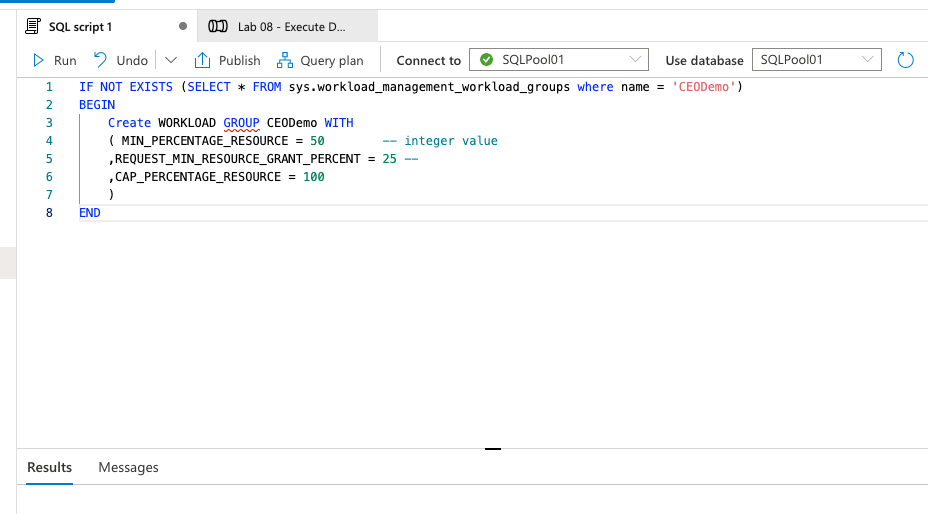
Work Load classification/prioritization

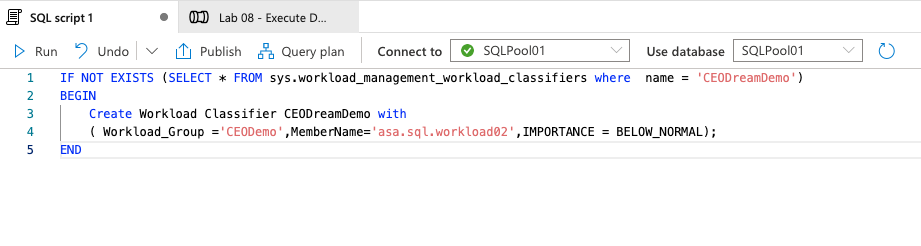


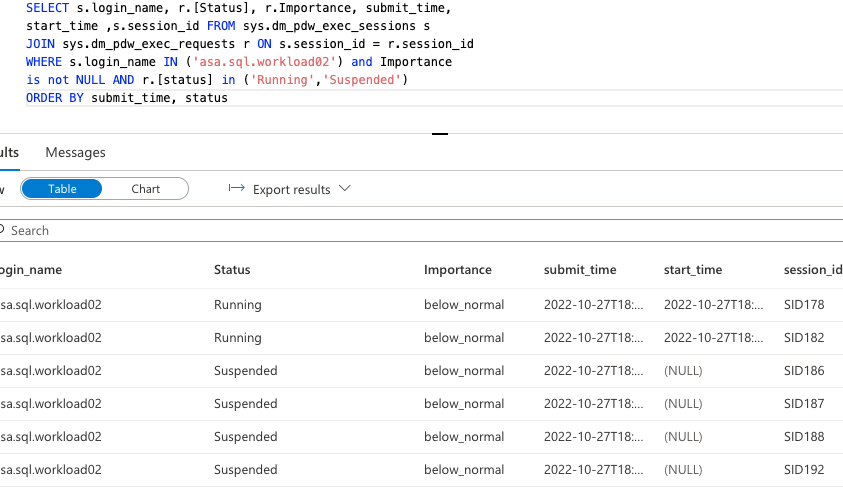


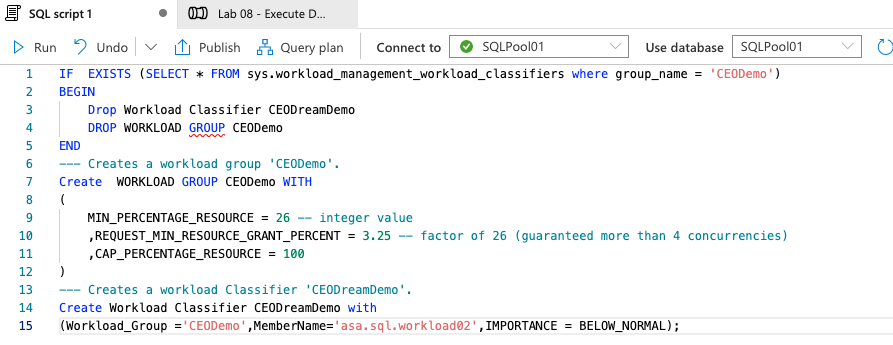




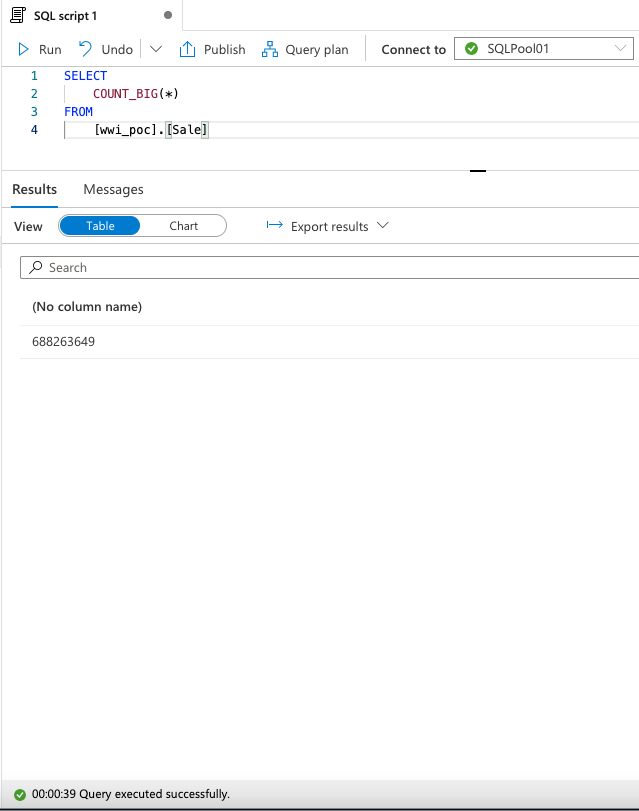


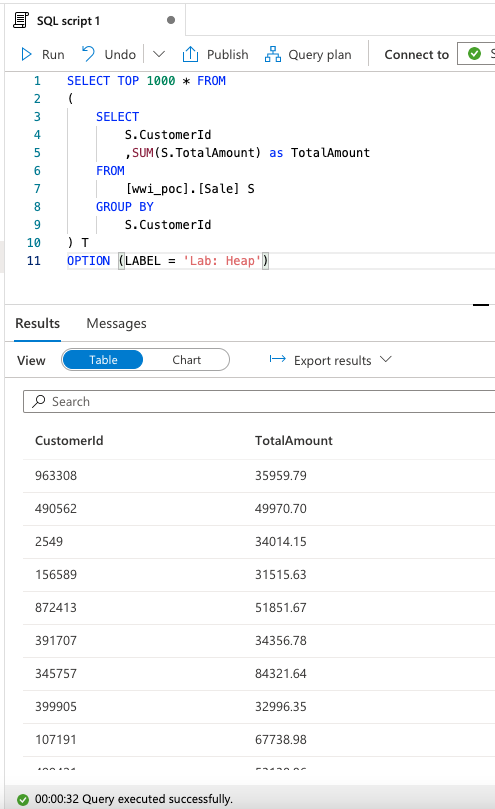


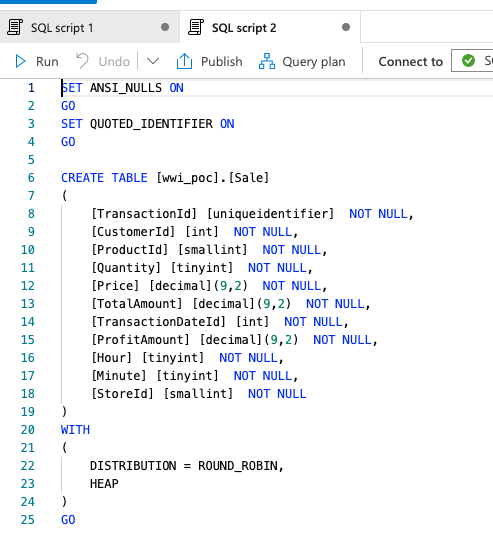


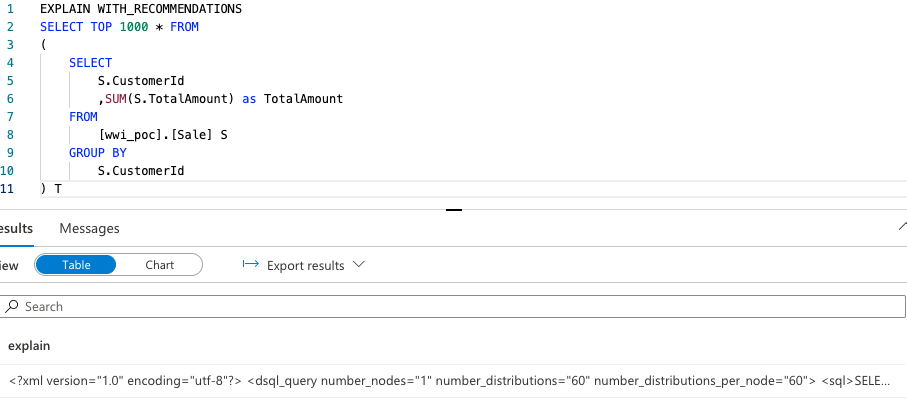


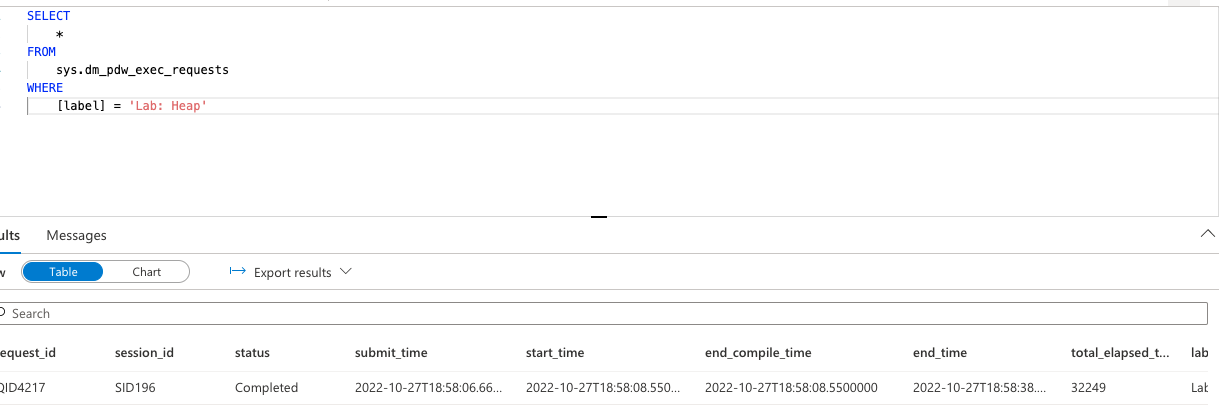
Exercise 3: Optimizing data warehouse query performance

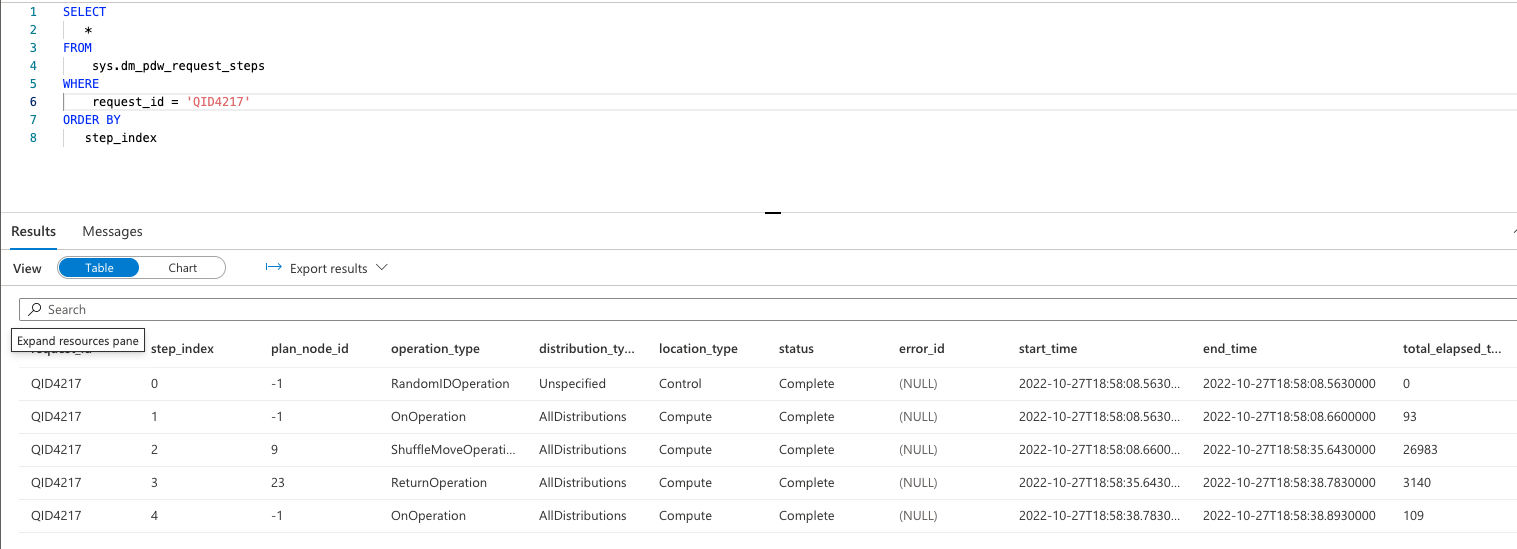


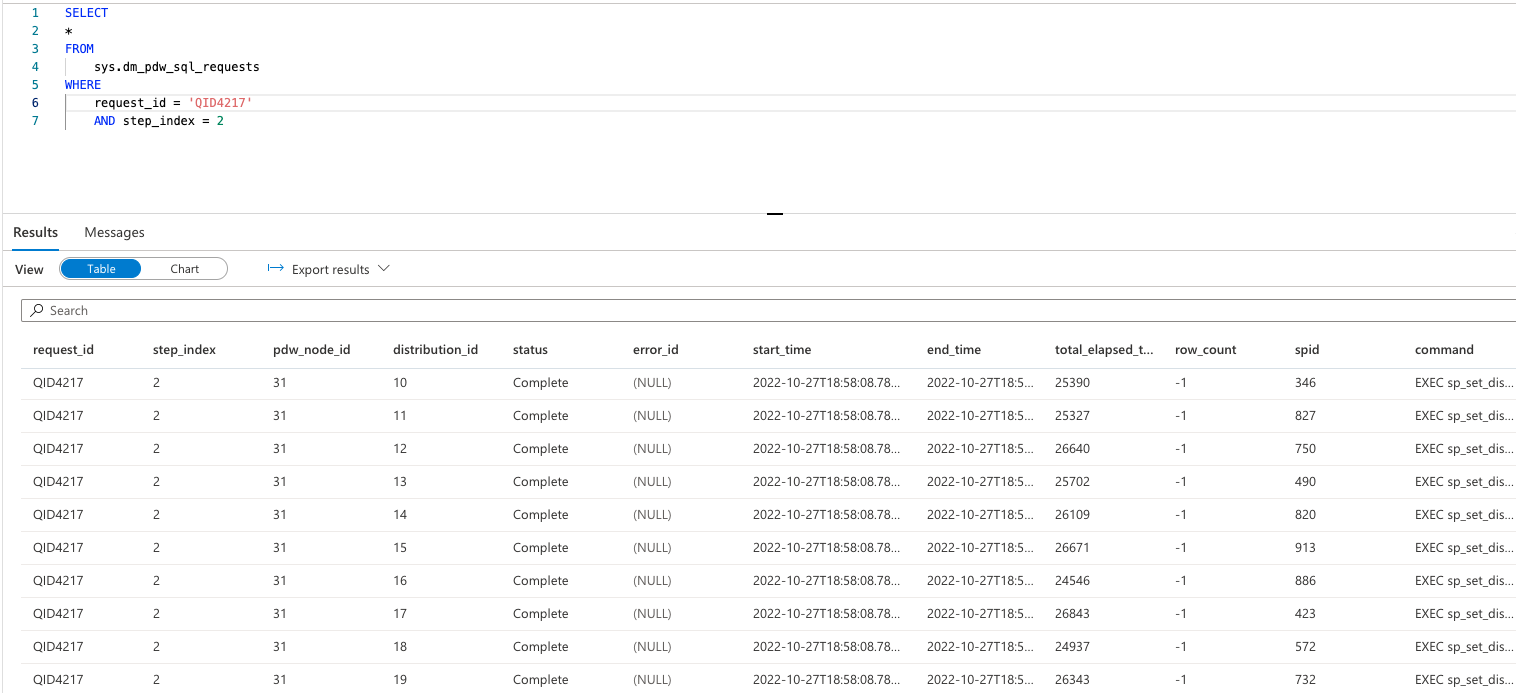


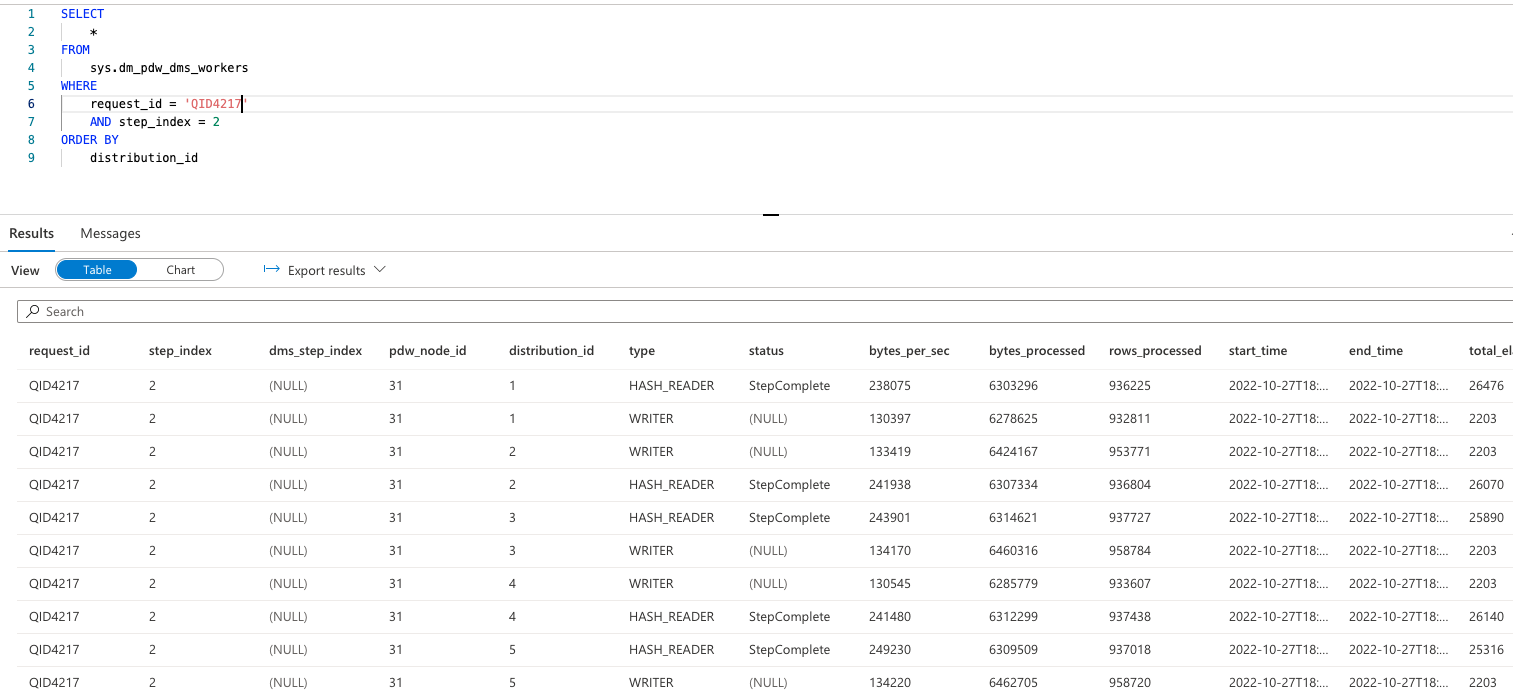


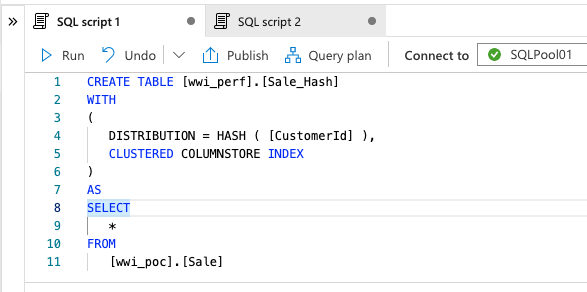


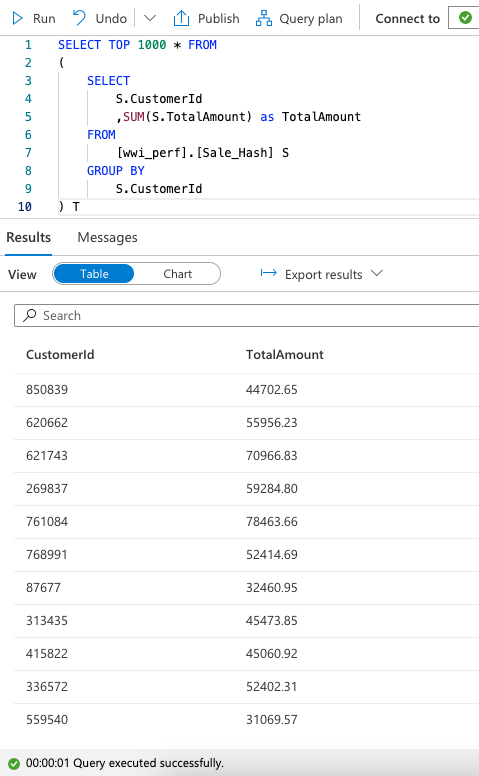


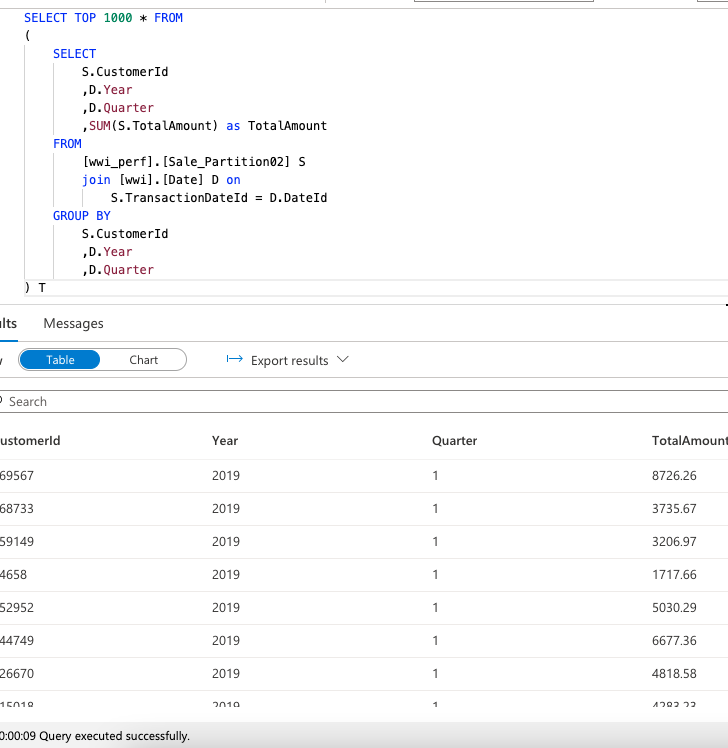


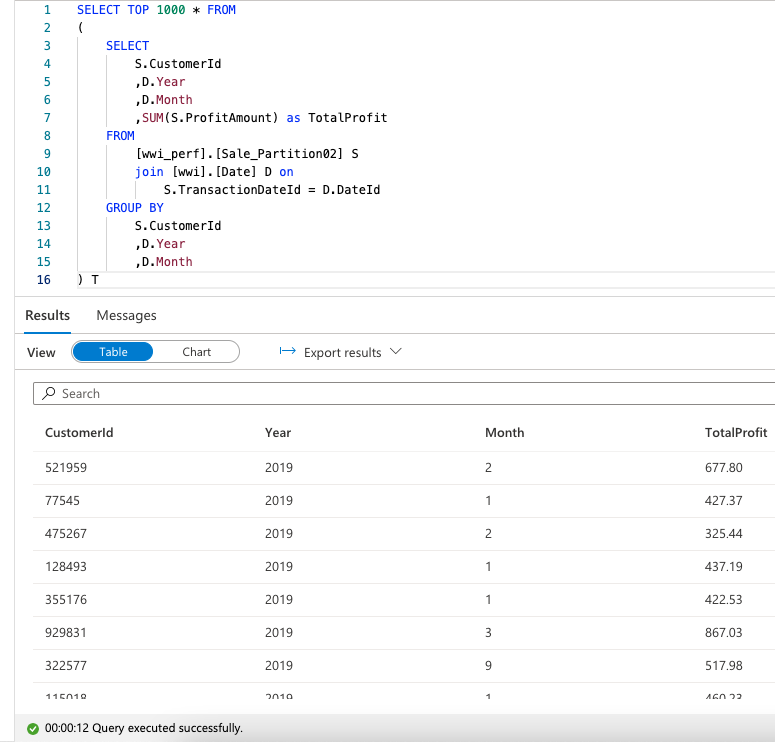


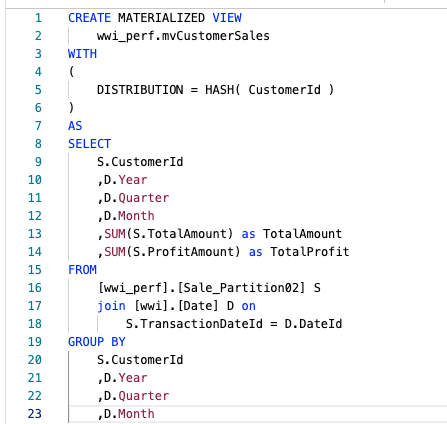


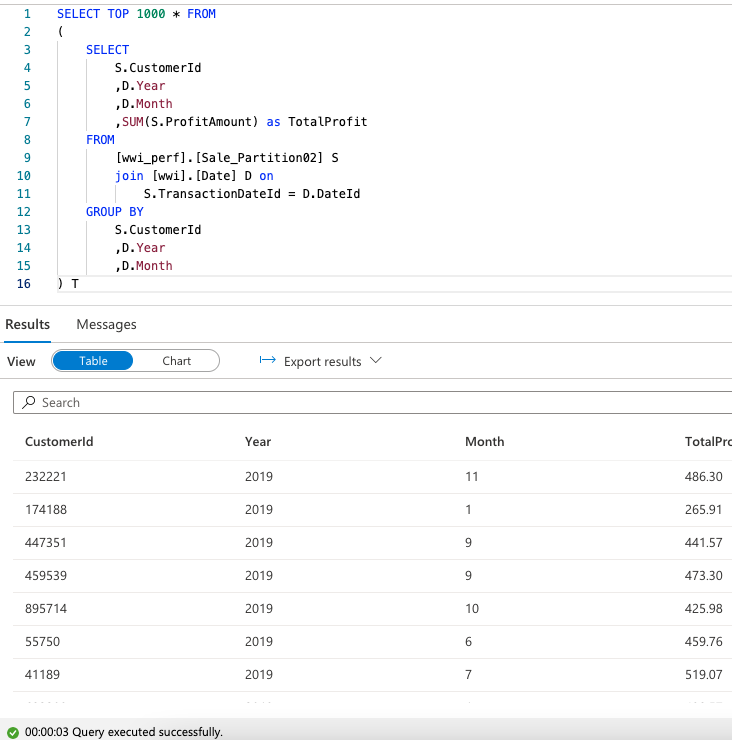


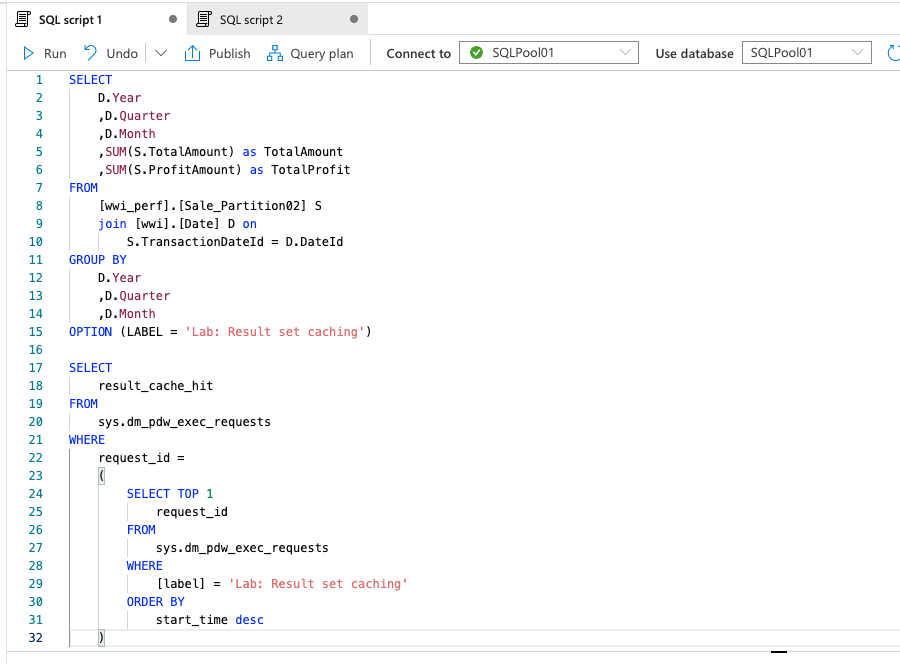


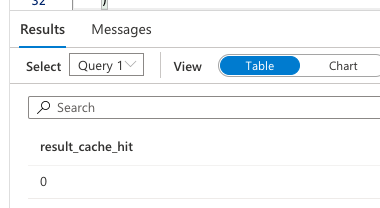


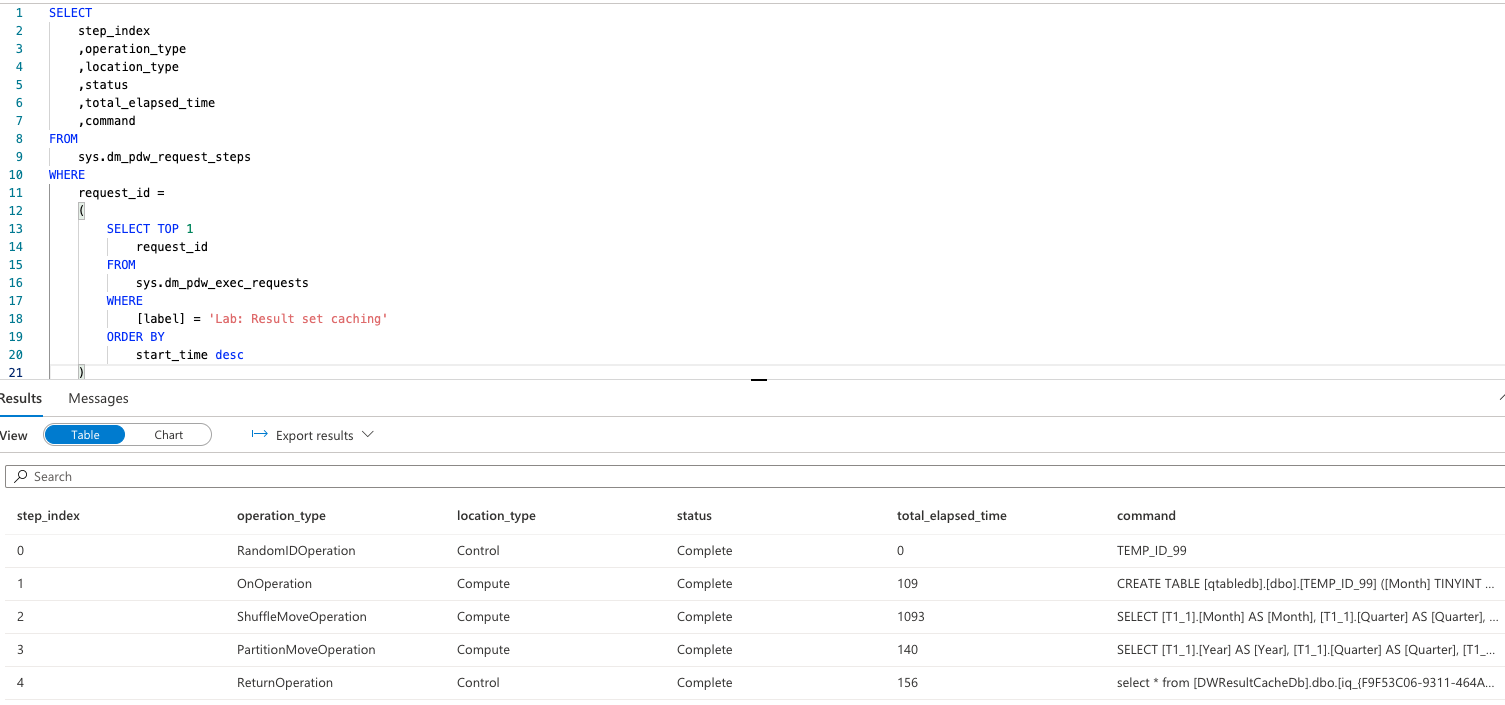


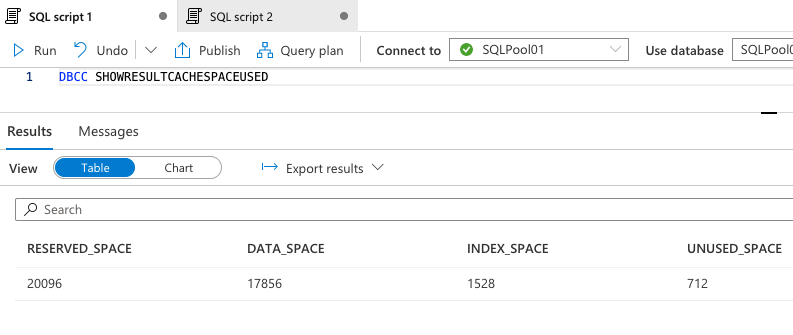


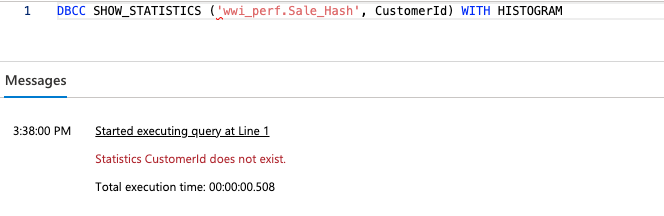


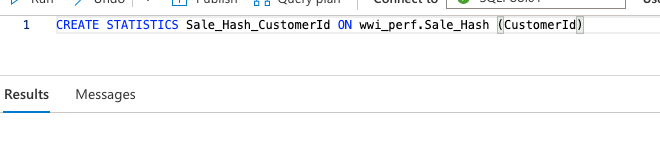


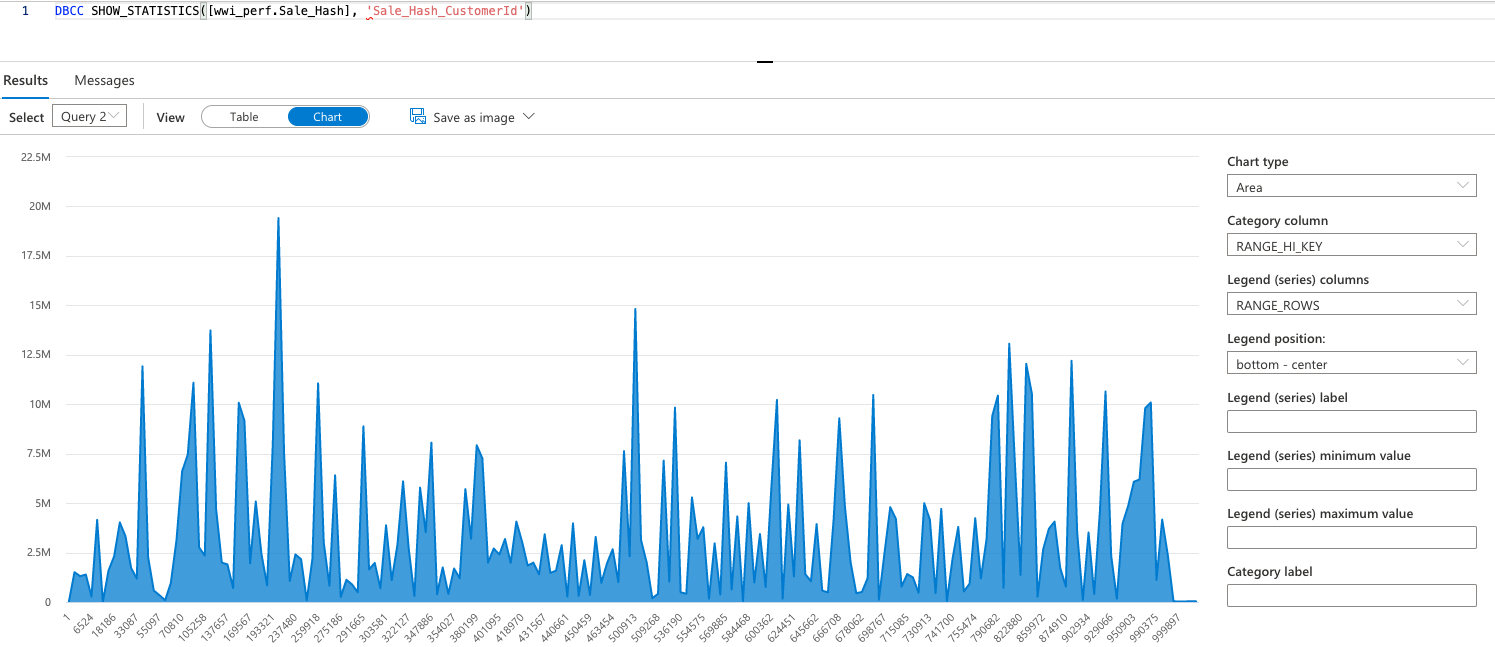


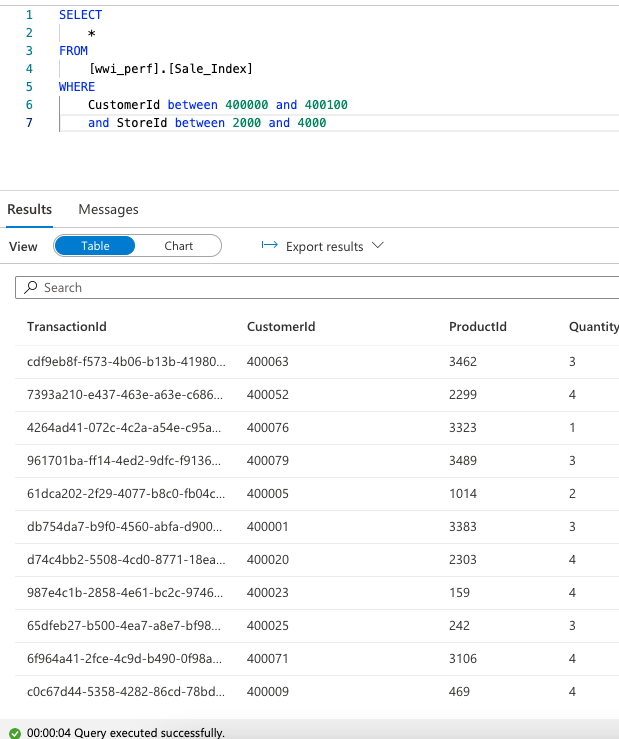


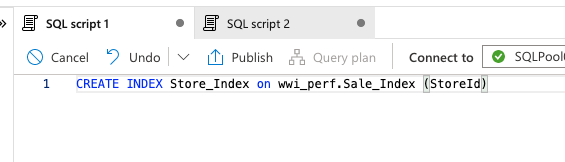












1. Why should one use Azure Key Vault when working in the Azure environment? What are the alternatives to using Azure Key Vault? What are the pros and cons of using Azure Key Vault?

* Azure Key Vault:
  + Protects cryptographic keys, certificates (and the private keys associated with the certificates), and secrets (such as connection strings and passwords) in the cloud.
  + Vulnerability is exposure to more than intended audience, customer may want more control.
  + Can Reduce the exposure of vaults by specifying which IP addresses have access to them. The virtual network service endpoints for Azure Key Vault allows to restrict access to a specified virtual network.
  + Azure Private Link Service enables to access Azure Key Vault and Azure hosted customer/partner services over a Private Endpoint in customer’s virtual network.
* Client-side encryption is performed outside of Azure. It includes:
  + Data encrypted by an application that’s running in the customer’s datacenter or by a service application.
  + Data that is already encrypted when it is received by Azure.
  + With client-side encryption, cloud service providers don’t have access to the encryption keys and cannot decrypt this data. Customer maintain complete control of the keys.

1. How do you achieve the loop functionality within an Azure Data Factory pipeline? Why would you need to use this functionality in a data pipeline?

This can be achieved using the ForEach Activity within Control flow (Azure Data Factory/Synapse Analytics. It is used to iterate over a collection and execute multiple activities in a loop. For eg: if order data ingested needs to be transformed and output published across more than one component – inventory and shipping, this can be used.

1. What are expressions in Azure Data Factory? How are they helpful when designing a data pipeline (please explain with an example)?

Expressions are used to evaluate values dynamically at runtime. These work well with parameters added to the pipeline. Functions can be called within expressions making it very powerful. In dataflow, we can build expressions to derive column values (Lot of examples in above screenshots – converting string to integer, removing extra characters like back slash). Creating files dynamically and naming them is common pattern

1. What are the pros and cons of parametrizing a dataset in Azure Data Factory pipeline’s activity?

Parameters help templatize pipelines and can be reused if built as a framework across multiple use cases. However, depending on the complexity of the pipelines, maintaining them can be a nightmare. There needs to be a balance between parameterizing and either choosing to build a different pipeline or make use of branches/conditionals. For eg: if a parameter will only be Boolean, doesn’t make sense to have it as a parameter.

1. What are the different supported file formats and compression codecs in Azure Data Factory? When will you use a Parquet file over an ORC file? Why would you choose an AVRO file format over a Parquet file format?

Supported file formats:

* [Avro format](https://learn.microsoft.com/en-us/azure/data-factory/format-avro)
* [Binary format](https://learn.microsoft.com/en-us/azure/data-factory/format-binary)
* [Delimited text format](https://learn.microsoft.com/en-us/azure/data-factory/format-delimited-text)
* [Excel format](https://learn.microsoft.com/en-us/azure/data-factory/format-excel)
* [JSON format](https://learn.microsoft.com/en-us/azure/data-factory/format-json)
* [ORC format](https://learn.microsoft.com/en-us/azure/data-factory/format-orc)
* [Parquet format](https://learn.microsoft.com/en-us/azure/data-factory/format-parquet)
* [XML format](https://learn.microsoft.com/en-us/azure/data-factory/format-xml)

When performance is priority over available space, parquet is chosen and if space efficiency is the main priority ORC is chosen. (Reference: https://medium.com/@dhareshwarganesh/benchmarking-parquet-vs-orc-d52c39849aef)

Avro

* Widely used as a serialization platform
* Row-based, offers a compact and fast binary format
* Schema is encoded on the file so the data can be untagged
* Files support block compression and are splittable
* Supports schema evolution

Parquet

* Column-oriented binary file format
* Uses the record shredding and assembly algorithm described in the Dremel paper
* Each data file contains the values for a set of rows
* Efficient in terms of disk I/O when specific columns need to be queried

(reference : https://www.slideshare.net/StampedeCon/choosing-an-hdfs-data-storage-format-avro-vs-parquet-and-more-stampedecon-2015)