MAESTRÍA EN CIENCIA DE LOS DATOS Y ANALÍTICA

TRABAJO 1, UNIDAD 1 INGENIERÍA DE DATOS

JOSE JORGE MUÑOZ MERCADO

LINA MARÍA BELTRÁN DURANGO

ST1800 ALMACENAMIENTO Y RECUPERACIÓN DE LA INFORMACIÓN

4/06/2022

RESUMEN

En este trabajo hemos diseñado e implementado un ecosistema de almacenamiento y procesamiento de datos a partir de dos bases de datos: CAMBIO CLIMÁTICO o CALENTAMIENTO GLOBAL con datos de varios países obtenidos de Kaggle, y una Base de Datos Relacional con 8 tablas simulando los datos de una empresa ficticia a partir de código SQL. Para esto se diseñó e implementó un Datalake para almacenar los datos en S3, se catalogaron (con Glue), se implementaron ETL para cada Base de Datos, se hicieron consultas con SQL (Athena y Hive), se cargaron datos desde S3 a Jupyter Notebook y se hizo un Análisis Exploratorio de Datos con Spark (Jupyter/pyspark) para los datos relacionados con Cambio Climático.

Se concluye que con el diseño en implementación del DataLake con las zonas de Raw, Trusted y Refined se pudo almacenar y procesar datos de diferentes orígenes, tipos y estructuras, también se pudo comprobar la visión integrada de Lakehouse al poder consultar los datos de manera indistinta desde Athena, Hive, SparkSQL.

CASOS DE ESTUDIO

• Cambio climático

Descripción del caso**:** El sitio web de cambio de temperatura de FAOSTAT difunde estadísticas del cambio de temperatura superficial promedio por país, con actualizaciones anuales. La data actual cubre el período entre 1961 y 2019. Los datos se basan en los datos GISTEMP disponibles públicamente y los datos de cambio de temperatura de la superficie global distribuidos por el Instituto Goddard de Estudios Espaciales de la Administración Nacional de Aeronáutica y del Espacio (NASA-GISS).

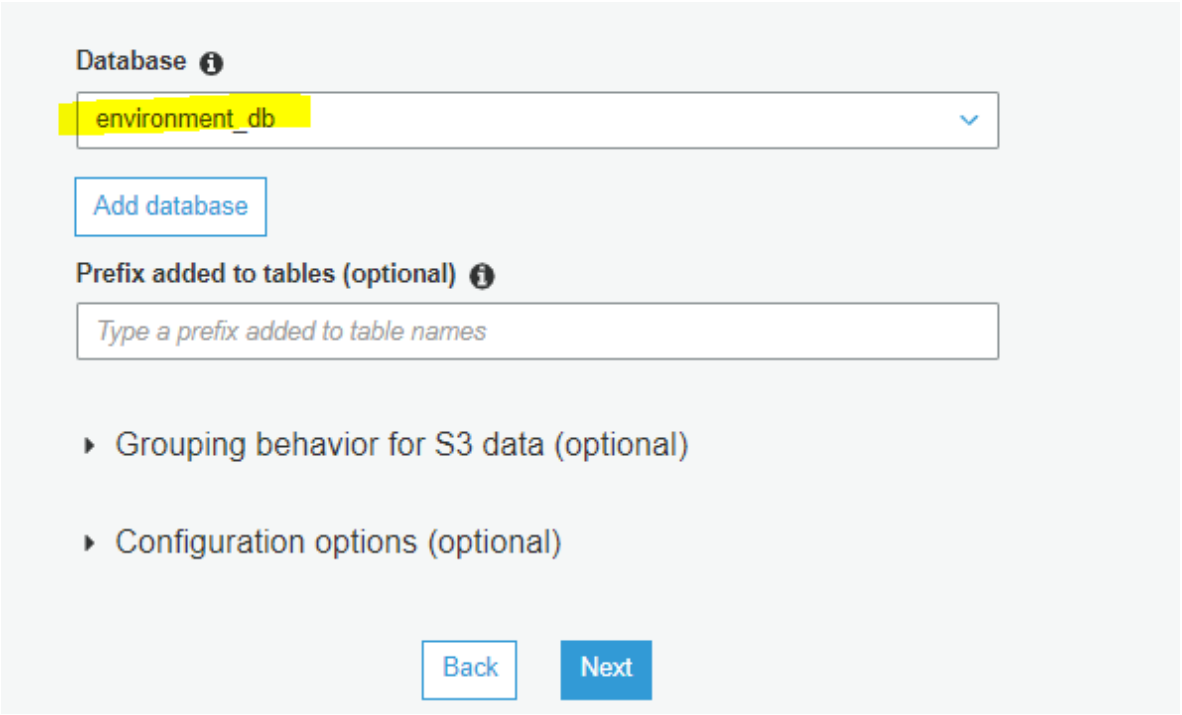
Esta base de datos está conformada por las siguientes tablas almacenadas en S3 de la zona Raw:

- Environment_data.csv
- FAOSTAT_2020.csv
- FAOSTAT_2022.csv.

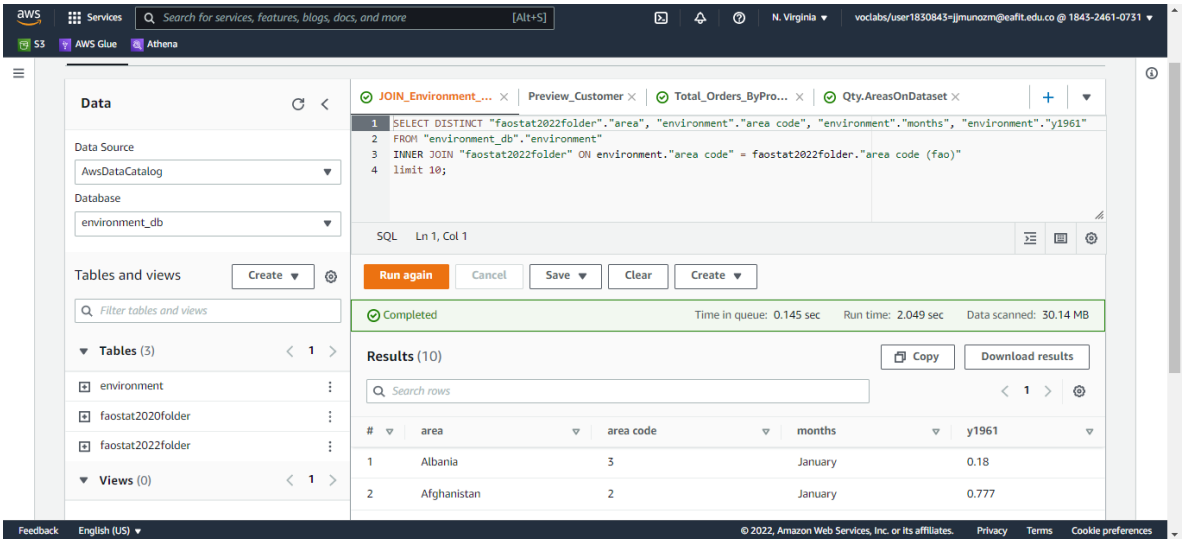
Debido a que estas tres tablas no mantienen el mismo formato, son guardadas en carpetas individuales para cada una.

En AWS Glue se catalogan estas tablas con crawlers individuales para cada una, pero depositadas en una misma base de datos llamada “environment_db”:

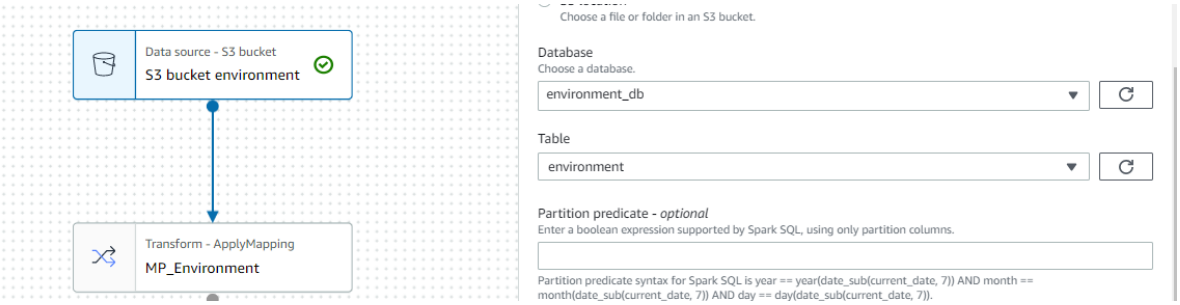
☐	FaoStat2020_crawler	Ready	Logs	1 min	1 min	1	0
☐	FaoStat2022_crawler	Ready	Logs	48 secs	48 secs	0	1
☐	customer_crawler	Ready	Logs	5 mins	5 mins	0	1
☐	employees_crawler	Ready	Logs	40 secs	40 secs	0	1
☐	environment_crawler	Ready	Logs	48 secs	48 secs	0	1



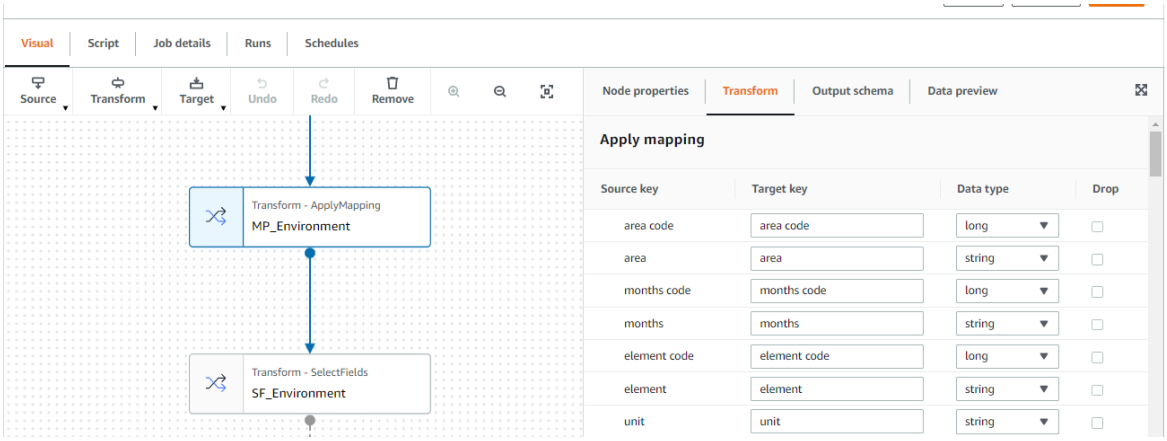
Posteriormente a esto se pueden hacer consultas en Athena o alguna transformación sencilla, como algún JOIN debido a que estas tablas mantienen Keys:

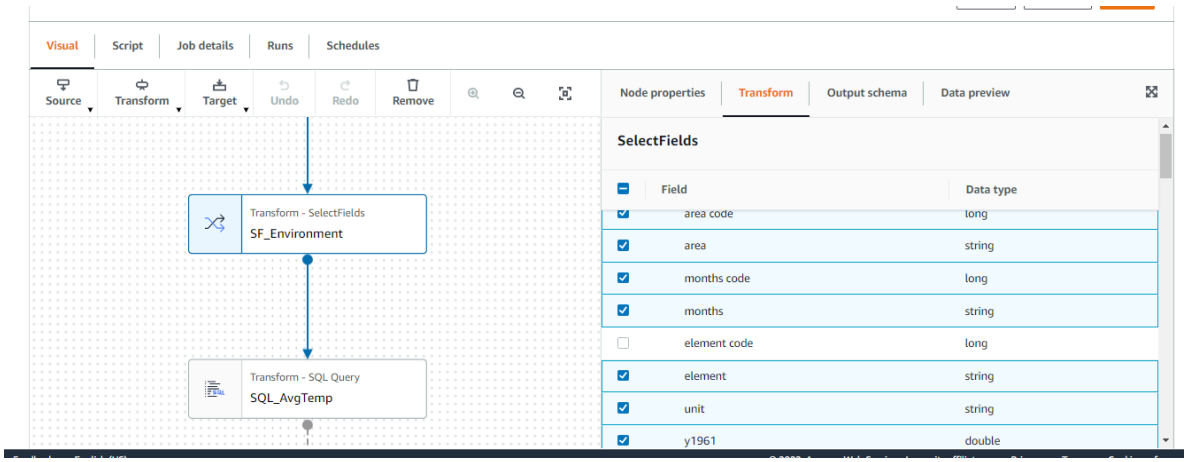


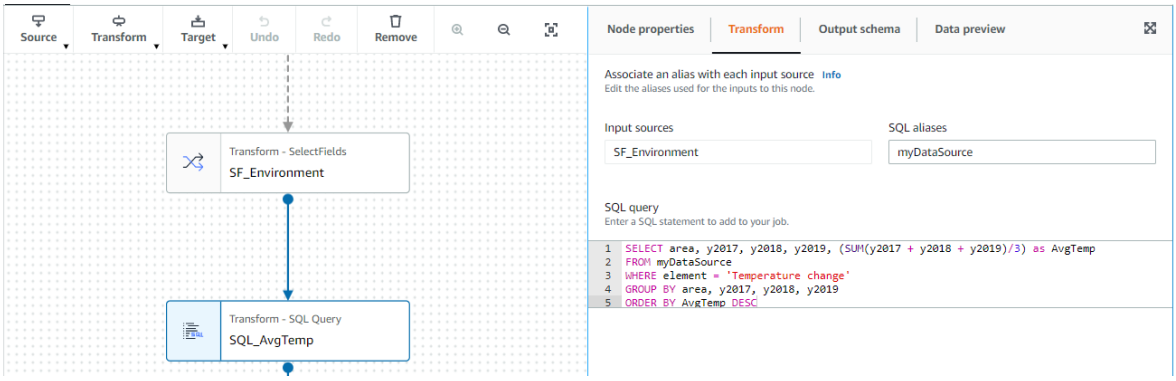
En AWS Glue Studio se realizó un ETL sencillo con la tabla “environment” ya catalogada anteriormente:

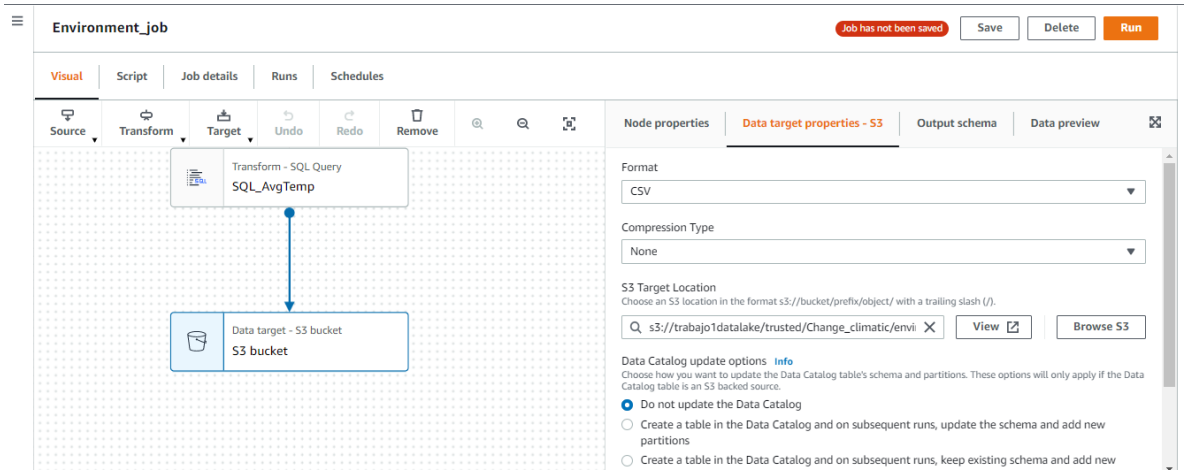


Este ETL consta de 4 nodos, en el primer nodo se cargan los datos desde Glue, en el segundo se hace una transformación en las columnas, aquí se cambiaron algunos tipos de datos ya que en el cuarto nodo presentaba errores porque algunos campos los reconocía como “string”, en el tercer nodo se escogieron solamente unas columnas de interés, en el cuarto nodo se ejecutó un código SQL con un query sencillo y en el quinto nodo se guardo este resultado a la zona Trusted de S3 perteneciente a la carpeta de cambio climático:

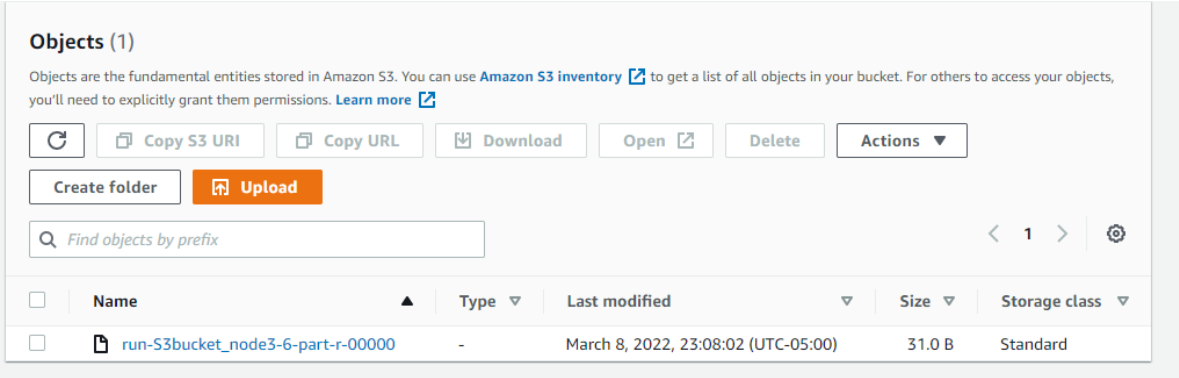




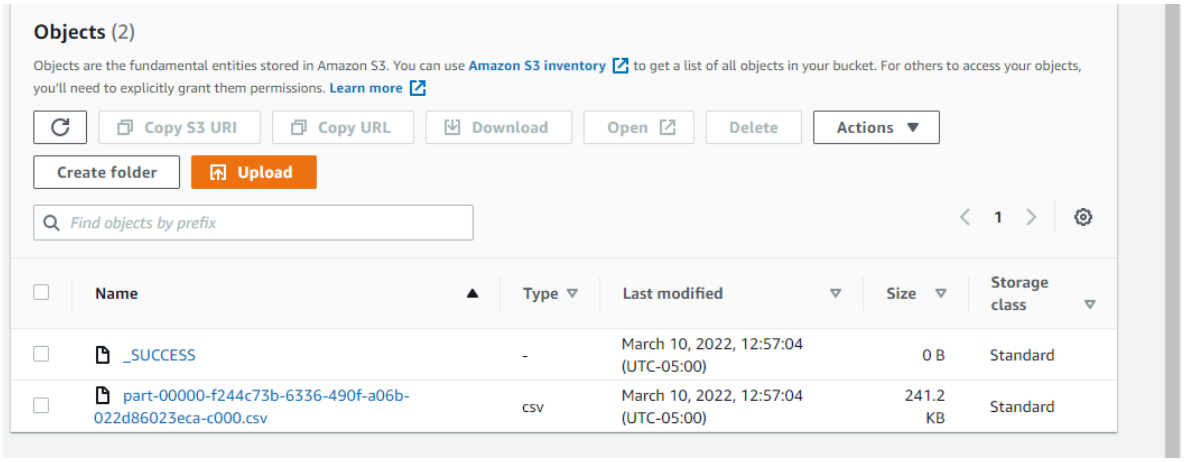




Esta salida se muestra guardada en la ubicación dada de la zona trusted:



Paralelamente en Jupyter PySpark podemos cargar los datos de environment, se hace un análisis exploratorio de datos y se puede guardar en la zona Refined:



A continuación, se muestra el código:

```
In [1]: spark
        sc
```

```
In [2]: environment=spark.read.csv('s3://trabajo1datalake/raw/Change_climatic/environment/',inferSchema=False,header=True)
        FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

```
In [3]: environment.describe()
```

```
In [4]: environment.printSchema()
```

```
In [4]: filter_element=environment.filter(environment['element code']=='7271')
        FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

```
In [44]: (filter_element).describe().show(5)
```

```
In [5]: New_environment=filter_element.select('area','y2017','y2018','y2019')
        FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

```
In [54]: New_environment.printSchema()
```

```
In [49]: New_environment.show(5)
```

```
In [6]: New_environment.select('area').distinct().show()
```

```
In [56]: print("The Variable",'area', "has: ", environment.filter(environment['area'].isNull()).count(), "Null Values")
         print("The Variable",'y2017', "has: ", environment.filter(environment['y2017'].isNull()).count(), "Null Values")
         print("The Variable",'y2018', "has: ", environment.filter(environment['y2018'].isNull()).count(), "Null Values")
         print("The Variable",'y2019', "has: ", environment.filter(environment['y2019'].isNull()).count(), "Null Values")
```

Shape

```
In [50]: print((New_environment.count(),len(New_environment.columns)))
```

```
In [44]: (environment.select('y2017','y2018','y2019')).describe().show()
```

```
In [7]: environment_avg=New_environment.withColumn('Avg3LastYears', (New_environment.y2017+New_environment.y2018+New_environment.y2019)/
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

```
In [56]: environment_avg.show(5)
```

```
In [8]: environment_avg.sort(environment_avg.Avg3LastYears.desc()).show(5)
```

```
In [9]: finalDf=environment_avg
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

```
In [10]: finalDf.coalesce(1).write.format("csv").option("header","true").save("s3://trabajo1datalake/refined/Change_climatic/Change_clima
```

El resultado guardado en S3 de la zona Refined es el siguiente:

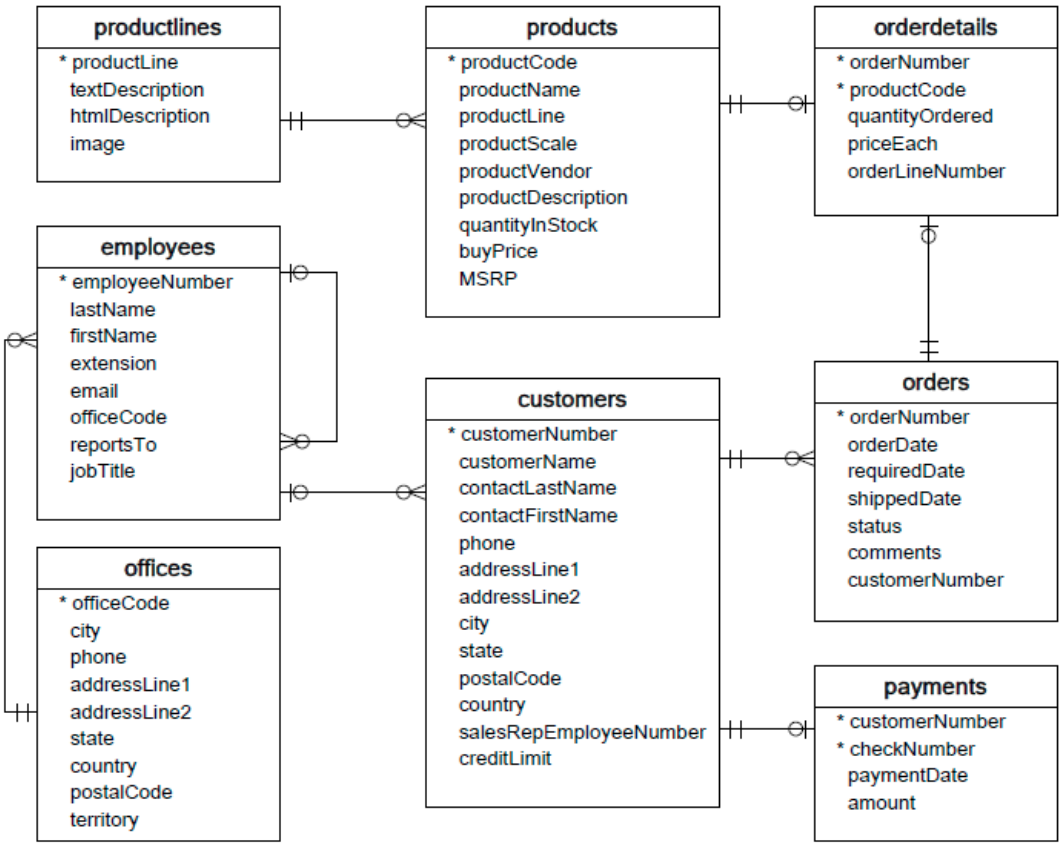
area	y2017	y2018	y2019	Avg3LastYears
Afghanistan	1.201	1.996	2.951	2.049333333
Afghanistan	-0.323	2.705	0.086	0.822666667
Afghanistan	0.834	4.418	0.234	1.828666667
Afghanistan	1.252	1.442	0.899	1.197666667
Afghanistan	3.28	0.855	0.647	1.594
Afghanistan	2.002	1.786	-0.289	1.166333333
Afghanistan	0.901	1.815	1.885	1.533666667
Afghanistan	0.102	0.982	0.773	0.619
Afghanistan	0.93	1.063	2.004	1.332333333
Afghanistan	2.092	-0.103	1.264	1.084333333
Afghanistan	2.089	0.882	-1.051	0.64
Afghanistan	0.441	1.311	1.426	1.059333333
Afghanistan	1.436	1.714	1.449	1.533
Afghanistan	1.789	2.238	0.593	1.54
Afghanistan	1.002	1.528	0.79	1.106666667
Afghanistan	1.704	0.614	0.739	1.019
Afghanistan	1.483	1.524	0.893	1.3
Albania	-2.47	2.304	-0.816	-0.327333333
Albania	2.622	0.765	1.563	1.65
Albania	3.419	1.768	3.119	2.768666667

• Empresa ficticia

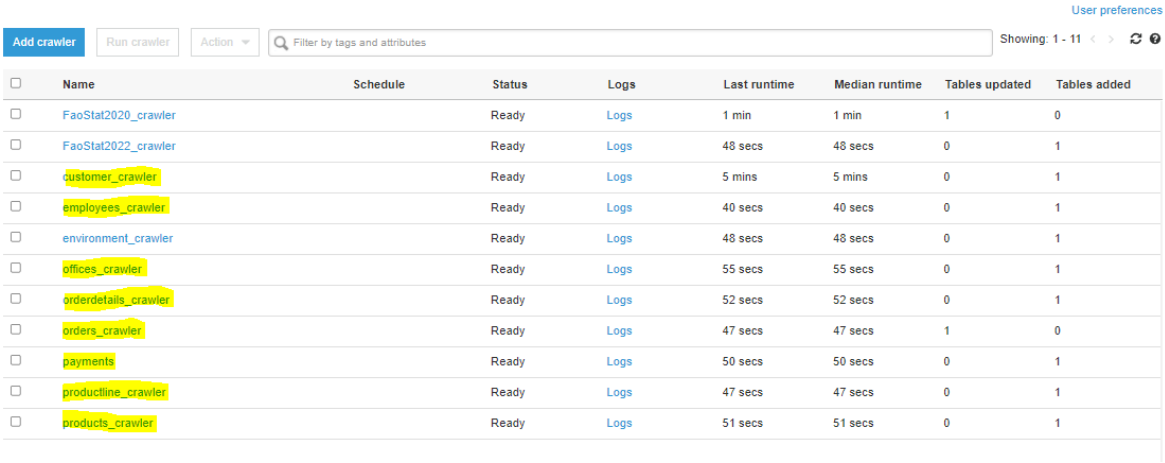
Esta base de datos fue creada con motor SQL. Se trata de una empresa cualquiera, puede ser manufacturera o solo de ventas, la cual maneja un catálogo de n cantidad de productos. Dentro de esta BD se encuentran las siguientes tablas relacionales:

- Productlines, correspondiente a las líneas de productos
- Products, correspondiente a los productos en venta
- Orderdetails correspondiente a los detalles de las ordenes
- Employees la cual detalla la información de los empleados
- Customers correspondiente a la información de los clientes
- Orders correspondiente a la información rasa de las ordenes
- Offices la cual detalla la información de las oficinas
- Payments la cual detalla la información de pago

A continuación, se presenta una imagen de la relación de las tablas:

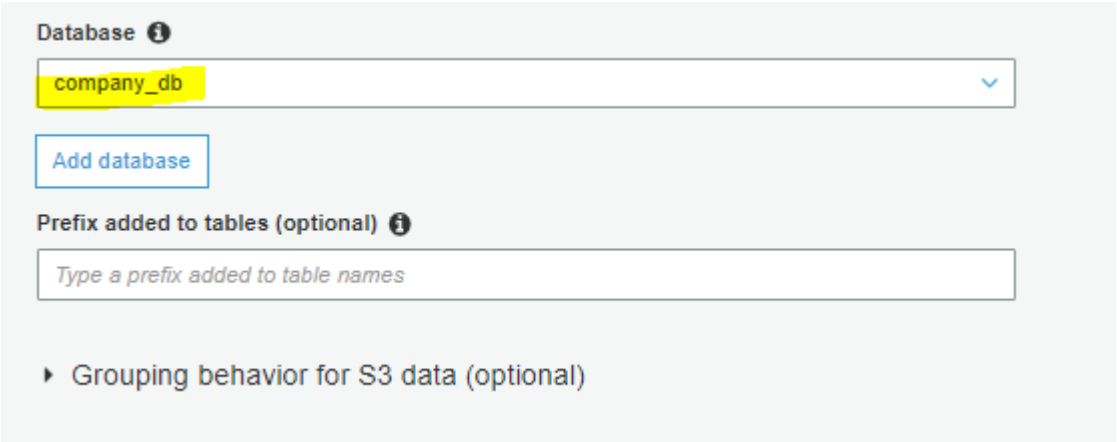


Para este caso se pueden resolver algunas preguntas de negocio utilizando Athena con los datos catalogados en Glue o utilizando Hive. A continuación, se presentan los crawlers para cada tabla:



Name	Schedule	Status	Logs	Last runtime	Median runtime	Tables updated	Tables added
FaoStat2020_crawler		Ready	Logs	1 min	1 min	1	0
FaoStat2022_crawler		Ready	Logs	48 secs	48 secs	0	1
customer_crawler		Ready	Logs	5 mins	5 mins	0	1
employees_crawler		Ready	Logs	40 secs	40 secs	0	1
environment_crawler		Ready	Logs	48 secs	48 secs	0	1
offices_crawler		Ready	Logs	55 secs	55 secs	0	1
orderdetails_crawler		Ready	Logs	52 secs	52 secs	0	1
orders_crawler		Ready	Logs	47 secs	47 secs	1	0
payments		Ready	Logs	50 secs	50 secs	0	1
productline_crawler		Ready	Logs	47 secs	47 secs	0	1
products_crawler		Ready	Logs	51 secs	51 secs	0	1

Estos crawler se almacenan en una sola base de datos llamada “company_db”



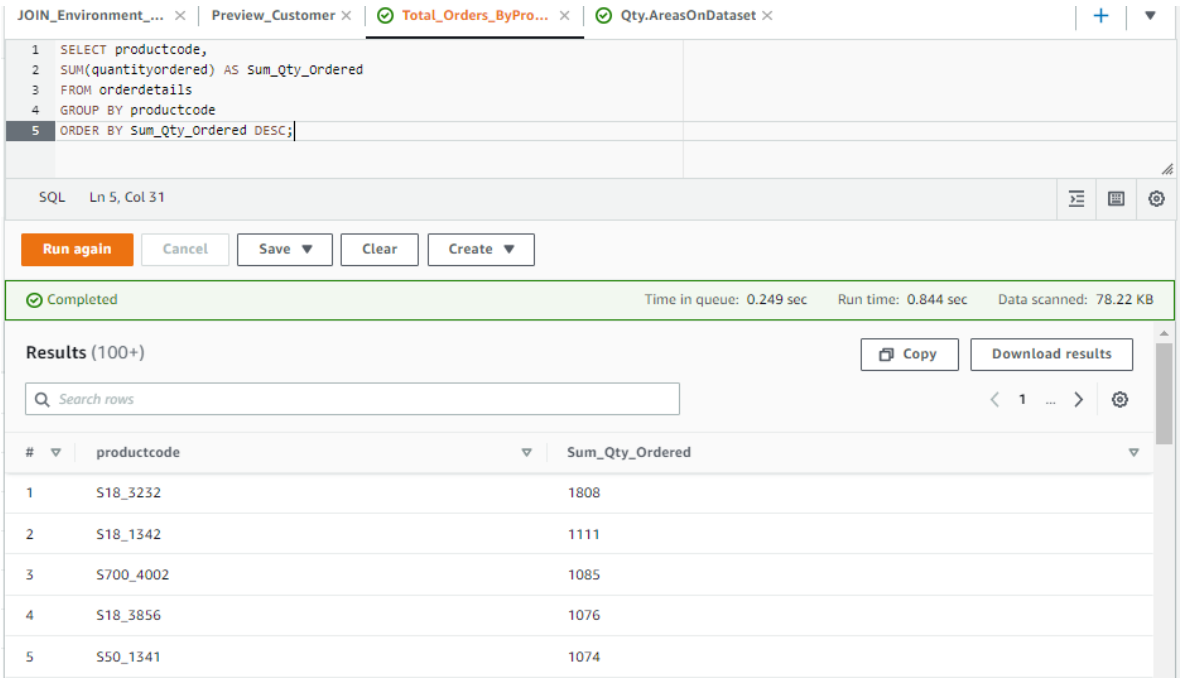
Para esta base de datos se hace un ETL un poco más complejo donde se tienen dos tablas de entrada, es decir, dos nodos con tablas bases las cuales son “products” y “orderdetails”. Al nodo de la tabla “products” se le realiza una transformación de columnas, donde se le cambia el nombre a estas para no ser confundidas posteriormente con la nueva tabla que resulta de un JOIN hecho en un nodo posterior. Luego de esta transformación en las columnas de esta tabla (la transformación se puede realizar con cualquiera de las dos tablas), se realiza un nodo de JOIN con la tabla “orderdetails” con la columna llave “productcode”, donde en la tabla “products” se convierte y es llamada mp_product. En el siguiente nodo se realiza un nodo personalizado donde se ejecuta una función con Spark. En este nodo se realiza un query para responder a la pregunta de qué producto contiene la mayor cantidad de ventas, esto a partir de la columna “quantityordered” de la tabla “orderdetails”.

Antes de escoger el nodo para exportar los resultados a S3, debemos ejecutar un nodo “SelectFromCollection” para convertir una recopilación de DynamicFrames la cual se ejecuta en el nodo personlaizado anterior en un solo DynamicFrame. Por último, se guarda con un nodo S3 OUT en la zona Trusted de S3. A continuación, se presenta el código del nodo Custom Transform y el resultado:

```
def MyTransform (glueContext, dfc) -> DynamicFrameCollection:
    df = dfc.select(list(dfc.keys())[0]).toDF()
    df.createOrReplaceTempView("FinalTable")
    query = spark.sql("SELECT mp_productname AS product_code, SUM(quantityordered) AS Sum_Qty_Ordered FROM FinalTable GROUP BY mp_productname ORDER BY Sum_Qty_Ordered DESC")
    resultQuery = DynamicFrame.fromDF(query, glueContext, "sum_quantity")
    return (DynamicFrameCollection({"CustomTransformQuantity": resultQuery}, glueContext))
```

product_code	sum_qty_ordered
1992 Ferrari 360 Spider red	1808
1937 Lincoln Berline	1111
American Airlines: MD-11S	1085
1941 Chevrolet Special Deluxe Cabriolet	1076
1930 Buick Marquette Phaeton	1074
1940s Ford truck	1061
1969 Harley Davidson Ultimate Chopper	1057

En Athena Podemos hacer un Query similar, pero sin tener un JOIN. Para este caso lo podemos responder la misma pregunta, pero para el código de producto (“productcode”):



#	productcode	Sum_Qty_Ordered
1	S18_3232	1808
2	S18_1342	1111
3	S700_4002	1085
4	S18_3856	1076
5	S50_1341	1074