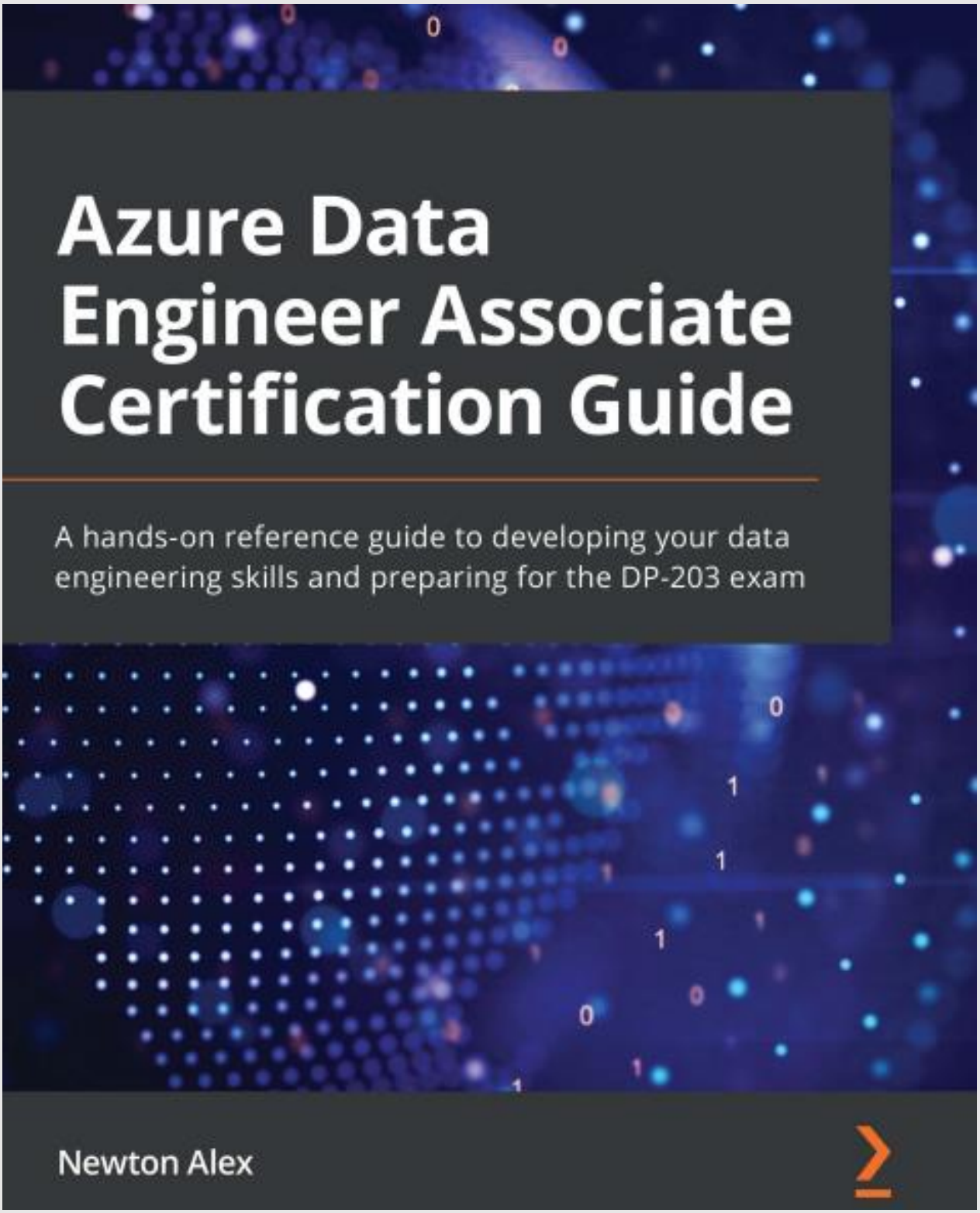
Azure Data Engineer Associate Certification Guide
A hands-on reference guide to developing your data engineering skills and preparing for the DP-203 exam
Newton Alex

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 4: Diseño de la capa de servicio

En este capítulo, aprenderemos las técnicas y tecnologías involucradas en el diseño de una capa de servicio de datos. Como hemos visto anteriormente en el diseño del data lake, los datos pasan por varias zonas. Parten de una Landing Zone, desde donde se procesan a formatos más útiles en la Transformation Zone, y finalmente, los datos derivados aterrizan en la Serving Zone (también llamada Capa de Servicio). La Serving Zone sirve los datos procesados y la información a los usuarios finales. La Landing Zone y la Transformation Zone de un data lake se centran en aspectos como el almacenamiento eficiente de los datos, el procesamiento de grandes volúmenes de datos, la optimización de las consultas, etc. La capa de servicio, por su parte, se centra principalmente en cómo servir los datos de forma rápida y eficiente a las herramientas de inteligencia empresarial (BI).

Las capas de servicio suelen construirse utilizando data stores relacionales (almacenes basados en el lenguaje de consulta estructurado (SQL)). Esto se hace por dos razones: los almacenes de datos relacionales pueden guardar los datos en tablas normalizadas eficientes y realizar consultas más rápidamente que los servicios analíticos de big data. Esto funciona bien porque la cantidad de datos en la Serving Zone suele ser de magnitudes de orden más pequeñas que en la Transformation Zone; además, SQL es el lenguaje preferido por la mayoría de los analistas de datos, y es universalmente soportado por las herramientas de BI.

Azure nos proporciona una variedad de servicios que se pueden utilizar para construir la capa de servicio, como Azure Synapse Analytics, Azure SQL, Cosmos DB, Apache Hive, Apache HBase, etc. En este capítulo exploraremos los más importantes para la certificación.

En este capítulo, nos centraremos en los siguientes temas:

- Aprender los fundamentos del modelado de datos y los esquemas
- Diseño de esquemas Star y Snowflake
- Diseño de dimensiones de cambio lento (SCD)
- Diseño de una solución para datos temporales
- Diseño de una jerarquía dimensional
- Diseño para la carga incremental
- Diseño de almacenes analíticos
- Diseño de metastores en Azure Synapse Analytics y Azure Databricks

Este es el último de los capítulos centrados en el diseño. Los próximos tres capítulos estarán dedicados a los detalles de implementación de lo que hemos aprendido hasta ahora.

¡Empecemos!

4.1. Requisitos técnicos

Para este capítulo, necesitarás una cuenta de Azure (gratuita o de pago).

4.2. Aprender los fundamentos del modelado de datos y los esquemas

El modelado de datos es un proceso de diseño de cómo se representarán los datos en los data stores. Muchas técnicas de modelado de datos fueron diseñadas originalmente para bases de datos y warehouses. Dado que las capas de servicio suelen construirse con almacenes de datos relacionales, como los data warehouses, algunas de las técnicas de modelado de datos pueden aplicarse también al diseño de la capa de servicio. Pero recuerde que la capa de servicio puede construirse utilizando otras tecnologías de almacenamiento, como bases de datos de documentos, almacenes de clave-valor, etc., en función de los requisitos del cliente.

A diferencia de los data lakes, en las bases de datos o los data warehouses no tenemos el lujo de almacenar enormes volúmenes de datos en el formato que queramos. Las bases de datos y los data warehouses pueden realizar consultas excepcionalmente rápidas, siempre que los datos se almacenen en formatos predeterminados y tengan un tamaño limitado. Por lo tanto, al diseñar la capa de servicio, tenemos que identificar los datos específicos que deben almacenarse, el formato en el que deben almacenarse y la cantidad de datos que deben almacenarse. En concreto, tenemos que decidir qué tablas SQL se necesitan, cuál sería la relación entre estas tablas y qué restricciones hay que imponer a estas tablas.

Existen diferentes métodos de modelado de datos, como el modelado entidad-relación (ER), el modelado jerárquico, el modelado dimensional de datos, el modelado relacional de datos, el modelado de datos orientado a objetos (OO), etc. Entre ellos, el modelado dimensional es el más relevante para el data warehousing, por lo que en este capítulo nos centraremos únicamente en las técnicas de modelado dimensional.

4.2.1. Modelos dimensionales

Los modelos dimensionales se centran en facilitar y agilizar la recuperación de la información, mientras que los demás modelos suelen centrarse en la optimización del almacenamiento. Los modelos de datos dimensionales más utilizados son los esquemas Star y los esquemas Snowflake. En las siguientes secciones nos centraremos en estos dos esquemas.

4.3. Diseño de esquemas Star y Snowflake

Los esquemas son directrices para organizar las entidades de datos, como las tablas SQL, en un data store. El diseño de un esquema se refiere al proceso de diseñar las distintas tablas y las relaciones entre ellas. Los esquemas Star y Snowflake son dos de los más utilizados en el mundo del análisis de datos y del BI. De hecho, los esquemas Star se utilizan con más frecuencia que los Snowflake. Ambos tienen sus propias ventajas y desventajas, así que vamos a explorarlos en detalle.

4.3.1. Esquemas en estrella

Un esquema en estrella es el más sencillo de los esquemas de data warehouses. Tiene dos conjuntos de tablas: uno que almacena información cuantitativa, como las transacciones que se producen en un comercio minorista o los viajes que se realizan en una compañía de taxis, y otro que almacena el contexto o las descripciones de los eventos que se almacenan en la tabla cuantitativa.

Las tablas cuantitativas se denominan tablas de hechos y las descriptivas o de contexto se llaman tablas de dimensiones.

El siguiente diagrama muestra un ejemplo de esquema Star:

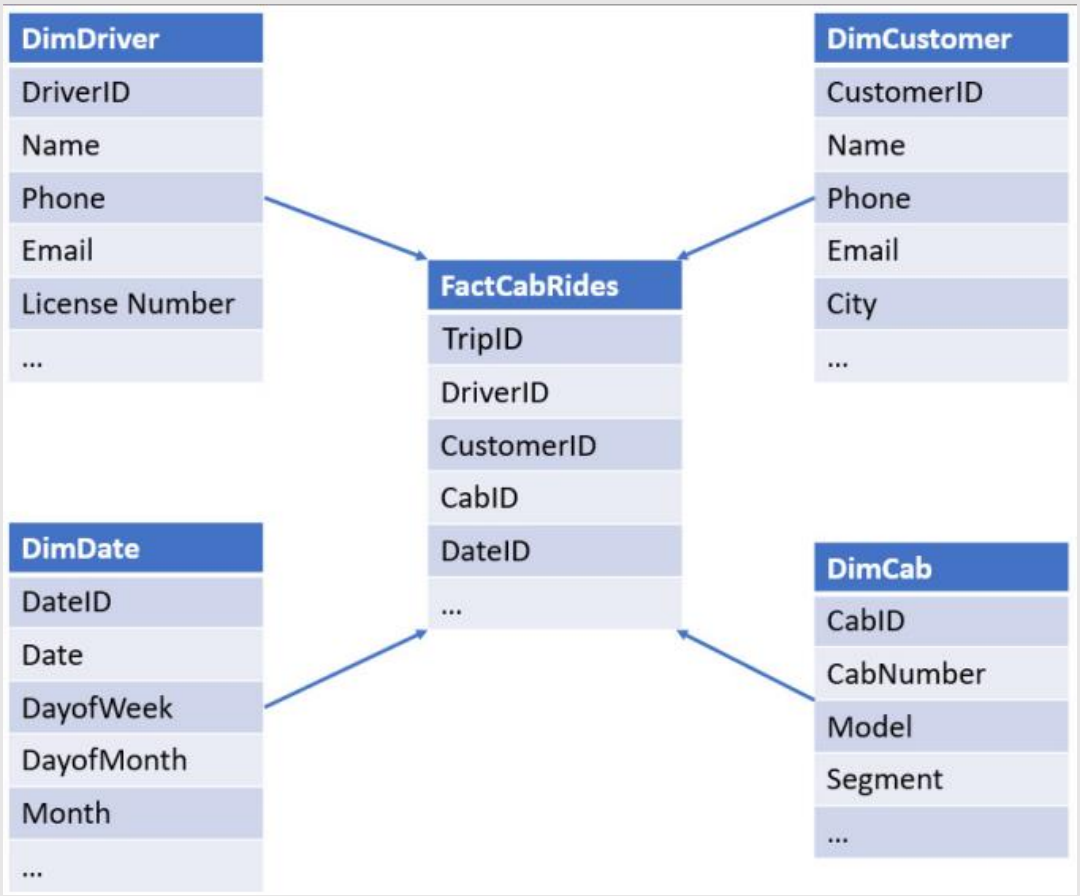

Figura 4.1 - Ejemplo de un esquema en estrella

Como puede resultar obvio por los nombres de las tablas, FactTrips es nuestra tabla de hechos, y todas las demás -como DimDriver, DimCustomer, DimDate y DimCab- son tablas de dimensiones.

Dado que el diagrama de relaciones del esquema Star tiene la forma de una estrella, se denomina esquema Star. La tabla de hechos que se encuentra en el medio es el centro de la estrella, y las tablas de dimensiones son los brazos de la estrella.

Veamos algunos puntos importantes sobre los esquemas en estrella, a saber:

- Las tablas de hechos suelen tener un volumen mucho mayor que las tablas de dimensiones.
- Las tablas de dimensiones no están conectadas; son independientes unas de otras.
- Los datos no están normalizados en un esquema Star. Es muy común encontrar datos replicados en múltiples tablas. Las tablas están diseñadas para ser rápidas y fáciles de usar.
- Están optimizadas para las consultas de BI. Las consultas suelen ser muy sencillas ya que sólo tienen un nivel de uniones.
- Las consultas suelen ser también mucho más rápidas debido al menor número de uniones.

Ahora, veamos en qué se diferencian los esquemas de Snowflake.

4.3.2. Esquemas de snowflake

Un esquema de copo de nieve es una extensión del esquema de estrella. En este modelo, la tabla de hechos sigue siendo la misma, pero las tablas de dimensiones se dividen además en sus formas normalizadas, a las que se hace referencia mediante claves externas. Puede haber varios niveles de jerarquía entre las tablas de dimensiones.

El siguiente diagrama muestra cómo el mismo ejemplo utilizado para un esquema Star puede extenderse a un esquema Snowflake:

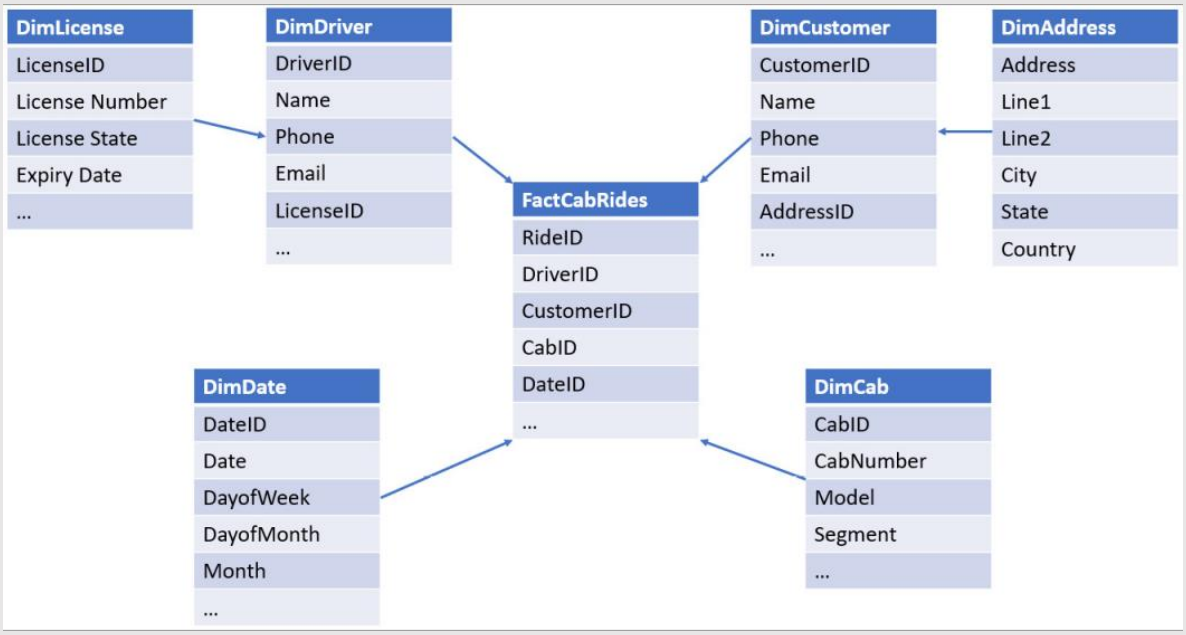

Figura 4.2 - Ejemplo de un esquema Snowflake

Como podrá observar en el diagrama anterior, la tabla DimDriver se ha normalizado para tener los detalles de la licencia por separado. Del mismo modo, hemos normalizado los detalles de la dirección fuera de la tabla DimCustomer.

Puede elegir un esquema Snowflake si tiene aplicaciones BI y no BI que comparten el mismo data warehouses. En estos casos, desde una perspectiva global, puede ser mejor tener los datos normalizados.

Veamos algunos puntos importantes sobre los esquemas Snowflake, como sigue:

- Las tablas de hechos, también en este caso, son similares a los esquemas Star y tienen un volumen mucho mayor que las tablas de dimensiones.

- Los datos de las dimensiones están normalizados, evitando así cualquier dato redundante.

- Las dimensiones pueden estar conectadas entre sí.

- Los datos están optimizados para el almacenamiento y la integridad, pero no para la velocidad.

- El esquema es más complejo que un esquema Star, por lo que podría no ser la opción más preferida para los casos de uso de BI e informes.

- Las consultas suelen ser más lentas en comparación con los esquemas Star debido a las uniones de varios niveles que se requieren.

Espero que ahora tenga una buena idea sobre los esquemas Star y Snowflake. Hay un tipo más de esquema llamado esquema de Constelación de Hechos o esquema de Galaxia. Con este tipo de esquema, puede haber más de una tabla de hechos, y las tablas de dimensiones se comparten. Este

esquema no es tan importante como los esquemas Star y Snowflake desde el punto de vista de la certificación, pero si quiere saber más sobre este esquema, puede empezar por aquí: https://www.javatpoint.com/data-warehouse-what-is-fact-constellation-schema.

Veamos ahora cómo manejar los datos que cambian lentamente en las tablas de dimensión.

4.4. Diseño de SCDs

Los SCDs se refieren a los datos en las tablas de dimensión que cambian lentamente en el tiempo y no con una cadencia regular. Un ejemplo común para los SCDs son los perfiles de los clientes-por ejemplo, una dirección de correo electrónico o el número de teléfono de un cliente no cambia tan a menudo, y estos son candidatos perfectos para SCD. En esta sección, veremos cómo diseñar para tales cambios.

Servicios como Azure SQL proporcionan soporte incorporado para SCD, pero en data warehouses como los de Synapse dedicated pools, tendremos que implementarlos nosotros mismos.

Estos son algunos de los principales aspectos que tendremos que considerar al diseñar un SCD:

- ¿Debemos hacer un seguimiento de los cambios? En caso afirmativo, ¿qué parte del historial debemos mantener?

- O, ¿deberíamos simplemente sobrescribir los cambios e ignorar el historial?

En función de nuestros requisitos para mantener el historial, hay unas siete maneras de llevar a cabo el seguimiento de los cambios. Se denominan SCD1, SCD2, SCD3, y así sucesivamente, hasta SCD7.

Entre ellas, SCD1, SCD2, SCD3, SCD4 y SCD6 son las más importantes, y nos centraremos sólo en ellas en este capítulo. También son los más importantes desde el punto de vista de la certificación.

4.4.1. Diseño de SCD1

En el SCD tipo 1, los valores se sobrescriben y no se mantiene ningún historial, por lo que una vez que se actualizan los datos, no hay forma de averiguar cuál era el valor anterior. Las nuevas consultas siempre devolverán el valor más reciente. A continuación se muestra un ejemplo de tabla SCD1:

CustomerID	Name	City	Email	...
1	Adam	New York	adam@....	...

CustomerID	Name	City	Email	...
1	Adam	New Jersey	adam@....	...

Figura 4.3 - Ejemplo de SCD tipo 1

En este ejemplo, el valor de la columna City está cambiando de Nueva York a Nueva Jersey. El valor simplemente se sobrescribe.

4.4.2. Diseño de SCD2

En SCD2, mantenemos un historial completo de cambios. Cada vez que hay un cambio, añadimos una nueva fila con todos los detalles sin borrar los valores anteriores. Hay múltiples maneras de lograr esto. Veamos los enfoques más comunes.

Usando una bandera (flag)

En este enfoque, utilizamos una bandera para indicar si un valor particular está activo o si es actual. He aquí un ejemplo de ello:

SurrogateID	CustomerID	Name	City	isActive	...
1	1	Adam	New York	True	...

SurrogateID	CustomerID	Name	City	isActive	...
1	1	Adam	New York	False	...
2	1	Adam	New Jersey	False	...
3	1	Adam	Miami	True	...

Figura 4.4 - Ejemplo de SCD tipo 2: bandera

En la segunda tabla, cada vez que hay un cambio, añadimos una nueva fila y actualizamos la columna isActive de las filas anteriores a False. De este modo, podemos consultar fácilmente los valores activos filtrando por el criterio isActive=True.

NOTA

Las claves sustitutas (Surrogate keys) son claves de identificación de filas secundarias. Se añaden en todos los casos de SCD2 porque la clave de identificación primaria ya no será única con las filas recién añadidas.

Uso de números de versión

En este enfoque, utilizamos los números de versión para hacer un seguimiento de los cambios. La fila con la versión más alta es el valor más actual. He aquí un ejemplo de ello:

SurrogateID	CustomerID	Name	City	Version	...
1	1	Adam	New York	0	...

SurrogateID	CustomerID	Name	City	Version	...
1	1	Adam	New York	0	...
2	1	Adam	New Jersey	1	...
3	1	Adam	Miami	2	...

Figura 4.5 - Ejemplo de SCD tipo 2: números de versión

En el ejemplo anterior, tenemos que filtrar en la columna MAX(Version) para obtener los valores actuales.

Uso de intervalos de fechas

En este enfoque, utilizamos rangos de fechas para mostrar el periodo en el que un determinado registro (fila) estuvo activo, como se ilustra en el siguiente ejemplo:

SurrogateID	CustomerID	Name	City	StartDate	EndDate
1	1	Adam	New York	01-Jan-2020	NULL

SurrogateID	CustomerID	Name	City	StartDate	EndDate
1	1	Adam	New York	01-Jan-2020	25-Mar-2020
2	1	Adam	New Jersey	25-Mar-2020	01-Dec-2020
3	1	Adam	Miami	01-Dec-2020	NULL

Figura 4.6 - Ejemplo de SCD tipo 2: rangos de fechas

En el ejemplo anterior, cada vez que cambiamos un campo, añadimos un nuevo registro a la tabla. Al mismo tiempo, actualizamos la columna EndDate del registro anterior y la columna StartDate del nuevo registro con la fecha de hoy. Para obtener el registro actual, tenemos que filtrar por el criterio EndDate=NULL o, en su lugar, podemos poner una fecha muy futurista en lugar de NULL, como por ejemplo 31-Dic-2100.

Como variación del enfoque de rango de fechas, también podríamos añadir una columna de bandera para identificar fácilmente los registros activos o actuales. El siguiente ejemplo muestra este enfoque:

SurrogateID	CustomerID	Name	City	StartDate	EndDate	isActive
1	1	Adam	New York	01-Jan-2020	25-Mar-2020	False
2	1	Adam	New Jersey	25-Mar-2020	01-Dec-2020	False
3	1	Adam	Miami	01-Dec-2020	NULL	True

Figura 4.7 - Ejemplo de SCD tipo 2: rangos de fechas y bandera

Veamos ahora el diseño del SCD3.

4.4.3. Diseño de SCD3

En SCD3, mantenemos sólo un historial parcial y no un historial completo. En lugar de añadir filas adicionales, añadimos una columna extra que almacena el valor anterior, por lo que sólo se conservará una versión de los datos históricos. Al igual que con la opción SCD2, aquí también podemos optar por añadir columnas de fecha para llevar la cuenta de las fechas modificadas, pero en este caso no necesitamos claves sustitutas ya que la clave de identificación del registro no cambia. He aquí un ejemplo de ello:

CustomerID	Name	City	PrevCity	...
1	Adam	New York	NULL	

CustomerID	Name	City	PrevCity	...
1	Adam	New Jersey	New York	

Figura 4.8 - Ejemplo de SCD tipo 3

En el ejemplo anterior, hemos añadido una nueva columna llamada PrevCity. Cada vez que el valor de Ciudad cambia, añadimos el valor anterior a PrevCity y actualizamos la columna Ciudad con la ciudad actual.

4.4.4. Diseño de SCD4

El SCD4 se introdujo para atributos de dimensión que cambian con relativa frecuencia. En el tipo 4, dividimos los atributos que cambian rápidamente de la tabla de dimensiones en otra tabla de dimensiones más pequeña y también referenciamos la nueva tabla de dimensiones directamente desde la tabla de hechos.

Por ejemplo, en el siguiente diagrama, si suponemos que el pase de coche compartido (carpool) (también conocido como vehículos de alta ocupación) debe comprarse cada mes, podemos mover ese campo a una minidimensión más pequeña y referenciarlo directamente desde la tabla de hechos:

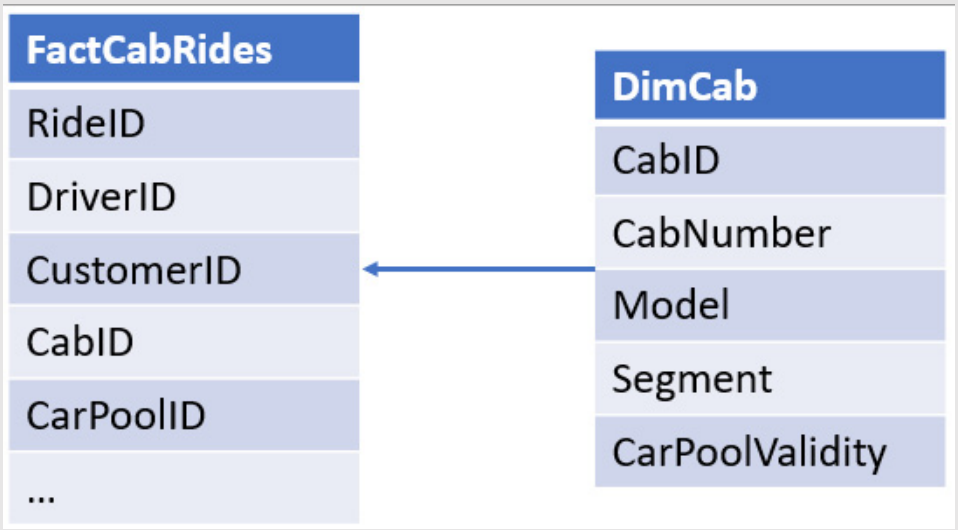

Figura 4.9 - Ejemplo de SCD4: antes de la división

Podemos dividir la tabla en una mini dimensión DimCarPool, como en el siguiente diagrama:

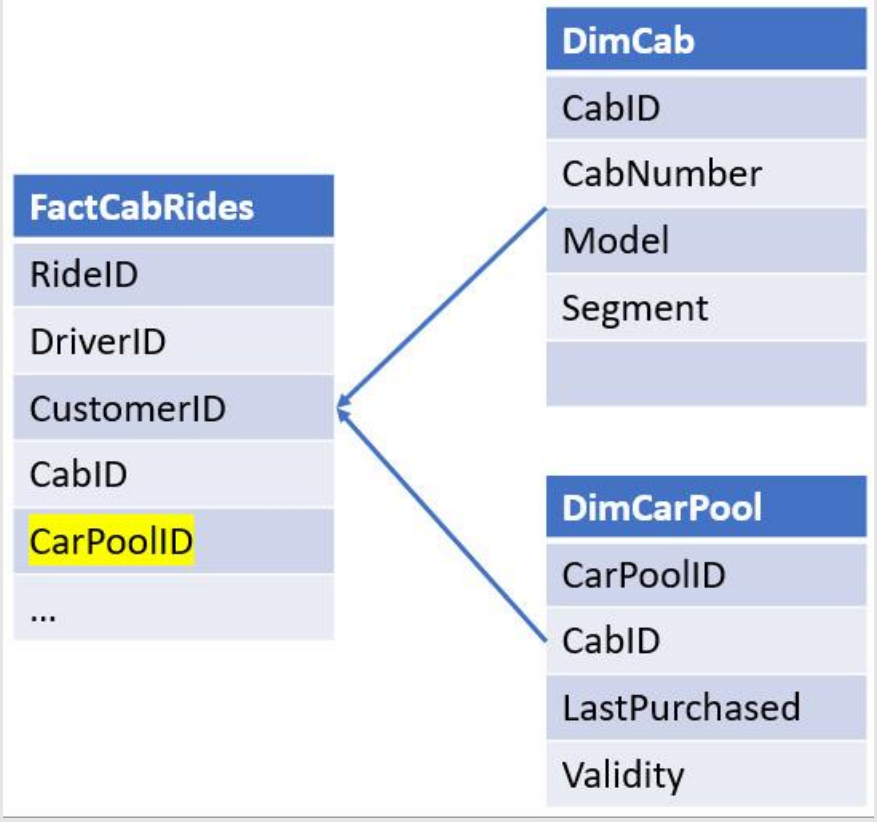

Figura 4.10 - Ejemplo de SCD tipo 4: después de la división

Esta subdivisión ayuda a modificar con frecuencia sólo una pequeña cantidad de datos en lugar de la fila completa.

4.4.5. Diseño de los SCD5, SCD6 y SCD7

El resto de los SCD -SCD5, SCD6 y SCD7- son derivados de los cuatro SCD anteriores. Entre estos derivados, el SCD6 es relativamente importante, por lo que lo estudiaremos en la siguiente subsección.

Diseño del SCD6

El tipo 6 es una combinación de los tipos 1, 2 y 3. En este tipo, junto con la adición de nuevas filas, también actualizamos el último valor en todas las filas, como se ilustra en la siguiente captura de pantalla:

Surrogate ID	Customer ID	Name	CurrCity	PrevCity	StartDate	EndDate	isActive
1	1	Adam	Miami	NULL	01-Jan-2020	25-Mar-2020	False
2	1	Adam	Miami	New York	25-Mar-2020	01-Dec-2020	False
3	1	Adam	Miami	New Jersey	01-Dec-2020	NULL	True

Figura 4.11 - Ejemplo de SCD tipo 6

En el ejemplo anterior, habrá observado que se ha actualizado el valor CurrCity de todos los registros pertenecientes al cliente Adam. Esta es otra de las ventajas de extraer los últimos valores.

Esto explica el tipo de SCD 6. Si está interesado en conocer los SCDs 5 y 7, puede encontrar más información en los siguientes enlaces:

SCD5: https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/type-5/

SCD7: https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/type-7

Ahora que ya tienes una idea sobre los SCD, vamos a explorar cómo manejar los temporales.

4.5. Diseñar una solución para los datos temporales

Los datos temporales se refieren a datos en puntos específicos en el tiempo. El almacenamiento de datos temporales es necesario en situaciones como la auditoría de datos, las investigaciones forenses, el mantenimiento de los SCD, las recuperaciones en un momento dado, etc. Azure SQL y SQL Server proporcionan un mecanismo llamado tablas temporales para almacenar datos temporales.

Las tablas temporales son tablas especializadas que hacen un seguimiento de los cambios de datos en el tiempo. Hacen un seguimiento del historial de cambios de datos como el que ya habíamos visto en las tablas SCD, pero en este caso el sistema se encarga de gestionar el periodo de validez temporal de cada fila, en lugar de tener que hacerlo nosotros manualmente. De ahí que estas tablas se llamen también tablas temporales versionadas por el sistema.

NOTA

La tabla temporal es un concepto de la base de datos Azure SQL y de SQL server. No estaba disponible en Azure Synapse pools en el momento de escribir este libro.

Veamos un ejemplo de cómo crear tablas temporales en Azure SQL:

```
CREATE TABLE Customer
(
  [customerId] INT NOT NULL PRIMARY KEY CLUSTERED,
  [name] VARCHAR(100) NOT NULL,
  [address] VARCHAR(100) NOT NULL,
  [email] VARCHAR (100) NOT NULL,
  [phone] VARCHAR(12) NOT NULL,
  [validFrom] DATETIME2 GENERATED ALWAYS AS ROW START,
  [validTo] DATETIME2 GENERATED ALWAYS AS ROW END,
  PERIOD FOR SYSTEM_TIME (validFrom, validTo),
)
WITH (SYSTEM_VERSIONING = ON);
```

Si te fijas, hay tres sentencias que destacarían:

```
[validFrom] DATETIME2 GENERATED ALWAYS AS ROW START,
[validTo] DATETIME2 GENERATED ALWAYS AS ROW END,
PERIOD FOR SYSTEM_TIME (validFrom, validTo),
```

Para una tabla temporal, necesitamos definir la columna de inicio del período y la columna de fin del período. En nuestro ejemplo, la columna de inicio del período es validFrom y la columna de fin del período es validTo. Y le decimos al motor de la Base de Datos que estos son nuestros indicadores de tiempo de inicio y fin usando la línea PERIOD FOR SYSTEM_TIME (validFrom, validTo).

NOTA

Sólo debe haber un PERIOD FOR SYSTEM_TIME definido con dos columnas datetime2. Estas dos columnas datetime2 deben ser declaradas como GENERATED ALWAYS AS ROW START / END.

Ahora, actualicemos una de las entradas de la tabla y veamos cómo la tabla temporal mantiene un registro de los cambios.

```
UPDATE [dbo].[Customer] SET [address] = '111 Updated Lane, LA' WHERE [customerId] = 101;
```

Consultemos la tabla basándonos en los dos límites temporales y veamos todos los cambios que se han producido en el cliente con customerId=101.

```
SELECT [customerId]
    , [name]
    , [address]
    , [validFrom]
    , [validTo]
    , IIF (YEAR(validTo) = 9999, 1, 0) AS IsActual
FROM [dbo].[Customer]
FOR SYSTEM_TIME BETWEEN '2022-01-22' AND '2022-01-24'
WHERE CustomerId = 101
ORDER BY validFrom DESC;
```

Results Messages

Search to filter items...

customerId	name	address	validFrom	validTo	IsActual
101	Alan Li	111 Updated Lane, LA	2022-01-23T07:38:32....	9999-12-31T23:59:59....	1
101	Alan Li	101 Test Lane, LA	2022-01-23T07:36:26....	2022-01-23T07:38:32....	0

Figura 4.12 - Ejemplo de resultado de una consulta temporal

Como puede ver, la tabla Temporal ha mantenido un registro de los cambios junto con las timestamps. Hemos obtenido el IsActual basándonos en el valor de la columna validTo.

Cuando creamos una tabla Temporal, entre bastidores se crean dos tablas. Una se llama tabla Temporal (la que hemos definido) y otra tabla con el mismo esquema llamada tabla histórica. Cada vez que los datos cambian, los valores actuales se mantienen en la tabla temporal y los valores antiguos se mueven a la tabla histórica con la hora de finalización actualizada a la hora actual, indicando que esa fila ya no está activa. También hay opciones para definir nuestras propias tablas de Historial con indexación adicional, distribuciones y demás y proporcionárselas a Azure SQL para que las utilice como tabla de historial.

Todo lo que hay que hacer es definir la tabla History y proporcionar el nombre como se muestra durante la creación de la tabla Temporal.

```
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.CustomerHistory));
```

Ahora ya sabes cómo aprovechar el soporte de las tablas temporales en Azure SQL para crear aplicaciones que necesiten registrar cambios en el tiempo.

Puedes aprender más sobre las tablas temporales aquí: https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-ver15

Veamos ahora cómo diseñar una jerarquía dimensional.

4.6. Diseñando una jerarquía dimensional

La jerarquía dimensional se refiere a la forma en que agrupamos y organizamos los datos dimensionales en múltiples niveles. En una estructura jerárquica, suele haber una relación padre-hijo de uno a muchos o de muchos a muchos. Ejemplos de estructuras jerárquicas pueden ser las estructuras organizativas, las categorías de productos en una tienda online, un sistema de archivos, etc. La principal característica de la estructura jerárquica es que todos los nodos son idénticos e incluyen punteros a sus nodos padre o hijos.

Para conseguir una jerarquía dimensional, podemos utilizar una técnica denominada relación de autorreferencia o autouniones dentro de la tabla de dimensiones. Tomemos un ejemplo de una dimensión Empleado.

```
CREATE TABLE DimEmployee (
[employeeId] VARCHAR(20) NOT NULL,
[name] VARCHAR(100),
[department] VARCHAR(50),
[title] VARCHAR(50),
[parentEmployeeId] VARCHAR(20)
)
```

Aquí tenemos una columna llamada parentEmployeeID que hace referencia a la columna employeeID. Si la base de datos admite la restricción de clave foránea, la columna parentEmployeeID se definiría como la clave foránea de employeeID. Synapse SQL todavía no soporta claves foráneas, por lo que necesitamos asegurar la corrección a nivel de la aplicación.

NOTA

La clave padre debe permitir entradas NULL ya que los elementos raíz de la jerarquía no tendrán un padre.

Una tabla jerárquica de ejemplo con entradas tendría el siguiente aspecto:

```
5   CREATE TABLE DimEmployee (
6       [employeeId] VARCHAR(20) NOT NULL,
7       [name] VARCHAR(100),
8       [department] VARCHAR(50),
9       [title] VARCHAR(50),
10      [parentEmployeeId] VARCHAR(20)
11  )
12
```

Results Messages

Search to filter items...

employeeId	name	department	title	parentEmployeeId
100	Alan Li	Manufacturing	Manager	
200	Brenda Jackman	Manufacturing	Supervisor	100
300	David Hood	Manufacturing	Machine operator	200

Figura 4.13 - Ejemplo de tabla de dimensiones jerárquicas

Podemos ver que David depende de Brenda y Brenda depende de Alan. Hemos creado efectivamente una jerarquía utilizando la misma tabla.

CONSEJO

Si usted está usando la Jerarquía Dimensional como parte de un SCD, siempre agregue la clave Padre apuntando a la Clave Sustituta en lugar de la clave primaria del Negocio. Porque las claves sustitutas serán únicas, y asegurará que la jerarquía dimensional no se rompa cuando ocurran cambios en la clave de negocio.

Espero que tengas una idea clara sobre las jerarquías dimensionales. Veamos ahora cómo diseñar una solución de carga incremental para la capa de servicio.

4.7. Diseño de la carga incremental

La carga incremental o carga delta se refiere al proceso de cargar incrementos reducidos de datos en una solución de almacenamiento; por ejemplo, podríamos tener datos diarios que se cargan en un data lake o datos por hora que fluyen en un pipeline de extracción, transformación y carga (ETL), etc. Durante los escenarios de consumo de datos, es muy común hacer una carga masiva seguida de cargas incrementales programadas.

Azure tiene un servicio muy versátil llamado Azure Data Factory (ADF) que puede ayudar con la carga incremental. Dado que es la primera vez que utilizamos ADF en este libro, vamos a aprender un poco más sobre él ahora, ya que la información será muy útil en futuros capítulos.

ADF es un servicio gestionado en la nube que se puede utilizar para coordinar y orquestar canalizaciones complejas basadas en la nube o híbridas (en las instalaciones). ADF proporciona la capacidad de construir ETL y pipelines de extracción, carga y transformación (ELT). Con ADF, puede hacer lo siguiente

- Ingerir datos desde una amplia variedad de fuentes como bases de datos, archivos compartidos, centros de Internet de las Cosas (IoT), Amazon Web Services (AWS), Google Cloud Platform (GCP), y más.

- Construir complejos pipelines utilizando variables, parámetros, ramas, etc.

- Transformar datos utilizando servicios de cómputo como Synapse, HDInsight, Cosmos DB, etc.

- Programar y supervisar ingesta, flujo de control y operaciones de flujo de datos.

Este es el aspecto del espacio de trabajo del ADF:

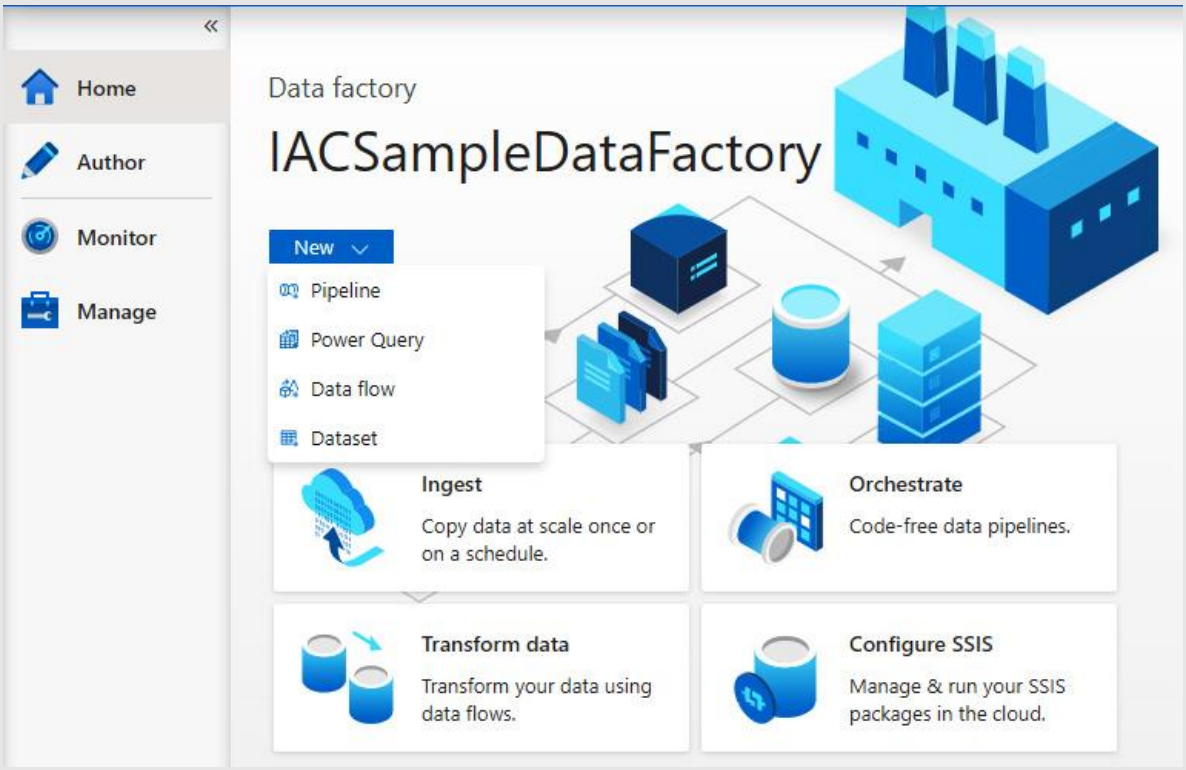

Figura 4.14 - Pantalla de inicio del espacio de trabajo de Data Factory

El ADF está formado por un conjunto básico de componentes. Aquí se enumeran los más importantes:

- **Pipelines**-Un pipeline es una colección de actividades que están enlazadas para realizar algún flujo de control o transformación de datos.
- **Actividades**-Actividades en ADF se refieren a los pasos en el pipeline como copiar datos, ejecutar un trabajo de Spark, etc.
- **Conjuntos de datos**-Son los datos sobre los que operan sus pipelines o actividades.
- **Servicios vinculados**: los servicios vinculados son conexiones que ADF utiliza para conectarse a una variedad de almacenes de datos y cálculos en Azure. Son como cadenas de conexión que le permiten acceder a datos de fuentes externas.
- **Triggers**-Los triggers son eventos que se utilizan para iniciar pipelines o iniciar una actividad.

Utilizaremos muchas de las terminologías anteriores al usar ADF, así que es bueno entenderlas.

Ahora que sabemos lo que es el ADF, vamos a explorar las diferentes formas en las que podemos diseñar la carga incremental utilizando el ADF. Basado en el tipo de fuente de datos, podemos tener diferentes técnicas para implementar la carga incremental. Algunas de ellas se enumeran aquí:

- **Usando marcas de agua (watermarks)** - Si la fuente de datos es una base de datos o un sistema basado en tablas relacionales
- **Usar marcas de tiempo de archivos (file timestamps)** - Si la fuente es un sistema de archivos o un blob storage
- **Uso de datos de partición**: si la fuente está dividida en función del tiempo.
- **Uso de la estructura de carpetas**: si la fuente está dividida en función del tiempo

Exploremos cada una de estas técnicas en detalle.

4.7.1. Marcas de agua (Watermarks)

La marca de agua es una técnica muy sencilla por la que **sólo se lleva la cuenta del último registro cargado (nuestra marca de agua) y se cargan todos los nuevos registros más allá de la marca de agua en la siguiente ejecución incremental**.

En las tecnologías de almacenamiento relacional, como las bases de datos SQL, podemos almacenar los detalles de la marca de agua como una simple tabla más y actualizar automáticamente la marca de agua con procedimientos almacenados. Cada vez que se cargue un nuevo registro, el procedimiento almacenado debe activarse, lo que actualizará nuestra tabla de marcas de agua. El siguiente pipeline de copia incremental puede utilizar esta información de marca de agua para identificar el nuevo conjunto de registros que necesitan ser copiados. **Veamos cómo podemos implementar un diseño de marca de agua con ADF utilizando Azure SQL como fuente**. Supongamos que tenemos una tabla simple llamada **FactTrips** que necesita ser cargada incrementalmente en una tabla de Azure SQL. Proceda de la siguiente manera:

1. Selecciona el servicio Azure SQL en el panel de control de Azure y crea una nueva instancia de Azure SQL si aún no tienes una. Cree una tabla simple de FactTrips como se muestra en la siguiente captura de pantalla e inserte algunos valores ficticios en ella utilizando la opción del editor de consultas (vista previa):

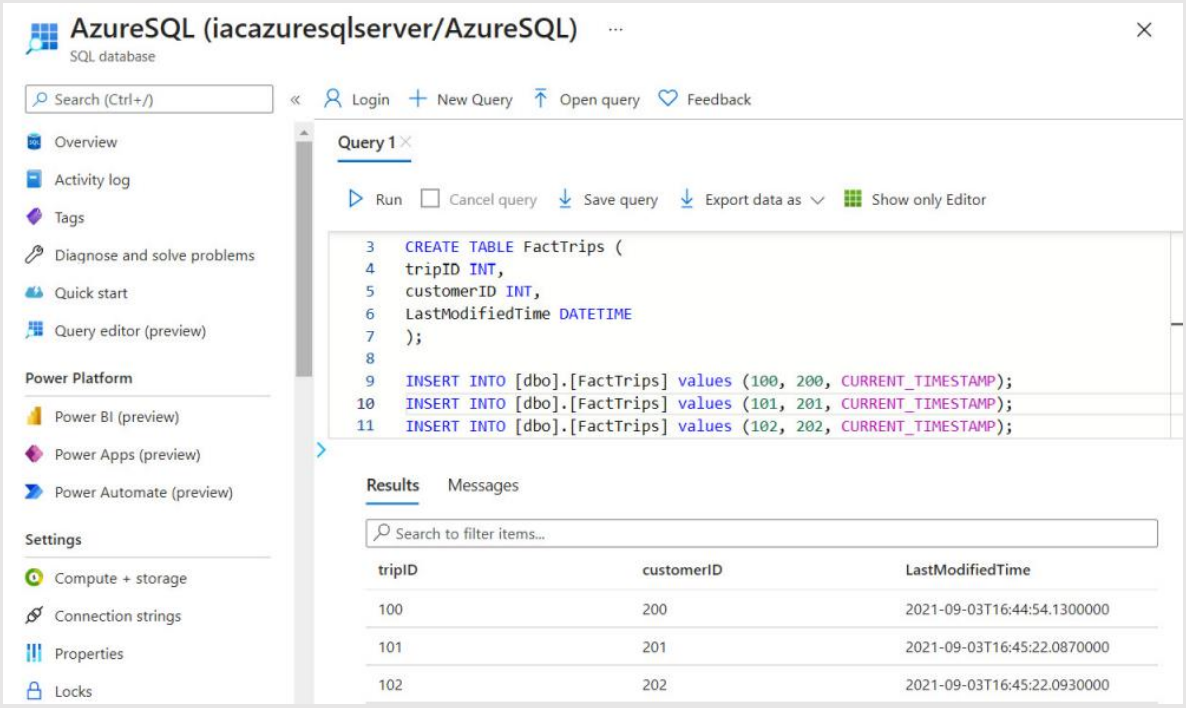

Figura 4.15 - Creación de una tabla simple en Azure SQL

2. Cree una tabla de marcas de agua, de la siguiente manera:

```
CREATE TABLE WatermarkTable
(
 [TableName] VARCHAR(100),
 [WatermarkValue] DATETIME,
);
```

3. Cree un procedimiento almacenado para actualizar automáticamente la tabla de marcas de agua cada vez que haya nuevos datos.

```
CREATE PROCEDURE [dbo].uspUpdateWatermark @LastModifiedtime DATETIME, @TableName VARCHAR(100)
AS
BEGIN
UPDATE [dbo].[WatermarkTable]
SET [WatermarkValue] = @LastModifiedtime
WHERE [TableName] = @TableName
END
```

4. Ahora, en el lado del ADF, necesitamos crear un nuevo pipeline para encontrar el delta entre las marcas de agua antiguas y las nuevas y luego iniciar una copia incremental.

5. Desde la página Pipeline en el ADF, cree dos actividades Lookup. Se encuentran en Actividades -> General -> Lookup. A continuación se muestra una pantalla de ejemplo del ADF:

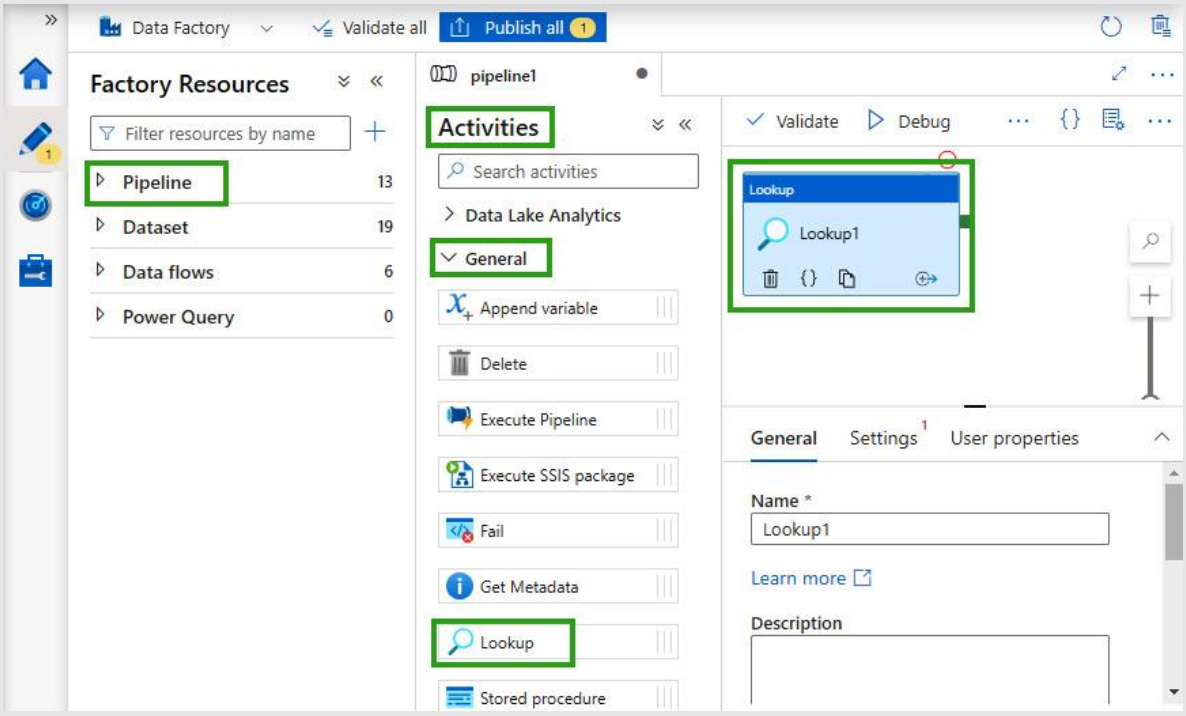

Figura 4.16 - Página de creación del ADF

6. Configure el primero para que busque la entrada anterior de la tabla de marcas de agua, como se muestra en la siguiente captura de pantalla. El conjunto de datos Watermark ha sido configurado para apuntar a la Azure WatermarkTable.

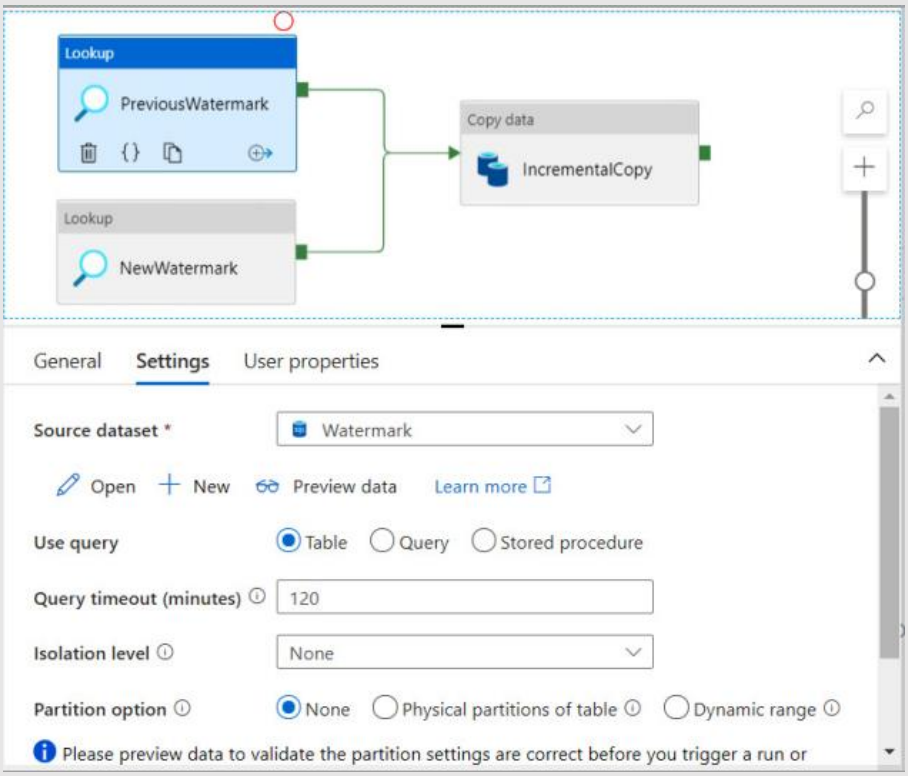

Figura 4.17 - Configuración de la búsqueda mediante la tabla de marcas de agua

7. Configure la siguiente actividad de Lookup para buscar la última hora de modificación del archivo en la tabla de origen, que en nuestro caso sería la tabla FactTrips:

```
SELECT MAX(LastModifiedTime) AS NewWatermarkValue FROM FactTrips;
```

Tendrá que introducir la siguiente consulta en el cuadro de texto Consulta de la pestaña Configuración:

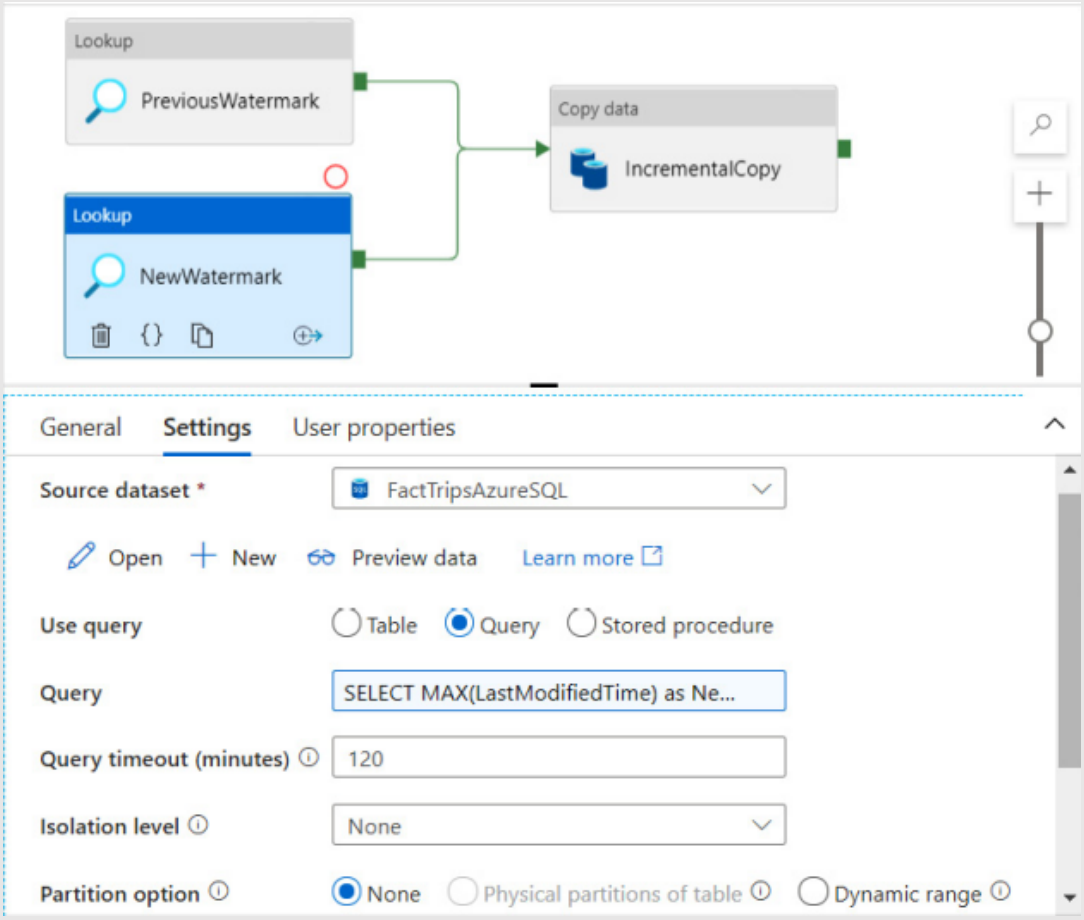

Figura 4.18 - Nueva configuración de búsqueda de marcas de agua usando LastModifiedTime

8. Por último, añada una nueva actividad de Copia desde Actividades -> Mover y Transformar -> Copiar Datos y configúrela como sigue:

```
SELECT * FROM FactTrips
WHERE LastModifiedTime > '@{activity('PreviousWatermark').output.firstRow.WatermarkValue}'
AND
LastModifiedTime <= '@{activity('NewWatermark').output.firstRow.WatermarkValue}';
```

En la sección de consultas de la pestaña Source, introduzca la siguiente consulta:

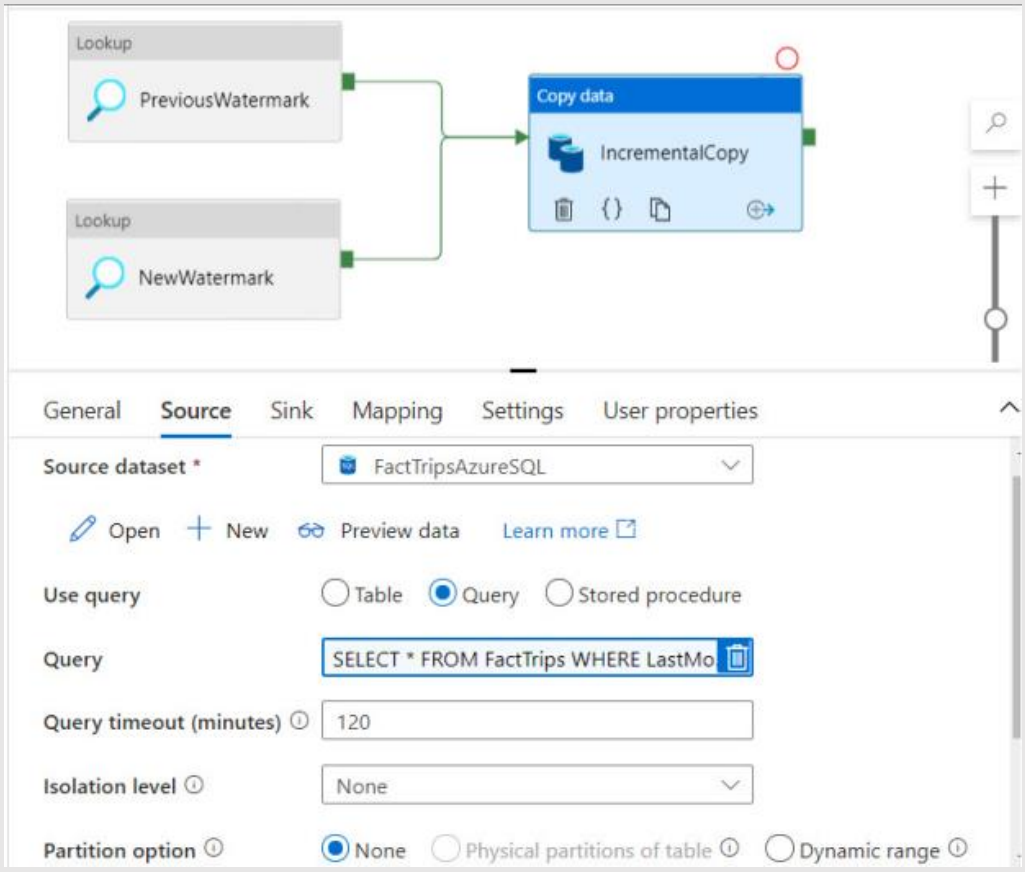

Figura 4.19 - Actividad de copia del ADF con generación de delta basada en marcas de agua

9. Guarde (publique) el pipeline anterior y configure un trigger programado utilizando el botón Add Trigger de la pantalla Pipeline. Ahora, cada vez que haya cambios en la tabla FactTrips, se copiará en nuestra tabla de destino a intervalos regulares.

A continuación aprenderemos a realizar copias incrementales utilizando marcas de tiempo de archivos.

4.7.2. File timestamps

Hay otra técnica disponible para cargar incrementalmente sólo los nuevos archivos de un origen a un destino: La funcionalidad de la herramienta de copia de datos del ADF. Esta herramienta proporciona una opción para escanear los archivos en el origen basándose en el atributo LastModifiedDate. Por lo tanto, todo lo que tenemos que hacer es especificar las carpetas de origen y de destino y seleccionar la opción Incremental load: LastModifiedDate para el campo de comportamiento de carga de archivos.

Puede iniciar la funcionalidad de la herramienta Copiar Datos desde la pantalla principal del ADF, como se muestra en la siguiente captura de pantalla:

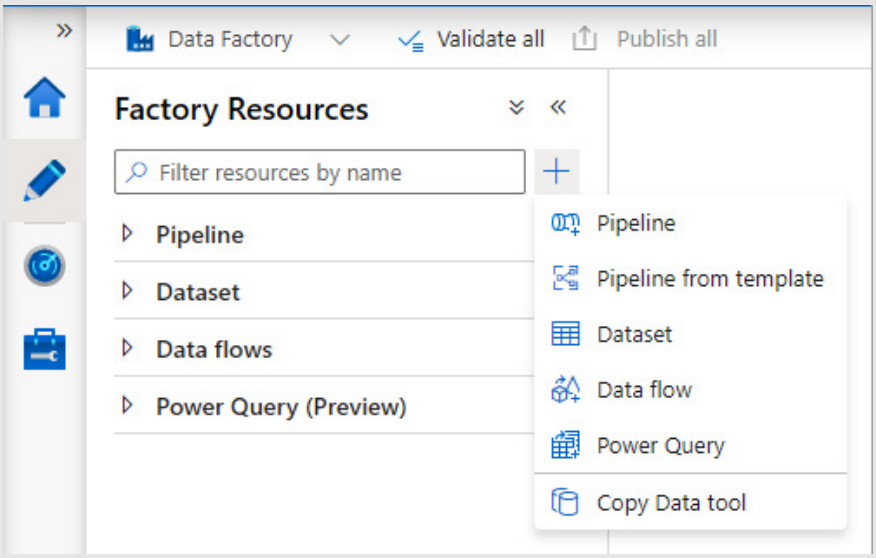

Figura 4.20 - Pantalla de inicio de la herramienta Copiar Datos del ADF

Una vez que haga clic en la herramienta Copiar Datos, se lanza una pantalla de asistente donde puede especificar los detalles de la carga incremental, como se ilustra en la siguiente captura de pantalla:

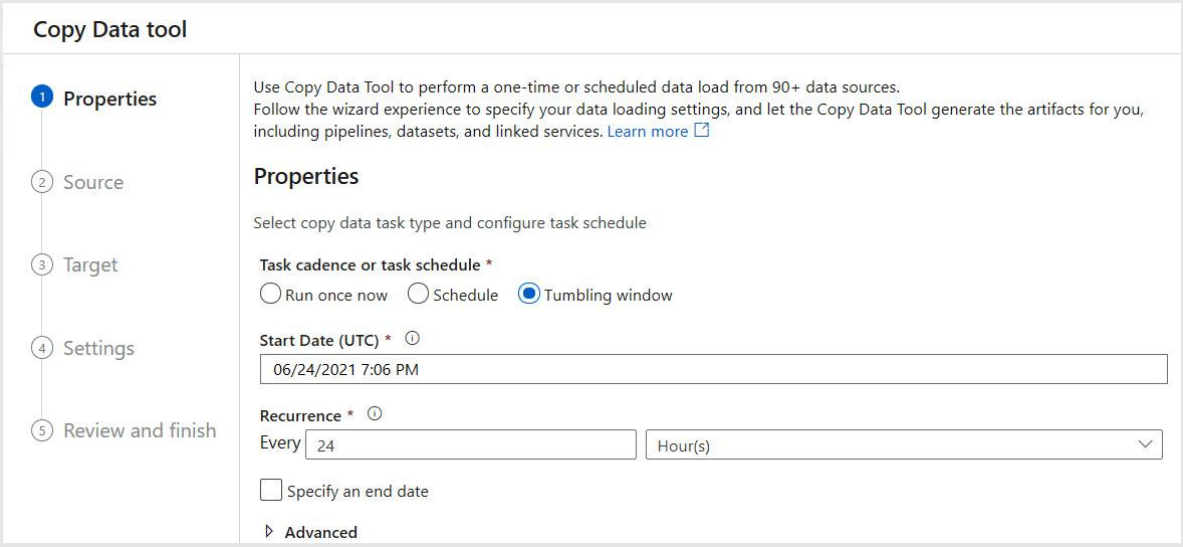

Figura 4.21 - Selección de tumbling window para la carga incremental basada en la hora de modificación del archivo

NOTA

En la pestaña de Propiedades que se muestra en la captura de pantalla anterior, es necesario seleccionar **Tumbling window** para la configuración de la **cadencia de tareas o del programa de tareas**; de lo contrario, la opción de carga incremental no se mostrará.

En la ventana Source (Origen), seleccione la opción **Incremental Load (Carga incremental): LastModified Date**, como se muestra en la siguiente captura de pantalla: Puede lanzar la funcionalidad de la herramienta Copiar Datos desde la pantalla principal del ADF, como se muestra en la siguiente captura de pantalla:

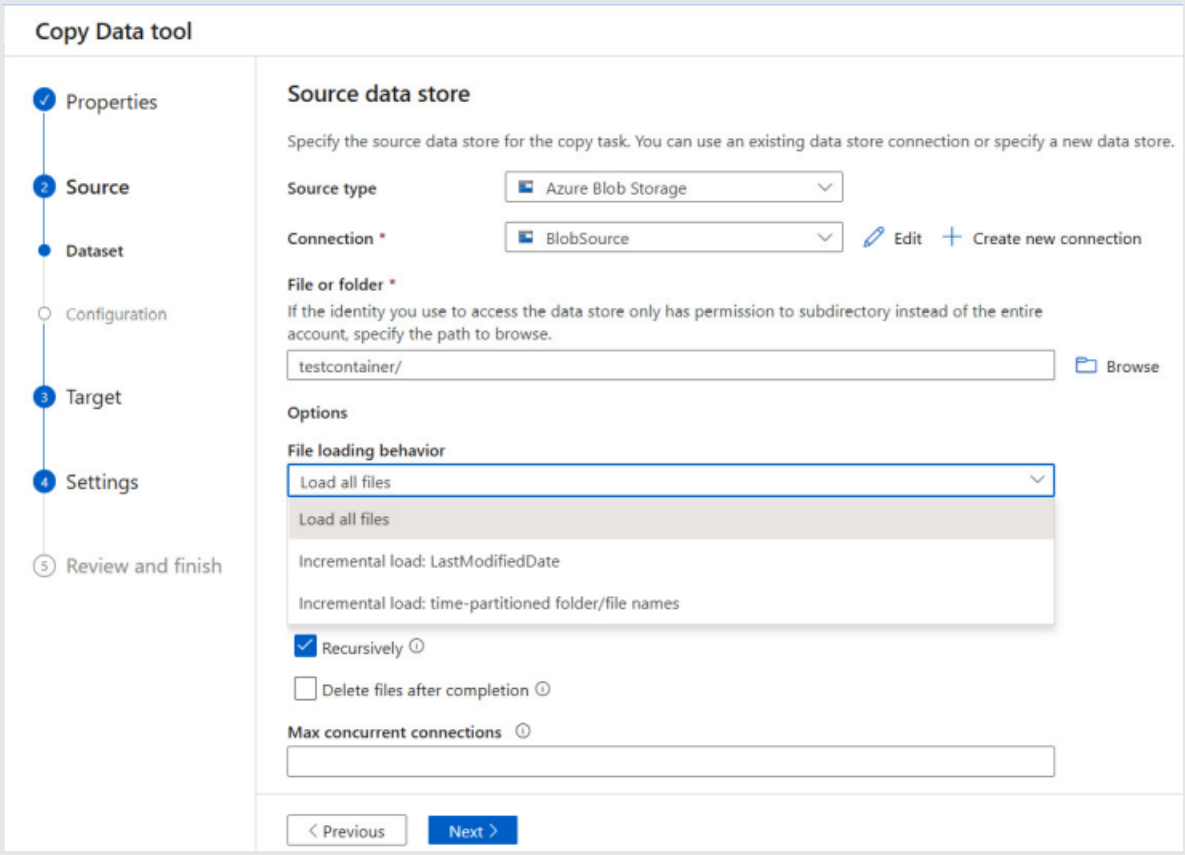

Figura 4.22 - Opción de carga incremental del ADF con marcas de tiempo LastModifiedDate

Rellene el resto de los campos y seleccione Siguiente en la pantalla de resumen para crear una cadena de copias incrementales utilizando las fechas de modificación de los archivos.

A continuación vamos a aprender a realizar copias incrementales utilizando estructuras de carpetas.

4.7.3. Particiones de archivos y estructuras de carpetas

Para ambas opciones de partición de archivos y datos organizados en estructuras de carpetas basadas en fechas, podemos utilizar la funcionalidad de la herramienta ADF Copy Data para realizar la carga incremental. Los archivos y carpetas en ambos enfoques tendrán una jerarquía de carpetas similar basada en la fecha/hora. Si recuerda la sección sobre la estructura de carpetas en el Capítulo 2, Diseño de una estructura de almacenamiento de datos, diseñamos una estructura de carpetas utilizando una jerarquía de fechas. Supongamos que nuestros datos de entrada están aterrizando en una carpeta con estructura de fechas, como se muestra aquí:

```
New York/Trips/In/2022/01/01
```

Intentemos subir estos datos de forma incremental a otra ubicación del blob storage de forma regular. La herramienta de Copia de Datos del ADF tiene soporte para la copia incremental de archivos y carpetas que están particionados usando la fecha/hora. De forma similar a cómo instanciamos una actividad de copia en la sección anterior para la copia incremental basada en marcas de tiempo de fechas de modificación de archivos, necesitamos instanciar la funcionalidad de la herramienta Copiar Datos con el campo de comportamiento de carga de archivos (File Loading behavior) establecido en Carga incremental: nombres de carpetas/archivos particionados por tiempo (Incremental load: time-partitioned folder/file names). En esta pantalla, una vez que comience a escribir el formato de entrada utilizando variables de fecha como {año}/{mes}/{día}, la sección Opciones se expande para mostrar los campos de formato de año, formato de mes y formato de día. Puede seleccionar el formato de estructura de carpetas que prefiera utilizando los desplegables y completar el resto del flujo. La siguiente captura de pantalla muestra un ejemplo de esto:

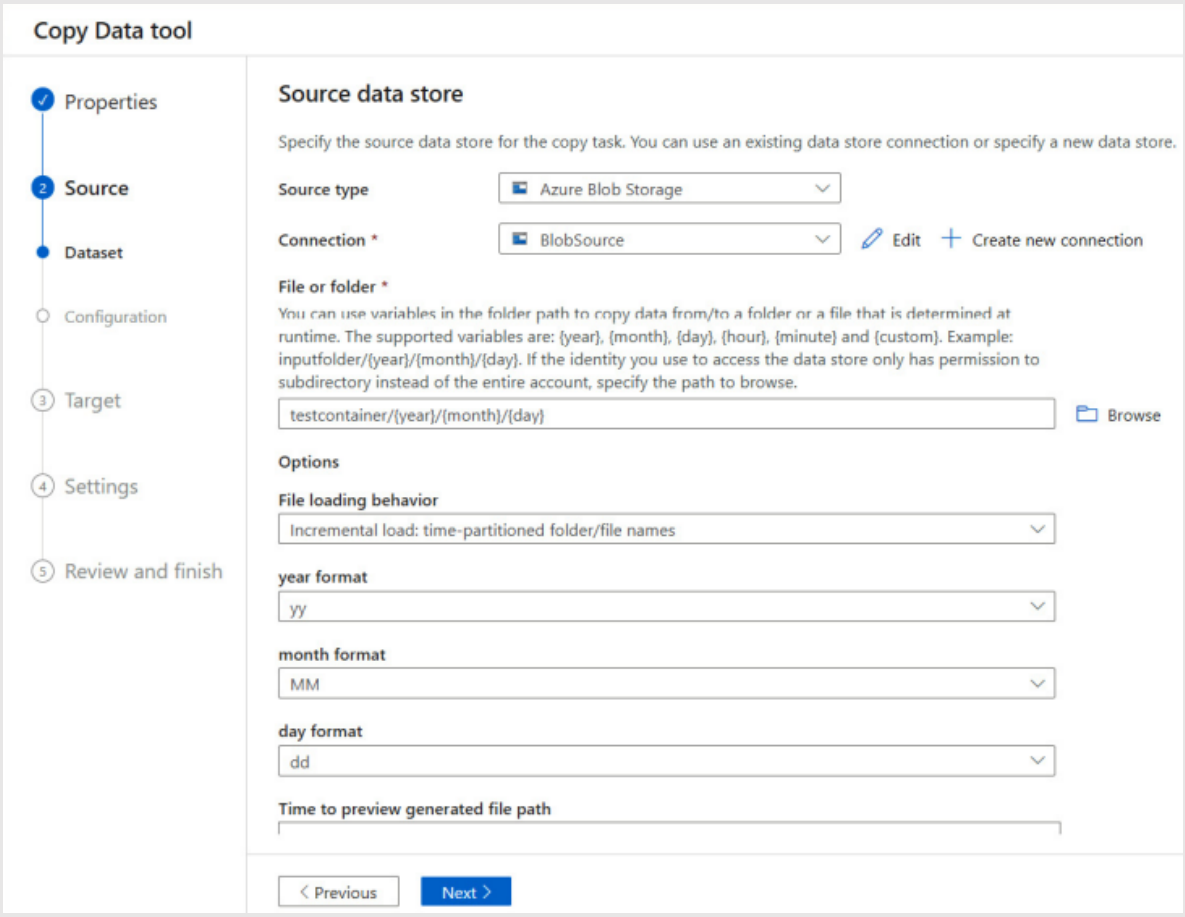

Figura 4.23 - Opción de carga incremental del ADF con carpetas separadas por tiempo

Una vez que revise los detalles y haga clic en Siguiente en la pantalla de resumen, se desplegará el pipeline incremental para estructuras de datos/carpetas particionadas.

Ahora hemos aprendido tres métodos diferentes para realizar copias incrementales. A continuación, vamos a explorar cómo seleccionar el data store analítico adecuado para nuestra capa de servicio.

4.8. Diseño de almacenes analíticos

Los almacenes analíticos pueden ser data stores SQL o NoSQL desplegados en la Serving Zone del data lake. El trabajo principal de un data store analítico es servir los datos generados por los pipelines de transformación de datos a las herramientas de BI de una manera rápida y sencilla. Los almacenes analíticos suelen ser objeto de consultas ad hoc por parte de los analistas de negocio y otros usuarios finales. Por lo tanto, estos almacenes tienen que funcionar muy bien con lecturas aleatorias. Azure proporciona una variedad de tecnologías de almacenamiento que pueden satisfacer estos requisitos. A continuación se indican algunas de las más importantes:

- **Azure Synapse Analytics (Serverless/dedicated SQL pools y Spark pools)**-Synapse Analytics proporciona tanto pools de SQL como pools de Spark. Entre ellos, los SQL dedicated pools son data warehouses de procesamiento masivo en paralelo (MPP) y suelen ser ideales para la mayoría de las situaciones de almacenamiento analítico. Los Serverless SQL pools pueden utilizarse para realizar consultas ad hoc. Los pools de Spark, por otro lado, pueden soportar cargas de trabajo analíticas a través de su almacén en memoria y su amplio soporte de columnstore. Ambos admiten interfaces similares a SQL/SQL.

- **Azure Databricks**: Azure Databricks es otra distribución de Spark que puede proporcionar capacidades limitadas de almacenamiento analítico a través de sus almacenes en memoria y su amplio soporte de columnstore. También admite interfaces similares a las de SQL.

- **Azure Cosmos DB**-Cosmos DB ofrece soporte para bases de datos de documentos, almacenes de clave-valor, almacenes de grafos, almacenes de columnas amplios, etc. También admite interfaces de tipo SQL/SQL.

- **Azure SQL Database/SQL Database en máquinas virtuales (VM) personalizadas**: Azure SQL proporciona soporte para bases de datos relacionales y, por supuesto, admite la interfaz SQL.

- **HBase/Phoenix en HDInsight**-Proporciona soporte de almacén analítico a través del almacén en memoria y un amplio soporte de columnstore. También admite interfaces similares a SQL/SQL.

- **Hive LLAP en HDInsight**: proporciona soporte de almacén analítico a través del almacén en memoria. También admite interfaces de tipo SQL/SQL.

- **Azure Data Explorer**: proporciona soporte para el almacén relacional y las series temporales. También admite interfaces de tipo SQL/SQL.

Hay otros criterios, como la seguridad y la escalabilidad, que puede considerar.

4.8.1. Consideraciones de seguridad

A continuación, se muestra una tabla comparativa de seguridad de alto nivel reproducida de la documentación de Azure que puede ayudarle a decidir el almacén analítico adecuado para usted:

Service	Authentication options	Supports data encryption at rest	Supports row-level security	Supports dynamic data masking	Supports firewalls
Synapse Analytics	SQL/Azure Active Directory (Azure AD)	Yes	Yes	Yes	Yes
Cosmos DB	Azure AD	Yes	No	No	Yes
Azure SQL	SQL/Azure AD	Yes	Yes	Yes	Yes
HBase/Phoenix	Local/Azure AD	Yes	Yes	Yes	Yes (with Azure virtual networks (VNets))
Hive LLAP	Local/Azure AD	Yes	Yes	Yes	Yes (with Azure VNets)
Azure Data Explorer	Azure AD	Yes	No	Yes	Yes
Azure Analysis Services	Azure AD	Yes	Yes	No	Yes

Figura 4.24 - Comparación de seguridad de los diferentes almacenes analíticos de Azure

Veamos a continuación la comparación de la escalabilidad.

4.8.2. Consideraciones sobre la escalabilidad

A continuación, se presenta otra tabla comparativa que puede ayudarle a decidir sobre un almacén analítico en función de sus consideraciones sobre el tamaño de los datos y la velocidad:

Service	Supports redundant regional servers for high availability (HA)	Supports query scale-out	Supports scale-up	Supports in-memory caching
Synapse Analytics	No	Yes	Yes	Yes
Cosmos DB	Yes	Yes	Yes	No
Azure SQL	Yes	No	Yes	Yes
HBase/Phoenix	Yes	Yes	No	No
Hive LLAP	No	Yes	No	Yes
Azure Data Explorer	Yes	Yes	Yes	Yes
Azure Analysis Services	No	Yes	Yes	Yes

Figura 4.25 - Comparación de la escalabilidad de los diferentes almacenes analíticos de Azure

Puede obtener más información sobre los criterios de selección de almacenes analíticos aquí: https://docs.microsoft.com/en-us/azure/architecture/data-guide/technology-choices/analytical-data-stores

Estamos casi al final de este capítulo. En la última sección, vamos a explorar qué son los metastores y cómo configurarlos en Azure Synapse Analytics y Azure Databricks.

4.9. Diseño de metastores en Azure Synapse Analytics y Azure Databricks

Los metastores almacenan los metadatos de los datos en servicios como Spark o Hive. Piensa en un metastore como un catálogo de datos que puede decirte qué tablas tienes, cuáles son los esquemas de las tablas, cuáles son las relaciones entre las tablas, dónde están almacenadas, etc. **Spark admite dos opciones de metastore: una versión en memoria y una versión externa**.

Los metastores en memoria tienen una accesibilidad y una escala limitadas. Pueden ayudar a los trabajos que se ejecutan en la misma máquina virtual Java (JVM), pero no mucho más allá. Además, los metadatos se pierden una vez que se apaga el clúster.

A efectos prácticos, Spark utiliza un metastore externo, y el único metastore externo soportado en el momento de escribir este libro era Hive Metastore. El metastore de Hive está maduro y proporciona interfaces de programación de aplicaciones (API) genéricas para acceder a él. Por lo tanto, en lugar de reconstruir un nuevo metastore, Spark simplemente utiliza el maduro y bien diseñado metastore de Hive para su propia catalogación.

Exploremos las características del metastore disponibles tanto en Azure Synapse Analytics como en Azure Databricks.

4.9.1. Azure Synapse Analytics

Synapse Analytics admite la compartición de metadatos entre sus pools computacionales, como Spark y SQL. Se puede acceder fácilmente a las bases de datos de Spark y a las tablas externas creadas con el formato Parquet desde los pools de SQL.

En la versión actual de Synapse se pueden compartir dos componentes principales de metadatos: las bases de datos y las tablas. Veámoslos en detalle.

Bases de datos

Cualquier base de datos que cree utilizando Spark SQL puede ser accedida directamente por los dedicated o serverless SQL pools de Synapse, siempre que ambos pools tengan acceso a nivel de almacenamiento (storage-level access) a la base de datos recién creada.

Por ejemplo, cree una tabla FactTrips en Spark ejecutando el siguiente código

```
spark.sql("CREATE DATABASE FactTrips")
```

Entonces, léelo desde SQL, así

```
SELECT * FROM sys.databases;
```

Usted no tiene que hacer ninguna configuración; Synapse hace automáticamente las bases de datos visibles a través de sus pools de cómputo. Ahora, veamos el otro componente de metadatos que puede ser compartido en la versión actual de Synapse: las tablas.

Tablas

Cualquier tabla basada en Parquet que Spark almacene en el directorio del warehouse se comparte automáticamente con los pools de SQL.

Cualquier tabla externa basada en Parquet creada por Spark también puede ser compartida con los pools SQL, proporcionando la ubicación de la tabla.

4.9.2. Azure Databricks (y Azure Synapse Spark)

Como vimos brevemente en el Capítulo 2, Diseño de una estructura de almacenamiento de datos, Azure Databricks es la versión Databricks de Spark alojada en Azure. Está disponible como un servicio independiente de Azure y está bien conectado con la mayoría de los servicios de Azure. Tanto Azure Databricks como Azure Synapse Spark utilizan Hive Metastore como su metastore externo. Estos metastores externos permiten el acceso a los datos entre diferentes clústeres de Spark y también entre otros cálculos como Hive. En teoría, cualquier servicio que pueda hablar con el metastore de Hive y tenga los niveles de acceso adecuados puede leer los catálogos de Spark persistidos en el metastore. El propio metastore de Hive suele desplegarse sobre una base de datos SQL como Azure SQL. La configuración de un metastore Hive externo es un procedimiento manual tanto para Databricks como para Synapse Spark. Hay dos conjuntos de configuraciones que tendrás que establecer, uno para Spark y otro para Hive.

En el lado de Spark, al crear un clúster, sólo tienes que añadir los siguientes parámetros en el campo de opciones de configuración de Spark. Aquí, asumimos que ya tienes un clúster Hive y una base de datos Azure SQL (que se utilizará como metastore) en funcionamiento. Puedes optar por utilizar cualquier base de datos SQL, no necesariamente Azure SQL para el metastore. Proceda como sigue:

1. Especifique la cadena de conexión de la conectividad de la base de datos de Java (JDBC) para un metastore JDBC, como sigue:

```
spark.hadoop.javax.jdo.option.ConnectionURL <cadena-de-conexión-mssql>
```

2. Especifique el nombre de usuario que se utilizará para el metastore, de la siguiente manera:

```
spark.hadoop.javax.jdo.option.ConnectionUserName <mssql-username>
```

3. Especifique la contraseña que se utilizará para el metastore, de la siguiente manera:

```
spark.hadoop.javax.jdo.option.ConnectionPassword <mssql-password>
```

Especifique el nombre de la clase del controlador para un metastore JDBC, de la siguiente manera:

```
spark.hadoop.javax.jdo.option.ConnectionDriverName
com.microsoft.sqlserver.jdbc.SQLServerDriver
```

Especifique la versión del metastore Hive, como sigue:

```
spark.sql.hive.metastore.version <hive-version>
```

Especifique la ubicación de los jars del metastore, archivos Java ARchive (JAR), de la siguiente manera:

```
spark.sql.hive.metastore.jars <hive-jar-source>
```

NOTA

Se añade un prefijo spark.hadoop para asegurarse de que estas opciones específicas de Hive se propagan al cliente del metastore.

Y en el servidor Hive, actualice las siguientes configuraciones con los valores correctos:

Especifique la cadena de conexión JDBC para un metastore JDBC, como sigue:

```
javax.jdo.option.ConnectionURL <mssql-connection-string>
```

Especifique el nombre de usuario para la base de datos del metastore, de la siguiente manera:

```
javax.jdo.option.ConnectionUserName <mssql-username>
```

Especifique la contraseña de la base de datos metastore, de la siguiente manera:

```
javax.jdo.option.ConnectionPassword <mssql-password>
```

Especifique el nombre de la clase del controlador de conexión para el metastore JDBC, de la siguiente manera:

```
javax.jdo.option.ConnectionDriverName com.microsoft.sqlserver.jdbc.SQLServerDriver
```

Si tus versiones de Spark/Hive no coinciden con las proporcionadas por Azure por defecto, tendrás que descargar la versión correcta de los archivos JAR y subirlos al espacio de trabajo. El siguiente enlace tiene más detalles sobre esto: https://docs.microsoft.com/en-us/azure/databricks/data/metastores/external-hive-metastore.

Resumen

Así concluye nuestro cuarto capítulo. Enhorabuena por haber llegado hasta aquí.

Sólo para recapitular, comenzamos con los fundamentos del modelado de datos y aprendimos sobre los esquemas Star y Snowflake. Luego aprendimos sobre el diseño de SCDs, los diferentes subtipos de SCDs, las jerarquías dimensionales, el manejo de datos temporales mediante el uso de dimensiones de tiempo, la carga de datos de forma incremental utilizando ADF, y la selección del almacén analítico correcto basado en los requisitos del cliente. Por último, aprendimos a crear metastores para Synapse y Azure Databricks. Todos estos temas completan el programa de estudios de DP203 - Diseño de la capa de servicio. Ahora has aprendido a diseñar tu propia capa de servicio en Azure.

Hemos llegado al final de nuestros capítulos de diseño. A partir del próximo capítulo nos centraremos en los detalles de implementación. Muy bien.