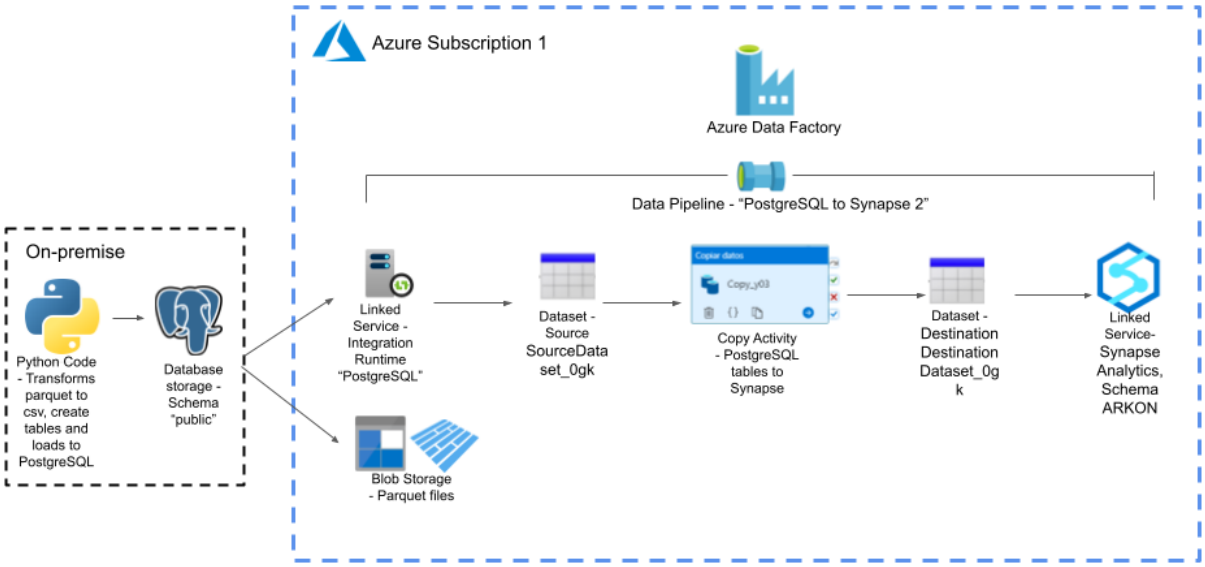
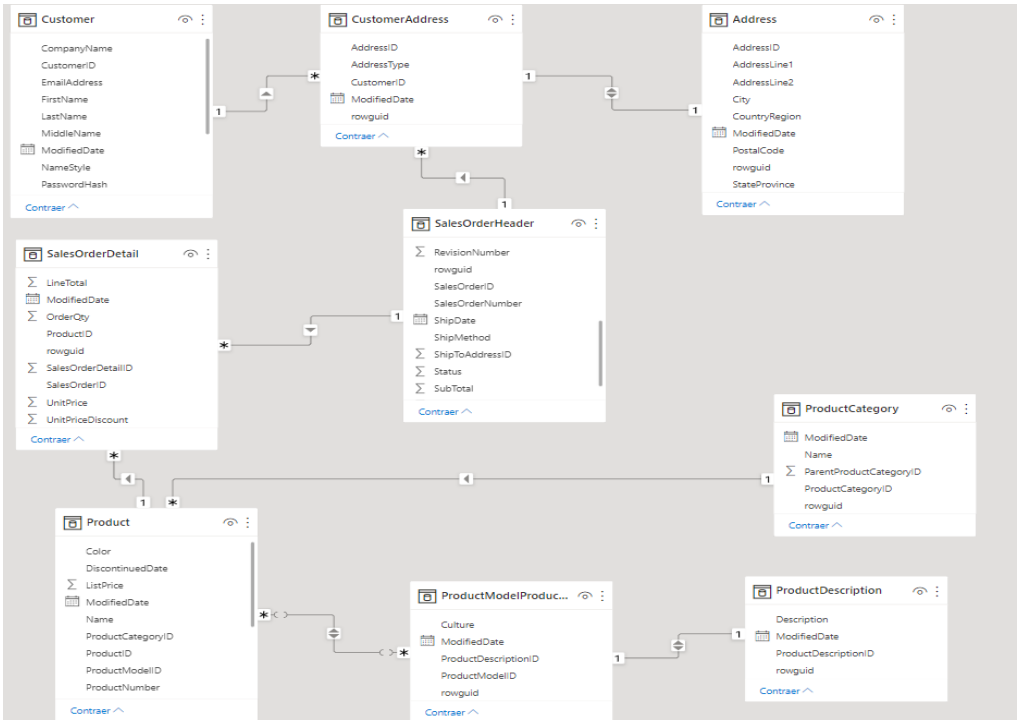
1. Diagrama de Solución

Para la solución de la prueba y la creación de un data pipeline se elaboró la siguiente propuesta de solución. Esta propuesta y el paso a paso se explica a detalle a partir de la sección 2 de este documento.



2. Modelo de Datos y DLLs

Para tener una perspectiva de la relación de tablas que serán cargadas a una base de datos PostgreSQL se cargaron primeramente los archivos en formato csv (transformados de parquet a csv con el código Python del punto 3) a PowerBI. La siguiente imagen muestra la relación entre tablas:



La creación de un servidor de PostgreSQL se realizó manualmente y se conectó a DBEAVER, que es una herramienta de administración de bases de datos. Se creó el esquema “public” y se definieron las tablas con los queries que se han agregado a anexos (al final de este documento). Las llaves para la conexión al servidor se presentan en el código Python de la sección 3.

3. Carga de datos a PostgreSQL (con Python)

El código escrito fue parte de la solución para un examen previo pero se tomó como primer paso para la solución a esta nueva prueba. El código fuente puede consultarse en el repositorio:

https://github.com/hromolozano/Arkon-Simple-Data-Pipeline.git

El código contiene comentarios que permiten el entendimiento del mismo pero se describe su secuencia a continuación:

1. Se importa el módulo para manipulación de datos, se determina una lista con el nombre de los archivos parquet y se lee cada documento para convertirlo a un dataframe.
2. Se creó una base de datos llamada “postgres” y los queries que crean las tablas dentro del esquema “public”. En el código se definen los nombres de los archivos de extensión txt para ser leídos y convertidos a un string. Esto resultó en una

lista donde cada elemento es el query a ejecutar en la siguiente celda de código.

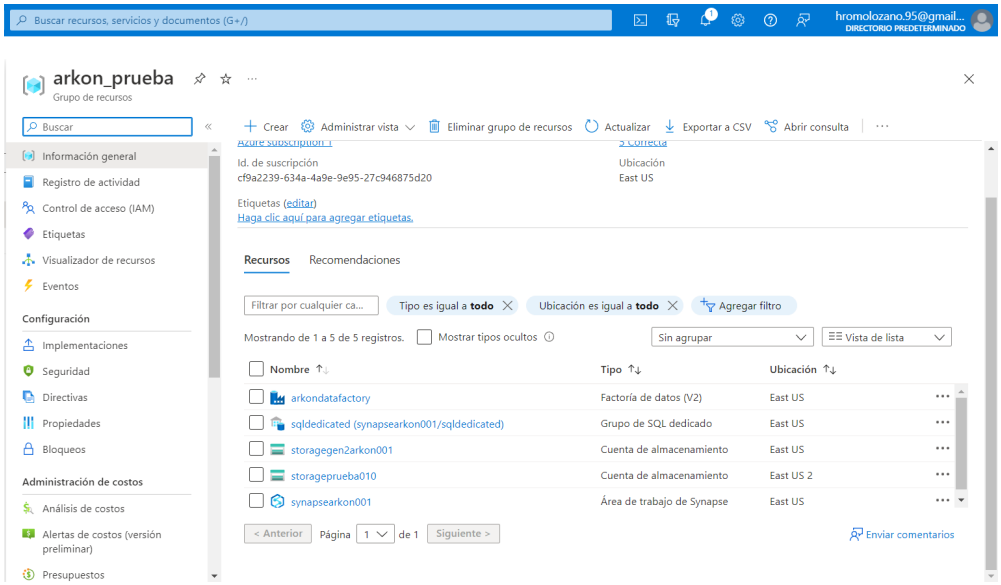
3. Se importó la librería SQLAlchemy para conectarnos desde Python a la base de datos utilizando las llaves determinadas durante la creación de la misma. Un ciclo for lee cada query y crea la tabla en la conexión a PostgreSQL. Si la tabla existe se imprime un mensaje de que la tabla ya existe.
4. Una vez creadas las tablas se inserta la información de cada archivo csv (creados en el paso 1) mediante otro ciclo for. En este caso cada ejecución de la inserción de datos reemplaza a la información existente en la tabla.

***Consideraciones para futuras correcciones y implementaciones de este código:

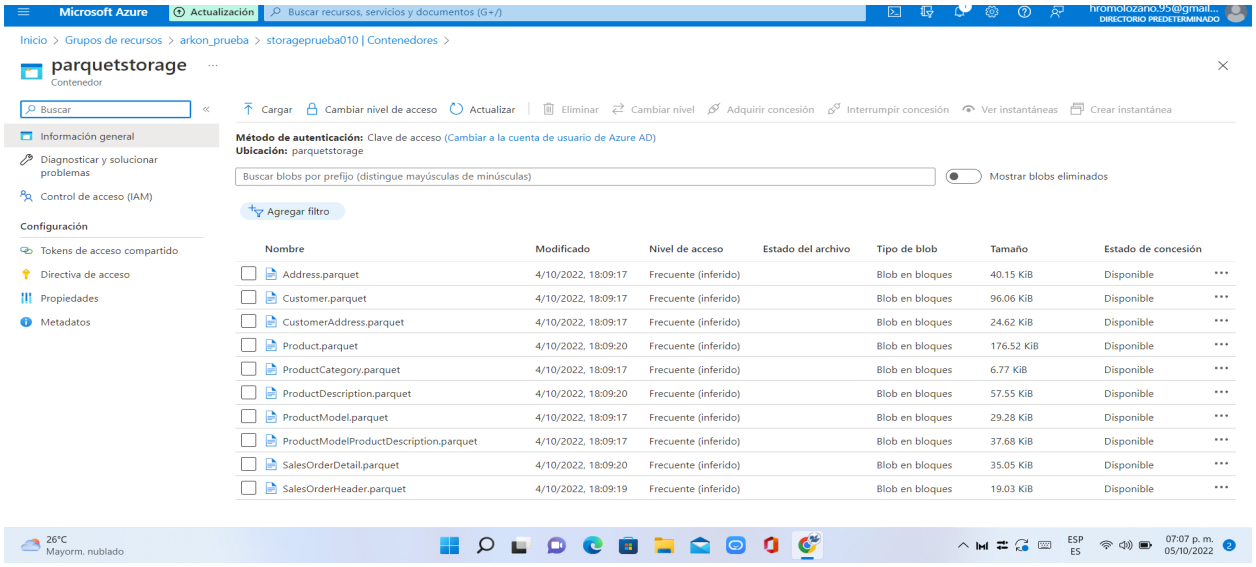
- Que el código lea desde el local path los archivos con extensión .parquet automáticamente y no mediante la determinación manual de los nombres de los archivos. El código debe ser ejecutado automáticamente cada que los archivos sean actualizados o sean agregados nuevos archivos parquet.
- Que el código también asigne el nombre de la tabla, tomando automáticamente el prefijo del archivo o los archivos leídos con la porción de código anterior.
- Que el DDL sea creado automáticamente con la lectura del csv, donde el usuario deberá pasar argumentos como las primary keys de la tabla.
- La inserción de datos debe considerar si la tabla es incremental o full, es decir, que el código permita insertar solo la información de la tabla faltante por medio de una comparación entre lo que ya está en PostgreSQL y lo que se actualizó en el nuevo archivo parquet. En el caso de este código supone una inserción full (borra en cada ejecución del código la información e inserta todo de nuevo).

4. Carga de archivos Parquet a Blob Storage

Después de haber creado la base de datos se pensó en una solución dentro de los servicios de la nube de Azure. Se creó una cuenta gratuita y se crearon los recursos necesarios dentro del Azure Portal. El grupo de recursos se nombró “arkon_prueba” y contiene los servicios de un Data Factory, un Blob storage y un Synapse Analytics que incluye un storage. Además se creó un SQL Pool dedicado para recibir la información ingestada.



Para ingestar los archivos de formato parquet se creó un contenedor dentro del servicio de Blob storage donde se cargó manualmente cada archivo.

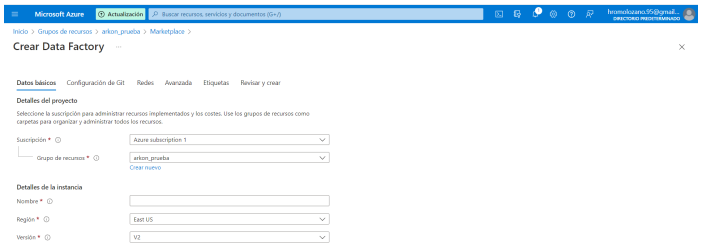


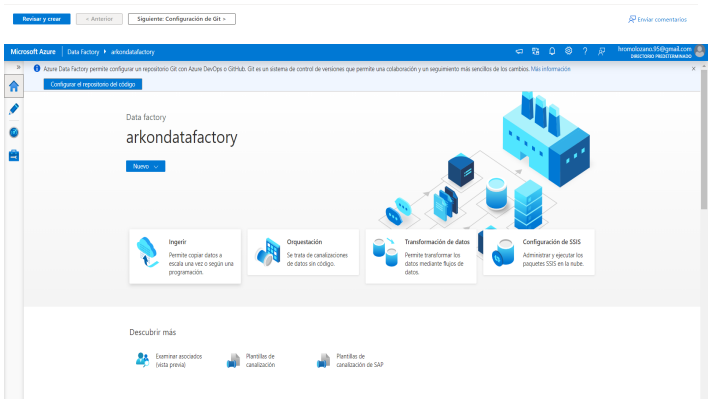
****Consideraciones para futuras implementaciones de esta parte de la solución:

- Que el código Python pueda conectarse a la nube y que cargue el archivo parquet a este contenedor cada que los archivos son actualizados o son agregados al local path. Si esto se realiza es posible que se pueda eliminar la base de datos PostgreSQL y la ingesta a Synapse sea desde Blob storage.

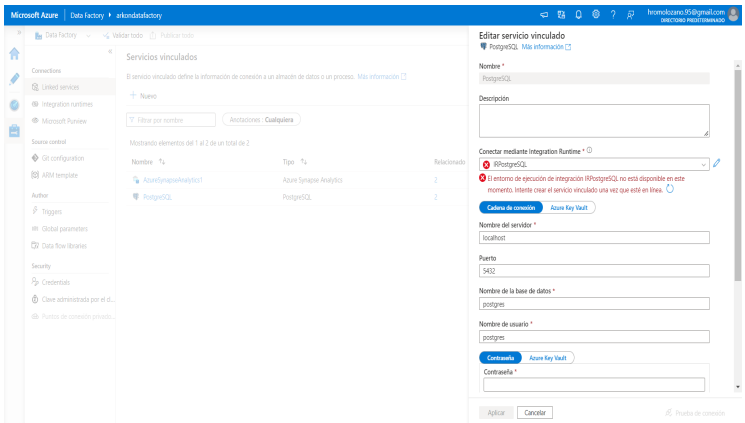
5. Pipeline Azure Data Factory a Synapse Analytics

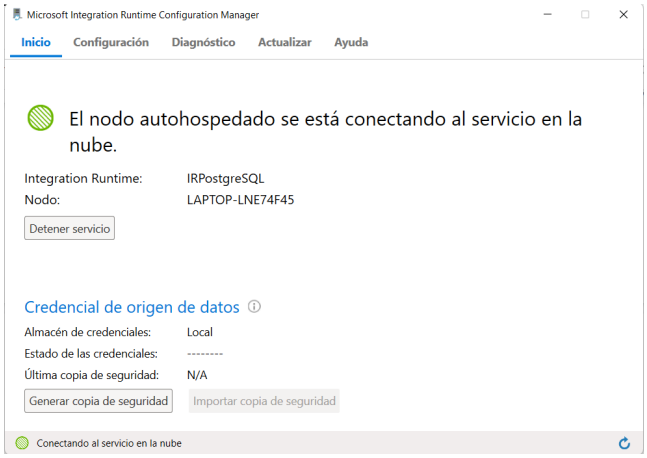
Se creó el servicio de Data Factory y posteriormente se crearon los linked services que permitirán la conexión a la fuente de datos y a la instancia final donde serán ingestados.



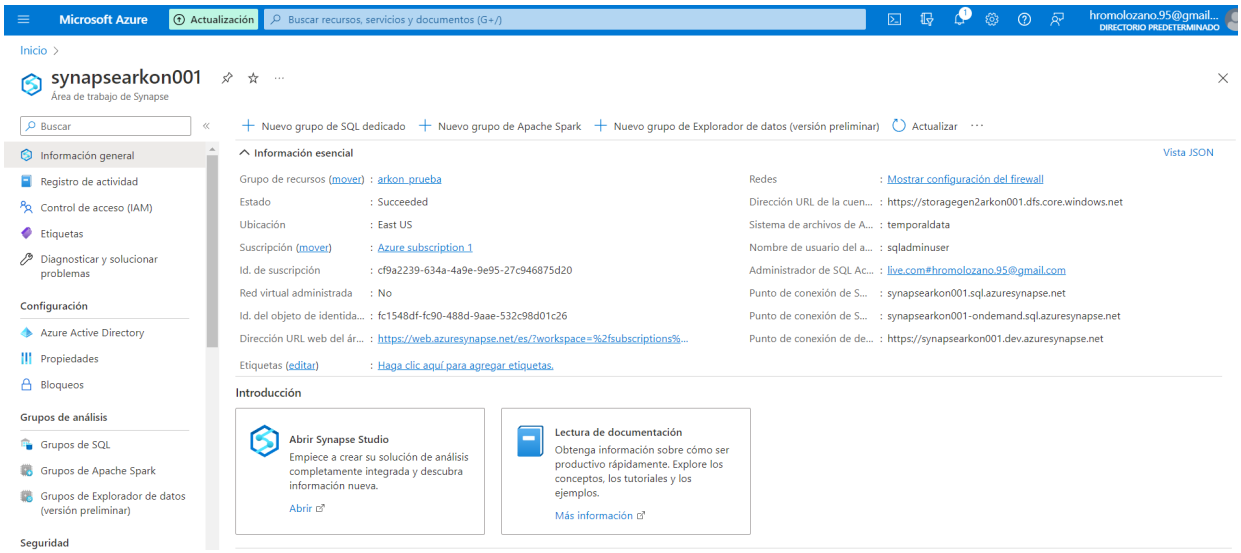


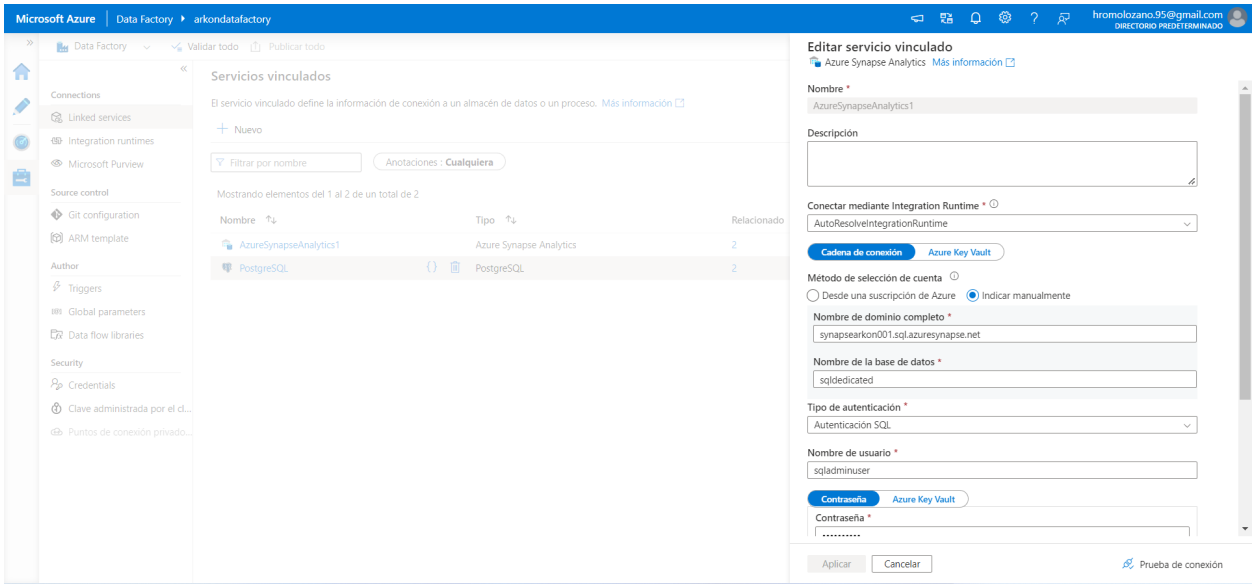
La primera conexión fue poder enlazar el servidor de PostgreSQL (en mi computadora local) con la nube. Esto se realizó con la aplicación de Integration Runtime. El procedimiento para esto es crear el linked service primero y después ingresar las keys a la aplicación descargada de Integration Runtime. Esto conecta nuestra base de datos para ser la fuente en el pipeline de Data Factory.



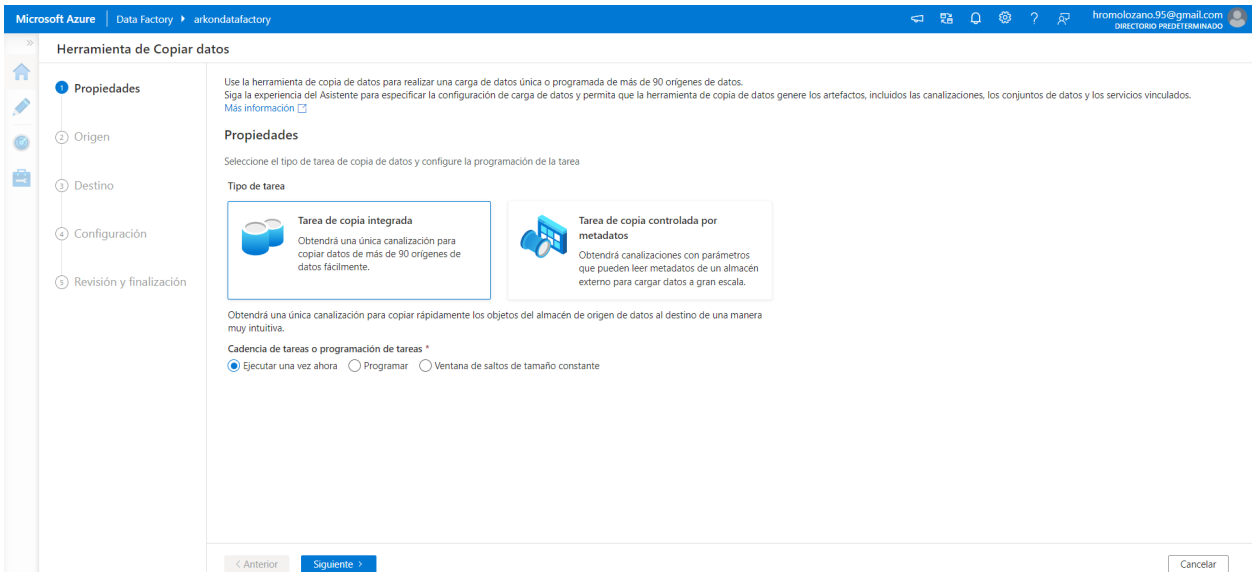


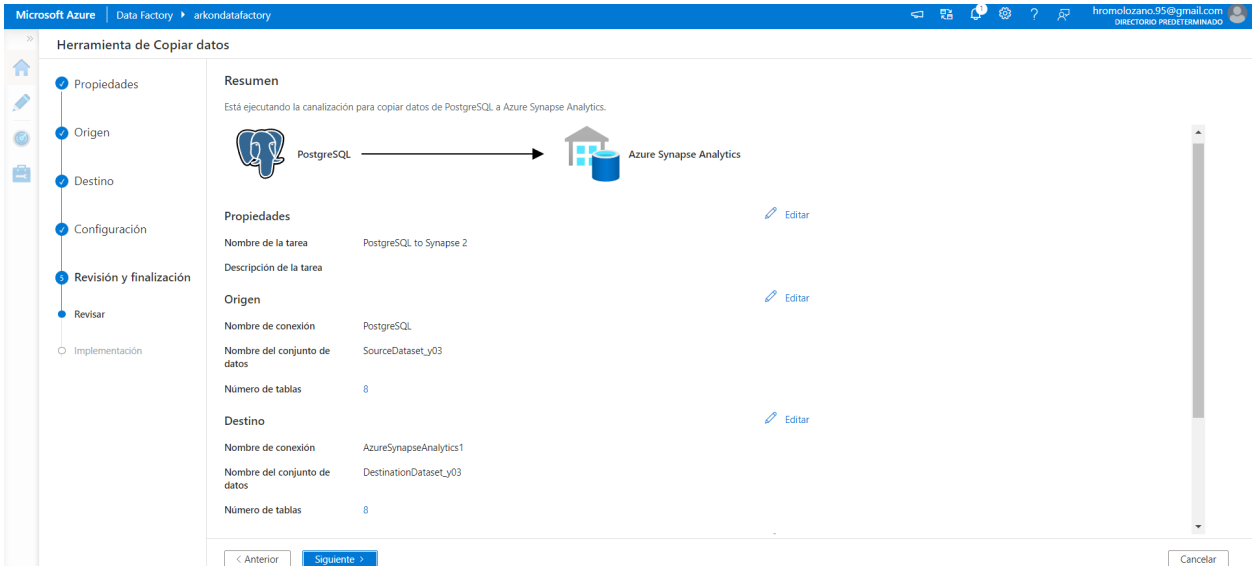
Posteriormente se creó el servició de Synapse Analytics y se creó el linked service para esta instancia. Para el linked service también se creó un SQL pool. Dentro de esta nueva base de datos dedicada se creó el esquema “ARKON”, desde una conexión de DBeaver y no desde los scripts de SQL.



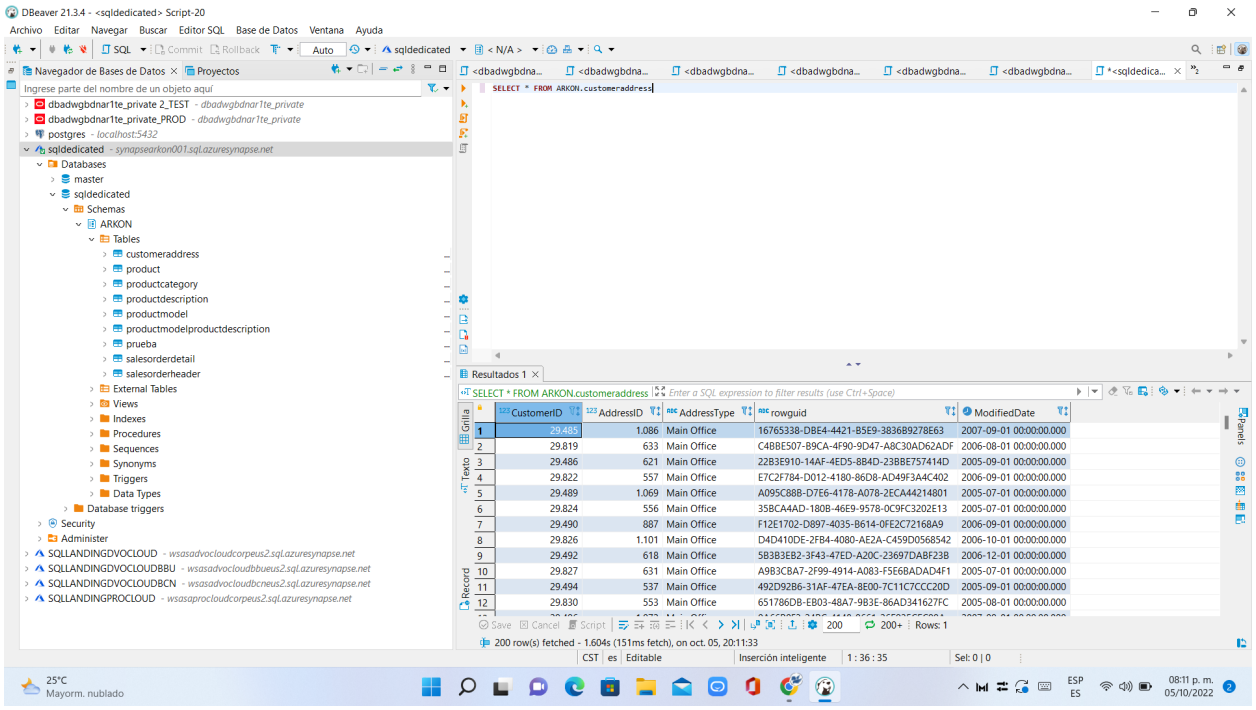


Una vez creada esta infraestructura se tiene acceso a la opción de Ingesta de datos desde el Azure Data Factory Studio. Aquí podemos configurar paso a paso la copia de datos de una fuente a nuestra base de datos final. Este proceso nos guía de una manera más práctica y sencilla para elegir nuestros servicios vinculados y la creación de los datasets. Este pipeline se ejecutó de manera inmediata pero puede ser programado para que se haga una vez al día una vez que con el código Python hemos actualizado la información (en un escenario productivo).





Terminando este proceso y la ejecución se validó la información para concluir que el data está ingestado en Synapse.



6. Anexos

Se anexan los queries para la creación de tablas del modelo de datos presentado:

```
CREATE TABLE IF NOT EXISTS public.address
(
AddressID integer PRIMARY KEY,
AddressLine1 varchar(4000),
AddressLine2 varchar(4000),
City varchar(4000),
StateProvince varchar(4000),
CountryRegion varchar(4000),
PostalCode integer,
rowguid varchar(4000),
ModifiedDate date
)

CREATE TABLE public.customer
(
CustomerID integer PRIMARY KEY,
NameStyle boolean,
Title varchar(4000),
FirstName varchar(4000),
MiddleName varchar(4000),
LastName varchar(4000),
Suffix varchar(4000),
CompanyName varchar(4000),
SalesPerson varchar(4000),
EmailAddress varchar(4000),
Phone integer,
PasswordHash varchar(4000),
PasswordSalt varchar(4000),
rowguid varchar(4000),
ModifiedDate date
)


CREATE TABLE public.customeraddress
(
CustomerID integer,
AddressID integer,
AddressType varchar(4000),
rowguid varchar(4000),
ModifiedDate date,
PRIMARY KEY (CustomerID, AddressID)
)
```

```
CREATE TABLE public.product
(
ProductID integer,
Name varchar(4000),
ProductNumber varchar(4000),
Color varchar(4000),
StandardCost float,
ListPrice float,
Size varchar(6),
Weight float,
ProductCategoryID integer,
ProductModelID integer,
SellStartDate date,
SellEndDate date,
DiscontinuedDate date,
ThumbNailPhoto varchar(4000),
ThumbnailPhotoFileName varchar(4000),
rowguid varchar(4000),
ModifiedDate date,
PRIMARY KEY (ProductID, ProductCategoryID, ProductModelID)
)

CREATE TABLE public.productcategory
(
ProductCategoryID integer PRIMARY KEY,
ParentProductCategoryID integer,
Name varchar(4000),
rowguid varchar(4000),
ModifiedDate date
)

CREATE TABLE public.productdescription
(
ProductDescriptionID integer PRIMARY KEY,
Description varchar(4000),
rowguid varchar(4000),
ModifiedDate date
)

CREATE TABLE public.productmodelproductdescription
(
ProductModelID integer,
ProductDescriptionID integer,
Culture varchar(12),
```

```
rowguid varchar(4000),
ModifiedDate date,
PRIMARY KEY (ProductModelID, ProductDescriptionID)
)

CREATE TABLE public.salesorderdetail
(
SalesOrderID integer,
SalesOrderDetailID integer,
OrderQty integer,
ProductID integer,
UnitPrice float,
UnitPriceDiscount float,
LineTotal float,
rowguid varchar(4000),
ModifiedDate date,
PRIMARY KEY (SalesOrderID, ProductID)
)

CREATE TABLE public.salesorderheader
(
SalesOrderID integer,
RevisionNumber integer,
OrderDate date,
DueDate date,
ShipDate date,
Status integer,
OnlineOrderFlag boolean,
SalesOrderNumber varchar(4000),
PurchaseOrderNumber varchar(4000),
AccountNumber varchar(4000),
CustomerID integer,
ShipToAddressID integer,
BillToAddressID integer,
ShipMethod varchar(4000),
CreditCardApprovalCode varchar(4000),
SubTotal float,
TaxAmt float,
Freight float,
TotalDue float,
Comment varchar(4000),
rowguid varchar(4000),
ModifiedDate date,
PRIMARY KEY (CustomerID, SalesOrderID)
```

)

Create a POSTGRESQL

Create Python Code for data loading to database

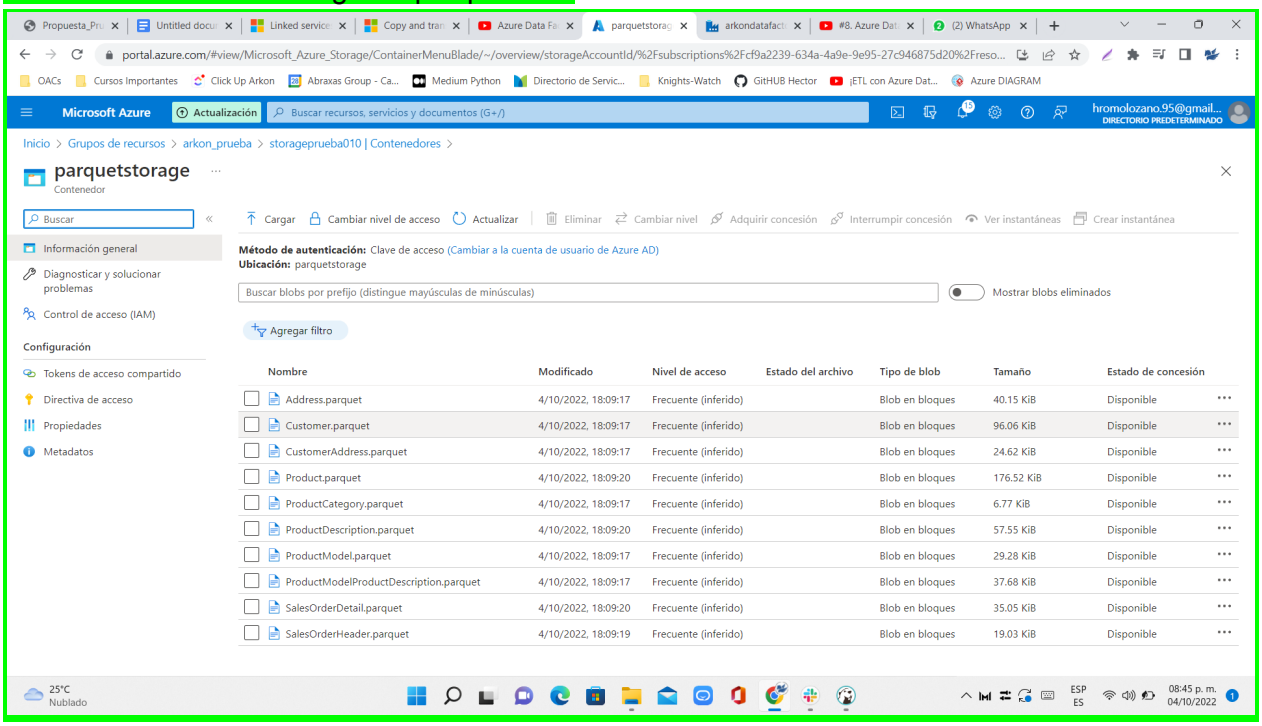
Create an Azure Data Factory Pipeline
Linked Service to Postgresql
Create Blob Storage

Creación de grupo de recursos

Creación de Azure Data Factory

Creación de Synapse Analytics
Lake Storage Gen 2

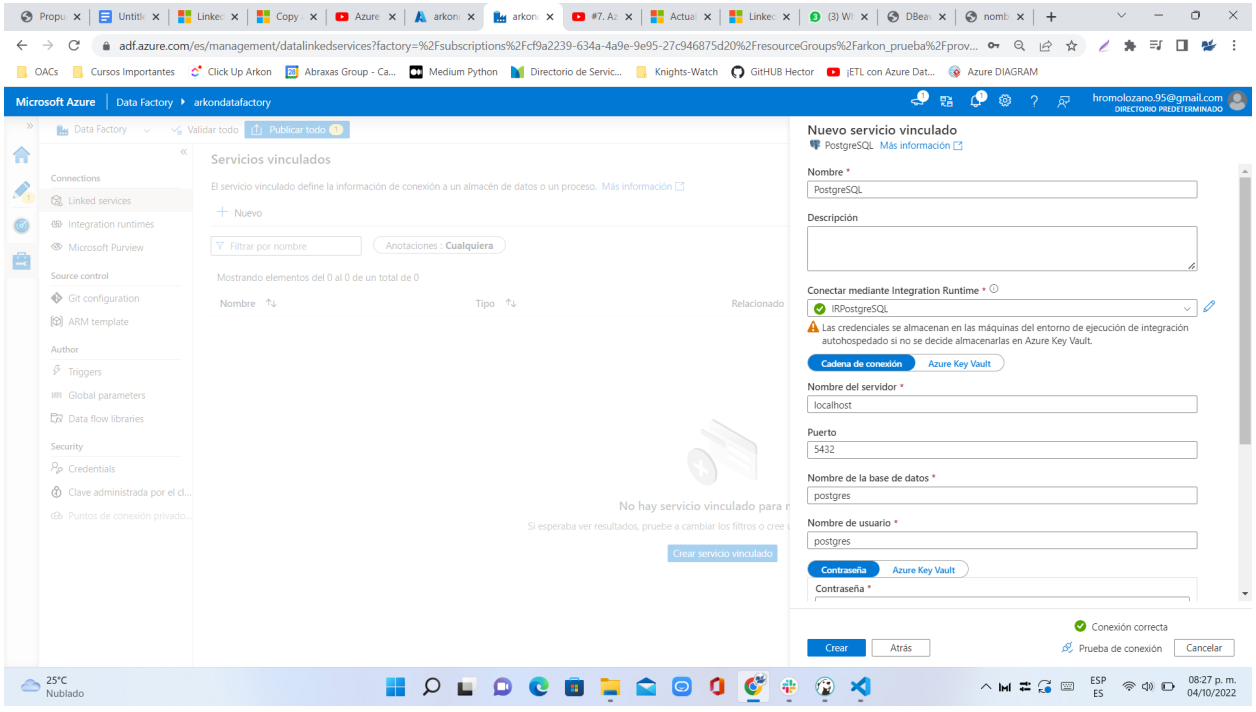
Storage Account
Blob Storage for parquet files

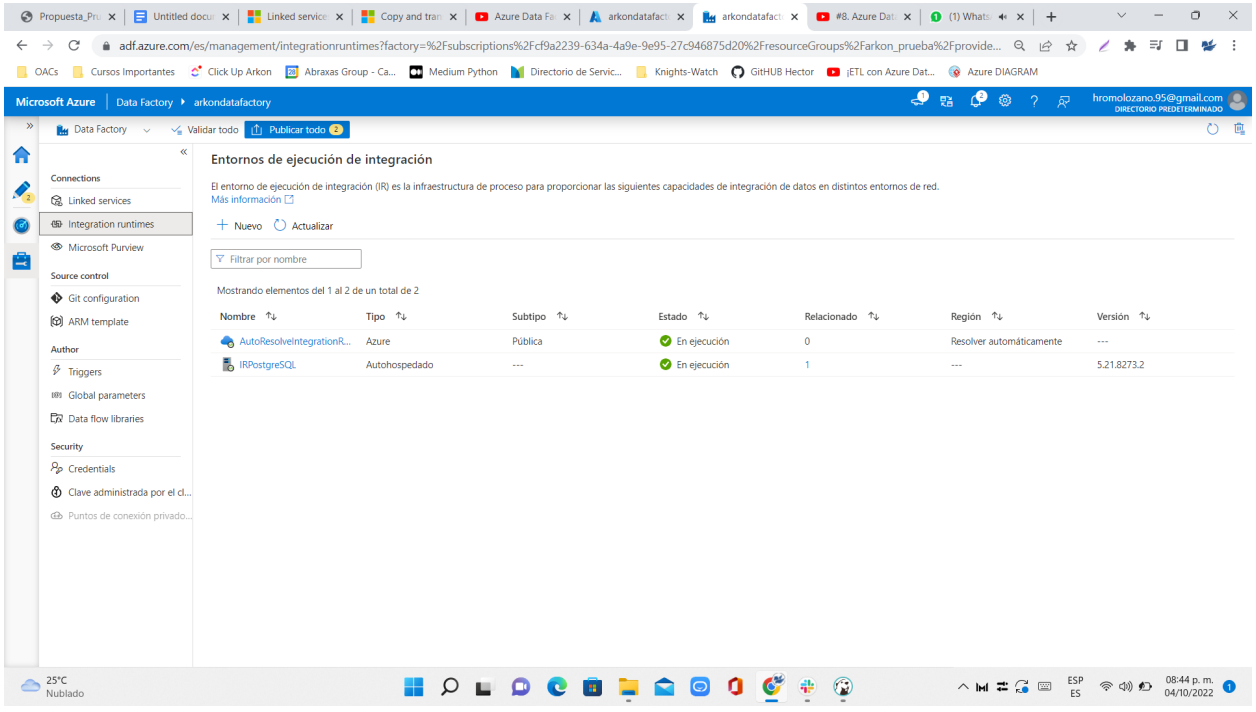


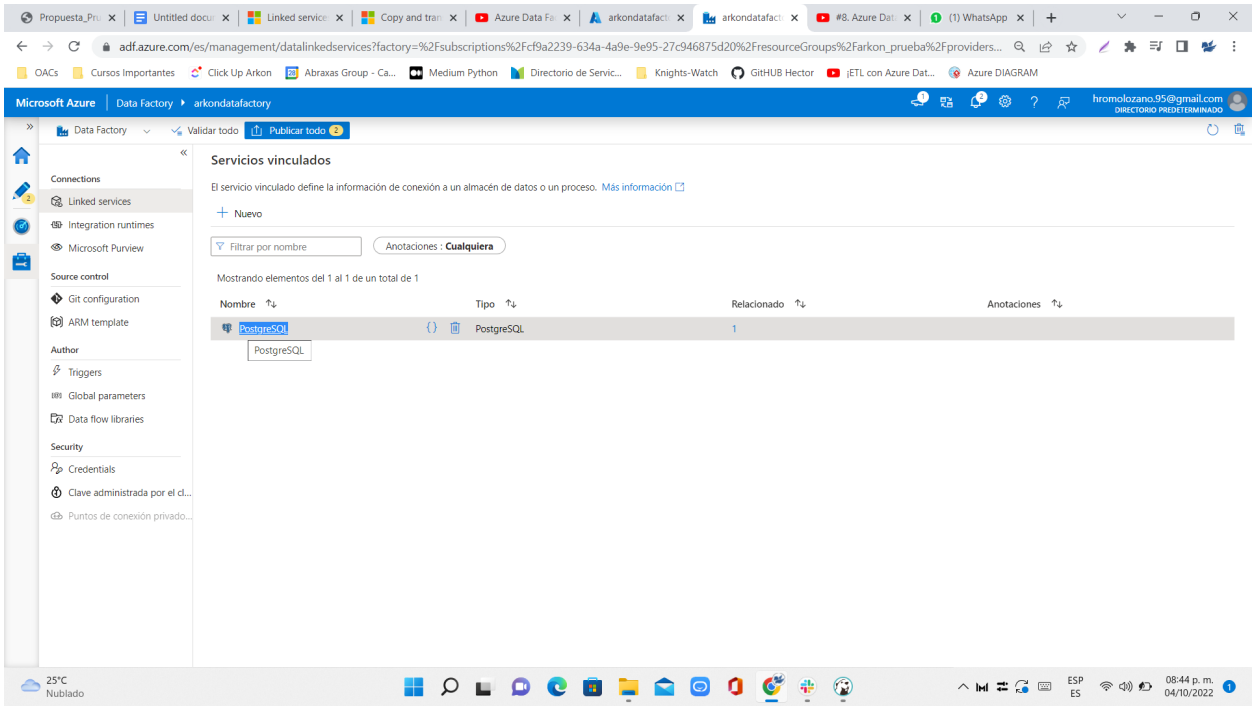
Connect PosgreSQL server to Azure

Integration Runtime Service

Linked Service to Integration Runtime (server and database keys to PostgreSQL)

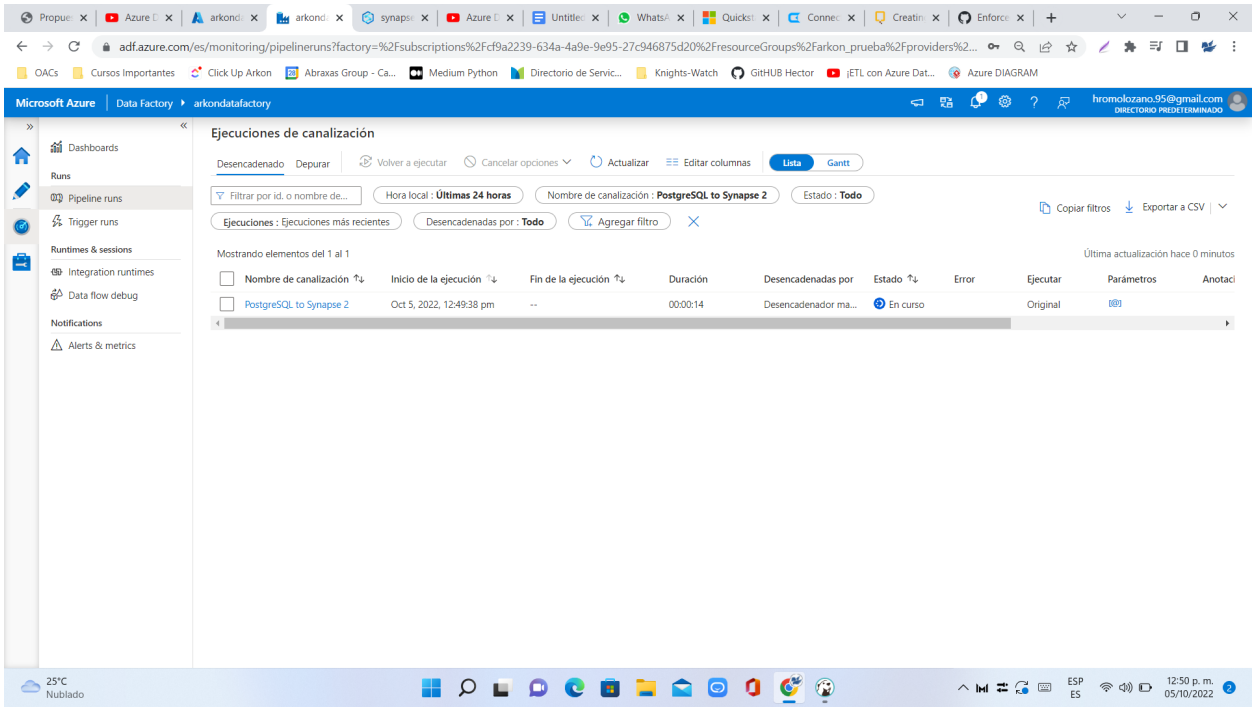


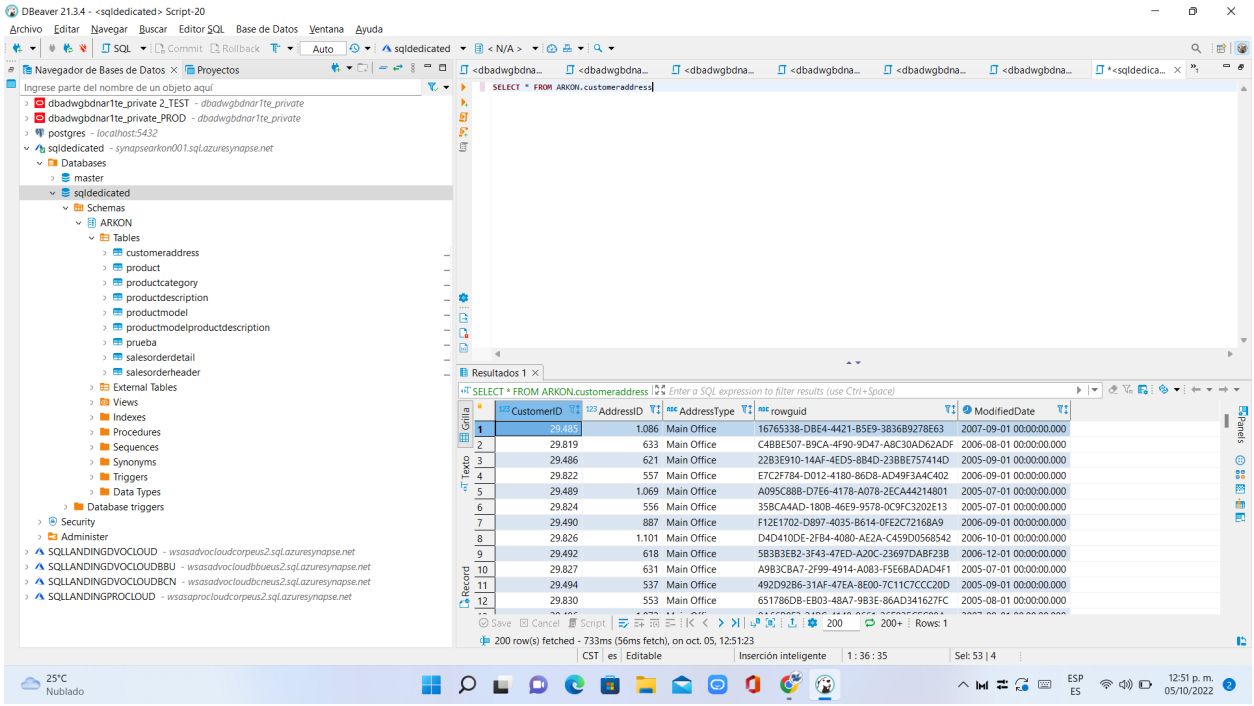




Data Pipeline Creation

Final




DBeaver 21.3.4 - <sqldedicated> Script-20
Archivo Editar Navegar Buscar Editor SQL Base de Datos Ventana Ayuda
Navegador de Bases de Datos
Proyectos
Ingrese parte del nombre de un objeto aquí
dbadwgbdnar1te_private 2_TEST - dbadwgbdnar1te_private
dbadwgbdnar1te_private_PROD - dbadwgbdnar1te_private
postgres - localhost:5432
sqldedicated - synapsearkon001.sql.azuresynapse.net
Databases
master
sqldedicated
Schemas
ARKON
Tables
customeraddress
product
productcategory
productdescription
productmodel
productmodelproductdescription
prueba
salesorderdetail
salesorderheader
External Tables
Views
Indexes
Procedures
Sequences
Synonyms
Triggers
Data Types
Database triggers
Security
Administer
SQLLANDINGDVOCLOUD - wsasadvocloudcorpeus2.sql.azuresynapse.net
SQLLANDINGDVOCLOUDBBU - wsasadvocloudbbueus2.sql.azuresynapse.net
SQLLANDINGDVOCLOUDBCN - wsasadvocloudbcneus2.sql.azuresynapse.net
SQLLANDINGPROCLOUD - wsasaprocloudcorpeus2.sql.azuresynapse.net
SELECT * FROM ARKON.customeraddress
Resultados 1
SELECT * FROM ARKON.customeraddress
Enter a SQL expression to filter results (use Ctrl+Space)
CustomerID AddressID AddressType rowguid ModifiedDate
1 29.485 1.086 Main Office 16765338-DBE4-4421-B5E9-3836B9278E63 2007-09-01 00:00:00.000
2 29.819 633 Main Office C4BBE507-B9CA-4F90-9D47-A8C30AD62ADF 2006-08-01 00:00:00.000
3 29.486 621 Main Office 22B3E910-14AF-4ED5-8B4D-23BBE757414D 2005-09-01 00:00:00.000
4 29.822 557 Main Office E7C2F784-D012-4180-86D8-AD49F3A4C402 2006-09-01 00:00:00.000
5 29.489 1.069 Main Office A095C88B-D7E6-4178-A078-2ECA44214801 2005-07-01 00:00:00.000
6 29.824 556 Main Office 35BCA4AD-180B-46E9-9578-0C9FC3202E13 2005-07-01 00:00:00.000
7 29.490 887 Main Office F12E1702-D897-4035-B614-0FE2C72168A9 2006-09-01 00:00:00.000
8 29.826 1.101 Main Office D4D410DE-2FB4-4080-AE2A-C459D0568542 2006-10-01 00:00:00.000
9 29.492 618 Main Office 5B3B3EB2-3F43-47ED-A20C-23697DABF23B 2006-12-01 00:00:00.000
10 29.827 631 Main Office A9B3CBA7-2F99-4914-A083-F5E6BADAD4F1 2005-07-01 00:00:00.000
11 29.494 537 Main Office 492D92B6-31AF-47EA-8E00-7C11C7CCC20D 2005-09-01 00:00:00.000
12 29.830 553 Main Office 651786DB-EB03-48A7-9B3E-86AD341627FC 2005-08-01 00:00:00.000
Save Cancel Script
200 row(s) fetched - 733ms (56ms fetch), on oct. 05, 12:51:23
CST es Editable Inserción inteligente 1 : 36 : 35 Sel: 53 | 4
25°C Nublado
ESP ES
12:51 p. m. 05/10/2022