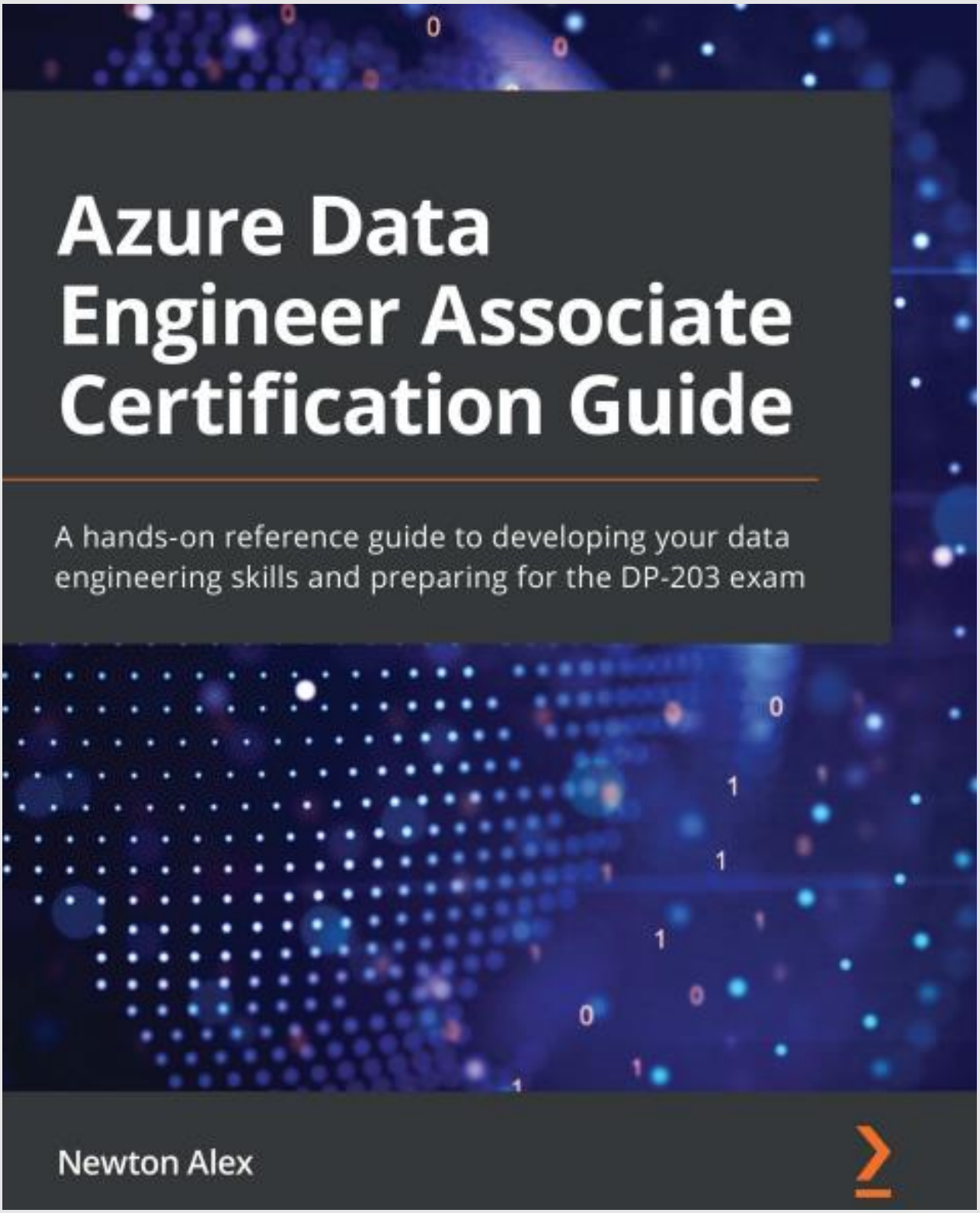
Azure Data Engineer Associate Certification Guide
A hands-on reference guide to developing your data engineering skills and preparing for the DP-203 exam
Newton Alex

Parte 2: Data Storage

Esta parte se sumerge en los detalles de los diferentes tipos de almacenamiento, las estrategias de partición de datos, los esquemas, los tipos de archivos, la alta disponibilidad, la redundancia, etc.

Esta sección comprende los siguientes capítulos:

- Capítulo 2, Diseño de una estructura de almacenamiento de datos
- Capítulo 3, Diseño de una estrategia de partición
- Capítulo 4, Diseño de la capa de servicio
- Capítulo 5, Implementación de estructuras físicas de almacenamiento de datos
- Capítulo 6, Implementación de estructuras lógicas de datos
- Capítulo 7, Implementación de la capa de servicio

Capítulo 6: Implementación de estructuras de datos lógicas

Espero que hayas disfrutado aprendiendo sobre los detalles de implementación de las estructuras de almacenamiento. En este capítulo, cubriremos los detalles de implementación de las estructuras lógicas de datos. Aprenderás a implementar conceptos avanzados de carga de datos como las slowly changing dimensions, soluciones basadas en el almacenamiento para optimizar el rendimiento de las consultas y técnicas para leer datos externos sin tener que copiarlos en el almacenamiento local.

En este capítulo cubriremos los siguientes temas:

- Construcción de una solución de datos temporales
- Construcción de una slowly changing dimension
- Construcción de una estructura lógica de carpetas
- Implementación de estructuras de archivos y carpetas para una consulta y poda de datos (pruning data) eficiente
- Creación de tablas externas

6.1. Requisitos técnicos

Para este capítulo, necesitará lo siguiente

- Una cuenta de Azure (gratuita o de pago)
- Azure CLI instalado en su estación de trabajo
- Un espacio de trabajo Synapse activo

¡Empecemos!

6.2. Creación de una solución de datos temporales

Los datos temporales se refieren a datos en puntos específicos del tiempo. La construcción de una solución temporal se ocupa de construir sistemas que puedan manejar y almacenar dichos datos basados en el tiempo. Ya hemos visto cómo utilizar la función de tablas temporales de Azure SQL para construir un sistema de este tipo en el capítulo 4, Diseño de la capa de servicio, en la sección Diseño de una solución para datos temporales. Como ya hemos explorado este tema en detalle allí, no lo repetiremos de nuevo. Por favor, consulte el Capítulo 4 para refrescar su memoria sobre las tablas temporales.

A continuación aprenderemos a implementar una dimensión de cambio lento.

6.3. Construir una slowly changing dimension

En el Capítulo 4, Diseño de la capa de servicio, aprendimos las diferentes formas de construir dimensiones de cambio lento (SCD). En esta sección, aprenderemos a implementar algunas de ellas utilizando Mapping flows de ADF/Synapse Pipelines. Implementaremos la SCD de tipo 2 ya que implica un flujo de trabajo un poco más complicado. Una vez que sepa cómo implementar uno de los SCD, la implementación de los demás será similar.

Consideremos el siguiente escenario de ejemplo:

- Tenemos una tabla de dimensión DimDriver en un Synapse SQL dedicated pool que contiene los datos del driver. Estos datos no cambian muy a menudo, por lo que es una buena opción para un SCD.

- Supongamos que los cambios en los datos del driver aparecen periódicamente como un archivo CSV en una carpeta de Azure Data Lake Gen2.

- Tenemos que construir una pipeline ADF para tomar los datos del archivo CSV y aplicarlos a la tabla DimDriver manteniendo su historial.

- Construiremos la opción SCD basada en flags en este ejemplo.

Crear un pipeline en ADF es como construir los pasos de un diagrama de flujo. Tendremos que llevar los datos a través de varios pasos para llegar al resultado final deseado. El siguiente ejemplo de SCD va a ser un poco más largo. He tratado de explicar todos y cada uno de los pasos para que sea fácil de entender. Pero, si le resulta difícil de seguir, le recomendaría que mirara primero el capítulo 8, sección Ingesta y transformación de datos: Transformación de datos mediante el uso de ADF para entender bien las actividades de ADF. Esa sección tiene muchos ejemplos de ADF más pequeños que pueden ayudarle a entender mejor el siguiente ejemplo de ADF.

Veamos los pasos necesarios para el ejemplo de SCD2. En un nivel alto, vamos a realizar los siguientes pasos:

1. Si los datos entrantes son para nuevos drivers, simplemente los añadimos directamente a la tabla DimDriver de Synapse SQL con la columna isActive puesta a 1.

2. Si los datos entrantes son una actualización de una fila existente en la tabla DimDriver, entonces tenemos que tener en cuenta dos condiciones:

    a) La primera condición actualiza las últimas filas del archivo CSV como la fila activa para los correspondientes DriverIds estableciendo la columna isActive a 1.

    b) La segunda condición actualiza todas las filas anteriores para esos DriverIds como inactivas, poniendo la columna isActive a 0.

La lógica anterior puede ejecutarse utilizando tres flujos separados dentro de un flujo de datos, uno para las nuevas filas y dos para las filas modificadas. Veamos primero el caso de las nuevas filas.

6.3.1. Actualización de nuevas filas

Estos son los pasos para construir el flujo para actualizar las nuevas filas:

- El primer paso es crear un nuevo Data flow, que puede ser utilizado para encadenar los pasos de transformación, también llamados Actividades. Haga clic en el signo + y elija la opción de Data flow en el ADF o Synapse Studio, como se muestra:

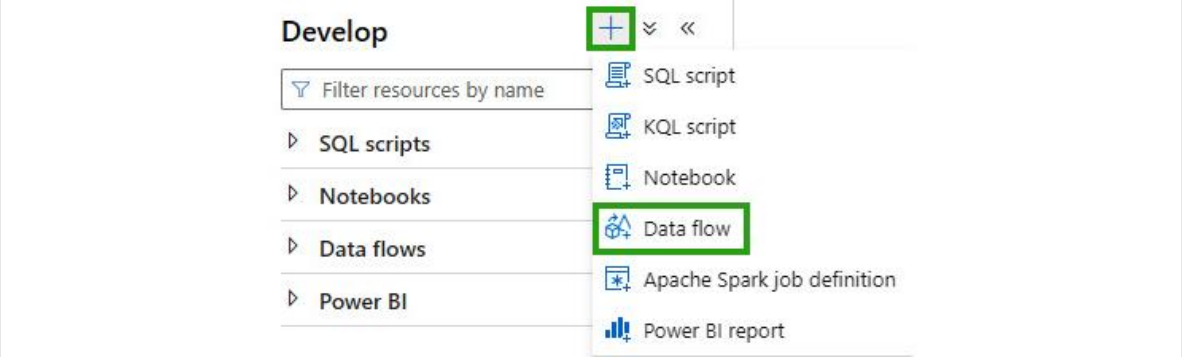

Figura 6.1 - Creación de un nuevo flujo de datos en Synapse Pipelines

- El siguiente paso es crear un dataset de origen para nuestro archivo CSV. Haga clic en el enlace Añadir origen en la región del lienzo de flujo de datos, como se muestra en la siguiente imagen.

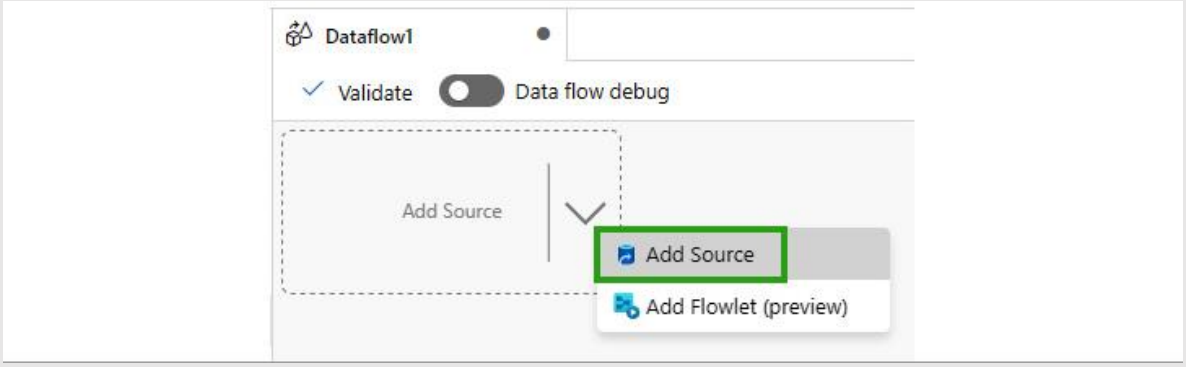

Figura 6.2 - Creación de una nueva fuente de datos

- Al hacer clic en el enlace Añadir origen, aparece la pantalla de un nuevo dataset de origen. En esa pantalla, puede crear un nuevo dataset utilizando el enlace + Nuevo, para apuntar a la carpeta CSV del driver de entrada, como se muestra a continuación:

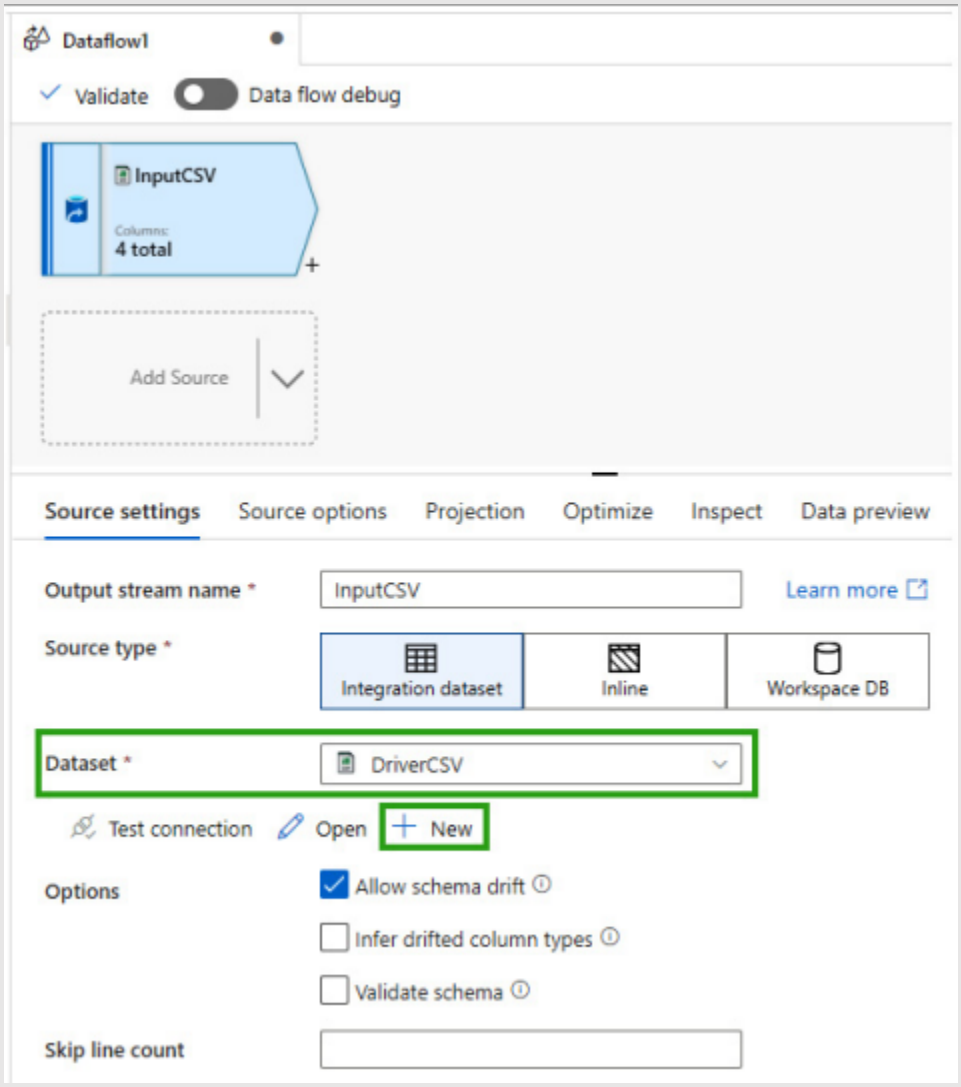

Figura 6.3 - Ejemplo de creación de un conjunto de datos de entrada

- Tenemos que crear un dataset de origen similar para encontrar el MaxSurrogateID de la tabla DimDriver de Synapse SQL. Al crear un nuevo dataset que apunte a Synapse SQL, tenemos que especificar la consulta a ejecutar para obtener el max surrogateId. Utilizaremos este valor más tarde para distinguir las filas recién añadidas de las existentes en la tabla DimDriver. A continuación se muestra una captura de pantalla de cómo especificar la consulta.

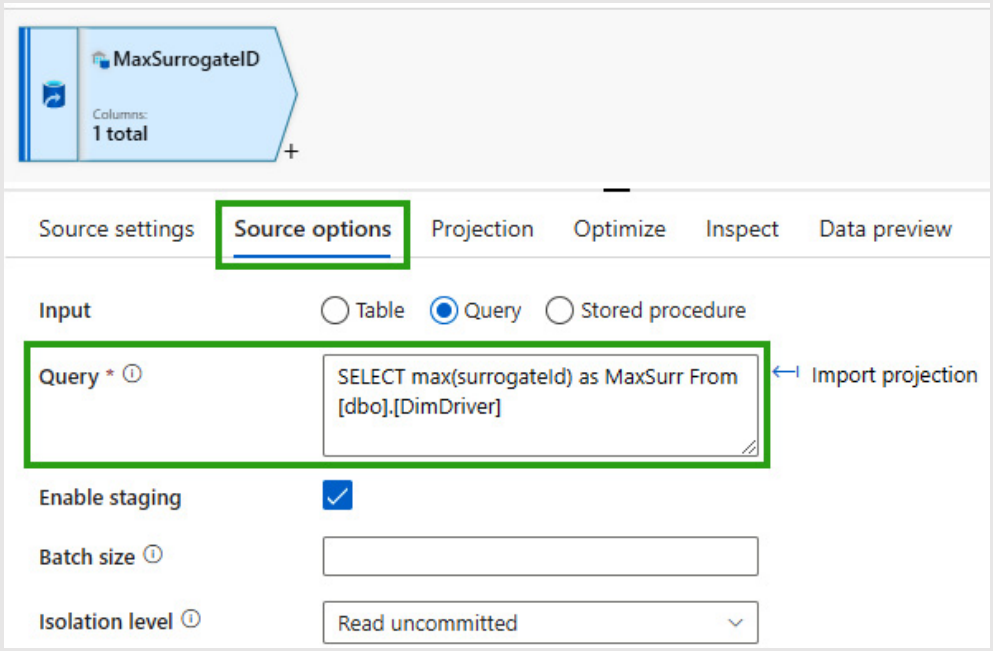

Figura 6.4 - Búsqueda del valor máximo de las claves sustitutivas

- Ahora, vamos a añadir el valor maxSurrogate recién calculado al entrante CSV driver dataset como una nueva columna utilizando la actividad Join, como se muestra en la siguiente captura de pantalla:

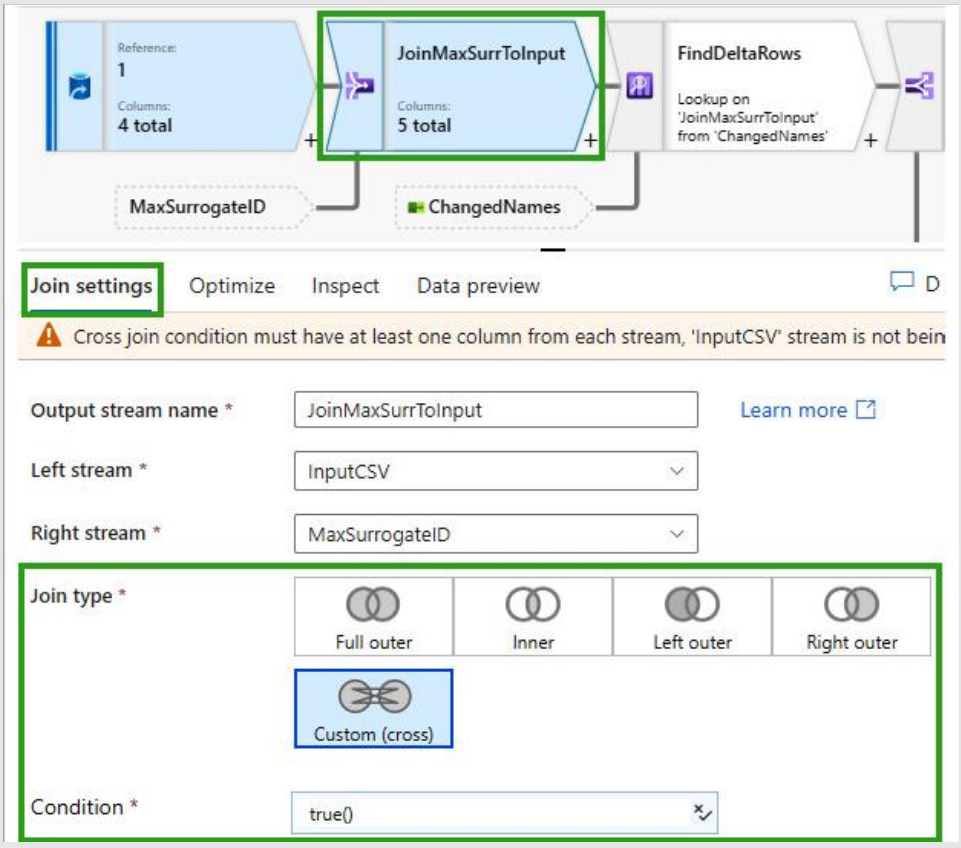

Figura 6.5 - Añadir la columna MaxSurrogateID a los datos CSV de entrada

- Ahora que tenemos el MaxSurrogateID junto con nuestros datos del Driver CSV, necesitamos encontrar la lista de todas las filas modificadas frente a las filas recién añadidas. Podemos usar una actividad Lookup con LEFT OUTER JOIN entre el Driver CSV y la tabla DimDriver de Synapse SQL para lograr esto. Aquí hay una captura de pantalla de cómo lograr esto:

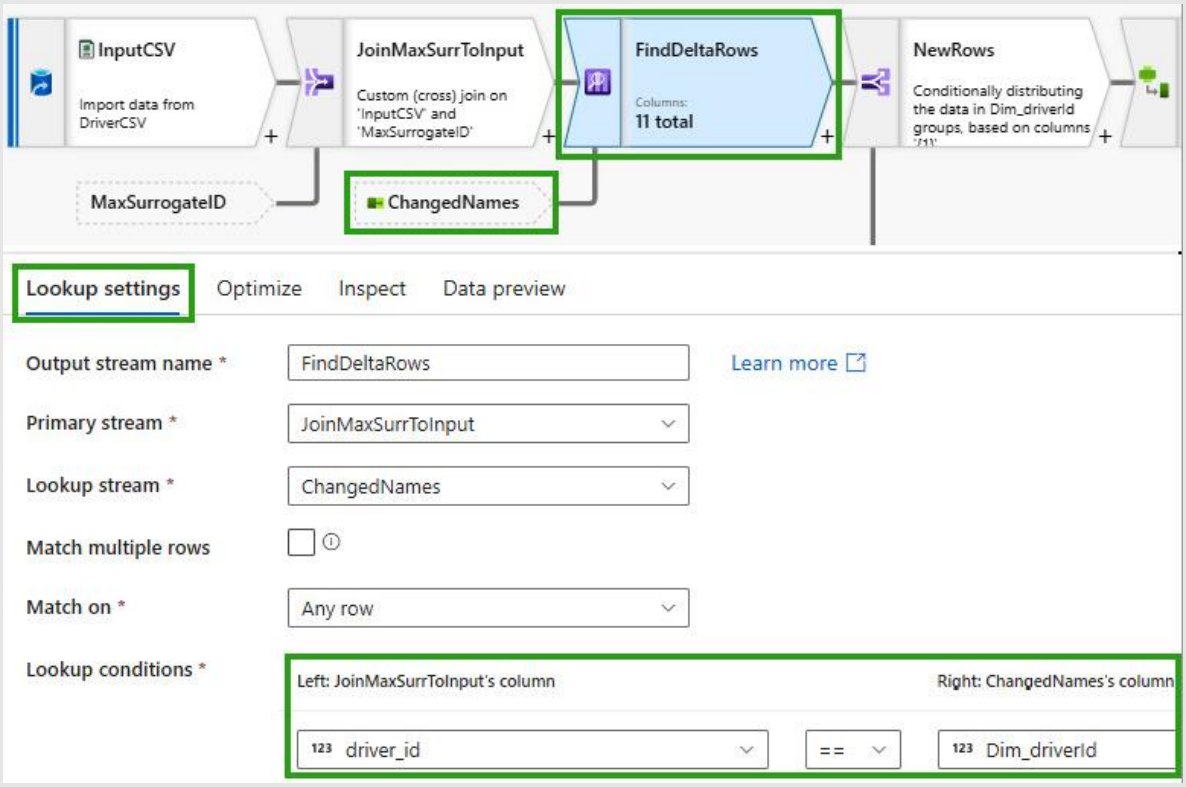

Figura 6.6 - Usando LEFT OUTER JOIN para obtener las filas actualizadas

- En la configuración de Lookup anterior, habría utilizado una tabla DimDriver ligeramente modificada llamada ChangedNames para la unión. Esta tabla se obtiene renombrando los nombres de las columnas de la tabla DimDriver de Synapse SQL. Esto es necesario ya que las tablas de la izquierda y de la derecha para la unión tendrán exactamente los mismos nombres de columna. Así es como podemos renombrar los nombres de las columnas usando una actividad Select.

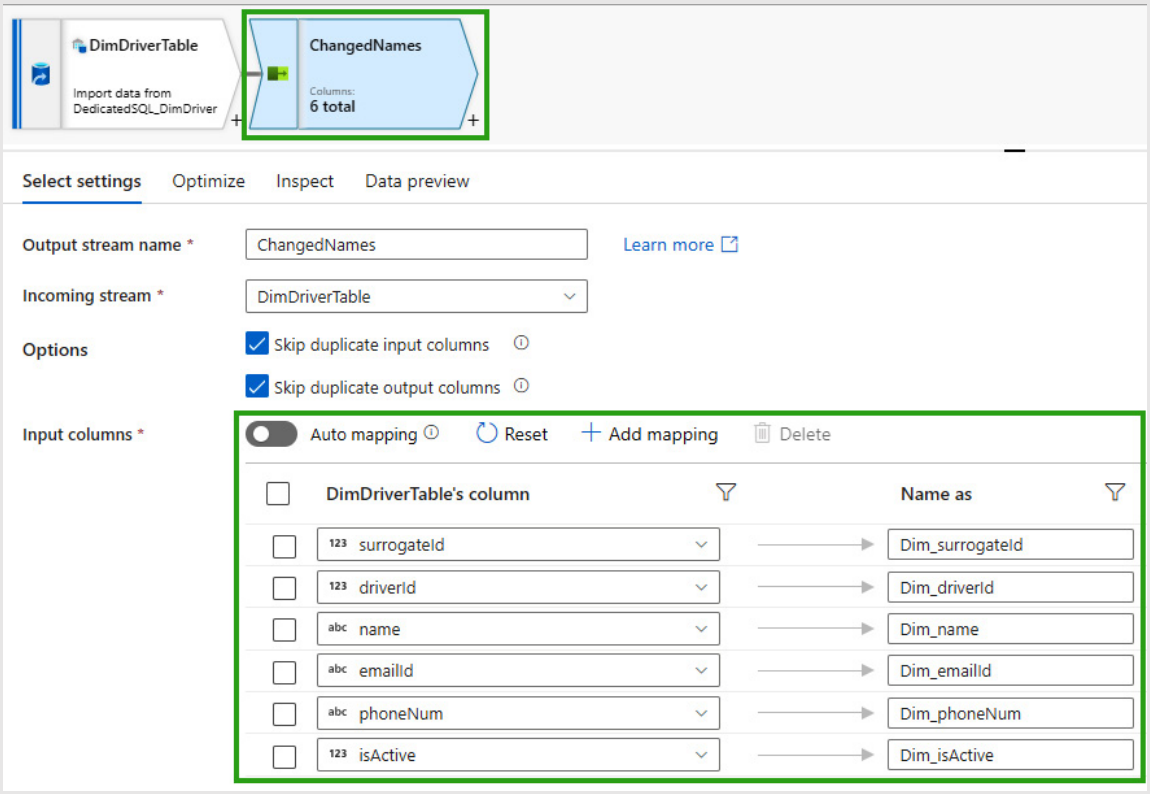

Figura 6.7 Modificación de los nombres de las columnas mediante la actividad Select

- Ahora que hemos identificado las filas a actualizar, vamos a dividir el data flow en dos ramas utilizando Conditional split: una que se encarga de los valores recién añadidos y otra que actualiza las filas modificadas:

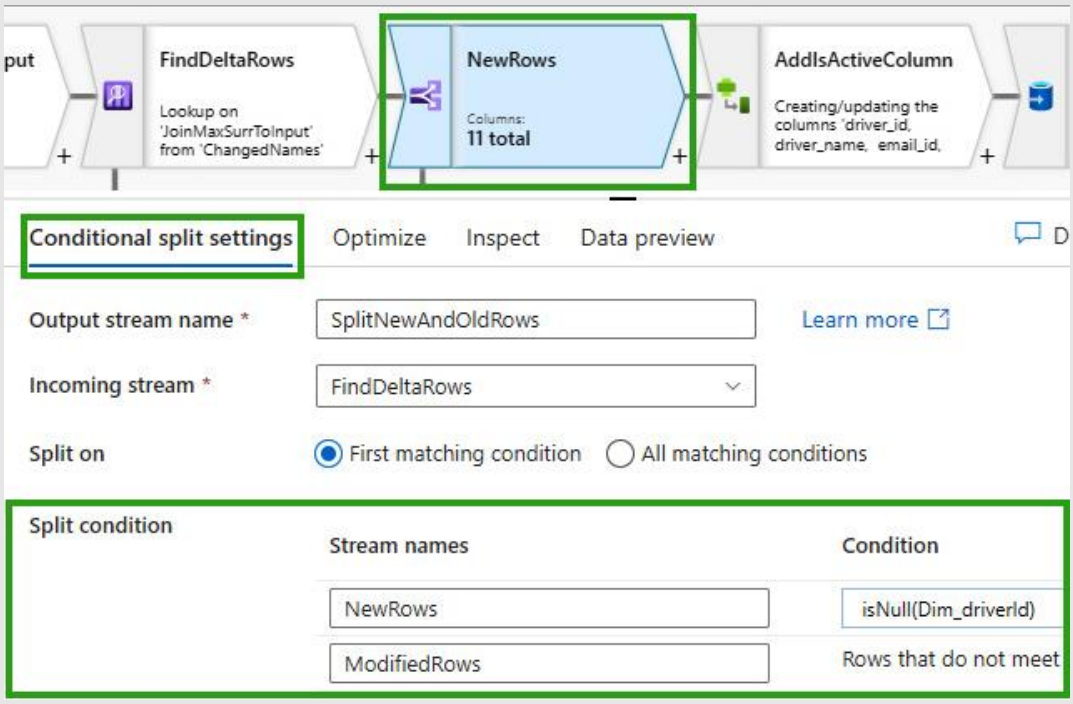

Figura 6.8 - Dividir el flujo en dos ramas - filas nuevas y modificadas

- En la rama NewRows, basta con añadir una columna (isNewActive=1), utilizando la actividad Derived Column. Cambiaremos este nombre a isActive justo antes de actualizarlo a las tablas de Synapse SQL en la última actividad Sink.

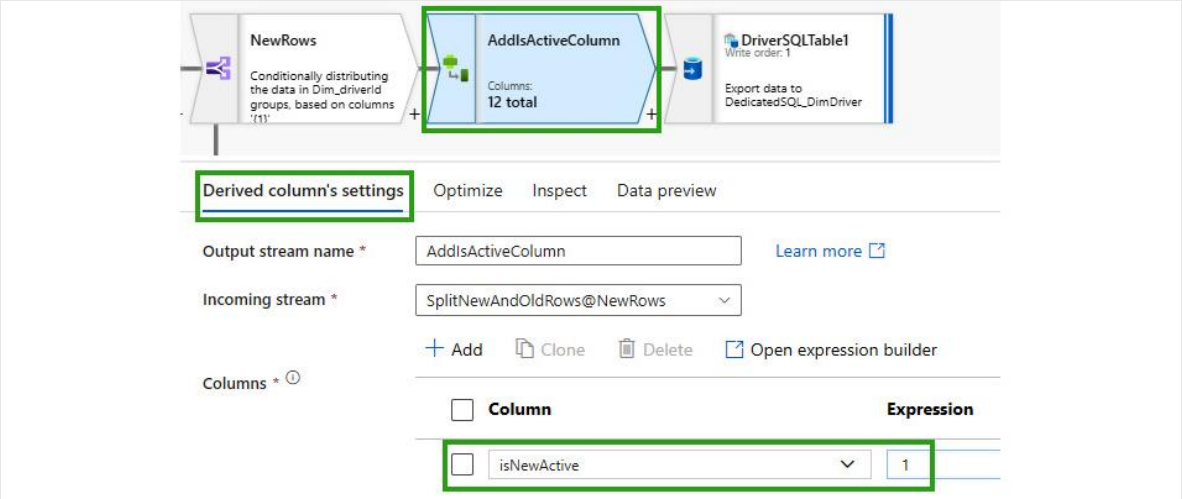

Figura 6.9 - Añadir la columna isActive a las filas de entrada

- Ahora envíe todas estas nuevas filas con la columna isNewActive renombrada como isActive, al destino usando una actividad Sink. Estamos utilizando una bandera temporal isNewActive para evitar colisiones de nombres de columnas. Tendrá que desactivar el mapping automático y asegurarse de que el mapping de campos se realiza correctamente en el SQL Sink, como se muestra en la siguiente imagen:

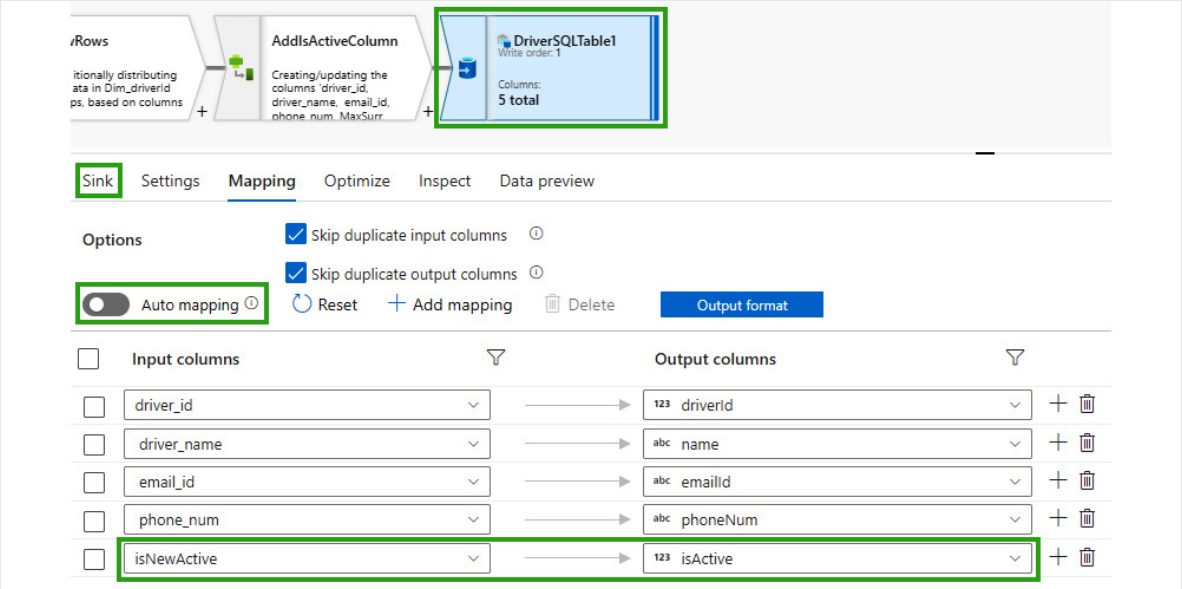

Figura 6.10 - Un ejemplo de escritura en el Sink, que es la tabla SQL de Synapse en nuestro caso

Con esto hemos completado el flujo para las filas recién añadidas. Ahora veamos los pasos para la rama de filas modificadas, aquella en la que dividimos condicionalmente la rama en función de NuevasFilas vs. FilasModificadas (Figura 6.8).

6.3.2. Actualización de las filas modificadas

Estos son los pasos para actualizar las filas modificadas.

1. Para la rama de filas modificadas, hay dos subtareas. Las filas recién actualizadas pueden añadirse directamente con isActive=1 y las filas antiguas para los correspondientes DriverIds deben actualizarse con isActive=0. Por lo tanto, tenemos que dividir el conjunto de datos ModifiedRows en dos ramas utilizando la opción New branch. Haga clic en el pequeño signo + después del cuadro de actividad para obtener la opción Nueva rama, como se muestra:

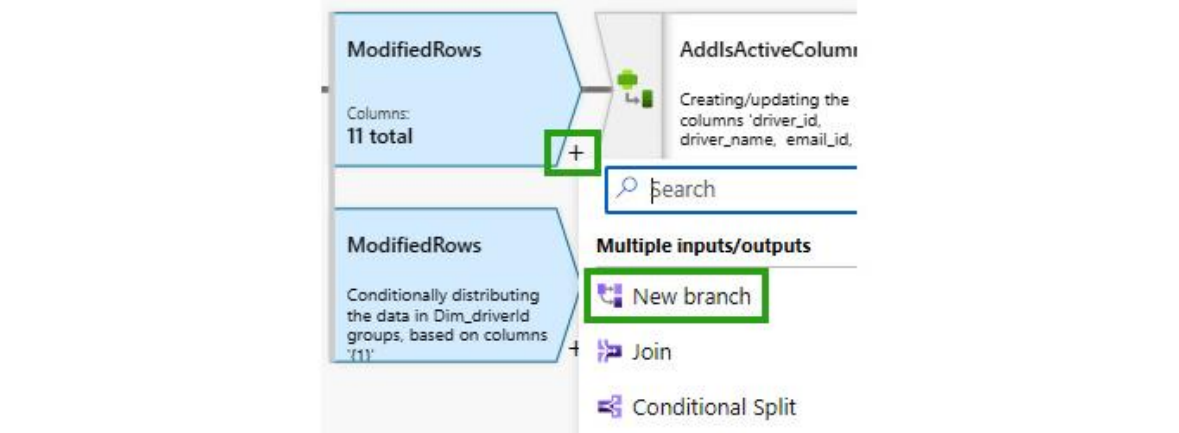

Figura 6.11 creando una nueva rama

2. En una de estas ramas, simplemente añada la columna isActive con el valor 1 y escríbala en la tabla Sink de Synapse DimDriver, de forma similar a lo que hicimos anteriormente para las nuevas filas (consulte la Figura 6.9) . Esto debería encargarse de la primera subtarea de filas modificadas.

3. Ahora, tenemos que hacer que isActive=0 para las filas anteriores con el mismo driverID. Para ello, podemos añadir una actividad de columna derivada para añadir una columna isActive a todas las filas, seguida de una actividad de filtro para filtrar sólo las filas antiguas antes de escribirlas en el sink. De esta manera, la última fila para ese DriverId, que se actualizó en el paso anterior no se modifica. Aquí también, utilizo una columna temporal llamada isOldRowActive para evitar el conflicto de nombres de columnas. Ésta se cambiará a isActive justo antes de escribirla en el sink SQL DimDriver.

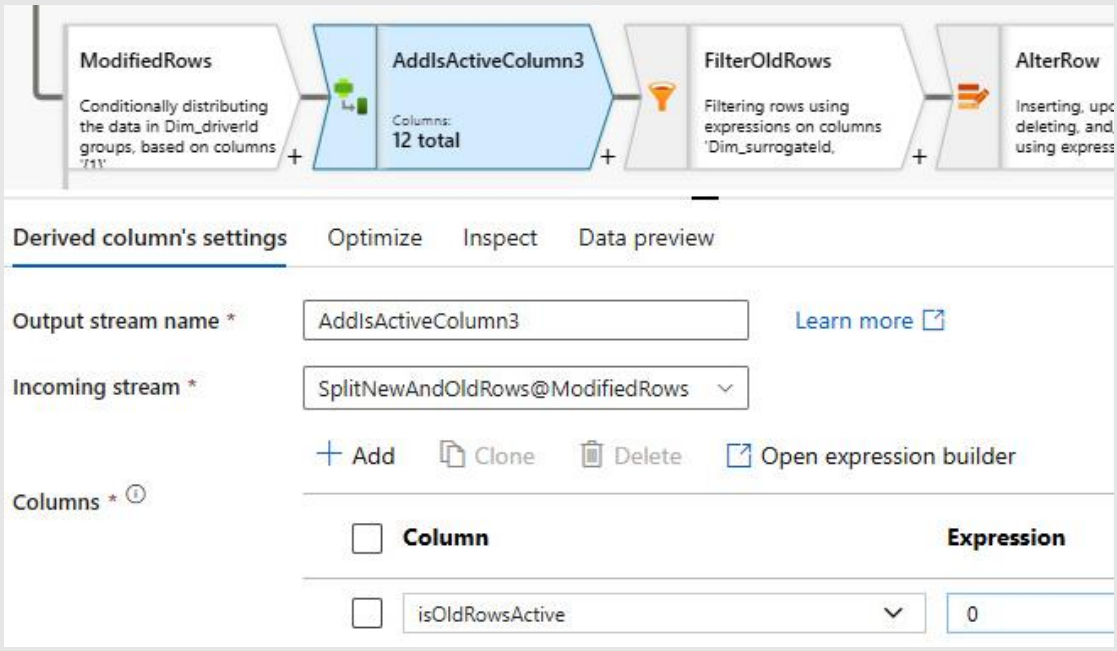

Figura 6.12 Añadir una columna extra usando una columna derivada

4. Ahora filtre las filas antiguas, usando la condición Dim_surrogateId <= MaxSurr, como se muestra:

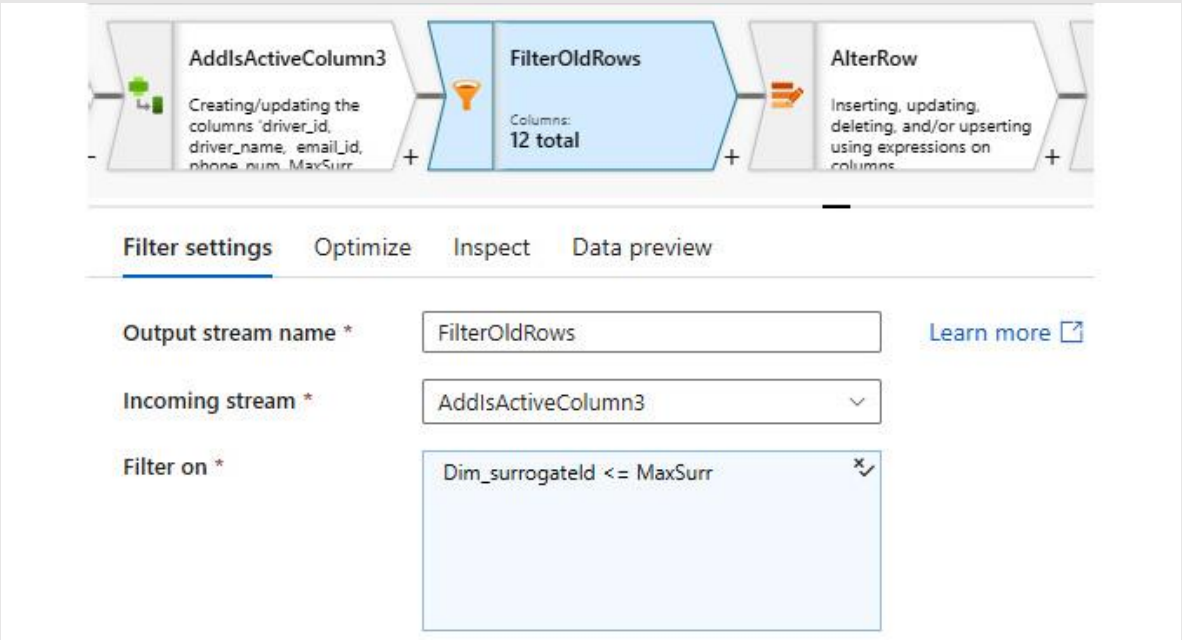

Figura 6.13 - Filtrar sólo las filas antiguas

5. Utilice un paso de Alter row para actualizar sólo las filas cuyos surrogates IDs son menores que el valor max surrogate en el Synapse SQL utilizando una actividad Sink:

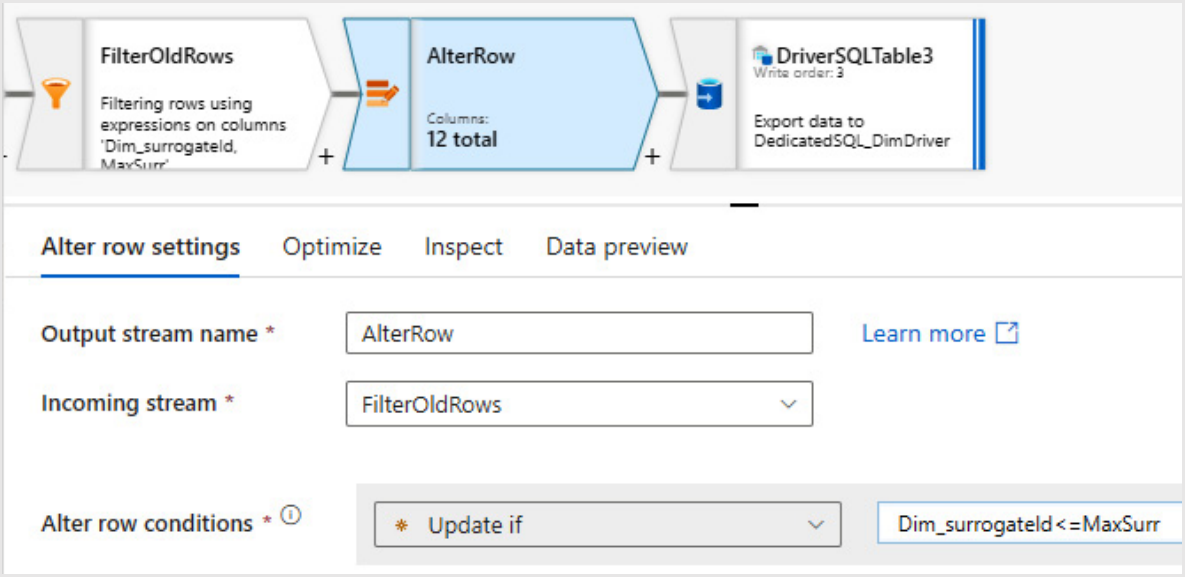

Figura 6.14 - Alterar el número de filas usando el valor MaxSurr como referencia

6. Así es como se verá la configuración de Sink:

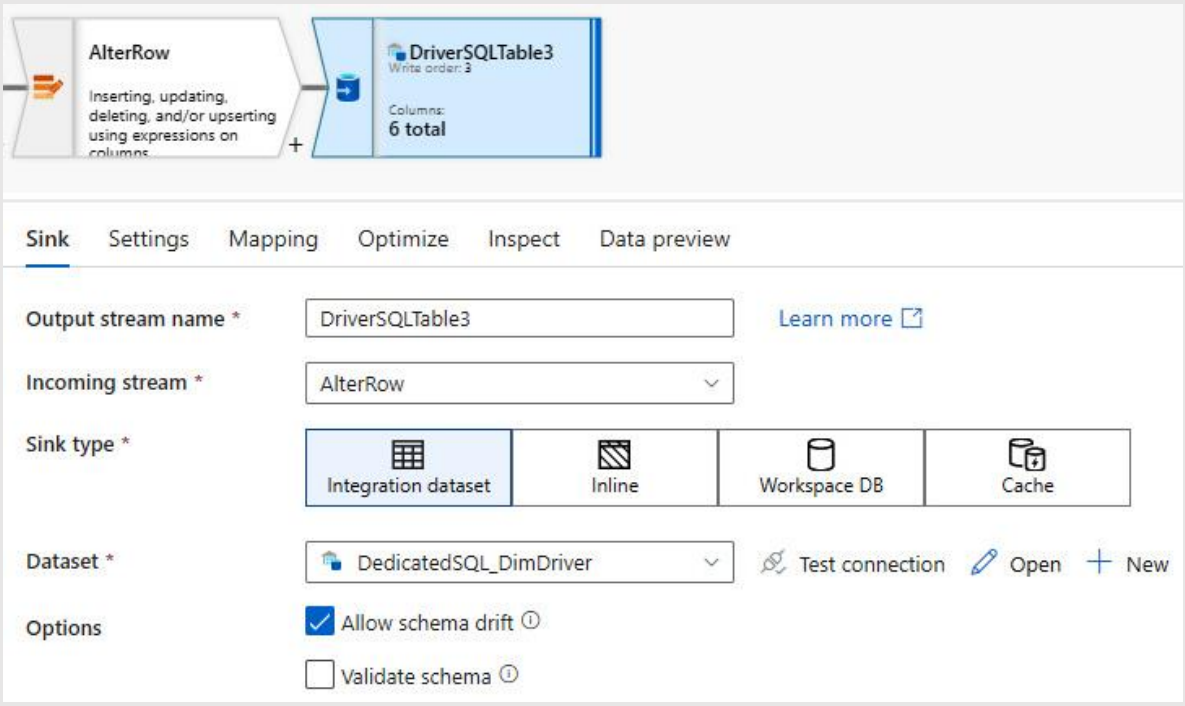

Figura 6.15 Configuración de Synapse SQL Sink

7. Por último, configure un pipeline y actívelo para ejecutar el data flow periódicamente. Su pipeline ADF debería ser similar al siguiente:

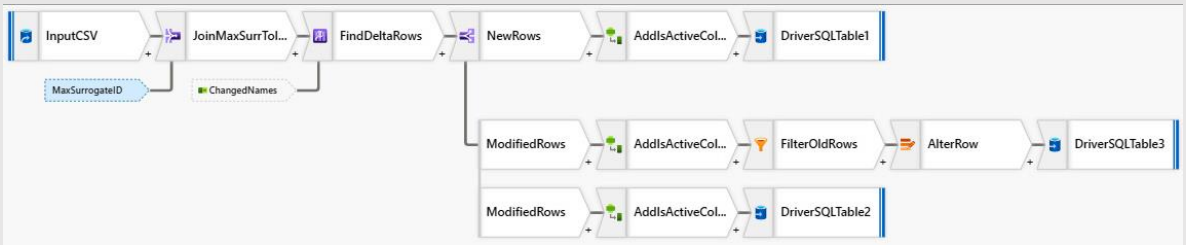

Figura 6.16 - Ejemplo de pipeline ADF para SCD 2

Así es como se construyen los SCDs utilizando Synapse Pipelines (o ADF). Puede que le resulte difícil seguir todos los pasos la primera vez. Así que intente leer esta sección de nuevo después de completar el Capítulo 8, Ingesta y transformación de datos, que tiene más ejemplos de ADF/Synapse Pipelines.

Ahora, vamos a aprender cómo implementar estructuras lógicas de carpetas.

6.4. Construir una estructura lógica de carpetas

Aprendimos acerca de las estructuras de carpetas eficientes en el Capítulo 2, Diseño de una estructura de almacenamiento de datos, donde exploramos las mejores prácticas para el almacenamiento de datos para el procesamiento por lotes y los escenarios de streaming. La regla general es almacenar los datos en una estructura jerárquica de carpetas de fechas, con la parte de la fecha añadida hacia el final, como se muestra aquí:

```
{Region}/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/
```

Podemos tener más carpetas intermedias en la ruta de la carpeta en función de nuestros requisitos empresariales. Consulta el capítulo 2, Diseño de una estructura de almacenamiento de datos, para refrescar tu memoria sobre el diseño de estructuras de carpetas eficientes.

Para crear un contenedor en Azure Data Lake Gen 2, puedes utilizar el siguiente comando a través de Azure CLI:

```
az storage fs create -n <container name> --account-name <account name> --auth-mode login
```

Una vez que hayas creado un contenedor, puedes crear fácilmente carpetas en él utilizando el siguiente comando:

```
az storage fs directory create -n <folder path> -f <container name> --account-name <account name>
--auth-mode login
```

También puedes utilizar herramientas de interfaz de usuario como el portal de Azure Storage o Azure Storage Explorer para crear la estructura de carpetas.

Dado que ya hemos cubierto este tema en detalle en los capítulos anteriores, vamos a pasar a ver qué estructuras de datos podemos utilizar para una consulta y poda (pruning) eficaz.

6.5. Implementación de estructuras de archivos y carpetas para una consulta y poda de datos (data pruning) eficaz

Los conceptos que exploramos en la sección anterior también se aplican aquí. Una vez que tenemos una estructura jerárquica de carpetas basada en fechas, el rendimiento de las consultas puede mejorarse mediante la partición de los datos. Si dividimos los datos en particiones y nos aseguramos de que las particiones se almacenan en diferentes estructuras de carpetas, las consultas pueden omitir el análisis de las particiones irrelevantes. Este concepto, como ya sabemos, se llama poda de datos (data pruning).

Otra ventaja del particionamiento es la mayor eficiencia en la carga y eliminación de datos al realizar el cambio y la eliminación de particiones. Aquí, en lugar de leer cada fila y actualizarla, se pueden añadir o eliminar enormes particiones de datos con simples operaciones de metadatos. En el Capítulo 2, Diseño de una estructura de almacenamiento de datos, ya se trataron ejemplos de cómo las consultas pueden beneficiarse de la poda de datos omitiendo la lectura de particiones innecesarias. En esta sección, aprenderemos a mejorar el rendimiento de la carga y descarga de datos mediante el cambio de particiones.

NOTA

Las particiones tienen que estar perfectamente alineadas en los límites para la conmutación de particiones.

Consideremos la misma tabla Fact de ejemplo que hemos estado utilizando hasta ahora:

```
CREATE TABLE dbo.TripTable
(
   [tripId] INT NOT NULL,
   [driverId] INT NOT NULL,
   [customerId] INT NOT NULL,
   [tripDate] INT,
   [startLocation] VARCHAR (40),
   [endLocation] VARCHAR (40)
)
WITH
(
   CLUSTERED COLUMNSTORE INDEX,
   DISTRIBUTION = HASH ([tripId]),
   PARTITION ([tripDate] RANGE RIGHT FOR VALUES
      ( 20220101, 20220201, 20220301 )
  )
)
```

Supongamos que sólo necesitamos almacenar los datos de 3 meses. Nuestra tabla Fact, dbo.TripTable, contiene los datos de 20220101, 20220201 y 20220301. Ahora, vamos a aprender cómo eliminar los datos del primer mes y añadir los datos del último mes, 20220401, a la tabla.

6.5.1. Eliminar una partición antigua

Para eliminar una partición, necesitamos crear una tabla ficticia que tenga la misma estructura que la tabla original y luego cambiar la partición por la tabla ficticia. Esta sección le mostrará cómo cambiar la partición 20220101.

Cree una tabla ficticia que contenga la partición que necesita ser cambiada, como sigue:

```
CREATE TABLE dbo.TripTable_20220101
WITH
(
   CLUSTERED COLUMNSTORE INDEX,
   DISTRIBUTION = HASH ([tripId]),
   PARTITION ([tripDate] RANGE RIGHT FOR VALUES (20220101) )
  )
AS
SELECT * FROM dbo.TripTable WHERE 1=2 ;
```

Obsérvese que para cambiar las tablas temporales, podemos tener sólo la partición que pensamos cambiar. 20220101 es la segunda partición ya que la primera partición corresponderá a todos los valores anteriores a 20220101, que en nuestro caso estarían vacíos. Ejecuta el comando ALTER TABLE, como se muestra en el siguiente bloque de código, para intercambiar la partición:

```
ALTER TABLE dbo.TripTable SWITCH PARTITION 2 TO dbo.TripTable_20220101 PARTITION 2 WITH (TRUNCATE_TARGET = ON);
```

Ahora, dbo.TripTable contendrá 0 filas para la partición 2, que corresponde a 20220101.

A continuación, vamos a añadir una nueva partición, 20220401, a la tabla.

6.5.2. Añadir una nueva partición

Para añadir nuestra nueva partición, necesitamos dividir la última partición en dos particiones. Podemos utilizar el siguiente comando SPLIT para hacerlo:

```
ALTER TABLE dbo.TripTable SPLIT RANGE (20220401);
```

Esto dividirá la última partición en 20220301 y 20220401.

NOTA

La división no funcionará si la partición contiene datos. Por lo tanto, tendremos que intercambiar temporalmente la última partición a una tabla ficticia, como hicimos anteriormente en la sección Eliminar una partición antigua. Una vez que la última partición esté vacía, divida esa partición usando el comando split y luego agregue la partición intercambiada temporalmente de vuelta a la tabla original.

Una vez que hemos creado la nueva partición para 20220401, debemos crear una tabla ficticia con la misma alineación de la partición para cargar los datos de 20220401. El siguiente fragmento de código hace esto:

```
CREATE TABLE dbo.TripTable_new
WITH
(
   CLUSTERED COLUMNSTORE INDEX,
   DISTRIBUTION = HASH ([tripId]),
   PARTITION ([tripDate] RANGE RIGHT FOR VALUES
      (20220101, 20220201, 20220301, 20220401)
  )
)
AS
SELECT * FROM dbo.TripTable WHERE 1 = 2;
```

Por el bien de este ejemplo, vamos a añadir algunos valores a la nueva partición:

```
INSERT INTO dbo.TripTable_new
VALUES (333, 444, 555, 20220401, "New York", "New Jersey");
```

Una vez que hayamos cargado los datos de la partición en la tabla ficticia, podemos cambiar la partición a nuestra tabla Fact utilizando el comando ALTER, como se muestra en el siguiente bloque de código. Este comando cambiará la última partición de abril (20220401) a la TripTable original:

```
ALTER TABLE dbo.TripTable_new SWITCH PARTITION 5 TO dbo.TripTable PARTITION 5 WITH
(TRUNCATE_TARGET = ON);
```

Estos comandos ALTER TABLE volverán casi inmediatamente ya que son operaciones de metadatos y no implican copiar filas de una partición a otra. Y así es como podemos aumentar la eficiencia de las operaciones de carga y borrado de datos. Puedes consultar el ejemplo completo en el repositorio GitHub de este libro.

Veamos a continuación el acceso a los datos mediante tablas externas.

6.6. Construcción de tablas externas

Las tablas externas son similares a las tablas normales, salvo que los datos se almacenan en ubicaciones de almacenamiento externas, como Azure Data Lake, Azure Blob storage y HDFS. Con las tablas externas, no es necesario copiar los datos en las tablas internas para su procesamiento. Pueden leer directamente los datos desde fuentes externas, lo que ahorra el coste de transferencia de datos. En Synapse, tanto dedicated SQL y serverless SQL soportan tablas externas. Tenemos que definir las siguientes tres construcciones para acceder a los datos a través de tablas externas:

6.6.1. EXTERNAL DATA SOURCE

A continuación se muestra un ejemplo de cómo podemos crear una external data source llamada users_iacstoreacct:

```
CREATE EXTERNAL DATA SOURCE [Dp203DataSource]
WITH ( LOCATION  = 'abfss://path/to/data')
```

6.6.2. EXTERNAL FILE FORMAT

A continuación se muestra un ejemplo de cómo crear un external file format llamado SynapseParquetFormat:

```
CREATE EXTERNAL FILE FORMAT [Dp203ParquetFormat]
WITH ( FORMAT_TYPE = PARQUET )
```

6.6.3. EXTERNAL TABLE

Podemos utilizar la external data source y the external file format para crear la tabla externa:

```
CREATE EXTERNAL TABLE TestExtTable (
[tripId] INT,
[driverId] INT,
. . .
[endLocation] VARCHAR (50)
)
WITH (
LOCATION = '/path/to/*.parquet',
DATA_SOURCE = [Dp203DataSource],
FILE_FORMAT = [Dp203ParquetFormat]
)
```

También puede crear fácilmente tablas externas desde Synapse UI, como se muestra en la siguiente captura de pantalla:

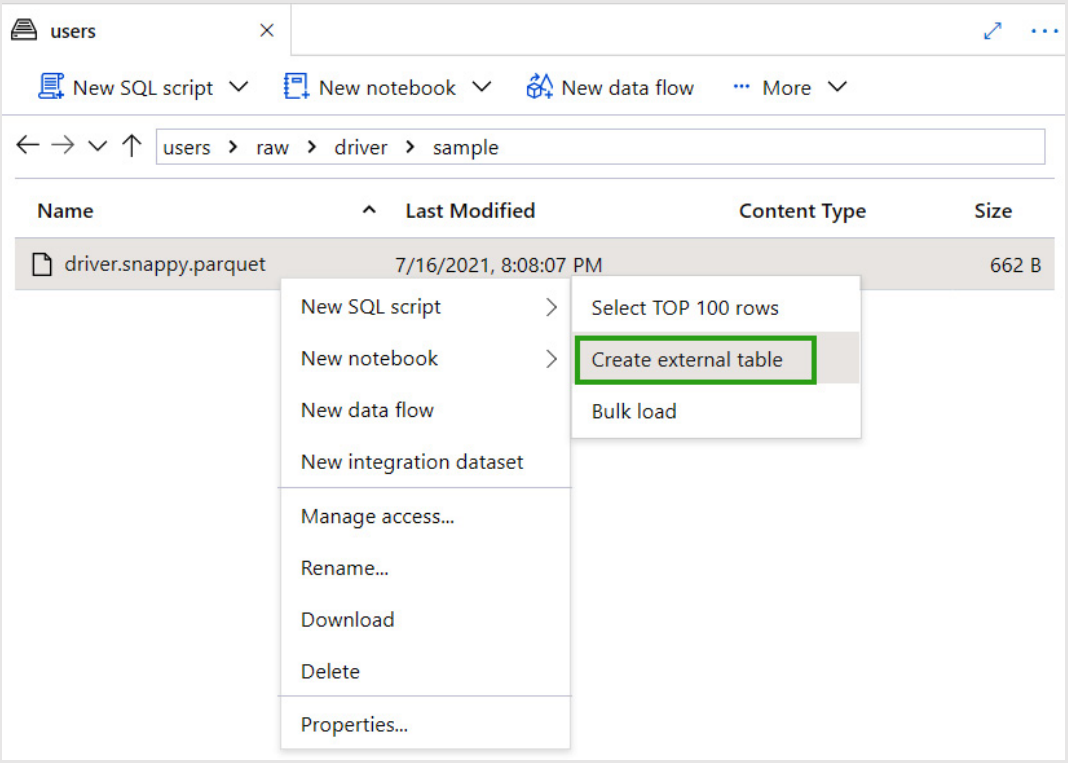

Figura 6.17 - Un ejemplo de creación de una external table desde Synapse UI

En la pantalla que aparece, debe hacer lo siguiente:

1. Seleccionar SQL Pool.
2. Introducir un nombre para la base de datos que se va a crear.
3. Introducir un nombre para la tabla externa que se va a crear.

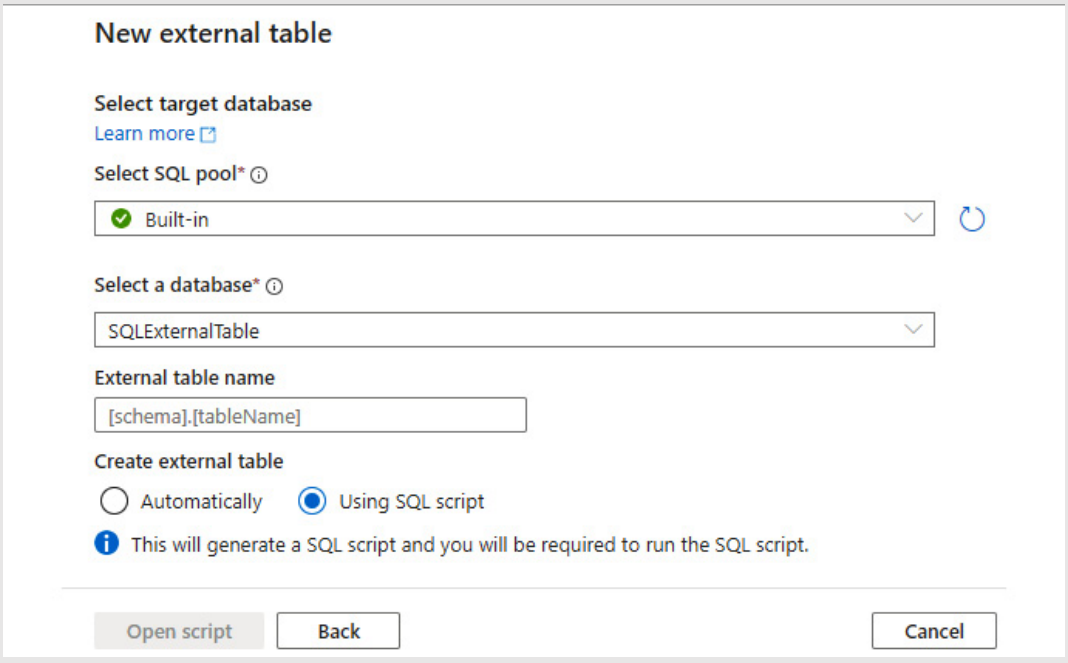

Figura 6.18 - Especificación de los detalles de la tabla externa

Una vez que haga clic en Abrir script, Synapse autogenerará el script SQL para crear tablas externas.

Resumen

Esto nos lleva al final de este capítulo. Este fue un capítulo más pequeño pero tuvo algunos conceptos muy importantes desde la perspectiva de la certificación. Comenzamos viendo la construcción de SCDs, que fue un proceso largo pero relativamente fácil, ya que en su mayoría tuvimos que arrastrar y soltar los componentes en los pipelines de Synapse y configurarlos. Luego, revisamos la regla general para construir una estructura de carpetas eficiente. También aprendimos a cambiar rápidamente las particiones dentro y fuera utilizando Synapse SQL. Por último, aprendimos lo fácil que es crear tablas externas en Synapse.

Todos los temas que se cubrieron en este capítulo deberían cubrir el programa de estudios para DP203 - Implementación de las estructuras lógicas de datos. En el próximo capítulo, nos centraremos en la implementación de la capa de servicio.