

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 5: Implementación de estructuras de almacenamiento de datos físicos	5
5.1. Requisitos técnicos.....	5
5.2. Introducción a Azure Synapse Analytics	5
5.3. Implementación de la compresión.....	7
5.3.1. Compresión de archivos usando Synapse Pipelines o ADF	7
5.3.2. Compresión de archivos con Spark	9
5.4. Implementación de la partición	10
5.4.1. Uso de pipelines ADF/Synapse para crear particiones de datos.....	10
5.4.2. Particionamiento para cargas de trabajo analíticas.....	12
5.5. Implementación de la partición horizontal o sharding	14
5.5.1. Sharding en los Synapse dedicated pools	14
Uso de dedicated SQL pools para crear particiones	14
5.5.2. Partición con Spark.....	16
Particionamiento en memoria	17
Particionamiento en disco.....	18
5.6. Implementación de distribuciones.....	19
5.6.1. Distribución Hash	19
5.6.2. Distribución Round-robin	20
5.6.3. Distribución replicada	20
5.7. Implementación de diferentes geometrías de tabla con los pools de Azure Synapse Analytics	21
5.7.1. Indexación Clustered Columnstore	21
5.7.2. Indexación Heap.....	21
5.7.3. Indexación Clustered.....	22
5.8. Implementar la redundancia de datos.....	23
5.8.1. Redundancia de almacenamiento de Azure en la región primaria.....	23
Almacenamiento redundante local (LRS).....	23
Almacenamiento redundante por zonas (ZRS)	23
5.8.2. Redundancia de almacenamiento Azure en regiones secundarias.....	23
Almacenamiento georredundante (GRS).....	24
Geo ZRS (GZRS).....	24
5.8.3. Replicación geográfica de Azure SQL	24

5.8.4. Replicación de datos SQL de Azure Synapse	24
5.8.5. Replicación de datos de CosmosDB	25
5.8.6. Ejemplo de configuración de la redundancia en Azure Storage	25
5.9. Implementación del archivo de datos (data archiving).....	28
Resumen.....	29

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 5: Implementación de estructuras de almacenamiento de datos físicos

Espero que hayas tenido una buena experiencia de aprendizaje hasta ahora. Continuemos nuestro viaje hacia la certificación con más temas interesantes en este capítulo. Hasta el capítulo anterior, nos hemos centrado en los aspectos de diseño, pero a partir de ahora, nos centraremos en los detalles de la implementación. Aprenderemos a implementar los conceptos a nivel de almacenamiento que hemos aprendido en los capítulos anteriores. Una vez que completes este capítulo, deberías ser capaz de decidir e implementar lo siguiente: qué tipo de fragmentación de datos se requiere, cuándo comprimir tus datos, cuántas particiones crear, qué tipo de redundancia de datos mantener, etc.

En este capítulo cubriremos los siguientes temas:

- Introducción a Azure Synapse Analytics
- Implementación de la compresión
- Implementación de la partición
- Implementación de la partición horizontal o sharding
- Implementación de distribuciones
- Implementación de diferentes geometrías de tablas con pools de Azure Synapse Analytics
- Implementación de la redundancia de datos
- Implementación del archivo de datos

5.1. Requisitos técnicos

Para este capítulo, necesitará lo siguiente

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

¡Empecemos a ponerlo en práctica!

5.2. Introducción a Azure Synapse Analytics

La mayoría de los ejemplos de este capítulo se ejecutarán en un área de trabajo de Azure Synapse Analytics, así que veamos cómo crear una. Proceda de la siguiente manera:

1. En el portal de Azure, busque Synapse. De los resultados, selecciona Azure Synapse Analytics.

2. Una vez dentro, haz clic en + Crear e introduce los detalles de un nuevo área de trabajo. La pantalla de Crear área de trabajo de Synapse se muestra en la siguiente captura de pantalla:

Microsoft Azure Search resources, services, and docs (G+/I)

Home > Azure Synapse Analytics >

Create Synapse workspace ...

Create a Synapse workspace to develop an enterprise analytics solution in just a few clicks.

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all of your resources.

Subscription * ⓘ Azure subscription 1

Resource group * ⓘ

Create new

Managed resource group ⓘ Enter managed resource group name

Workspace details

Name your workspace, select a location, and choose a primary Data Lake Storage Gen2 file system to serve as the default location for logs and job output.

Workspace name * Enter workspace name

Region * East US

Select Data Lake Storage Gen2 * ⓘ ☒ From subscription ☐ Manually via URL

Account name * ⓘ

Create new

Review + create < Previous Next: Security >

3. Una vez hecho esto, haz clic en el botón Review + create y, finalmente, en el botón Crear de esa pantalla para crear una nueva área de trabajo de Synapse.

Esta área de trabajo nos proporcionará Structured Query Language (SQL) y Spark pools, necesarios para experimentar con los distintos conceptos que hemos aprendido en los capítulos anteriores. Empecemos por implementar la compresión en la siguiente sección.

5.3. Implementación de la compresión

En los capítulos anteriores, hemos aprendido la importancia de la compresión de datos en los data lakes. A medida que el tamaño de los datos en los data lakes crece, es importante que almacenemos los datos en un formato comprimido para ahorrar costes. Existen numerosas formas de implementar la compresión en Azure. Hay muchas bibliotecas de compresión disponibles en el mercado y, técnicamente, todo lo que tenemos que hacer es escribir algunos scripts para llamar a esas bibliotecas para comprimir los datos. Pero escribir scripts ad hoc conlleva su propia complejidad de mantenimiento, así que veamos algunas formas fáciles de mantener de implementar la compresión en Azure usando Azure Synapse Pipelines. Lo mismo se puede lograr utilizando Azure Data Factory también. Azure Synapse Pipelines no es más que la misma implementación de Azure Data Factory dentro de Synapse Analytics.

5.3.1. Compresión de archivos usando Synapse Pipelines o ADF

En el capítulo anterior, aprendimos sobre ADF. Al igual que ADF, Synapse Pipelines puede utilizarse para crear pipelines para realizar una amplia gama de actividades, como extraer, transformar, cargar (ETL), extraer, cargar, transformar (ELT), copiar datos, etc. ADF y Synapse Pipelines admiten varias docenas de servicios como fuentes y destinos de datos, tanto dentro de Azure como fuera de Azure. Entre ellos, **servicios como Azure Blob Storage o Azure Data Lake Storage Gen2 (ADLS Gen2)** admiten la opción de comprimir los datos antes de copiarlos. **Veamos un ejemplo de cómo leer archivos sin comprimir y comprimirlos antes de escribirlos de nuevo en ADLS Gen2 utilizando Synapse Pipelines.**

Proceda como sigue:

1. Cree una instancia de la **herramienta Copiar Datos** haciendo clic en el signo +, como se muestra en la siguiente captura de pantalla:

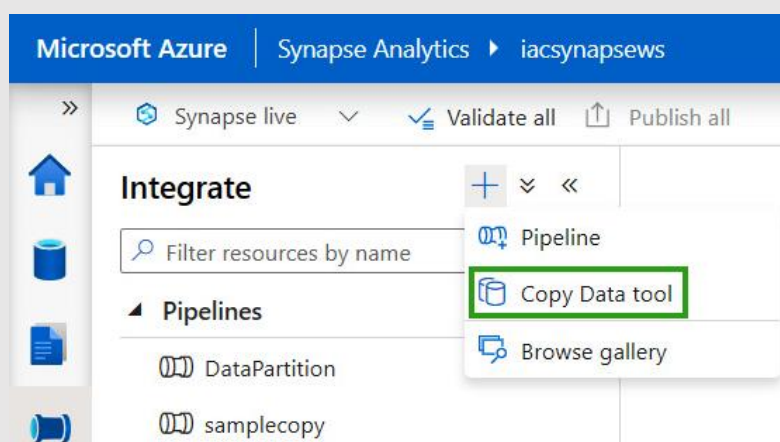


Figura 5.2 - Seleccionando la herramienta Copy Data en Synapse Pipelines

2. En la instancia de la **herramienta Copiar Datos**, introduzca las carpetas de **Origen y Destino** en su Azure data store.

3. En la página de **configuración**, puede especificar una opción de **tipo de compresión**, como se muestra en la siguiente captura de pantalla:

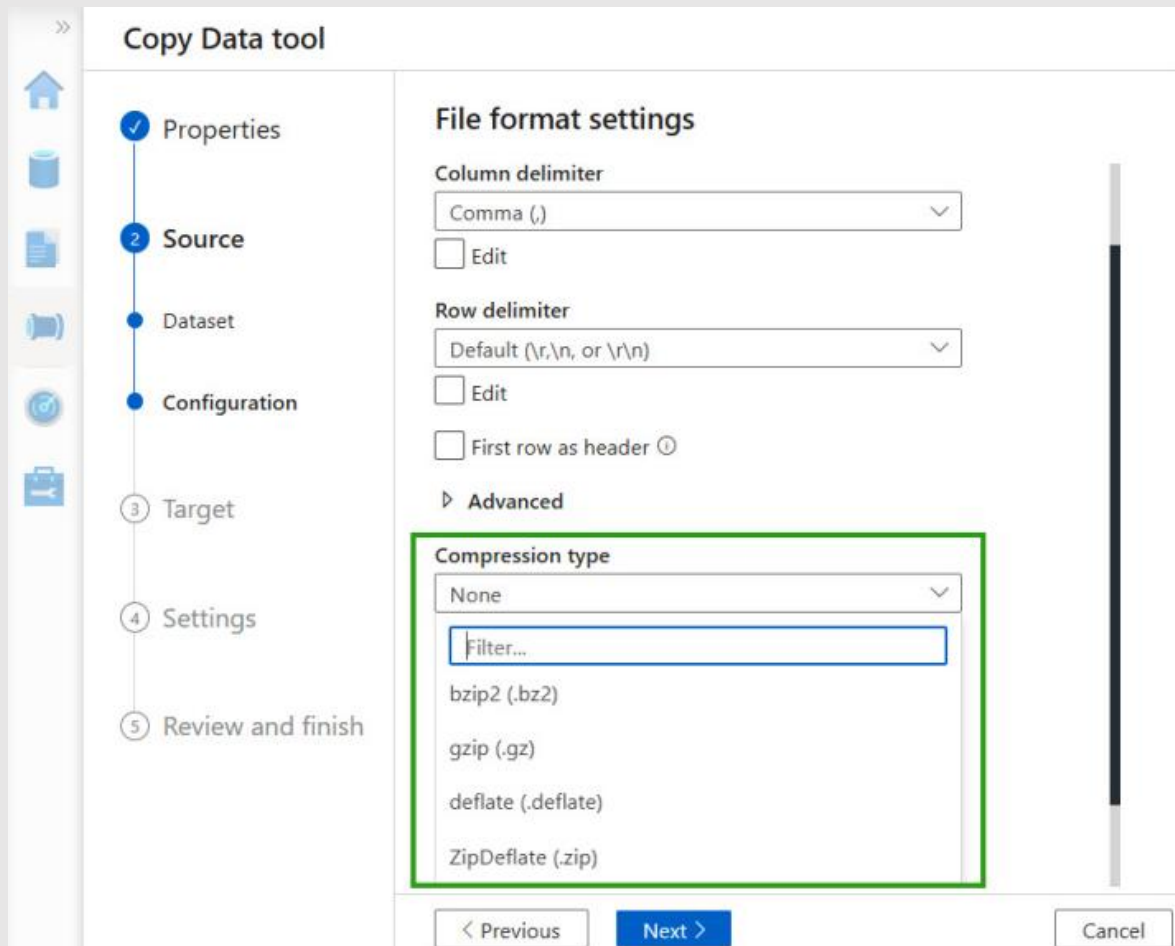


Figura 5.3 - Opciones de tipo de compresión en la página del **data store de destino** de la herramienta Copiar Datos

4. Seleccione una de las opciones de tipo de compresión, rellene los demás campos obligatorios en las otras pestañas y, finalmente, haga clic en Publicar en la página Revisar y finalizar para guardar esta instancia de la herramienta Copiar datos.
5. A continuación, puede activar esta instancia de la herramienta de copia de datos bajo demanda o configurar un activador recurrente para realizar la compresión a intervalos regulares desde la pantalla de la canalización del ADF.

Así de sencillo es configurar un trabajo de compresión regular utilizando Synapse Pipelines (de forma similar en ADF). **A continuación, vamos a explorar cómo comprimir archivos utilizando Spark**, que es uno de los pools de cómputo disponibles en Synapse. También puedes optar por utilizar Azure Databricks Spark. Los ejemplos proporcionados aquí funcionarán bien con cualquier tipo de Spark.

5.3.2. Compresión de archivos con Spark

Spark proporciona bibliotecas que pueden escribir directamente las salidas en formatos comprimidos como Parquet, ORC, etc. Además, podemos especificar los algoritmos de compresión a utilizar. Por ejemplo, el siguiente script de Python almacena los datos en formato Parquet utilizando la compresión gzip:

```
columnNames = ["nombre", "licencia", "género", "salario"]
driverData = [
    ('Alice', 'A224455', 'Mujer', 3000),
    ('Bryan', 'B992244', 'Male', 4000),
    ('Catherine', 'C887733', 'Mujer', 4000)
]

df = spark.createDataFrame(data= driverData, schema = columnNames)
df.write.option("compression", "gzip").parquet("abfss://path/to/output")
```

NOTA

Spark soporta estas opciones de compresión: **snappy**, **gzip**, **lzo**, **brotli**, **lz4** y **zstd**. Puedes aprender más sobre las opciones de compresión de Spark en <https://spark.apache.org/docs/latest/configuration.html>.

Ahora que hemos aprendido a comprimir datos utilizando ADF y Spark, vamos a ver las diferentes formas de particionar datos en Azure Synapse.

5.4. Implementación de la partición

En el Capítulo 3, Diseño de una estrategia de partición, cubrimos los fundamentos del particionamiento. **En esta sección, aprenderemos a implementar los diferentes tipos de partición.** Comenzaremos con el particionamiento en los Azure data stores y luego veremos el particionamiento para las cargas de trabajo analíticas.

En el caso de la partición basada en el almacenamiento, la técnica principal consiste en **particionar los datos en la estructura de carpetas correcta.** En los capítulos anteriores, aprendimos a almacenar los datos en un formato basado en la fecha. La recomendación de Azure es utilizar el siguiente patrón:

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

Vamos a aprender a implementar la creación de carpetas de forma automatizada utilizando ADF.

5.4.1. Uso de pipelines ADF/Synapse para crear particiones de datos

Puede utilizar ADF o Synapse Pipelines, ya que ambos utilizan la misma tecnología ADF. En este ejemplo, estoy utilizando Synapse Pipelines. Veamos los pasos para particionar los datos de forma automatizada:

1. En su área de trabajo de Synapse, seleccione la pestaña Pipelines y seleccione una actividad de Data flow, como se ilustra en la siguiente captura de pantalla:

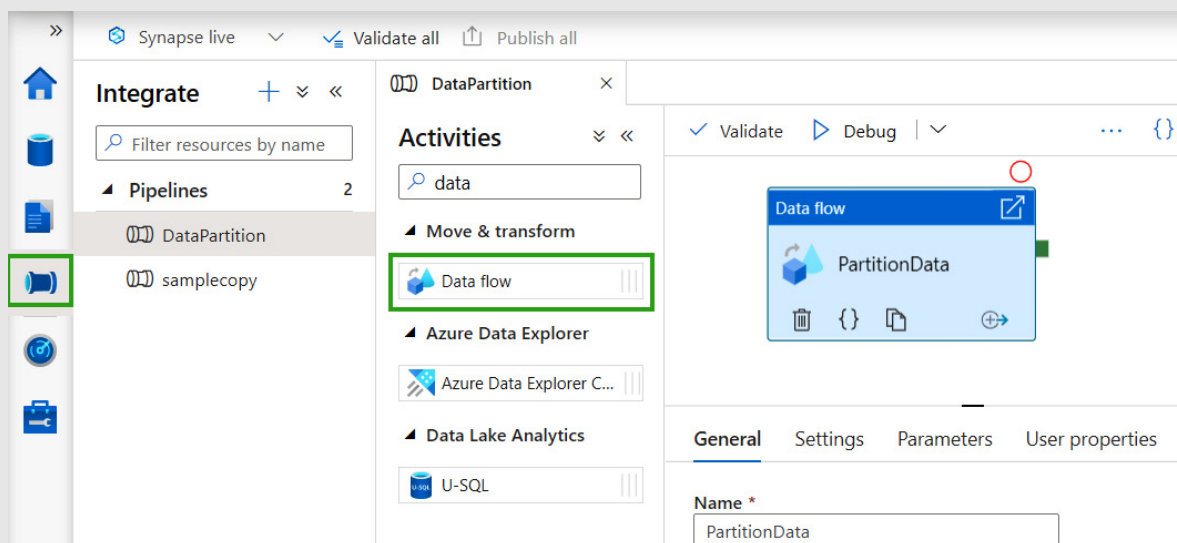


Figura 5.4 - Selección de la actividad de Data flow en Synapse Pipelines

2. Haga doble clic en la actividad de Data flow para mostrar los nodos de Origen y Destino.

3. Configure los detalles de la **Fuente**. En la pestaña **Opciones de origen**, rellene el campo **Rutas comodín**, de forma similar al ejemplo que se muestra en la siguiente captura de pantalla:

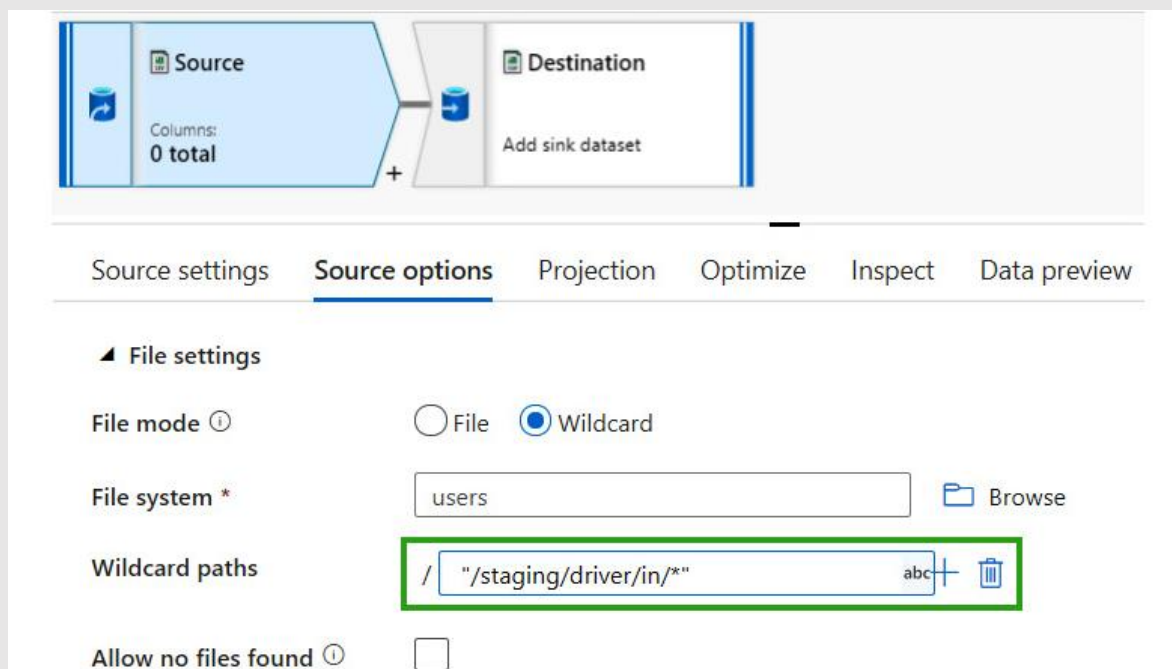


Figura 5.5 - Introducción de la información de origen en la actividad Data flow

4. Ahora, haga clic en el nodo **Destino** y rellene los detalles de la carpeta de destino, como se ilustra en la siguiente captura de pantalla:

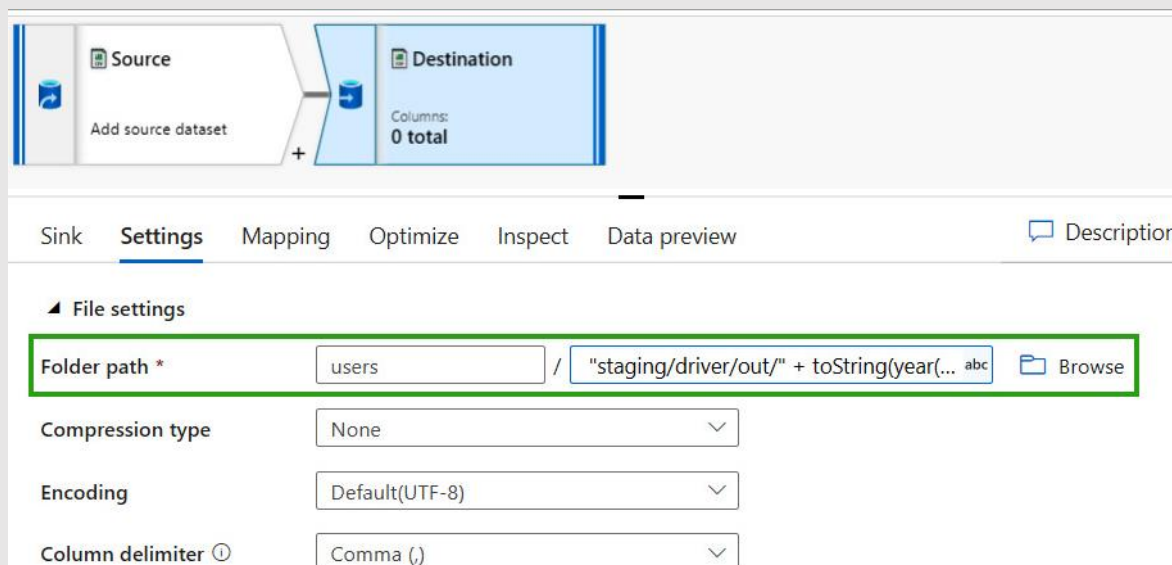


Figura 5.6 - Introducción de la información de destino en la actividad Data Flow

5. El campo importante aquí es la ruta de la carpeta. Este campo acepta expresiones dinámicas. Si hace clic en el cuadro de texto de la ruta de la carpeta, aparece una herramienta de construcción de expresiones que puede utilizarse para construir expresiones complejas que indiquen el formato de archivo que necesitamos.

6. He aquí una expresión de ejemplo que se puede utilizar para crear directorios de fechas:

```
"staging/driver/out/" + toString(year(currentDate())) + "/" + toString(month(currentDate())) + "/" + toString(dayOfMonth(currentDate()))
```

7. Si la fecha actual es el 1 de enero de 2022, la expresión anterior generaría una carpeta como ésta

```
staging/driver/out/2022/01/01
```

8. Y al día siguiente, se crearía una nueva carpeta con la fecha 02, y así sucesivamente.

9. Una vez que hayas terminado de añadir la ruta de la carpeta, publica el pipeline y actívalo.

Así es como podemos particionar y almacenar automáticamente los datos entrantes. Veamos ahora las opciones de partición disponibles para las cargas de trabajo analíticas.

5.4.2. Particionamiento para cargas de trabajo analíticas

En el Capítulo 3, Diseño de una estrategia de partición, aprendimos sobre los tres tipos de partición para cargas de trabajo analíticas, que se enumeran de nuevo aquí:

- Partición horizontal o sharding
- Partición vertical
- Partición funcional

Entre ellas, las técnicas de **particionamiento horizontal o sharding** son de naturaleza dinámica y se realizan después de la creación de los esquemas. Suelen realizarse regularmente durante la carga de datos o durante la fase de transformación de los mismos. Los Synapse dedicated pools y los Spark pools ofrecen varias opciones para realizar el sharding. A continuación dedicaremos una sección completa a la fragmentación.

El **particionamiento vertical o funcional**, por otro lado, se realiza en el momento de la construcción del esquema. Estos tipos de partición implican la determinación del conjunto adecuado de tablas en función de las consultas y la distribución de los datos. Una vez que se determinan las tablas requeridas en función de las necesidades de nuestro negocio, la implementación sólo implica la creación de múltiples tablas mediante el comando CREATE TABLE y la vinculación de las tablas mediante una restricción, como por ejemplo una restricción FOREIGN KEY.

NOTA

En el momento de escribir este libro, la restricción FOREIGN KEY todavía no está soportada en los dedicated SQL pools.

Desde el punto de vista de la certificación, el sharding es más importante, así que vamos a centrarnos en él.

5.5. Implementación de la partición horizontal o sharding

Vamos a explorar el sharding desde dos perspectivas diferentes: un dedicated SQL pool y Spark. Ten en cuenta que utilizaremos los términos partición horizontal y sharding indistintamente a lo largo del libro, pero significan lo mismo.

5.5.1. Sharding en los Synapse dedicated pools

Synapse SQL dedicated pools tienen tres tipos diferentes de tablas en función de cómo se almacenan los datos, que se describen a continuación:

- Clustered columnstore (almacén de columnas agrupadas)
- Clustered index (índice agrupado)
- Heap

Más adelante en este capítulo aprenderemos más sobre estos tipos de tablas. **Los Synapse dedicated pools soportan el sharding para todos estos tipos de tablas. Proporcionan tres maneras diferentes de fragmentar (shard) los datos, como se indica a continuación:**

- Hash
- Round-robin
- Replicado

Estos métodos a través de los cuales un SQL dedicated pool distribuye los datos entre sus tablas también se denominan técnicas de distribución. Las técnicas de sharding y distribución son tecnologías superpuestas que siempre se especifican juntas en las sentencias SQL CREATE TABLE.

NOTA

El dedicated SQL pool divide por defecto los datos en 60 distribuciones. Las particiones que especificamos explícitamente se suman a las particiones que el SQL dedicated pool ya crea. Por lo tanto, asegúrese de no acabar particionando demasiado los datos, ya que esto puede afectar negativamente al rendimiento de la consulta.

Veamos ahora ejemplos de cómo fragmentar (shard) datos en un dedicated SQL pool.

Uso de dedicated SQL pools para crear particiones

Los dedicated SQL pools tienen una opción llamada PARTITION que se puede utilizar durante el proceso de creación de la tabla para definir cómo se van a particionar los datos. Utilicemos el mismo ejemplo base que utilizamos en el Capítulo 2, Diseño de una estructura de almacenamiento de datos, para probar la partición. Procederemos como sigue:

1. Creemos una tabla simple que particione sobre el campo tripDate, de la siguiente manera:

```
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 )
    )
)
```

2. Inserte algunos datos de muestra en la tabla, como se muestra en el siguiente bloque de código:

```
INSERT INTO dbo.TripTable VALUES (100, 200, 300, 20220101, 'New York', 'New Jersey');
INSERT INTO dbo.TripTable VALUES (101, 201, 301, 20220101, 'Miami', 'Dallas');
INSERT INTO dbo.TripTable VALUES (102, 202, 302, 20220102, 'Phoenix', 'Tempe');
```

3. Ahora, vamos a ejecutar la consulta como se muestra en la siguiente captura de pantalla. No se preocupe si no entiende los detalles de la consulta - sólo trata de obtener los detalles de la partición en un formato fácil de leer. Esta consulta se puede encontrar en la documentación de Synapse:

The screenshot shows the SQLPartitions application interface. The top section contains a SQL query editor with the following code:

```

28
29 SELECT QUOTENAME(s.[name])+'.'+QUOTENAME(t.[name]) as Table_name
30 ,
31 i.[name] as Index_name
32 ,
33 p.partition_number as Partition_nmbr
34 ,
35 p.[rows] as Row_count
36 ,
37 p.[data_compression_desc] as Data_Compression_desc
38 FROM
39 sys.partitions p
40 JOIN sys.tables t ON p.[object_id] = t.[object_id]
41 JOIN sys.schemas s ON t.[schema_id] = s.[schema_id]
42 JOIN sys.indexes i ON p.[object_id] = i.[object_id]
43 AND p.[index_id] = i.[index_id]
44 WHERE t.[name] = 'TripTable'
45 ;

```

Below the query editor, the 'Results' tab is active, displaying a table with 5 columns: Table_name, Index_name, Partition_nmbr, Row_count, and Data_Compression_desc. The table contains 4 rows of data, all for the '[dbo].[TripTable]' table, with 4 different partitions, each having 250 rows and using COLUMNSTORE compression.

Table_name	Index_name	Partition_nmbr	Row_count	Data_Compression_desc
[dbo].[TripTable]	ClusteredIndex_495fcaa8f25a4b4...	1	250	COLUMNSTORE
[dbo].[TripTable]	ClusteredIndex_495fcaa8f25a4b4...	2	250	COLUMNSTORE
[dbo].[TripTable]	ClusteredIndex_495fcaa8f25a4b4...	3	250	COLUMNSTORE
[dbo].[TripTable]	ClusteredIndex_495fcaa8f25a4b4...	4	250	COLUMNSTORE

Figura 5.7 - Salida del comando de partición

4. Como puede ver en la sección de resultados, se han creado cuatro particiones.

Dos de los atributos utilizados al definir la partición son RANGE RIGHT y RANGE LEFT. Aprendimos sobre estas palabras clave en el Capítulo 2, Diseño de una estructura de almacenamiento de datos, mientras aprendíamos sobre la poda de datos. Sólo para recapitular, RANGE RIGHT asegura que el valor especificado en la sintaxis de PARTITION pertenecerá a la partición del lado derecho y viceversa para la palabra clave RANGE LEFT.

A continuación, veamos las opciones que ofrece Spark para el sharding.

5.5.2. Partición con Spark

Spark particiona por defecto los datos en función del número de núcleos disponibles o del número de bloques del Sistema de Archivos Distribuidos Hadoop (HDFS) (Hadoop Distributed File System (HDFS) blocks) (si se ejecuta en HDFS). Si necesitamos particionar los datos en cualquier otro formato personalizado, Spark también ofrece opciones para ello. **Spark soporta dos tipos de partición: partición en memoria y partición en disco.** Veamos estos dos tipos de partición en detalle.

Particionamiento en memoria

Spark proporciona los siguientes tres métodos para realizar el particionamiento en memoria:

- **repartition()**-Para aumentar el número de particiones.
- **coalesce()**-Para disminuir el número de particiones.
- **repartitionByRange()**-Es una especialización del comando repartition donde se pueden especificar los rangos.

Veamos un fragmento de Spark de ejemplo para entender estos tres tipos de partición, como sigue:

```
columnNames = ["name","license","gender","salary"]
driverData = [
    ('Alice', 'A224455', 'Female', 3000),
    ('Bryan','B992244','Male',4000),
    ('Catherine','C887733','Female',2000),
    ('Daryl','D229988','Male',3000),
    ('Jenny','J663300','Female', 6000)
]

df = spark.createDataFrame(data= driverData, schema = columnNames)
print("Default Partitions: " + str(df.rdd.getNumPartitions()))
repartitionDF = df.repartition(3)
print("Repartition Partitions: " + str(repartitionDF.rdd.getNumPartitions()))

coalesceDF=df.coalesce(2)
print("Coalesce Partitions: " + str(coalesceDF.rdd.getNumPartitions()))
repartitionRangeDF = df.repartitionByRange(1,'salary')
print("Range Partitions: " + str(repartitionRangeDF.rdd.getNumPartitions()))
```

La salida del fragmento de código anterior sería algo así

```
Default Partitions: 8
Repartition Partitions: 3
Coalesce Partitions: 2
Range Partitions: 1
```

Como puedes ver, con cada uno de los comandos de partición, el número de particiones se actualiza.

Ahora que hemos explorado las particiones en memoria en Spark, vamos a ver los métodos de partición en disco.

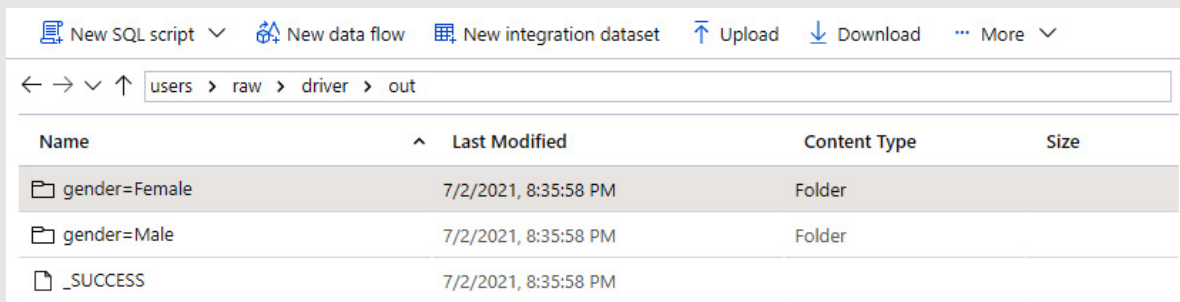
Particionamiento en disco

Spark proporciona el operador `partitionBy()`, que puede utilizarse para particionar los datos y almacenarlos en diferentes archivos mientras se escribe la salida. Aquí hay un fragmento de código de ejemplo para realizar `partitionBy()`. Este ejemplo continúa el ejemplo de la sección de partición en memoria:

```
df = spark.createDataFrame(data= driverData, schema = columnNames)

df.write.partitionBy("gender","salary")..parquet("abfss://path/to/output/")
```

El fragmento de código anterior creará dos carpetas separadas, una para mujeres y otra para hombres. La salida será como se muestra aquí:



Name	Last Modified	Content Type	Size
gender=Female	7/2/2021, 8:35:58 PM	Folder	
gender=Male	7/2/2021, 8:35:58 PM	Folder	
_SUCCESS	7/2/2021, 8:35:58 PM		

Figura 5.8 - Salida del comando PartitionBy

Con esto, hemos cubierto cómo podemos realizar el particionamiento usando Spark. Ahora, pasemos a las distribuciones, que es un concepto relacionado con las particiones en los dedicated SQL pools.

5.6. Implementación de distribuciones

Un motor de **procesamiento paralelo masivo (MPP)** de un dedicated SQL pool divide los datos en 60 particiones paralelas y las ejecuta en paralelo. Cada una de estas particiones más pequeñas, junto con los recursos informáticos para ejecutar las consultas, se denomina distribución. Una distribución es una unidad básica de procesamiento y almacenamiento para un dedicated SQL pool.

Los dedicated SQL pool ofrecen tres opciones de distribución. Veamos cada una de ellas en detalle.

5.6.1. Distribución Hash

Este tipo de distribución distribuye los datos basándose en una función hash. Las filas con los mismos valores para la columna hash se moverán siempre a la misma partición. Esto se puede implementar proporcionando el valor `DISTRIBUTION = HASH (COLUMN_ID)` en la cláusula `WITH` de `CREATE TABLE`. He aquí un ejemplo:

```
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 )
    )
)
```

CONSEJO

Utilice la distribución hash si el tamaño de la tabla es superior a **2 gigabytes (GB)** o si la tabla tiene actualizaciones muy frecuentes.

5.6.2. Distribución Round-robin

Este tipo de distribución simplemente distribuye los datos de forma aleatoria entre los nodos de forma round-robin. Se puede implementar proporcionando el valor `DISTRIBUTION = ROUND_ROBIN` en la cláusula `WITH` de `CREATE TABLE`. He aquí un ejemplo:

```
WITH  
(  
    CLUSTERED COLUMNSTORE INDEX,  
    DISTRIBUTION = ROUND_ROBIN  
    PARTITION (...)  
)
```

Elige distribuciones `round_robin` mientras se cargan los datos en las tablas de preparación o cuando no hay una buena indexación para elegir.

5.6.3. Distribución replicada

Este tipo de distribución simplemente copia la tabla completa en todos los nodos. Se puede implementar proporcionando el valor `DISTRIBUTION = REPLICATE` en la cláusula `WITH` de `CREATE TABLE`. He aquí un ejemplo:

```
WITH  
(  
    CLUSTERED COLUMNSTORE INDEX,  
    DISTRIBUTION = REPLICATE  
    PARTITION (...)  
)
```

Utilice `REPLICATE` para copiar datos pequeños pero de acceso frecuente, como tablas de catálogos minoristas, tablas de gráficos de precios, etc.

Espero que hayas comprendido bien la fragmentación y las distribuciones en un pool de Synapse SQL. Ahora que sabemos como implementar distribuciones, vamos a aprender sobre los otros atributos de las tablas de Synapse.

5.7. Implementación de diferentes geometrías de tabla con los pools de Azure Synapse Analytics

El término geometría de tabla no es la terminología estándar de las bases de datos, así que estoy tomando una conjetura educada que el equipo de certificación se refería a las diferentes características de las tablas de Azure Synapse dedicated pool.

Las principales características de las tablas pool dedicadas de Synapse son las particiones, los índices y las distribuciones. Ya hemos cubierto las particiones y las distribuciones en detalle, así que nos centraremos en la característica restante, que es la indexación.

Los SQL dedicated pools proporcionan los siguientes tres tipos de indexación:

- Indexación Clustered Columnstore
- Indexación Heap
- Indexación Clustered

Veámoslos en detalle.

5.7.1. Indexación Clustered Columnstore

Clustered columnstore es la opción de indexación por defecto de una tabla SQL dedicated pool. Este tipo de indexación funciona mejor para las tablas de hechos grandes. Es un almacenamiento de datos basado en columnas y proporciona niveles muy altos de compresión y un mejor rendimiento de las consultas que los índices basados en filas.

A continuación se muestra un ejemplo de cómo crear una tabla con indexación clustered columnstore:

```
CREATE TABLE dbo.Driver
(
    [driverId] INT NOT NULL,
    [name] VARCHAR(40),
    ...
)
WITH ( CLUSTERED COLUMNSTORE INDEX );
```

5.7.2. Indexación Heap

Las tablas heap se utilizan como tablas temporales de carga de datos, ya que proporcionan tiempos de carga más rápidos. Suelen utilizarse para almacenar datos antes de cargarlos en otras tablas refinadas. **La indexación heap funciona mejor para las tablas más pequeñas.**

A continuación se muestra un ejemplo de cómo crear una tabla con indexación heap:

```
CREATE TABLE dbo.Driver
(
    [driverId] INT NOT NULL,
    [name] VARCHAR(40),
    ...
)
WITH ( HEAP );
```

5.7.3. Indexación Clustered

Las tablas de índices clustered son tablas de almacenamiento basadas en filas. Suelen ser más rápidas para las consultas que necesitan búsquedas de filas con filtros muy selectivos en la columna del índice clustered.

A continuación, se muestra un ejemplo de cómo crear una tabla con indexación clustered:

```
CREATE TABLE dbo.Driver
(
    [driverId] INT NOT NULL,
    [name] VARCHAR(40),
    ...
)
WITH ( CLUSTERED INDEX (driverId) );
```

NOTA

Si el tamaño de su tabla es inferior a los 60 millones de filas recomendados para la indexación clustered columnstore, considere la posibilidad de utilizar tablas de índices heap o clustered.

Ahora ha aprendido algunas de las características importantes de las tablas SQL dedicated pool, como el particionamiento, la indexación y las distribuciones. Ahora, veamos nuestro siguiente tema: la redundancia de datos.

5.8. Implementar la redundancia de datos

La redundancia de datos es el proceso de almacenamiento de múltiples copias de datos en diferentes ubicaciones, con el fin de proteger los datos de eventos tales como fallos de alimentación, fallos de disco, fallos de red, e incluso catástrofes mayores como cortes de todo el centro de datos. Azure Storage proporciona múltiples opciones para la redundancia de datos tanto a nivel de centro de datos local como entre centros de datos. Podemos agrupar a grandes rasgos las opciones en estas dos categorías:

- Redundancia de la región primaria
- Redundancia de la región secundaria

Veamos estas opciones en detalle.

5.8.1. Redundancia de almacenamiento de Azure en la región primaria

Este tipo de redundancia cubre fallos localizados, como fallos de disco, fallos de máquina y fallos de rack. Hay dos tipos de redundancia en la región primaria, como se detalla a continuación.

Almacenamiento redundante local (LRS)

Con este tipo de redundancia, los datos se copian tres veces en varias máquinas de la misma región física dentro de un centro de datos. Esta opción proporciona protección contra fallos locales, como fallos de disco, de máquina o de red local. Si las tres copias de datos acaban residiendo en el mismo rack y si éste pierde energía, podríamos acabar perdiendo los datos. Esta es la más barata de todas las opciones de almacenamiento redundante.

Almacenamiento redundante por zonas (ZRS)

Este tipo de redundancia es similar al LRS, pero los datos se copian de forma inteligente en varias zonas de disponibilidad (AZ). Las AZs son regiones locales dentro de un centro de datos que tienen diferentes suministros de energía y de red y períodos de mantenimiento que no se solapan. Esto garantiza que, incluso si una de las zonas está fuera de servicio, las otras zonas pueden seguir sirviendo los datos. Esta opción es más cara que el LRS.

5.8.2. Redundancia de almacenamiento Azure en regiones secundarias

Este tipo de redundancia proporciona protección de datos contra fallos importantes de la región geográfica, como cortes totales del centro de datos causados por apagones masivos, terremotos, huracanes, etc. **Con este tipo de redundancia, junto con las copias locales, se almacena una copia de los datos en un centro de datos en una ubicación geológica diferente.** Si la región primaria se cae, las aplicaciones podrían fallar automática o manualmente a la región secundaria. Hay dos opciones para la redundancia de la región secundaria, como se detalla a continuación.

Almacenamiento georredundante (GRS)

El GRS es básicamente una configuración de LRS tanto en la región primaria como en la secundaria. Los datos se escriben primero en forma de LRS de manera sincrónica en la región primaria, y luego se realiza una copia remota de manera asincrónica en la región secundaria. En el sitio remoto, los datos se replican de nuevo tres veces utilizando LRS local.

Azure también proporciona una extensión de GRS de sólo lectura llamada GRS de acceso de lectura (RA-GRS). La única diferencia es que en este caso la copia remota es de sólo lectura.

Geo ZRS (GZRS)

GZRS es ZRS en la región primaria y LRS en la región secundaria. Al igual que el GRS, las copias locales se copian de forma sincrónica y las remotas de forma asincrónica. La ubicación remota sigue manteniendo una copia LRS, pero no una ZRS. Esta es la opción de mayor redundancia de datos que ofrece Azure y, como es lógico, también la más cara. Al igual que el GRS, el GZRS también ofrece una opción de copia remota de sólo lectura denominada RA-GZRS.

Al igual que las opciones de redundancia de datos de almacenamiento, hay algunas opciones adicionales de redundancia de datos especializados disponibles para servicios como Azure SQL, Synapse SQL, CosmosDB y otros servicios similares. Veámoslas brevemente también.

5.8.3. Replicación geográfica de Azure SQL

Azure SQL proporciona una característica llamada Replicación Geo Activa. Se trata de una función de redundancia de datos que replica la instancia completa del servidor Azure SQL a los servidores secundarios configurados en una región geográfica de respaldo diferente. Los servidores secundarios sirven como servidores de sólo lectura y pueden cambiarse a lectura-escritura durante los fallos. Las conmutaciones por error pueden iniciarse mediante programación, lo que reduce el tiempo de conmutación durante los fallos.

Puede obtener más información sobre Azure SQL Active Geo Replication aquí: <https://docs.microsoft.com/en-us/azure/azure-sql/database/active-geo-replication-overview>.

5.8.4. Replicación de datos SQL de Azure Synapse

Azure Synapse realiza copias de seguridad geográficas en sus centros de datos emparejados (normalmente distribuidos geográficamente) cada 24 horas, a menos que haya optado por no utilizar esta función. Puede restaurar desde sus copias de seguridad geográficas a cualquier pool de SQL de Synapse en cualquier región si esa región es compatible con los pools de SQL de Synapse.

Puede obtener más información sobre la redundancia de datos de Azure SQL Pool aquí:

<https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/backup-and-restore#geo-backups-and-disaster-recovery>.

5.8.5. Replicación de datos de CosmosDB

CosmosDB es una base de datos distribuida globalmente que replica de forma transparente sus datos a las ubicaciones geográficas configuradas. Proporciona latencias muy bajas al permitir la lectura y escritura desde las réplicas locales de la base de datos. Por lo tanto, sólo configurando CosmosDB en múltiples geolocalizaciones se obtiene redundancia de datos.

Además, CosmosDB realiza copias de seguridad automáticas y periódicas de tus datos. Puedes utilizar las copias de seguridad automáticas para recuperarte en caso de borrado accidental, actualización de tu cuenta CosmosDB, etc.

Puedes obtener más información sobre la replicación global de datos de CosmosDB aquí:

<https://docs.microsoft.com/en-us/azure/cosmos-db/distribute-data-globally>.

NOTA

No todos los servicios o regiones de Azure soportan las cuatro redundancias, así que por favor lee sobre el soporte de redundancia disponible para cada servicio y planifica la implementación de tu producto en consecuencia.

Ahora, vamos a ver cómo implementar la redundancia en Azure Storage.

5.8.6. Ejemplo de configuración de la redundancia en Azure Storage

Al crear una nueva cuenta de almacenamiento, puede elegir entre las cuatro opciones de redundancia de datos, como se muestra en la siguiente captura de pantalla:

Home > Storage accounts >

Create a storage account ...

Basics Advanced Networking Data protection Tags Review + create

Resource group * DP203SandboxRG

Instance details

If you need to create a legacy storage account, select **Legacy storage account**.

Storage account name ⓘ *

Region ⓘ *

Performance ⓘ *

Redundancy ⓘ *

Locally-redundant storage (LRS):
Lowest-cost option with basic protection against server rack and drive failures. Recommended for non-critical scenarios.

Geo-redundant storage (GRS):
Intermediate option with failover capabilities in a secondary region. Recommended for backup scenarios.

Zone-redundant storage (ZRS):
Intermediate option with protection against datacenter-level failures. Recommended for high availability scenarios.

Geo-zone-redundant storage (GZRS):
Optimal data protection solution that includes the offerings of both GRS and ZRS. Recommended for critical data scenarios.

Geo-redundant storage (GRS)

Review + create < Previous Next : Advanced >

Figura 5.9 - Elección de la opción de redundancia de datos durante la creación de la cuenta de almacenamiento

Si desea cambiar la redundancia después de crear el almacenamiento, puede hacerlo desde la siguiente pantalla de configuración:

Home > iacstoreacct

iacstoreacct | Configuration

Storage account

Search (Ctrl+/) Save Discard Refresh

Settings

- Configuration**
- Resource sharing (CORS)
- Advisor recommendations
- Endpoints
- Locks

Monitoring

- Insights
- Alerts
- Metrics
- Workbooks

Minimum TLS version ⓘ
Version 1.0

Blob access tier (default) ⓘ
☐ Cool ☒ Hot

Replication ⓘ

Read-access geo-redundant storage (RA-GRS)

Locally-redundant storage (LRS)

Geo-redundant storage (GRS)

Read-access geo-redundant storage (RA-GRS)

Identity-based access for file shares
Identity-based access for file shares options [have been migrated to the file shares page.](#)

Data Lake Storage Gen2

Figura 5.10 - Elección de la opción de redundancia de datos tras la creación de la cuenta de almacenamiento

Sin embargo, para cambiar la redundancia después de la creación será necesario realizar una migración manual o en vivo. La migración manual se refiere al proceso de copiar los datos al nuevo almacenamiento manualmente o automatizarlo usando el ADF. Esto suele provocar un tiempo de inactividad de la aplicación.

Por otro lado, si necesita que sus aplicaciones estén activas durante la migración, puede solicitar una migración en vivo al soporte de Microsoft. Esto puede dar lugar a algunos gastos de soporte.

5.9. Implementación del archivo de datos (data archiving)

En el capítulo 2, "Diseño de una estructura de almacenamiento de datos", ya hemos estudiado cómo diseñar e implementar el archivado de datos. Aprendimos sobre los niveles de almacenamiento Hot, Cold y Archive y cómo construir políticas de gestión del ciclo de vida de los datos. Como ya hemos cubierto los detalles, no los repetiremos aquí. Por favor, consulte la sección Diseño de una solución de archivo de datos del Capítulo 2, Diseño de una estructura de almacenamiento de datos, de nuevo si ha olvidado el tema del archivo.

Resumen

Con esto llegamos al final de este capítulo. Espero que lo hayas disfrutado tanto como yo al escribirlo. Comenzamos con el aprendizaje de cómo comprimir los datos de una manera limpia utilizando Synapse Pipelines y ADF, y de forma nativa utilizando Spark. A continuación, nos centramos en aprender sobre la implementación de particiones, sharding y distribuciones en SQL dedicated pools y Spark. Después, aprendimos sobre los diferentes tipos de tablas e indexación disponibles en los SQL dedicated pools, y finalmente, concluimos con el aprendizaje de cómo configurar la redundancia y el archivado de datos.

Todos los temas anteriores deberían cubrir el programa de estudios de DP203 - Implementación de Estructuras de Almacenamiento de Datos Físicos. Nos centraremos en la implementación de estructuras de almacenamiento de datos lógicos en el próximo capítulo.