

Azure Data Engineer Associate Certification Guide

A hands-on reference guide to developing your data engineering skills and preparing for the DP-203 exam

Newton Alex



Capítulo 8: Ingesta y transformación de datos	4
8.1. Requisitos técnicos	4
8.2. Transformación de datos mediante Apache Spark	5
8.2.1. ¿Qué son los RDDs?	5
Creación de RDDs	5
Transformaciones de RDDs	5
8.2.2. ¿Qué son los DataFrames?	6
Creación de DataFrames	6
Transformaciones DataFrame	7
8.3. Transformación de datos mediante T-SQL	9
8.4. Transformación de datos mediante ADF	11
8.4.1. Transformaciones de esquema	13
8.4.2. Transformaciones de filas	15
8.4.3. Transformaciones multi-I/O	16
8.4.4. ADF templates	18
8.5. Transformación de datos mediante Azure Synapse pipelines	21
8.6. Transformación de datos mediante Stream Analytics	22
8.7. Limpieza de datos	23
8.7.1. Tratamiento de los valores ausentes/nulos	23
Sustitución con valores por defecto	23
Filtrar valores nulos	23
8.7.2. Trimming inputs (Recorte de entradas)	24
8.7.3. Estandarización de valores	24
8.7.4. Manejo de valores atípicos	25
8.7.5. Eliminación de duplicados/agregación	25
8.8. Dividir datos	27
8.8.1. Divisiones de archivos	27
8.9. Shredding JSON	29
8.9.1. Extracción de valores de JSON con Spark	29
8.9.2. Extracción de valores de JSON utilizando SQL	30
8.9.3. Extracción de valores de JSON utilizando ADF	31
8.10. Codificación y decodificación de datos	33
8.10.1. Codificación y decodificación usando SQL	33

8.10.2. Codificación y decodificación en Spark	33
8.10.3. Codificación y decodificación con ADF.....	34
8.11. Configurar el manejo de errores para la transformación	35
8.12. Normalización y desnormalización de valores.....	37
8.12.1. Desnormalización de valores con Pivot.....	37
8.12.2. Normalización de valores con Unpivot	39
8.13. Transformación de datos utilizando Scala	41
8.14. Realizando Análisis Exploratorio de Datos (EDA).....	42
8.14.1. Exploración de datos mediante Spark.....	42
8.14.2. Exploración de datos mediante SQL	43
8.14.3. Exploración de datos con ADF.....	44
Resumen.....	45

Capítulo 8: Ingesta y transformación de datos

Bienvenido a la siguiente sección principal del libro. En esta sección, nos centraremos en el diseño y desarrollo de sistemas de procesamiento de datos.

En el último capítulo, aprendimos sobre la implementación de la capa de servicio y vimos cómo compartir datos entre servicios como Synapse SQL y Spark usando metastores. En este capítulo, nos centraremos en la transformación de datos, es decir, en el proceso de transformación de los datos desde su formato sin procesar a un formato más útil que pueda ser utilizado por herramientas y proyectos posteriores. Una vez que haya completado este capítulo, podrá leer datos utilizando diferentes formatos de archivo y codificaciones, realizar la limpieza de datos y ejecutar transformaciones utilizando servicios como Spark, SQL y Azure Data Factory (ADF).

En este capítulo cubriremos los siguientes temas:

- Transformación de datos mediante el uso de Apache Spark
- Transformación de datos mediante Transact-SQL (T-SQL)
- Transformación de datos mediante ADF
- Transformación de datos mediante el uso de pipelines Azure Synapse
- Transformación de datos mediante el uso de Stream Analytics
- Limpieza de datos
- Dividir datos
- Shredding JavaScript Object Notation (JSON)
- Codificación y decodificación de datos
- Configuración de la gestión de errores para la transformación
- Normalización y desnormalización de valores
- Transformación de datos utilizando Scala
- Realización de análisis exploratorios de datos (EDA)

8.1. Requisitos técnicos

Para este capítulo, necesitarás lo siguiente

- Una cuenta Azure (gratuita o de pago)
- Un área de trabajo de Synapse activa
- Un área de trabajo ADF activa

Comencemos.

8.2. Transformación de datos mediante Apache Spark

Apache Spark admite transformaciones con tres interfaces de programación de aplicaciones (API) diferentes: **Resilient Distributed Datasets (RDDs)**, **DataFrames** y **Datasets**. En este capítulo aprenderemos sobre las transformaciones RDDs y DataFrame. Los Datasets son sólo extensiones de los DataFrames, con características adicionales como ser a prueba de tipos (donde el compilador comprobará estrictamente los tipos de datos) y proporcionar una interfaz orientada a objetos (OO).

La información de esta sección se aplica a todas las versiones de Spark disponibles en Azure: Synapse Spark, Azure Databricks Spark y HDInsight Spark.

8.2.1. ¿Qué son los RDDs?

Los RDDs son una colección inmutable y tolerante a fallos de objetos de datos que pueden ser operados en paralelo por Spark. Son las estructuras de datos más fundamentales con las que opera Spark. Los RDDs soportan una amplia variedad de formatos de datos como JSON, valores separados por comas (CSV), Parquet, etc.

Creación de RDDs

Hay muchas maneras de crear un RDD. Aquí hay una forma fácil usando la función `parallelize()`:

```
val cities = Seq("New York", "Austin")
val rdd=spark.sparkContext.parallelize(cities)
```

Una vez que hemos creado un RDD, podemos ejecutar diferentes tipos de transformaciones sobre ellos. Vamos a explorar las más comunes.

Transformaciones de RDDs

Estas son algunas de las transformaciones de RDD más utilizadas en Spark:

map()-Aplica la función proporcionada como parámetro a todos los elementos del origen y devuelve un nuevo RDD. Este es un ejemplo para hacer un recuento de palabras:

```
val maprdd=rdd.map(word => (word,1))
maprdd.collect.foreach(println)
```

flatMap()-Similar a `map()`, pero puede aplicar la función proporcionada a más de un elemento, como se ilustra aquí:

```
val fmrdd = rdd.flatMap(word => word.split(" "))
fmrdd.collect.foreach(println)
```

filter()-Devuelve un nuevo RDD que satisface la condición del filtro, como se ilustra aquí:

```
val emptystrfilterrdd = rdd.filter(_.nonEmpty)
emptystrfilterrdd.collect.foreach(println)
```

groupByKey()-Recoge datos idénticos en grupos y puede realizar acciones agregadas sobre ellos, como se ilustra aquí:

```
val groupbyrdd = rdd.groupBy(word => word.charAt(0))
groupbyrdd.collect.foreach(println)
```

union()-Devuelve un nuevo RDD que es la unión de los dos conjuntos de datos, como se ilustra aquí:

```
val rdd1 = spark.sparkContext.parallelize(List(1, 2, 3))
val rdd2 = spark.sparkContext.parallelize(List(4, 5, 6))
val unionrdd = rdd1.union(rdd2)
unionrdd.collect().foreach(println)
```

distinct()-Devuelve un nuevo conjunto de datos que contiene sólo los elementos únicos del conjunto de datos de entrada, como se ilustra aquí:

```
val distrinctrdd = rdd.distinct()
distrinctrdd.collect.foreach(println)
```

Spark soporta una enorme lista de transformaciones. Si quieres ver la lista completa, consulta la documentación de Apache Spark aquí:

<https://spark.apache.org/docs/latest/rdd-programming-guide.html>.

Veamos ahora los DataFrames.

8.2.2. ¿Qué son los DataFrames?

Los DataFrames son similares a las tablas de las bases de datos relacionales. Son como los RDDs en el sentido de que también son inmutables, redundantes y distribuidos, pero representan una forma superior de abstracción de datos. Los DataFrames contienen esquemas, columnas y filas, al igual que las tablas relacionales, y son útiles para procesar grandes volúmenes de datos utilizando operaciones similares a las de las tablas relacionales.

Creación de DataFrames

Veamos las opciones disponibles para crear DataFrames. Se exponen aquí:

- Convertir un RDD en un DataFrame, de la siguiente manera

```
val df = rdd.toDF()
```

- Crear un DataFrame a partir de un archivo CSV, de la siguiente manera

```
csvDf = spark.read.csv("/path/to/file.csv")
```

- Creación de un DataFrame a partir de un archivo JSON, como sigue:

```
jsonDf = spark.read.json("/path/to/file.json")
```

- Crear un DataFrame con un esquema, como el siguiente:

```
data = [("Adam", "Smith", "Male", "CA"),  
        ("Brenda", "Jones", "Female", "FL")]  
schema = ["firstname", "lastname", "gender", "state"]  
df = spark.createDataFrame(data = data, schema = schema)
```

A continuación, vamos a ver las transformaciones de DataFrame.

Transformaciones DataFrame

Para el análisis de datos, las transformaciones DataFrame son más relevantes que las transformaciones RDD, ya que tratan con abstracciones similares a las de las tablas, lo que facilita el trabajo de los analistas de datos.

NOTA

Una transformación DataFrame se convierte finalmente en una transformación RDD dentro de Spark:

Veamos algunas transformaciones DataFrame importantes. Supongamos que df, df1 y df2 son DataFrames válidos.

- **select()**-Para seleccionar los datos de un subconjunto de columnas, de la siguiente manera

```
df.select("firstname", "lastname").show()
```

- **filter()**-Para filtrar las filas en función de una condición, de la siguiente manera

```
df.filter('location === "Florida").show()
```

- **distinct()**-Para seleccionar filas únicas de la entrada, como sigue:

```
df.distinct().show()
```

- **orderBy()**-Para ordenar las filas por una columna en particular, como sigue:

```
df.orderBy("location").show()
```

- **join()**-Para unir dos tablas basándose en las condiciones proporcionadas, como sigue:

```
df1.join(df2, df1("id") === df2("id"), "inner")
```

- **groupBy() y avg()**-Pueden utilizarse en combinación para agregar valores agrupados en algunas columnas, como la ubicación en el siguiente caso:

```
df.groupBy("location").avg("salary").show()
```

Los ejemplos anteriores deberían darle una buena idea de los tipos de transformaciones disponibles en Spark. Puedes aprender más sobre DataFrames aquí: <https://spark.apache.org/docs/latest/sql-programming-guide.html>.

Ahora, veamos las transformaciones disponibles en T-SQL.

8.3. Transformación de datos mediante T-SQL

T-SQL es un lenguaje procedimental que es utilizado por los dedicated and serverless SQL pools en Synapse. Al igual que las transformaciones que hemos visto en Spark, T-SQL también proporciona un rico conjunto de transformaciones. Veamos algunas de las más importantes:

- **SELECT**-Para seleccionar los datos de un subconjunto de columnas, de la siguiente manera:

```
[firstName], [lastName] from dbo.Driver WHERE [city] = 'New York';
```

- **ORDER BY**-Para ordenar las filas por una columna en particular, como sigue:

```
SELECT [firstName], [lastName] from dbo.Driver ORDER BY [firstName];
```

- **DISTINCT**-Para seleccionar filas únicas de la entrada, como sigue:

```
SELECT DISTINCT [firstName], [lastName] from dbo.Driver;
```

- **GROUP BY**-Para agrupar filas por columnas de manera que se puedan realizar operaciones de agregación sobre ellas, de la siguiente manera

```
SELECT [gender], AVG([salary]) AS 'AVG salary' from dbo.Driver GROUP BY [gender];
```

- **UNION**-Para combinar filas de dos tablas que contienen el mismo esquema, como sigue:

```
SELECT [firstName], [lastName] FROM  
dbo.Driver  
WHERE [city] = 'New York'  
UNION  
select [firstName], [lastName] FROM  
dbo.TempDriver  
WHERE [city] = 'New York';
```

- **JOIN**-Para unir dos tablas basándose en las condiciones proporcionadas, como sigue:

```
SELECT driver.[firstName], driver.[lastName], feedback.[rating], Feedback.[comment]  
FROM dbo.Driver AS driver  
INNER JOIN dbo.Feedback AS feedback  
ON driver.[driverId] = feedback.[driverId]  
WHERE driver.[city] = 'New York';
```

- **VIEW**-Aparte de las transformaciones estándar, T-SQL también proporciona una transformación VIEW, que puede ayudar en la elaboración de informes y consultas ad hoc. A continuación veremos un ejemplo sencillo de cómo crear y utilizar una transformación VIEW:

```
CREATE VIEW CompleteDriverView
AS
SELECT driver.[firstName], driver.[lastName], feedback.[rating], feedback.[comment] FROM
dbo.Driver AS driver
INNER JOIN dbo.Feedback AS feedback
ON driver.[driverId] = feedback.[driverId]
WHERE driver.[city] = 'New York';
```

Así es como se puede utilizar una transformación VIEW:

```
SELECT DISTINCT * from CompleteDriverView;
```

Esto cubre algunas de las transformaciones importantes en T-SQL. Para una lista más completa, consulte la documentación T-SQL disponible aquí: <https://docs.microsoft.com/en-us/sql/t-sql/language-reference>.

A continuación vamos a conocer las opciones de transformación disponibles en ADF.

8.4. Transformación de datos mediante ADF

Ya hemos visto algunos ejemplos de ADF en los capítulos anteriores. Sólo para refrescar, busca Azure Data Factory desde el portal de Azure y crea una nueva fábrica de datos de Azure. Una vez creada, puedes lanzar la fábrica de datos de Azure, como se muestra en la siguiente captura de pantalla:

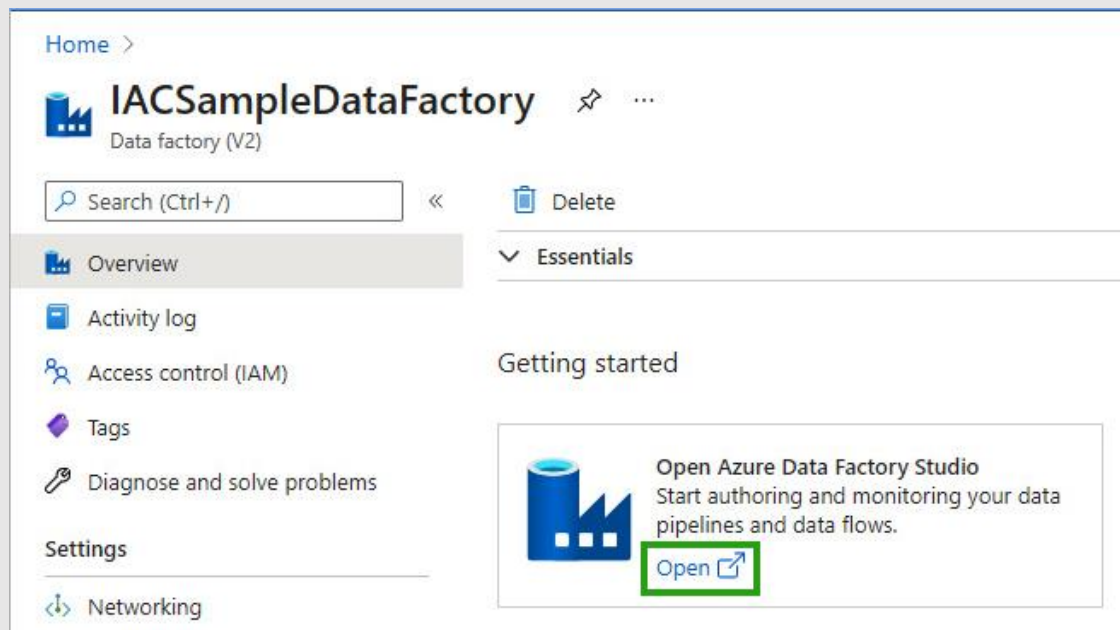


Figura 8.1 - Lanzamiento de la fábrica de datos de Azure

Esto abrirá el área de trabajo de ADF donde puedes construir tus pipelines, como se muestra en la siguiente captura de pantalla:

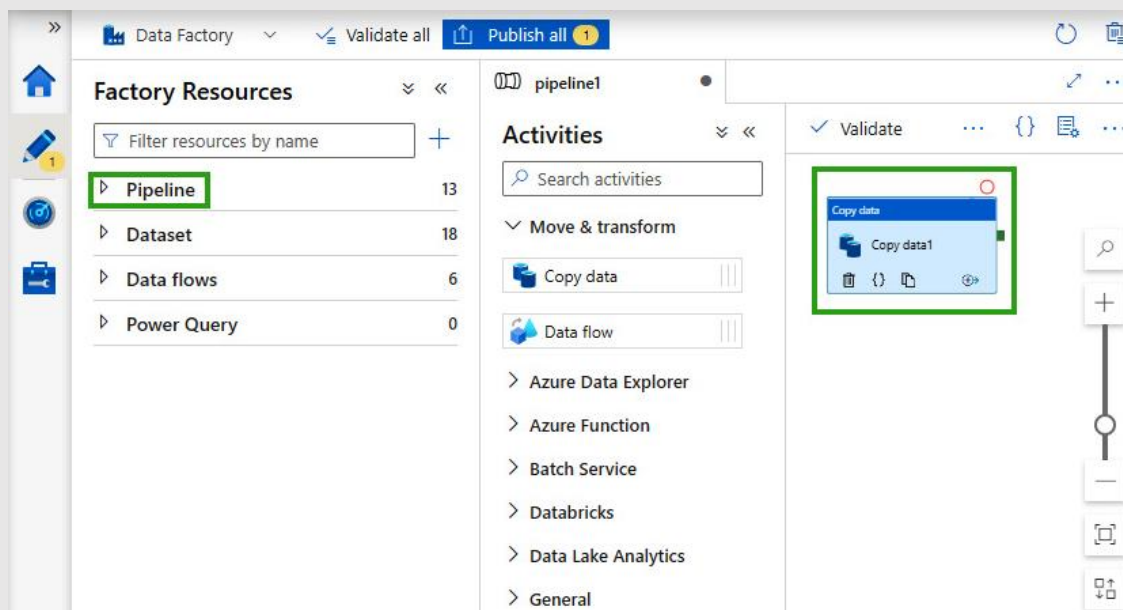


Figura 8.2 - ADF Studio

Todas las transformaciones de ADF ocurren en datasets. Por lo tanto, antes de poder realizar cualquier transformación, tendremos que crear datasets de los datos de origen. Puede hacer clic en el símbolo + para crear un nuevo dataset, como se muestra en la siguiente captura de pantalla:

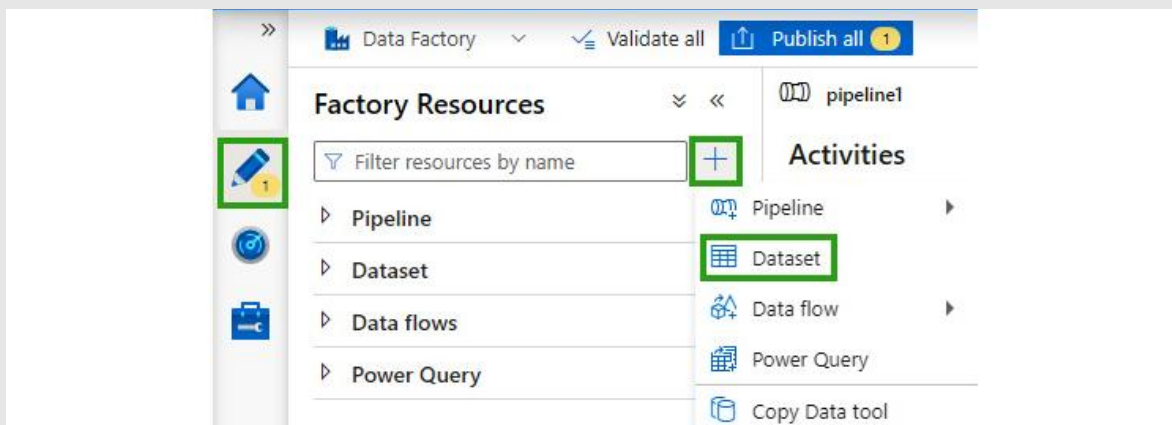


Figura 8.3 - Creación de un nuevo conjunto de datos en ADF

ADF proporciona una amplia gama de fuentes de datos de Azure y de otros proveedores, como se muestra en la siguiente captura de pantalla.

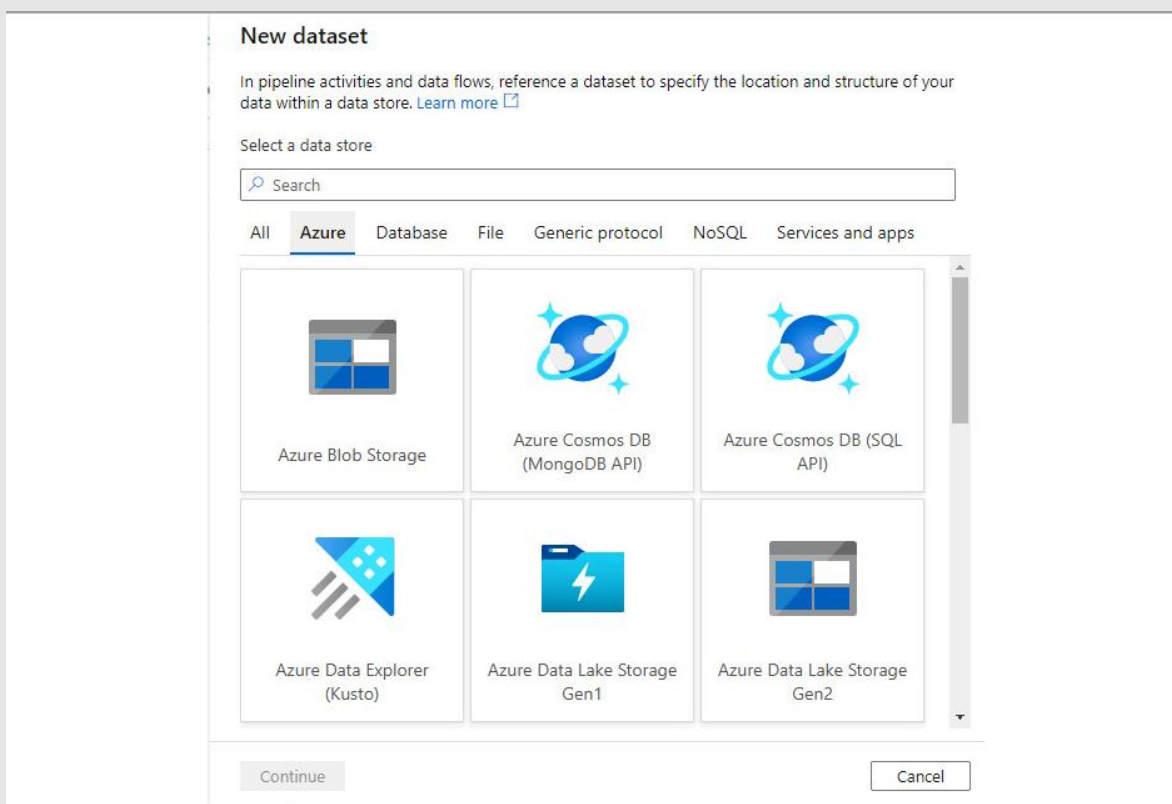


Figura 8.4 - Opciones de origen de conjuntos de datos en ADF

Puede seleccionar el origen de datos adecuado, hacer clic en Continuar y añadir la ubicación de los archivos o carpetas de datos de origen que deben transformarse. Esto creará conjuntos de datos de origen.

El ADF ofrece prácticas opciones de transformación sin código denominadas **mapping data flows** (flujos de datos de mapeo). Los mapping data flows **proporcionan tres tipos de transformaciones**, como se indica aquí:

- **Transformaciones de esquema:** como la adición de nuevas columnas.
- **Transformaciones de filas:** como la actualización de filas, la creación de vistas, etc.
- **Transformaciones de entrada/salida múltiple (I/O):** como la división de filas, la combinación de filas, etc.

Veamos en detalle algunas de las opciones importantes disponibles en estos tres tipos de transformación.

8.4.1. Transformaciones de esquema

Las transformaciones de esquema se refieren a las acciones que resultan en el cambio del esquema de la tabla o de los DataFrames. Veamos algunas de las transformaciones de esquema más utilizadas, a continuación:

- **Aggregate**-Para realizar Min, Max, Sum, Count, y así en los datos entrantes. Aquí hay una captura de pantalla de cómo encontrar el promedio de la columna Salario:

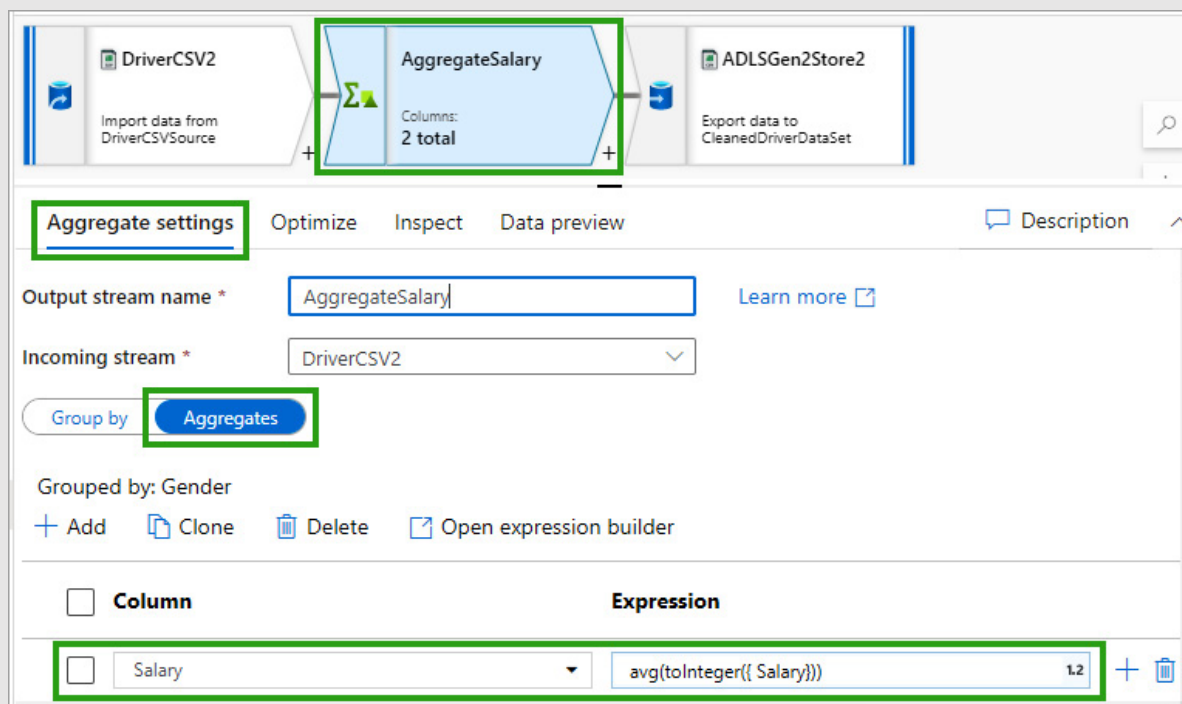


Figura 8.5 - Transformación agregada en el ADF

NOTA

Las transformaciones de agregación sólo mostrarán las columnas utilizadas en la agregación. Por lo tanto, realice una self-join con los datos de origen después de esta etapa si desea que las otras columnas estén presentes para las etapas posteriores.

- **Derived column** (Columna derivada): utilice esta transformación para añadir nuevas columnas a los datos existentes. En la siguiente captura de pantalla, estamos añadiendo una simple columna isActive a la tabla Driver extraída de un archivo CSV:

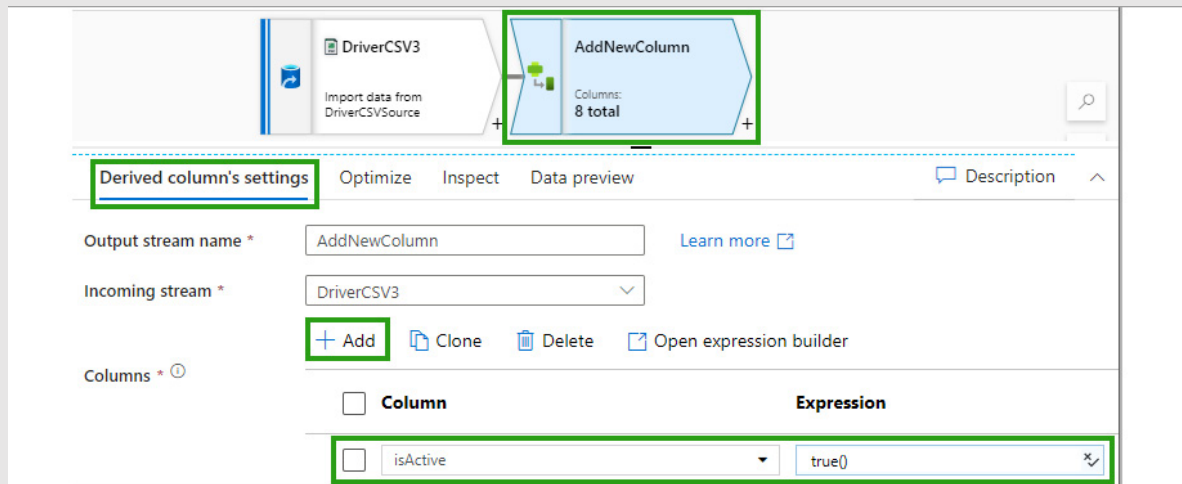


Figura 8.6 - Transformación de columna derivada en ADF

- **Select**-Puede utilizar esta transformación para seleccionar sólo las columnas necesarias de la entrada o para cambiar el nombre de las columnas antes de almacenarlas en un data store, como se demuestra en la siguiente captura de pantalla. Puede hacer clic en el icono de la papelera para eliminar las columnas que no sean necesarias:

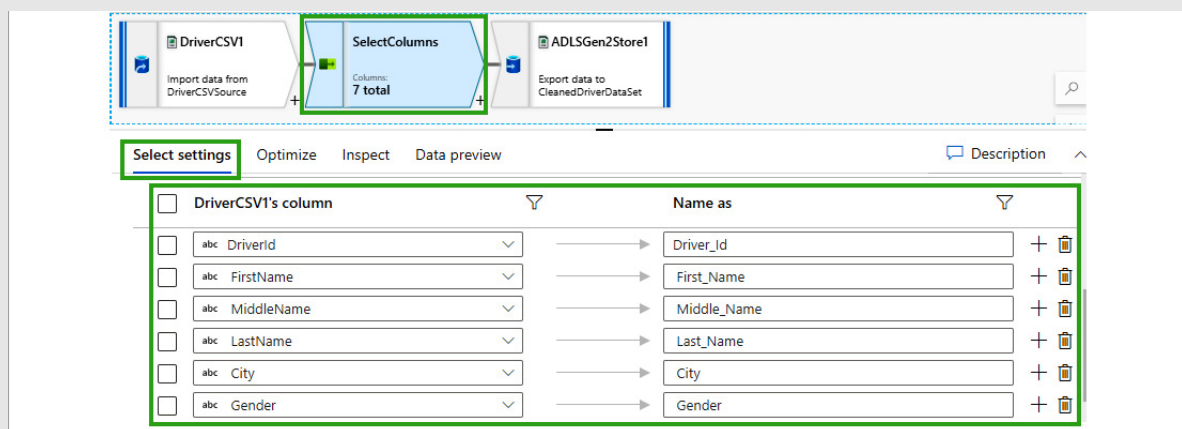


Figura 8.7 - Transformación de selección en el ADF

A continuación, veamos algunas transformaciones de filas.

8.4.2. Transformaciones de filas

Se trata de transformaciones que se aplican a las filas de la tabla, y que se describen aquí:

- **Alter row**: se utiliza para insertar, eliminar, actualizar y upsert (insertar o actualizar) filas en una base de datos o data warehouse. En la siguiente captura de pantalla, insertamos las filas en una base de datos sólo si el valor DriverId no es NULL:

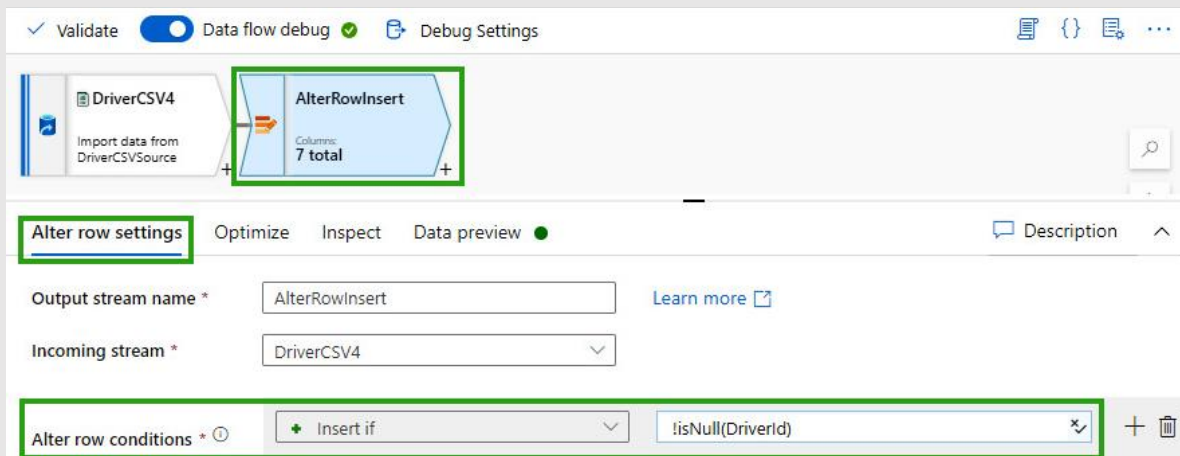


Figura 8.8 - Transformación de filas de alteración

NOTA:

Alter row sólo funciona en bases de datos, Cosmos DB o REpresentational State Transfer (REST) endpoint sinks.

- **Filter** (Filtro): para filtrar las filas en función de las condiciones. En la siguiente captura de pantalla, estamos filtrando las filas en las que el valor de Ciudad es Nueva York:

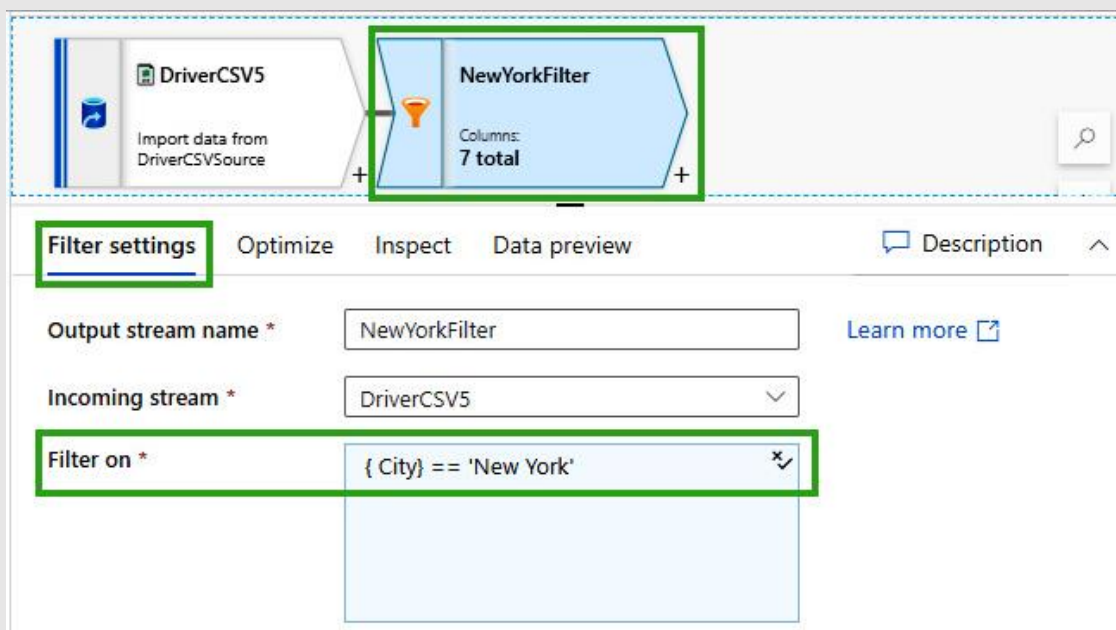


Figura 8.9 - Transformación de filtros en el ADF

- **Sort** (Ordenar) - Para ordenar las filas en base a cualquier columna o grupo de columnas, como se ilustra en la siguiente captura de pantalla:

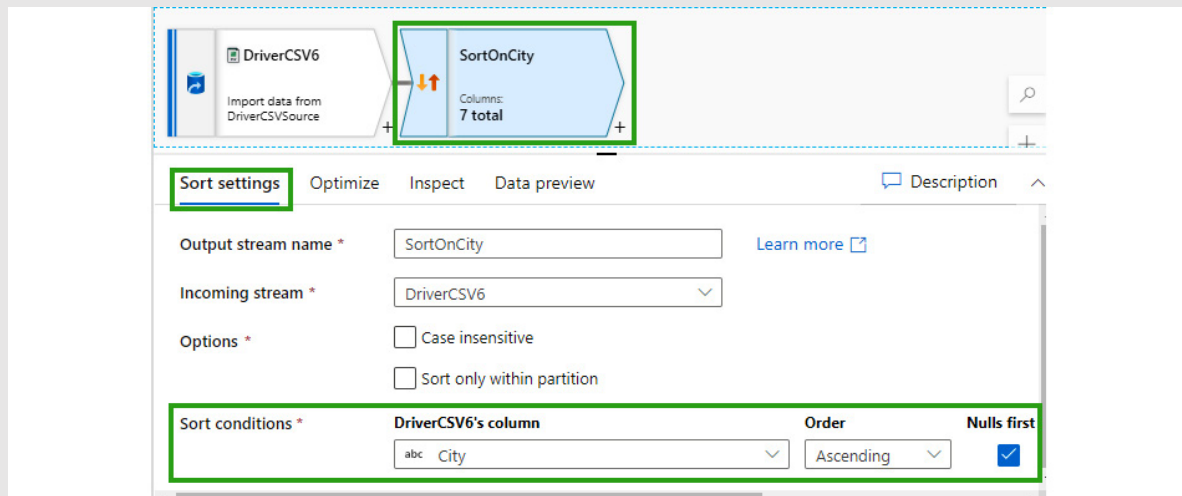


Figura 8.10 - Transformación de ordenación

A continuación, veamos las transformaciones multi-I/O.

8.4.3. Transformaciones multi-I/O

Son transformaciones que operan sobre más de una entrada o, por el contrario, dividen la entrada en más de una salida. Aquí se describen:

- **Conditional split** (división condicional): puede utilizarse para dividir la entrada en dos flujos de salida en función de las condiciones. En la siguiente captura de pantalla, dividimos la entrada basándonos en el Rating:

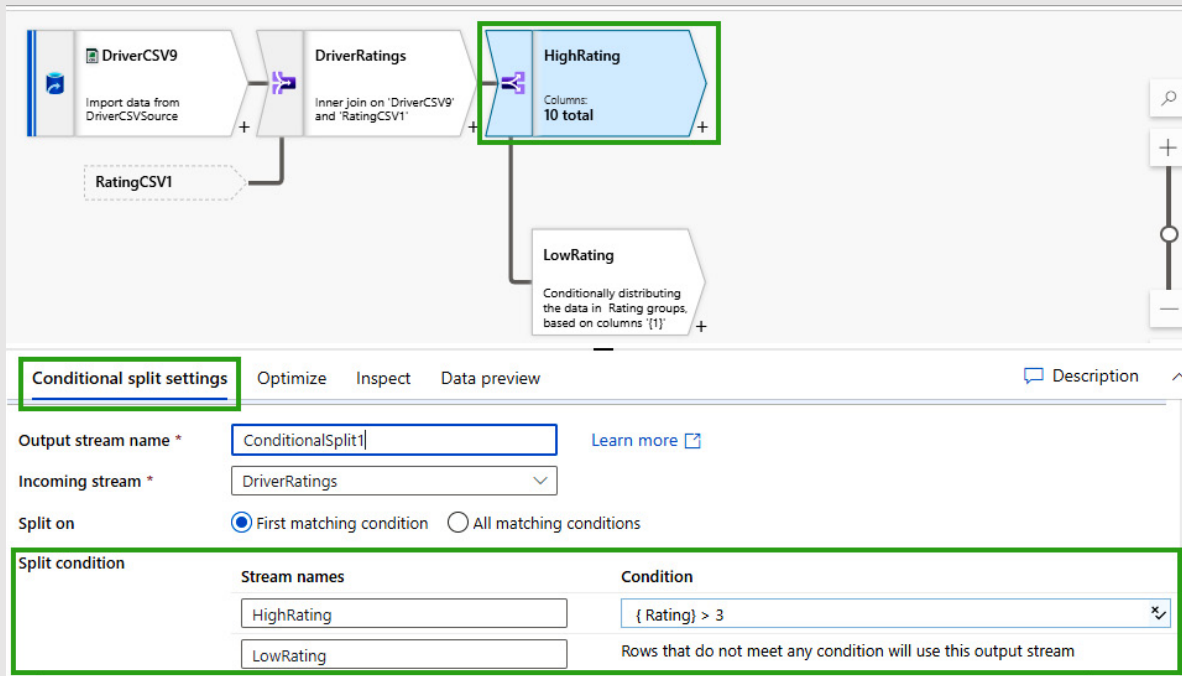


Figura 8.11 - Transformación de división condicional en ADF

- **Join**: se utiliza para unir dos flujos basándose en una o más condiciones de unión. La siguiente captura de pantalla muestra cómo combinar los conjuntos de datos DriverCSV y RatingsCSV utilizando el valor DriverId:

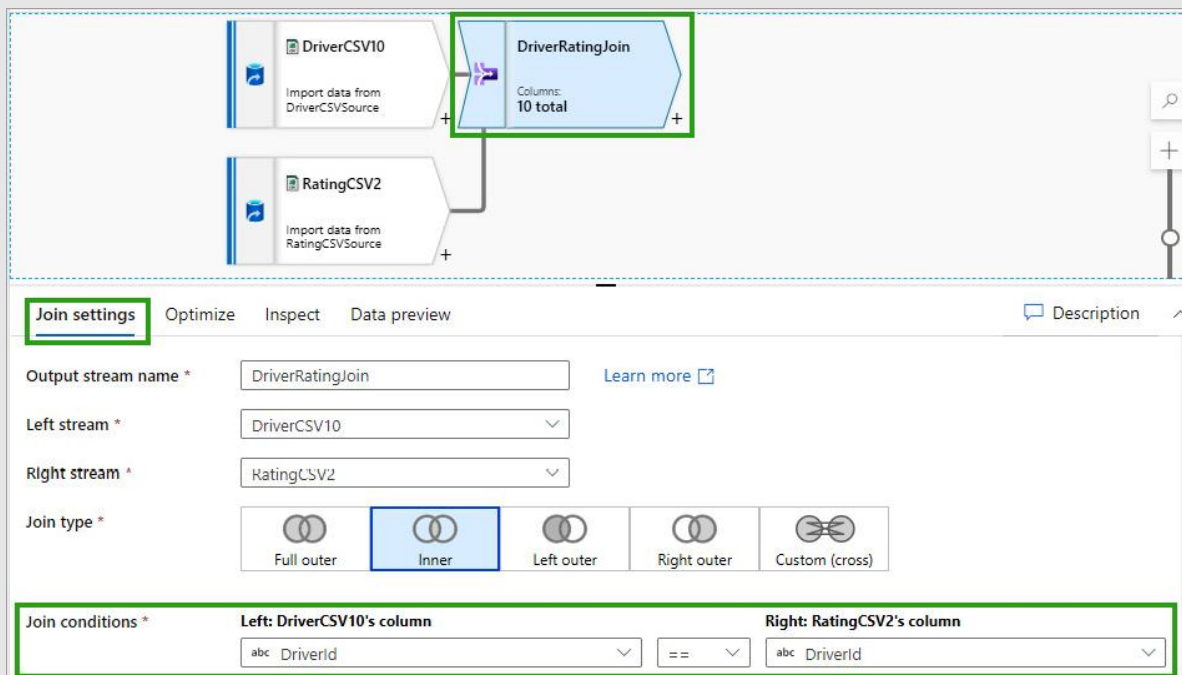


Figura 8.12 - Transformación de join en ADF

- **Unión**: combina dos conjuntos de datos de entrada con el mismo esquema en uno solo, como se ilustra en la siguiente captura de pantalla:

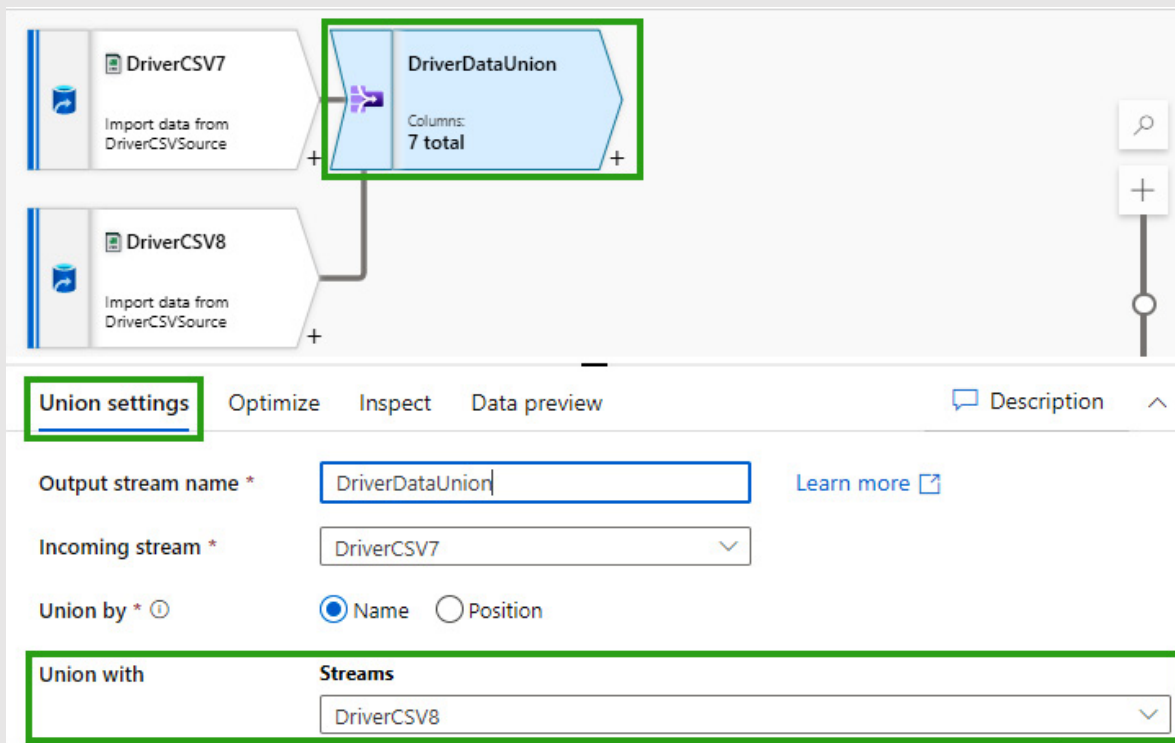


Figura 8.13 - Ejemplo de unión en ADF

Para obtener una lista completa de transformaciones, puede consultar la documentación de Azure ADF disponible aquí: <https://docs.microsoft.com/en-us/azure/data-factory/data-flow-transformation-overview>.

ADF proporciona plantillas convenientes que pueden realizar muchas de las actividades estándar del pipeline. Veámoslas a continuación.

8.4.4. ADF templates

ADF proporciona plantillas estándar que puede utilizar para diversas actividades de copia y transformación de datos. Puede explorar las plantillas bajo el enlace Pipeline templates en la página de inicio de ADF Studio, como se muestra en la siguiente captura de pantalla:

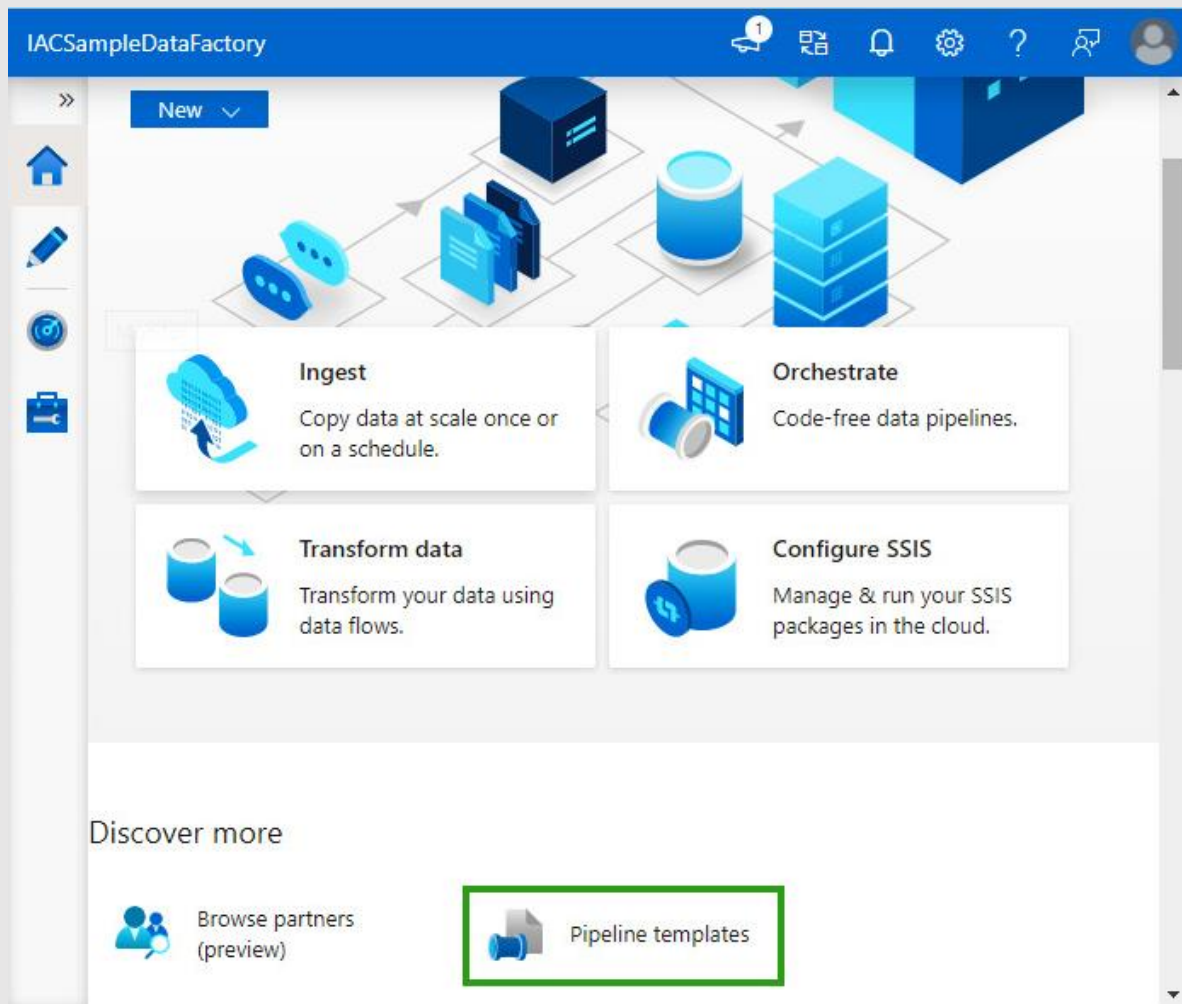


Figura 8.14 - Enlace de plantillas de pipelines

A continuación se muestra una muestra de la galería de plantillas:

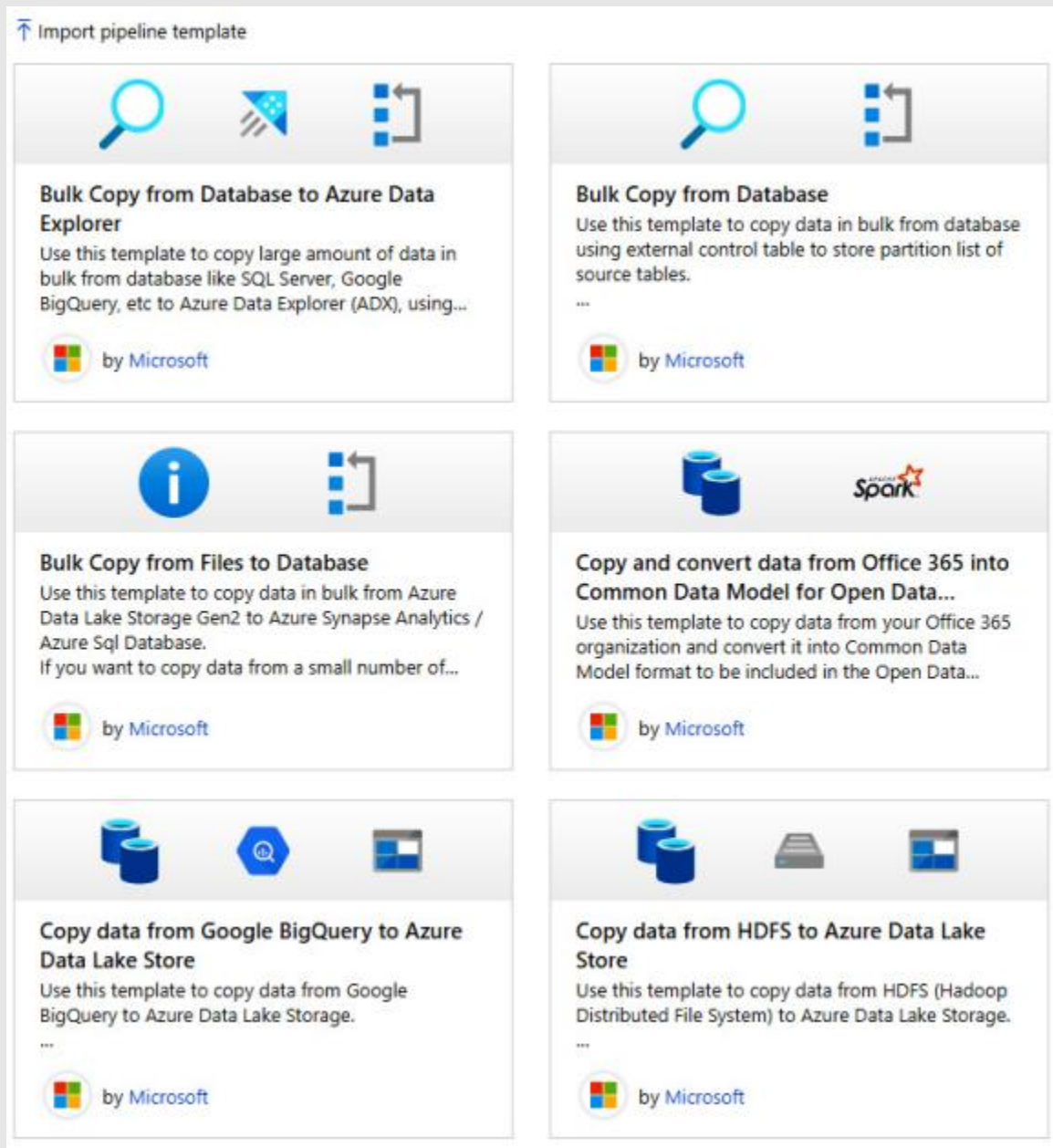


Figura 8.15 - Ejemplo de galería de plantillas del ADF

Por favor, utilice las plantillas siempre que sea posible en lugar de reinventar los procedimientos, ya que esto le ahorrará mucho tiempo y evitará errores comunes.

Veamos a continuación las transformaciones utilizando los pipelines de Synapse.

8.5. Transformación de datos mediante Azure Synapse pipelines

Azure Synapse pipelines no es más que ADF implementado dentro de Azure Synapse Analytics, por lo que los ejemplos de transformación que vimos en la sección de ADF se aplican aquí también. La única diferencia es la página de lanzamiento. Puede lanzar Synapse pipelines desde la pestaña Synapse Analytics, como se muestra en la siguiente captura de pantalla:

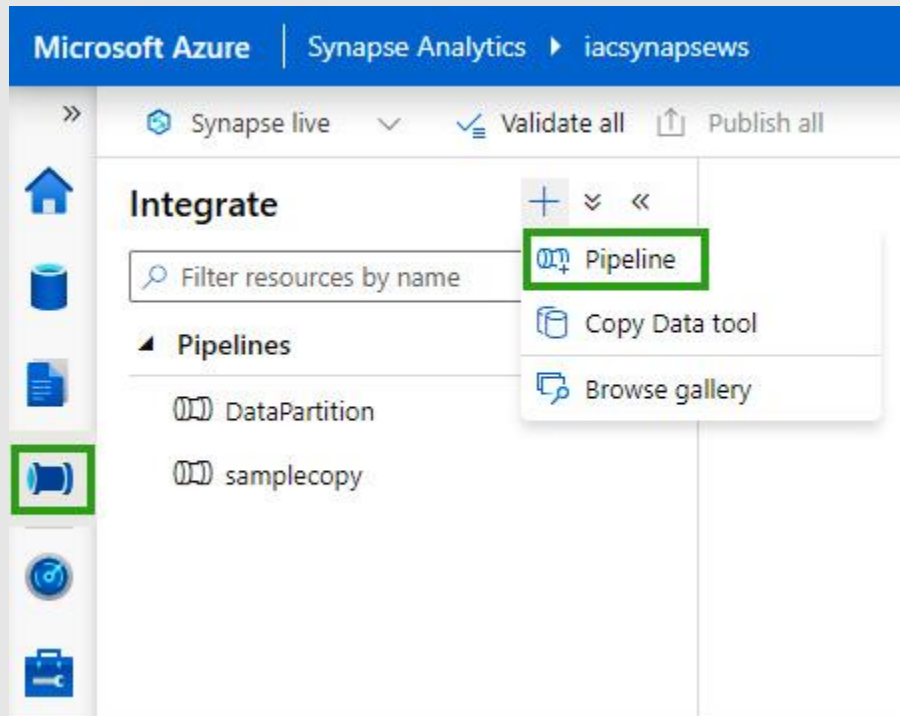


Figura 8.16 - Lanzamiento de pipelines de Synapse

Espero que haya comprendido las transformaciones disponibles en ADF y en los pipelines de Synapse. Veamos ahora las opciones de transformación disponibles en Stream Analytics.

8.6. Transformación de datos mediante Stream Analytics

La configuración de Stream Analytics, los conceptos y otros detalles se cubrirán en el Capítulo 10, Diseño y desarrollo de una solución de procesamiento de stream. Dado que necesitamos una base de streaming antes de poder hablar de las transformaciones, cubriremos las transformaciones de Stream Analytics como parte del Capítulo 10, Diseño y desarrollo de una solución de procesamiento de stream.

Veamos ahora el concepto de limpieza de datos.

8.7. Limpieza de datos

La limpieza de datos es una actividad importante en cualquier pipeline de datos. Como los datos fluyen desde varias fuentes, hay posibilidades de que los datos no se adhieran a los esquemas y puede haber valores perdidos, entradas no estándar, duplicados, etc. Durante la fase de limpieza, intentamos corregir estas anomalías. Veamos algunas técnicas comunes de limpieza de datos.

8.7.1. Tratamiento de los valores ausentes/nulos

Podemos tratar los valores perdidos o nulos de varias maneras. Podemos optar por filtrar esas filas, sustituir los valores que faltan por valores predeterminados, o sustituirlos por algunos valores significativos como la media, la mediana, el promedio, etc. Veamos un ejemplo de cómo podemos sustituir los valores que faltan por alguna cadena por defecto como 'NA'.

Sustitución con valores por defecto

La sustitución con valores por defecto se puede lograr utilizando la transformación de **columnas derivadas**, como se muestra en la siguiente captura de pantalla:

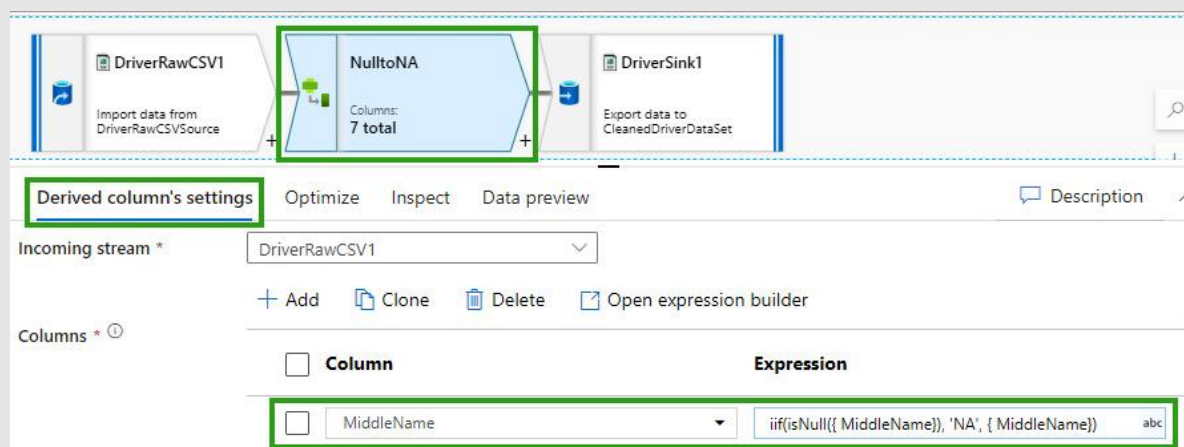


Figura 8.17 - Sustitución con valores por defecto

Veamos ahora cómo filtrar los valores nulos.

Filtrar valores nulos

Puede filtrar los valores nulos utilizando la transformación **Alter row**, como ya hemos visto en la Figura 8.8.

A continuación veremos cómo recortar los valores de entrada.

8.7.2. Trimming inputs (Recorte de entradas)

Los valores con espacios en blanco al final son un problema común. Puede recortar fácilmente estos valores utilizando el método trim() desde una transformación de columna derivada. A continuación se muestra un ejemplo de ello:

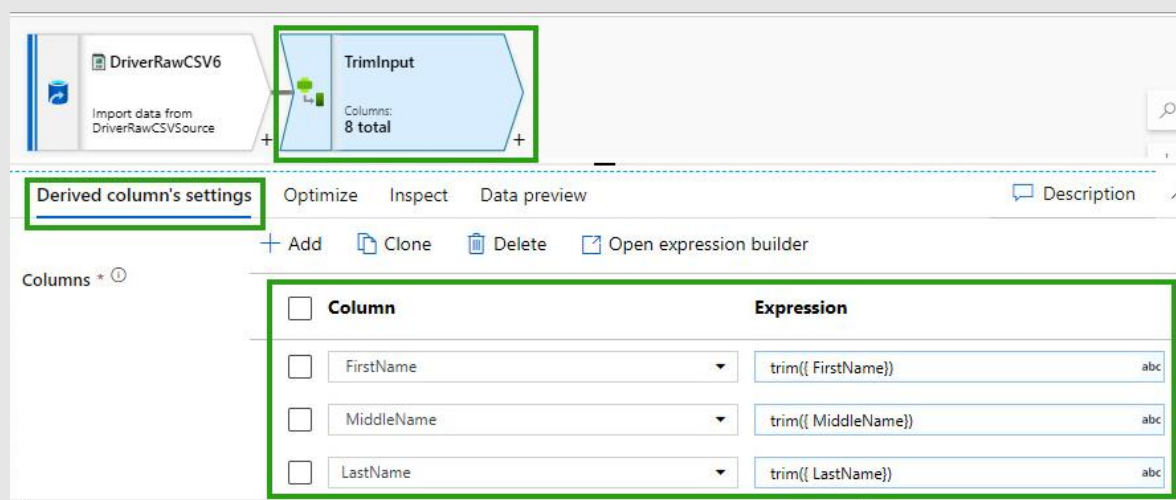


Figura 8.18 - Recorte de espacios en blanco

Veamos ahora cómo normalizar los valores de entrada.

8.7.3. Estandarización de valores

Diferentes fuentes de entrada pueden utilizar diferentes convenciones para los datos. Por ejemplo, si una de las fuentes de entrada utiliza el símbolo \$ para representar el valor del dólar y otra corriente de entrada utiliza el USD, es posible que queramos estandarizar las entradas antes de enviarlas para su posterior procesamiento. En la siguiente captura de pantalla, podemos ver cómo reemplazar el símbolo \$ en la columna Salario por USD. Sólo hay que utilizar el script **replace({Salary}, '\$', 'USD')** en el campo Expresión:

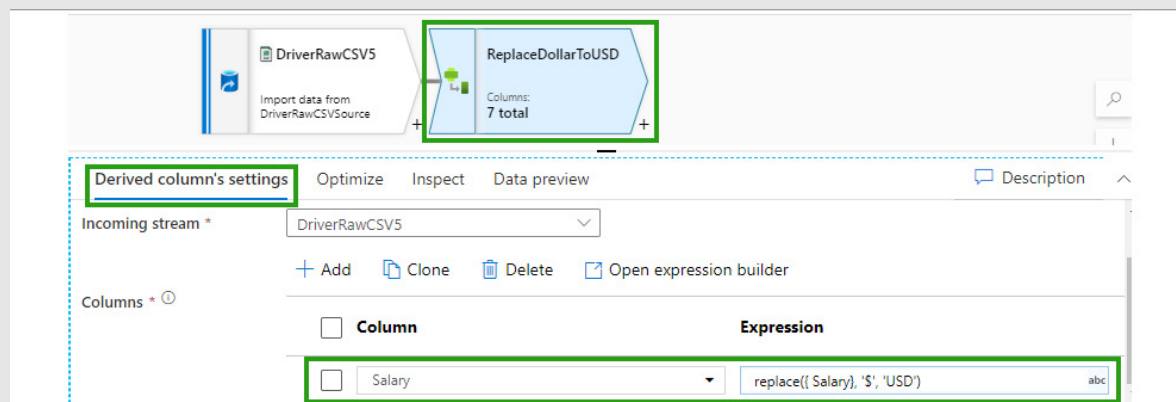


Figura 8.19 - Sustitución de valores mediante la Columna Derivada

Veamos a continuación cómo manejar los valores atípicos.

8.7.4. Manejo de valores atípicos

Si los valores de algunos de los campos parecen anormales, puede reemplazarlos por valores medios o medianos. Veamos un ejemplo de cómo hacer esto, como sigue:

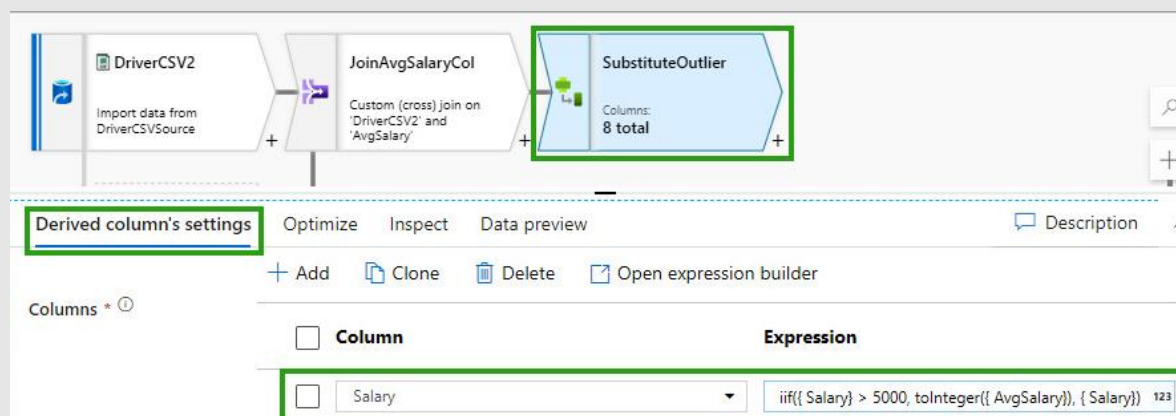


Figura 8.20 - Uso de la columna Derivada para sustituir valores

En la captura de pantalla anterior, en el campo Expresión, estamos sustituyendo cualquier valor > 5000 por el salario medio.

NOTA

Puede utilizar la transformación Aggregate para encontrar la media, la mediana, el mínimo, el máximo y otras funciones de agrupación matemática.

Veamos ahora cómo eliminar los duplicados.

8.7.5. Eliminación de duplicados/agregación

Puede eliminar las filas duplicadas utilizando la transformación Aggregate. A continuación se muestra un ejemplo de ello:

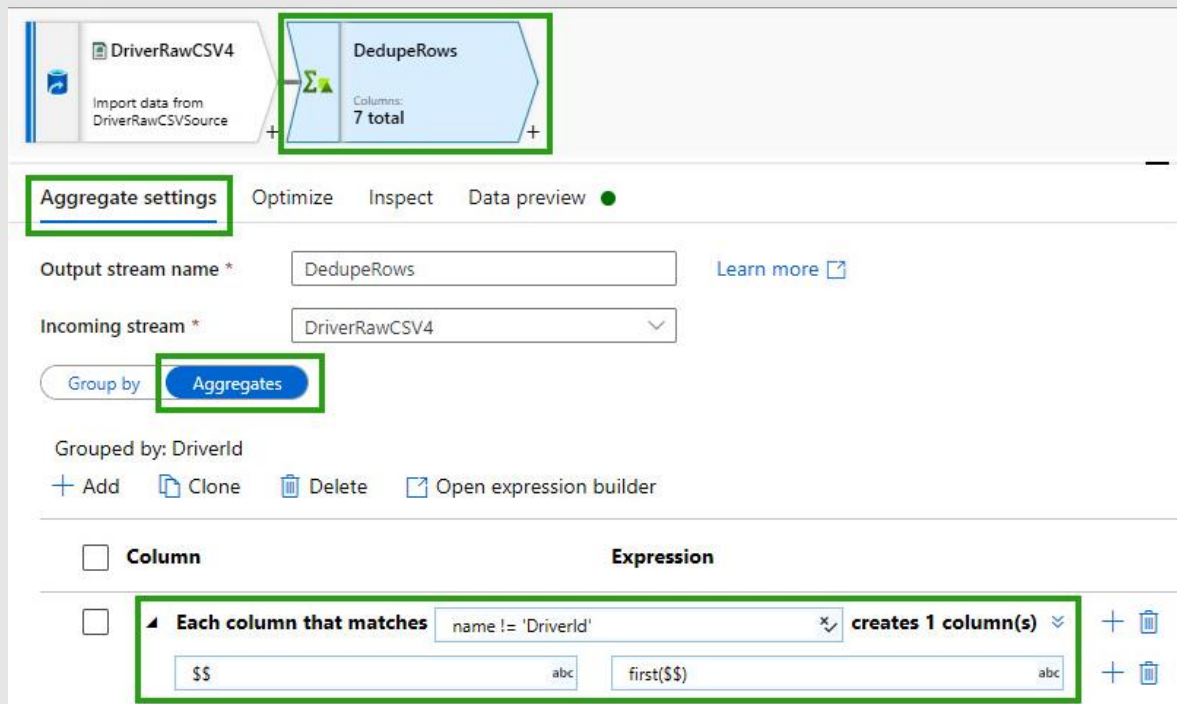


Figura 8.21 - Eliminación de duplicados mediante la transformación Aggregate

Aggregates por defecto emite sólo la columna sobre la que se opera. En este caso, como lo he agrupado en DriverId, sólo emitiría la columna DriverId. Para superar esto, en el campo Columna, para la opción Each column that matches (Cada columna que coincide), he especificado nombre != DriverId para que todas las demás columnas también aparezcan en la salida.

NOTA

Puedes llevar a cabo todo este trabajo de limpieza y preparación de datos utilizando las transformaciones de Spark o SQL que hemos discutido anteriormente en el capítulo. Éstas también funcionarán perfectamente.

Ya hemos cubierto los detalles de las operaciones de limpieza más utilizadas. Con la variedad de transformaciones disponibles en ADF, Spark y SQL, puede ser creativo en la forma de realizar sus tareas de manera eficiente. Veamos a continuación cómo dividir los datos en los pipelines de datos.

8.8. Dividir datos

ADF ofrece múltiples formas de dividir los datos en un pipeline. Las más importantes son la división condicional y la clonación (nueva rama). Mientras que la división condicional se utiliza para dividir los datos basándose en ciertas condiciones, la opción de nueva rama se utiliza para copiar el conjunto de datos para un nuevo flujo de ejecución. Ya hemos visto un ejemplo de División Condicional en la Figura 8.11. Veamos cómo podemos crear una nueva rama en el data pipeline.

Para crear una nueva rama, basta con hacer clic en el icono + junto a cualquier artefacto de origen de datos (como el bloque DriverCSV11 que se muestra en la siguiente captura de pantalla). Desde ahí, puede elegir la opción Nueva rama:

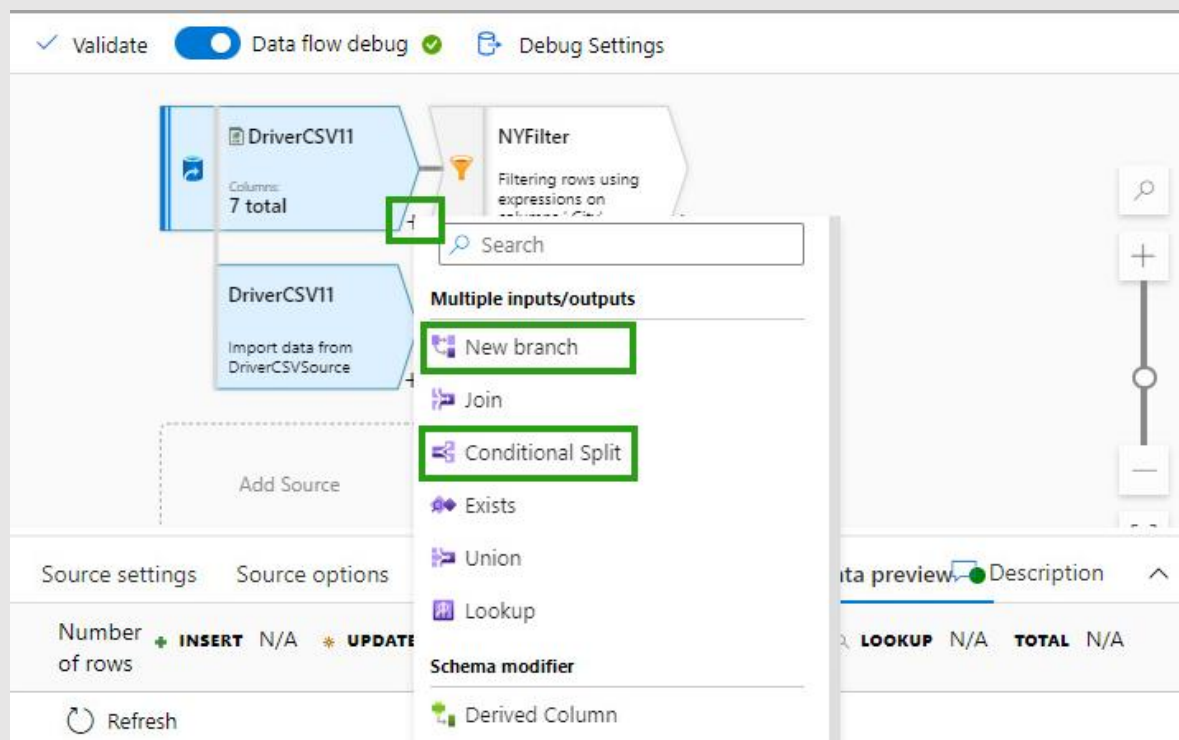


Figura 8.22 - Opción de nueva rama en ADF

Además de estas dos opciones, el ADF también ofrece la posibilidad de dividir los archivos de entrada en múltiples subarchivos mediante particiones. Veamos cómo lograrlo a continuación.

8.8.1. Divisiones de archivos

Para utilizar las divisiones de archivos, cree un nuevo artefacto Sink y, en la pestaña Optimizar, especifique el número de particiones necesarias. La siguiente captura de pantalla muestra cómo se puede hacer esto:

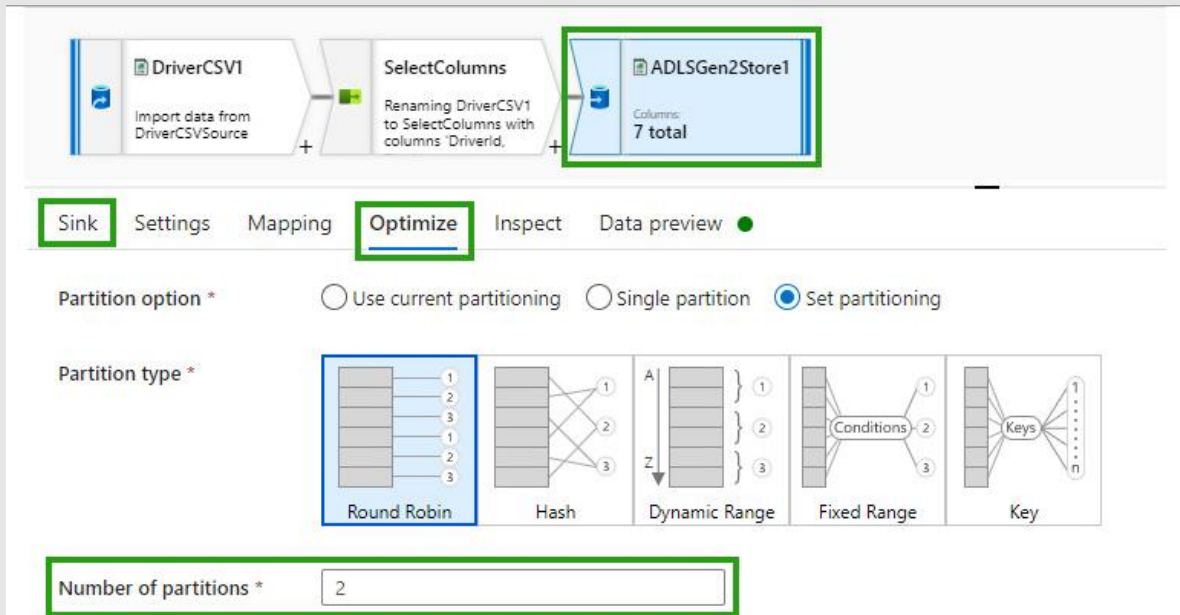


Figura 8.23 - Dividir archivos usando ADF

Ahora que sabemos cómo dividir los datos para su procesamiento, veamos a continuación cómo extraer los datos de los archivos JSON.

8.9. Shredding JSON

Shredding se refiere al proceso de extracción de datos de archivos JSON en tablas. Spark, Synapse SQL pools y ADF proporcionan soporte nativo para extraer datos de JSON. Veamos ejemplos para cada uno de los servicios.

8.9.1. Extracción de valores de JSON con Spark

Spark puede leer directamente archivos JSON y extraer el esquema de los mismos. Aquí hay un simple fragmento de código que puede lograr la lectura de JSON:

```
val dfJSON = spark.read.json("abfss://path/to/json/*.json")
dfJSON.printSchema()
dfJSON.show(false)
```

Este es el aspecto de la salida:

```
root
 |-- firstname: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- id: string (nullable = true)
 |-- lastname: string (nullable = true)
 |-- location: string (nullable = true)
 |-- middlename: string (nullable = true)
 |-- salary: long (nullable = true)

+-----+-----+---+-----+-----+-----+-----+
|firstname|gender|id |lastname|location |middlename|salary|
+-----+-----+---+-----+-----+-----+-----+
|Catherine|Female|102|        |California|Goodwin  |4300 |
|Jenny    |Female|104|Simons  |Arizona  |Anne    |3400 |
|Bryan    |Male  |101|Williams|New York |M       |4000 |
|Alice    |Female|100|Hood    |New York |        |4100 |
|Daryl    |Male  |103|Jones   |Florida  |        |5500 |
|Daryl    |Male  |103|Jones   |Florida  |        |5500 |
+-----+-----+---+-----+-----+-----+-----+
```

Figura 8.24 - Salida de la operación JSON

También puede especificar manualmente el esquema, como se muestra en el siguiente ejemplo:

```
val driverSchema = new StructType()
.add("firstname", StringType)
.add("middlename", StringType)
...
.add("salary", IntegerType)
val dfJSON = spark.read.schema(driverSchema).json("abfss://path/to/json/*.json")
```

Una vez que tienes los datos en el DataFrame, puedes utilizar cualquiera de las transformaciones disponibles en Spark sobre él para extraer y modificar los datos.

A continuación, vamos a ver cómo extraer valores de JSON utilizando SQL.

8.9.2. Extracción de valores de JSON utilizando SQL

T-SQL proporciona la función OPENROWSET para consultar data stores remotos. Podemos utilizar esta función para cargar datos de forma masiva en instancias dedicated or serverless SQL. A continuación se muestra un ejemplo de cómo podemos cargar y analizar archivos JSON desde un almacenamiento remoto utilizando SQL serverless:

```
SELECT
    JSON_VALUE(doc, '$.firstname') AS firstname,
    JSON_VALUE(doc, '$.lastname') AS lastname,
    CAST(JSON_VALUE(doc, '$.id') AS INT) as driverid,
    CAST(JSON_VALUE(doc, '$.salary') AS INT) as salary
FROM openrowset(
    BULK 'abfss://path/to/json/*.json',
    FORMAT = 'csv',
    FIELDTERMINATOR = '0x0b',
    FIELDQUOTE = '0x0b'
) WITH (doc nvarchar(max)) AS ROWS
GO
```

Los resultados de la consulta anterior serían algo así:

Results

Messages

View

Table

Chart

↗

Export results ▾

🔍

Search

firstname	lastname	driverid	salary
Alice	Hood	100	4100
Bryan	Williams	101	4000
Daryl	Jones	103	5500
Daryl	Jones	103	5500
Jenny	Simons	104	3400
Catherine		102	4300

Figura 8.25 - Ejemplo de salida del análisis sintáctico de JSON con OPENROWSET

NOTA

Es necesario especificar el valor FORMAT como CSV para JSON, como se resalta en el fragmento de código anterior.

8.9.3. Extracción de valores de JSON utilizando ADF

ADF proporciona la transformación Flatten para convertir estructuras de datos jerárquicas como JSON en estructuras planas como tablas. Existe otra transformación de desnormalización similar llamada Pivot, que conocerás más adelante en este capítulo.

Supongamos que tiene un conjunto de datos de origen con el siguiente JSON

```
{
  "firstname": "Alice",
  "middlename": "",
  "lastname": "Hood",
  "id": "100",
  "locations": [{"city": "San Francisco", "state": "CA"},
    {"city": "San Jose", "state": "CA"},
    {"city": "Miami", "state": "FL"}
  ],
  "gender": "Female"
}
```

Seleccione la transformación Flatten en ADF y especifique el mapeo de columnas como se muestra en la siguiente figura:

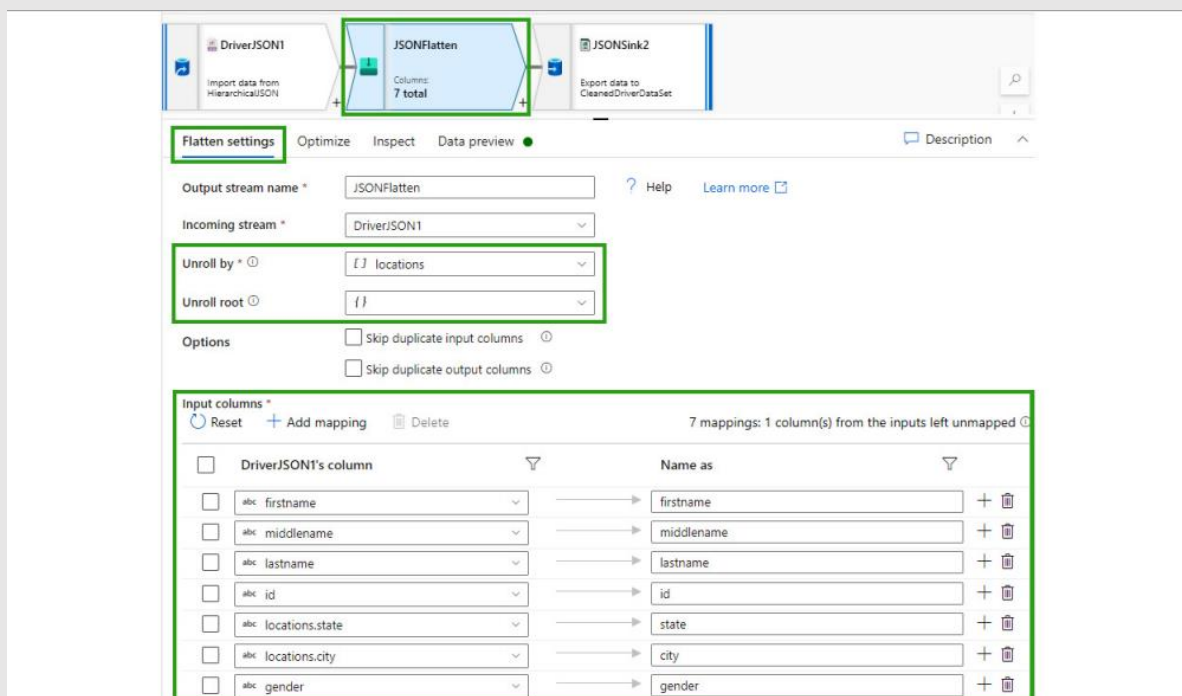


Figura 8.26 - Transformación Flatten en ADF

Para estructuras complejas como arrays dentro de JSON, puede utilizar Unroll by para dividirlos en varias filas. En nuestro ejemplo, puede ver que el campo Location, que era un array, ha sido desnormalizado en líneas separadas en la sección Input columns.

Puede aprender más sobre la transformación Flatten aquí: <https://docs.microsoft.com/en-us/azure/data-factory/data-flow-flatten>.

Veamos ahora cómo codificar y decodificar datos.

8.10. Codificación y decodificación de datos

En esta sección, veremos cómo ocuparnos de la codificación y decodificación de valores como el Código Estándar Americano para el Intercambio de Información (ASCII), el Formato de Transformación Unicode 8 (UTF-8), UTF-16, etc. mientras leemos o escribimos datos de diferentes fuentes. Aquí volveremos a ver ejemplos utilizando Spark, SQL y ADF.

8.10.1. Codificación y decodificación usando SQL

En Synapse SQL, collation define el tipo de codificación, el tipo de ordenación, etc. en las cadenas SQL. El collation puede establecerse tanto a nivel de base de datos como de tabla. En el nivel de la base de datos, se puede establecer el collation, como se muestra aquí:

```
CREATE DATABASE TripsDB COLLATE Latin1_General_100_BIN2_UTF8;
```

A nivel de tabla, puede establecerla como se muestra aquí:

```
CREATE EXTERNAL TABLE FactTrips (  
  [tripId] VARCHAR (40) COLLATE Latin1_General_100_BIN2_UTF8,  
  ...  
)
```

Una vez definida la collation correcta, Synapse SQL se encarga de almacenar los datos en el formato correcto y de utilizar el conjunto de codificación adecuado para las operaciones posteriores en ese conjunto de datos.

Veamos a continuación cómo realizar la codificación/descodificación en Spark.

8.10.2. Codificación y decodificación en Spark

Spark admite métodos denominados codificación y decodificación, que pueden utilizarse para realizar la conversión. El siguiente ejemplo de Spark SQL muestra una simple operación de codificación y decodificación. Convertimos los datos a formato hexadecimal (hex) para mostrar los resultados en un formato compacto y legible, pero también se pueden utilizar directamente los métodos de codificación y decodificación sin la función hex():

```
>SELECT hex(encode('Azure', 'UTF-16'));  
FEFF0041007A007500720065  
>SELECT decode(X'FEFF0041007A007500720065', 'UTF-16')  
Azure
```

Los métodos de codificación y decodificación también están disponibles en las versiones Python y Scala de Spark.

8.10.3. Codificación y decodificación con ADF

ADF ofrece la posibilidad de especificar la codificación correcta en los artefactos del conjunto de datos. Si haces clic en la pestaña Conexión, puedes ver el campo Codificación, como se muestra en la siguiente captura de pantalla:

The screenshot shows the 'Connection' tab in the ADF interface. It contains several configuration fields: 'Linked service' (set to 'AzureDataLakeStorage'), 'File path' (set to 'users / raw/driver/sample/csv / driver.csv'), 'Compression type' (set to 'None'), 'Column delimiter' (set to 'Comma (,)'), 'Row delimiter' (set to 'Default (\r,\n, or \r\n)'), 'Encoding' (set to 'Default(UTF-8)' and highlighted with a green box), and 'Escape character' (set to 'Backslash (\)'). There are also 'Test connection', 'Edit', and '+' icons.

Figura 8.27 - Función de codificación en los datasets de origen de ADF

Aparte de la opción de codificación de conjuntos de datos, ADF también proporciona funciones para codificar y decodificar identificadores de recursos uniformes (URI), Base64, etc., utilizando sus comandos de conversión. He compartido algunos comandos de conversión de la documentación de ADF, como sigue:

<code>base64ToBinary()</code>	Return the binary version for a Base64-encoded string.
<code>base64ToString()</code>	Return the string version for a Base64-encoded string.
<code>decodeBase64()</code>	Return the string version for a Base64-encoded string.
<code>dataUriToBinary()</code>	Return the binary version for a data URI.
<code>dataUriToString()</code>	Return the string version for a data URI.
<code>decodeDataUri()</code>	Return the binary version for a data URI.

Figura 8.28 - Tabla de funciones de codificación/decodificación disponibles en el ADF

Puede encontrar una lista detallada de las funciones de conversión en ADF aquí: <https://docs.microsoft.com/en-us/azure/data-factory/control-flow-expression-language-functions#conversion-functions>.

Veamos a continuación cómo configurar el manejo de errores en las transformaciones de ADF.

8.11. Configurar el manejo de errores para la transformación

En todos nuestros ejemplos de pipelines ADF hasta ahora, hemos visto sólo casos de éxito. ADF también proporciona un flujo separado para manejar errores y fallos. De hecho, ADF soporta cuatro flujos diferentes, como se muestra en la siguiente captura de pantalla:

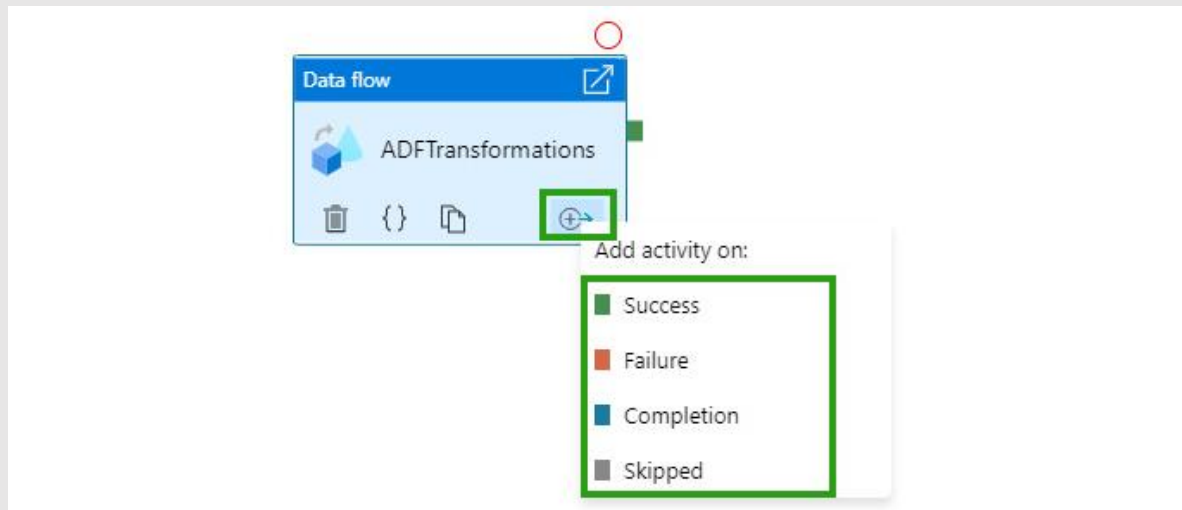


Figura 8.29 - Flujos de actividad de ADF

En caso de que se produzcan errores en cualquier paso del pipeline, podemos construir una rama de gestión de errores que se puede utilizar para arreglar los errores o almacenarlos para futuras acciones. La siguiente captura de pantalla muestra uno de estos flujos. Tendrá que conectar la línea naranja a la actividad de gestión de errores:

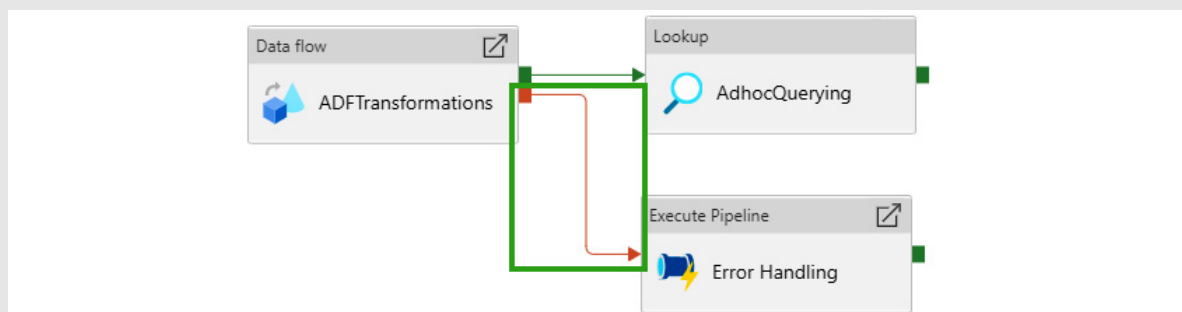


Figura 8.30 - Creación de un flujo de gestión de errores

Seleccione la actividad Ejecutar pipeline y vincule el flujo de fallos (la línea naranja) de cualquiera de las otras actividades a ella. Esta nueva actividad de Ejecutar Pipeline podría ser un pipeline completo en sí mismo, como con los pipelines de transformación que vimos anteriormente en el capítulo, o podría ser una simple actividad independiente para sólo registrar los registros de error. También puede configurarse para insertar los detalles de los errores en una base de datos, de modo que podamos analizarlos más tarde utilizando los conocidos scripts SQL.

ADF Sink también proporciona opciones para escribir automáticamente las líneas de error en un data store externo, como un blob store. Esta es otra opción conveniente que ayuda a analizar los errores de forma asíncrona. Puede configurarse mediante la opción Error row handling settings en la pestaña Settings de la actividad Sink, como se muestra en la siguiente imagen.

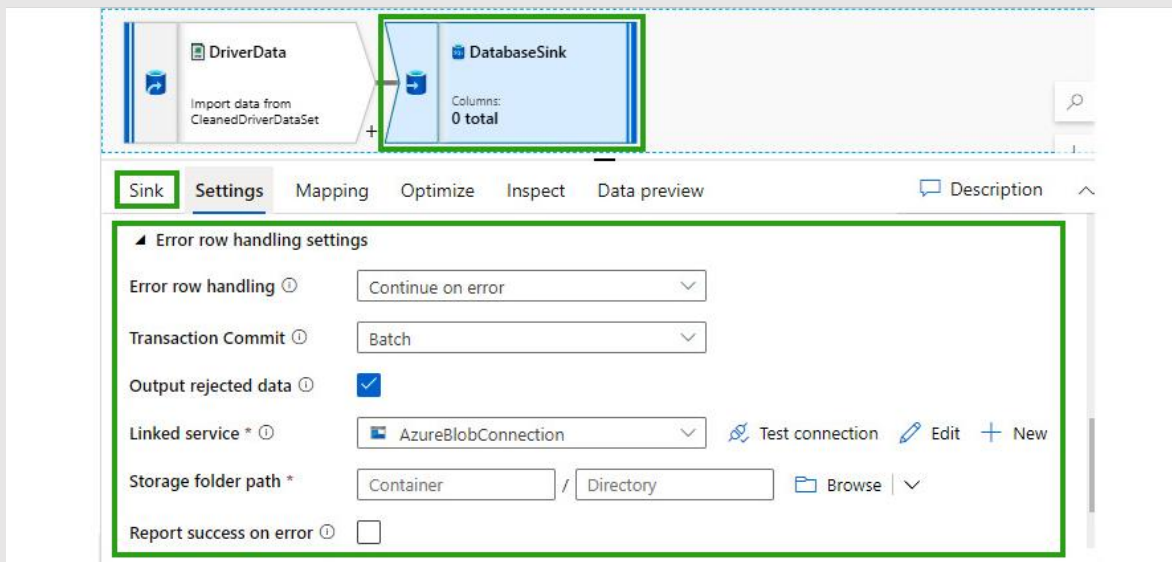


Figura 8.31 - Redirigir las líneas de error al blob storage

Veamos ahora las funciones Pivot y Unpivot, que se utilizan para normalizar y desnormalizar las tablas.

8.12. Normalización y desnormalización de valores

Ya hemos visto la actividad ADF Flatten, que ayuda a desnormalizar los datos. Hay otras dos transformaciones que ayudan a normalizar y desnormalizar los conjuntos de datos: Pivot y Unpivot. Veámoslas en detalle.

8.12.1. Desnormalización de valores con Pivot

Supongamos que tiene una tabla con una columna normalizada para almacenar los valores de la ciudad, pero por motivos de información, quiere tener una columna por ciudad en sus tablas. En este caso, puede utilizar la función Pivot para desnormalizar la tabla. La función Pivot toma los valores de la fila única y los convierte en columnas de la tabla. A continuación se muestra un ejemplo de cómo pivotar las tablas:

1. Consideremos la siguiente tabla de ejemplo:

DriverId	123	FirstName	abc	MiddleName	abc	LastName	abc	City	abc	Gender	abc	Salary	123
200		Alice		NULL		Hood		New York		Female		4100	
201		Bryan		M		Williams		New York		Male		4000	
202		Catherine		Goodwin		NULL		California		Female		4300	
203		Daryl		NULL		Jones		Florida		Male		5500	
204		Jenny		Anne		Simons		Arizona		Female		3400	
203		Daryl		NULL		Jones		Florida		Male		5500	

Figura 8.32 - Tabla de ejemplo antes de pivotar

2. Seleccione la actividad Pivot en el panel del Data Flow del ADF, y en la pestaña Group by (Agrupar por), especifique Gender (Género), como se ilustra en la siguiente captura de pantalla:

The screenshot shows the Data Flow Designer interface. At the top, a pipeline is visible with three activities: 'DriverCSV12' (Import data from DriverCSVSource), 'PivotOnCity' (Columns: 1 total), and 'UnpivotOnCity' (Unpivots columns into row values and ungroups columns). The 'PivotOnCity' activity is highlighted with a green box. Below the pipeline, the 'Pivot settings' tab is selected. The 'Output stream name' is 'PivotOnCity', and the 'Incoming stream' is 'DriverCSV12'. Under the '1. Group by' section, the 'Columns' list contains 'trim({ Gender})' and the 'Name as' field contains 'Gender'. Both the '1. Group by' section and the 'Columns' list are highlighted with green boxes.

Figura 8.33 - Configuración de Agrupar por para la operación de pivote

3. En la pestaña Pivot key, especifique City como clave de pivote, como se ilustra en la siguiente captura de pantalla:

1. Group by **2. Pivot key** 3. Pivoted columns

Pivot key * abc City

Value

Enter value (optional)... +

☐ Null value

Figura 8.34 - Pestaña de clave pivotante de la función Pivot

4. Y por último, especifique cualquier agregación que necesite, junto con el prefijo a utilizar para los nombres de las columnas, como se muestra en la siguiente captura de pantalla:

Pivot settings Optimize Inspect Data preview

Output stream name * PivotOnCity ? Help Learn more

Incoming stream * DriverCSV12

1. Group by 2. Pivot key **3. Pivoted columns**

Column name pattern * prefix(expression prefix)middle(Pivot key value)suffix

Prefix Middle Suffix

Column arrangement * Normal Lateral

avg({ Salary}) 1.2 Avg +

Figura 8.35 - Especificación del patrón de nombres de columnas y de la agregación para la función Pivot

5. Después de la operación de pivote, la tabla tendría el siguiente aspecto:

Gender	abc	AvgArizona 1.2	AvgCalifornia 1.2	AvgFlorida 1.2	AvgNew York 1.2
Female		3400.0	4300.0	NULL	4100.0
Male		NULL	NULL	5500.0	4000.0

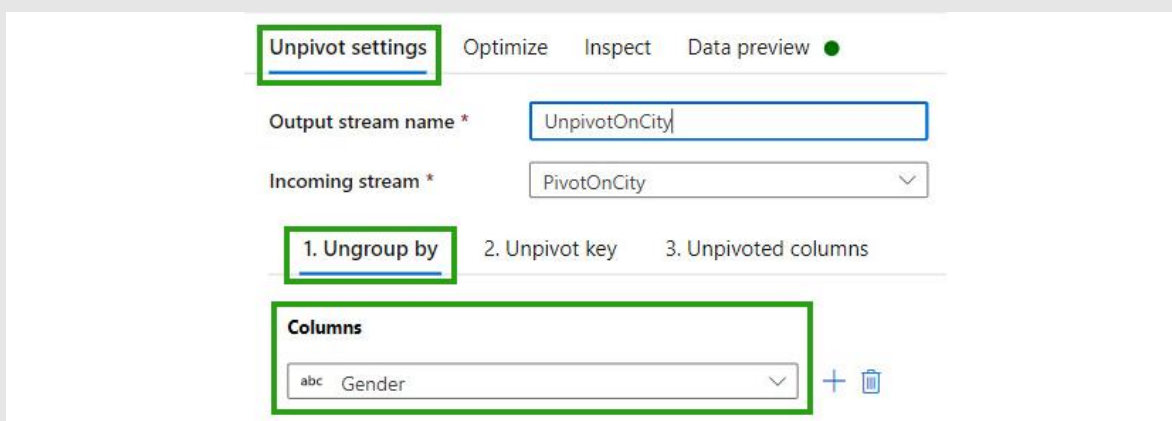
Figura 8.36 - Tabla post-pivotante

Puede ver que las filas de la tabla se han convertido en columnas aquí. A continuación, veamos la operación Unpivot.

8.12.2. Normalización de valores con Unpivot

Unpivot es la operación inversa a Pivot. Se puede utilizar para normalizar los datos; por ejemplo, si tenemos una columna para cada una de las ciudades, podemos unpivotarlas en una sola columna llamada Ciudad. Siga los siguientes pasos:

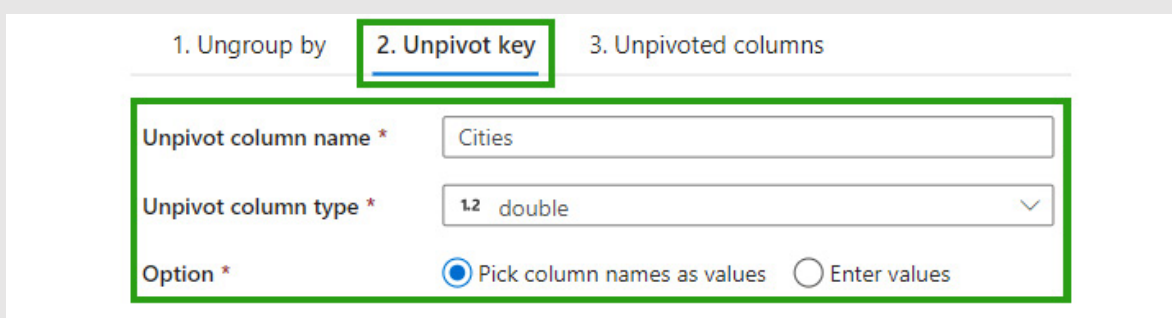
1. Para este ejemplo, especifique Género en la columna Desagrupar por, como se ilustra en la siguiente captura de pantalla:



The screenshot shows the 'Unpivot settings' tab in a software interface. It includes fields for 'Output stream name' (UnpivotOnCity) and 'Incoming stream' (PivotOnCity). Below these are three tabs: '1. Ungroup by', '2. Unpivot key', and '3. Unpivoted columns'. The '1. Ungroup by' tab is active, showing a 'Columns' field with 'Gender' selected.

Figura 8.37 - Pestaña Ungroup By para la operación Unpivot

2. Añada un nuevo nombre y tipo para la columna unpivotada en la pestaña Unpivot key, como se ilustra en la siguiente captura de pantalla:



The screenshot shows the '2. Unpivot key' tab. It contains three fields: 'Unpivot column name' (Cities), 'Unpivot column type' (double), and 'Option' (Pick column names as values).

Figura 8.38 - Especificación de Unpivot key para la operación Unpivot

3. En la pestaña de Columnas Unpivotadas, puede especificar las operaciones de agregación, como se ilustra en la siguiente captura de pantalla:

1. Ungroup by
2. Unpivot key
3. Unpivoted columns

Column arrangement *

Normal

Lateral

Drop rows with null ⓘ ☒

Columns *

Column name	Type	
AvgSalary	1.2 double	+

Figura 8.39 - Pestaña de columnas unpivotadas de la operación Unpivot

4. Después de la operación Unpivot, la tabla de salida tendrá el siguiente aspecto:

Gender	Cities	AvgSalary
Female	AvgArizona	3400.0
Female	AvgCalifornia	4300.0
Female	AvgNew York	4100.0
Male	AvgFlorida	5500.0
Male	AvgNew York	4000.0

Figura 8.40 - Tabla unpivotada

Ahora que has aprendido a desnormalizar y normalizar utilizando las operaciones Pivot y Unpivot, vamos a ver a continuación las transformaciones de Scala.

8.13. Transformación de datos utilizando Scala

Todos los ejemplos de transformación de Spark que vimos en las secciones iniciales de este capítulo estaban en Scala, así que ya sabes cómo transformar datos usando Scala. Como el aprendizaje de Scala no está en el ámbito de este libro, no profundizaremos en los aspectos del lenguaje Scala. Si estás interesado en aprender Scala, puedes encontrar algunos buenos recursos aquí: <https://www.scala-lang.org/>.

8.14. Realizando Análisis Exploratorio de Datos (EDA)

La exploración de datos es mucho más fácil desde el interior de Synapse Studio, ya que proporciona opciones fáciles de un solo clic para mirar en varios formatos de datos. Veamos algunas de las opciones disponibles para la exploración de datos utilizando Spark, SQL y ADF/Synapse pipelines.

8.14.1. Exploración de datos mediante Spark

Desde Synapse Studio, puede hacer clic con el botón derecho en el archivo de datos y seleccionar Cargar en DataFrame, como se muestra en la siguiente captura de pantalla:

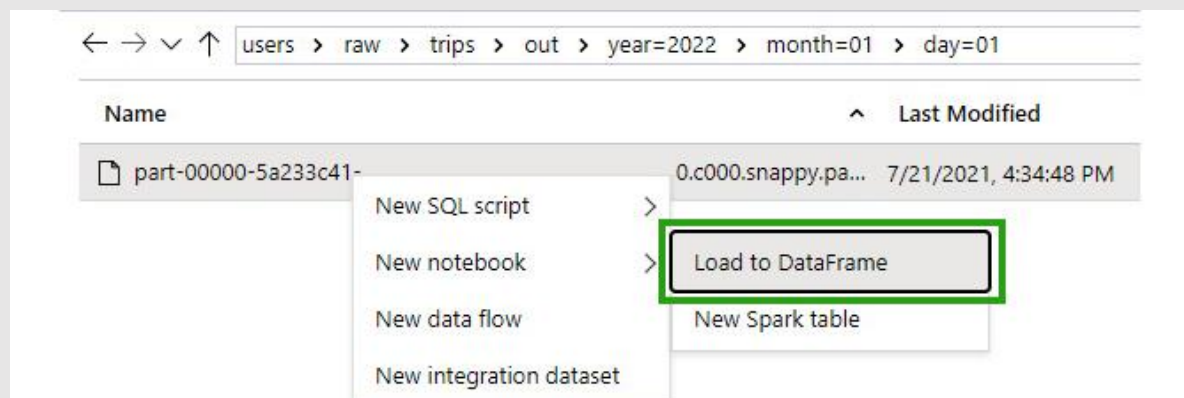


Figura 8.41 - Lanzamiento de un DataFrame desde el archivo de datos de Synapse

Una vez que haga clic en Cargar en DataFrame, Synapse crea un nuevo notebook, como se muestra en la siguiente captura de pantalla. Después de eso, todo lo que tiene que hacer es hacer clic en el icono Ejecutar (el pequeño símbolo de triángulo) para ver el contenido del archivo, que se muestran en la siguiente captura de pantalla:

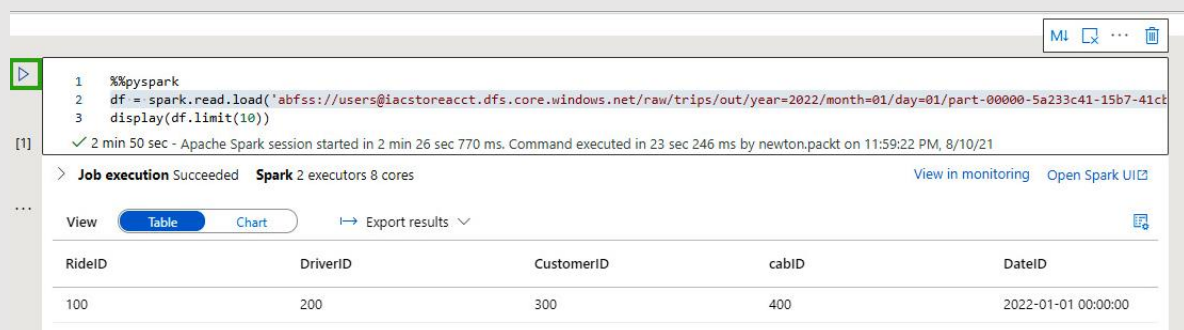


Figura 8.42 - Exploración mediante Spark Load to DataFrame

Esta es una forma sencilla de explorar el contenido de un archivo. Veamos a continuación cómo hacer lo mismo mediante SQL.

8.14.2. Exploración de datos mediante SQL

Synapse SQL también proporciona opciones similares para explorar los datos. Puede seleccionar el archivo, hacer clic en Nuevo script SQL y, a continuación, elegir Seleccionar TOP 100 filas, como se muestra en la siguiente captura de pantalla:

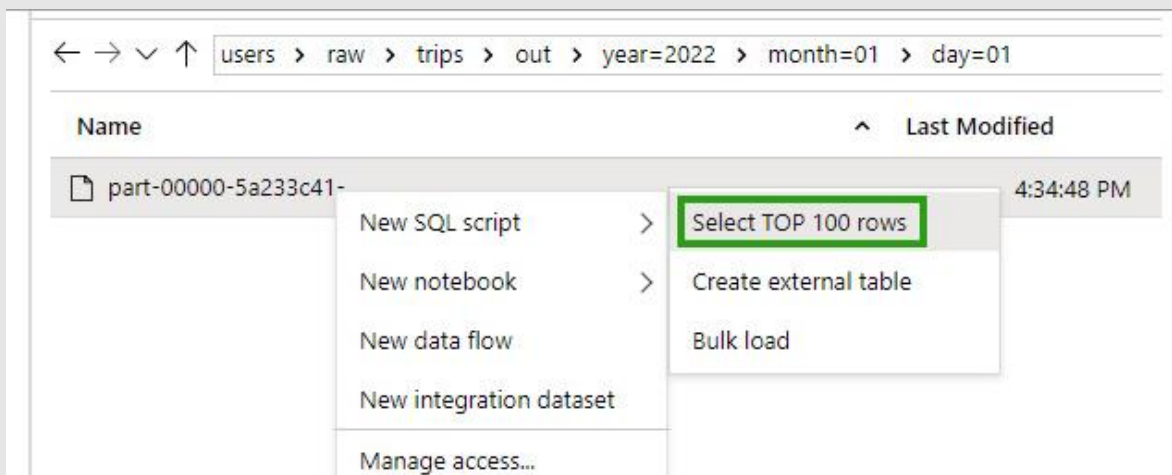


Figura 8.43 - Lanzamiento de un script SQL para explorar los datos del archivo de datos de Synapse

Una vez que haga clic en la opción Select TOP 100 rows, Synapse abre un nuevo script SQL, como se muestra en la siguiente captura de pantalla. Ahora, sólo tiene que hacer clic en Ejecutar para obtener las 100 primeras filas:

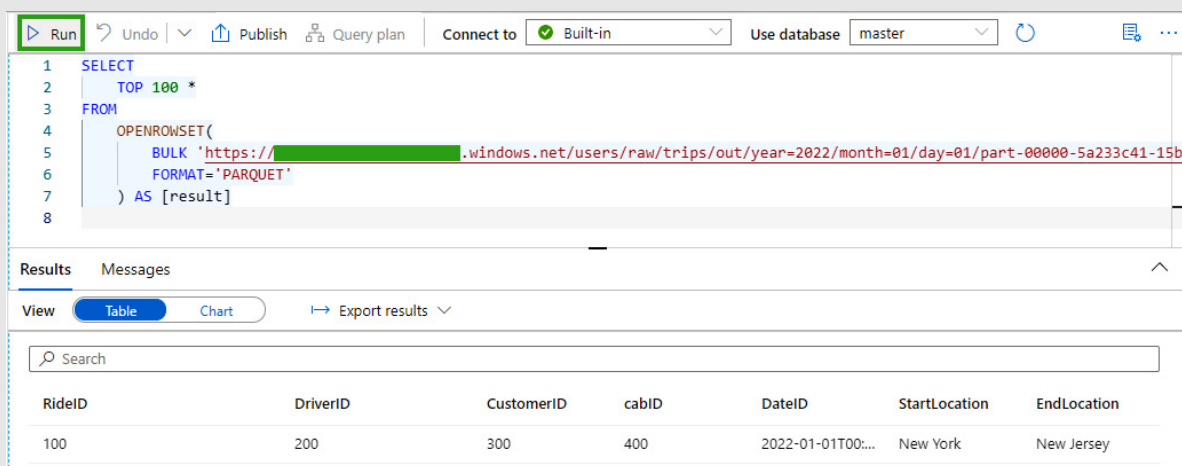


Figura 8.44 - Muestra del script SQL que Synapse lanza automáticamente para ayudarnos a explorar los datos

A continuación, veamos cómo lograr esto en ADF.

8.14.3. Exploración de datos con ADF

ADF proporciona una pestaña de vista previa de datos que funciona cuando tenemos la configuración de depuración del flujo de datos activada. Una vez que tenemos la depuración activada, un pequeño clúster de Azure Databricks (llamado ADF Integration Runtime) se ejecuta entre bastidores y obtiene datos reales para que podamos explorar y ajustar rápidamente nuestras transformaciones y pipelines. Un ejemplo se muestra en la siguiente captura de pantalla:

The screenshot shows the ADF Transformations interface. At the top, there's a 'Data flow debug' toggle which is turned on. Below it, a pipeline diagram shows a 'DriverCSV5' component connected to a 'NewYorkFilter' component. The 'NewYorkFilter' component shows 'Columns: 7 total'. Below the diagram, the 'Data preview' tab is selected, showing a table with 2 rows of data. The table has columns: DriverId, FirstName, MiddleName, LastName, and City. The data rows are: (200, Alice, NULL, Hood, New York) and (201, Bryan, M, Williams, New York). The interface also shows 'Filter settings', 'Optimize', and 'Inspect' tabs. The 'Data preview' tab is highlighted with a green box. The 'Data preview' tab also shows a summary of row counts: INSERT 2, UPDATE 0, DELETE 0, UPSERT 0, LOOKUP 0, and TOTAL 2.

DriverId	FirstName	MiddleName	LastName	City
200	Alice	NULL	Hood	New York
201	Bryan	M	Williams	New York

Figura 8.45 - Exploración de datos utilizando la pestaña de vista previa de datos del ADF

Como puede ver, Synapse y ADF hacen que sea muy fácil explorar los datos sin tener que salir del estudio.

Resumen

Con esto, hemos llegado al final de este interesante capítulo. Hubo un montón de ejemplos y capturas de pantalla para ayudarle a entender los conceptos. Puede ser abrumador a veces, pero la forma más fácil de seguir es abrir una sesión de Spark, SQL o ADF en vivo y tratar de ejecutar los ejemplos en paralelo.

Cubrimos muchos detalles en este capítulo, como la realización de transformaciones en Spark, SQL y ADF; las técnicas de limpieza de datos; la lectura y el análisis sintáctico de datos JSON; la codificación y decodificación; el manejo de errores durante las transformaciones; la normalización y desnormalización de conjuntos de datos; y, finalmente, un montón de técnicas de exploración de datos. Este es uno de los capítulos más importantes del temario. Ahora deberías ser capaz de construir cómodamente pipelines de datos con transformaciones que impliquen a Spark, SQL y ADF. Espero que te hayas divertido leyendo este capítulo. En el próximo capítulo exploraremos el diseño y desarrollo de una solución de procesamiento por lotes.