**Tema 1: Configuración de archivos de registro de datos.**

Una base de datos SQL. Server 2012 cuenta con dos tipos de archivos operativos: los archivos de datos y los archivos de registro. Los archivos de datos contienen datos y objetos tales como tablas e índices; los archivos de registro contienen el registro de transacciones necesario para recuperar las transacciones de la base de datos. Es posible agrupar los archivos de datos en grupos de archivos con el fin de que puedan ser más fácilmente administrados y puedan ofrecer un mayor rendimiento. A continuación explicaremos los diferentes tipos de archivo existentes, veremos las diferentes opciones de configuración disponibles, y las mejores prácticas para diseñar sus archivos de manera que tanto su recuperación como su rendimiento sean óptimos.

Archivos de datos.

En una base de datos SQL Server 2012, pueden crearse dos tipos de archivos de datos: primarios y secundarios.

* El archivo primario de datos es obligatorio y contiene información de arranque para el catálogo de la base de datos, señalando a otras bases de datos. Este archivo primario puede contener también objetos y datos de los usuarios. La extensión recomendada para el archivo de datos primarios es **.mdf**.
* El archivo secundario de datos, que es opcional y viene definido por el usuario, contiene tanto objetos como datos del usuario. Es posible colocar cada uno de los archivos secundarios en un disco diferente para mejorar el rendimiento. Una base de datos puede contener un máximo de 32,766 archivos secundarios. La extensión recomendada para este tipo de archivos **.ndf**.

Por ejemplo, usted puede tener una base de datos sencilla o una base de datos que utiliza en raras ocasiones con un único archivo primario de datos que almacena tanto la información del catálogo como las tablas, vistas, procedimientos y datos. Para una base de datos mayor o que se utilice más frecuentemente, podría configurar un archivo primario de datos, así como varios archivos secundarios alojados en diferentes discos para obtener un mejor rendimiento y una major disponibilidad y estabilidad.

*Archivos de bases de datos.  
Es recomendable almacenar todo los datos y objetos en archivos secundarios y dejar el catálogo en la base de datos en el archivo primario. Esta configuración ayuda a reducir la contención del acceso al disco.*

Archivos de registro.

Todas las bases de datos SQL Server 2012 tienen un registro de transacciones que registra todas las modificaciones que realiza cada transacción en la base de datos. SQL Server almacena esta información en los archivos de registro. Deberá tener al menos un archivo de registro para cada base de datos. Puede crear archivos de registro múltiples por cada base de datos para facilitar así una recuperación más rápida. La extensión recomendada para los archivos de registro es **. ldf**.

Grupos de archivos.

Un grupo de archivos es una estructura lógica que permite a los administradores de la base de datos agrupar archivos de datos y tratarlos como si fueran una unidad lógica. Con el fin de mejorar el rendimiento, puede destinar objetos de la base de datos, como por ejemplo las tablas, a grupos de archivo específicos. Al dividir los objetos de la base de datos en varios grupos de archivo, pueden aprovecharse los diferentes subsistemas de disco y permitir que SQL Server realice operaciones de disco paralelas. Además, con la configuración de grupos de archivo múltiples, los archivos pueden ser copiados y restaurados individualmente.

SQL Server soporta dos tipos de archivos de grupo: primarios y definidos por el usuario.

* Un grupo primario de archivos contiene el archivo de datos primario y todos los archivos de datos secundarios que no hayan sido almacenados en otro grupo. Todos los sistemas de tablas son asignados al grupo primario de archivos.
* El grupo de archivos definido por el usuario se crea con la finalidad de agrupar archivos secundarios y asignar objetos de la base de datos a grupos de archivos. Una base de datos puede contener hasta 32,766 grupos de archivo definidos por el usuario.

A la hora de administrar grupos de archivos, deberá prestar especial atención a las siguientes propiedades de los mismos, propiedades que podrá establecer y modificar desde el SSMS o utilizando el comando ALTER DATABASE.

* Cada base de datos contiene un grupo primario de archivos. Cuando se crea un objeto de la base de datos sin especificar un grupo de archivo, SQL Server asigna el objeto al grupo de archivos predeterminado.
* Se puede configurar un grupo de archivos como de sólo lectura. Los grupos de archivos de sólo lectura pueden utilizarse para objetos de la base de datos que no deban ser modificados, como por ejemplo las tablas de historia.

Todos los grupos de archivo pueden ser configurados como de sólo lectura salvo el grupo primario de archivos.

Si su base de datos tiene una tabla a la que se accede con mucha frecuencia (por ejemplo, Detalles de compra), es posible crear múltiples archivos secundarios de datos para la misma, almacenar los archivos en diferentes unidades de disco, y agrupar todos estos archivos en un mismo grupo de archivos. Posteriormente, podrá almacenar la tabla Detalles de compra en este grupo, de manera que todas las consultas que se realicen sobre esta tabla se extenderán por todos los discos disponibles.

*Diseño de grupos de archivo.*

*Cree al menos un grupo de archivos definido por el usuario para albergar archivos secundarios de datos y objetos de la base de datos. Configure este grupo de archivos como grupo predeterminado, de modo que SQL Server almcene todos los objetos creado en dicho grupo.*

Cómo configurar archivos de datos y archivos de registro.

Los archivos de datos y de registro se pueden configurar utilizando el comando de Transact-SQL CREATE DATABASE. Es posible modificar la configuración de los mismos mediante el comando ALTER DATABASE. De forma alternativa, estos archivos pueden también configurarse a partes de la página de Propiedades de la Base de Datos en SSMS. La tabla 1 contiene las opciones de configuración para cada tipo de archivo.

**Tabla 1**. *Opciones de configuración de archivos*.

***Opción Descripción***

Name El nombre lógico para el archivo.

Filename Ruta completa del sistema operativo y nombre de archivo.

Size Tamaño del archivo. Si no se específica un tamaño para el archivo primario, el motor de la base de datos utilizará el tamaño del archivo primario en la base de datos modelo. Si se especifica un archivo secundario o archivo de registro sin la opción de tamaño, el motor de la base de datos crear archivos de 1 MB.

Maxsize Es el tamaño máximo del archivo. Si no se especifica un tamaño máximo, o si se especifica el valor UNLIMITED, la fila crecerá hasta que la unidad esté llena. En SQL Server 2012, un archivo de registro tiene un tamaño máximo de 2 terabytes, y los archivos de datos, un máximo de 16 terabytes.

Filegrowth Especifica el crecimiento automático permitido para el archivo. Puede especificarse el valor en kilobytes, megabytes, gigabytes o terabytes, o como porcentaje del tamaño de archivo actual. Si se especifica un valor 0, no crecerá.

Como normal general, deberá crear archivos de base de datos lo más grandes posible, basándose en la cantidad máxima de datos que estima que contendrá la base de datos para poder acomodar el crecimiento futuro de la misma. Al crear grandes archivos, puede evitarse la fragmentación de archivos y obtener un mejor rendimiento de la base de datos. En muchos casos, es posible permitir el crecimiento automático de la base de datos; asegúrese, no obstante, de limitar al autocrecimiento especificando un tamaño de crecimiento máximo que deje libre cierta cantidad de espacio en el disco duro. Al agrupar diferentes grupos de archivo en discos diferentes, podrá también eliminar la fragmentación física de sus archivos a medida que éstos vayan creciendo.

Nota: Para el siguiente ejercicio, cree una carpeta llamada C:\AdSQL y dentro de ella dos carpetas, Project\_Data y Project\_Data2. Si cuenta con una unidad de disco externa, cree Project\_Data2 en esa unidad. En el ejemplo asumiremos que existe una unidad externa “E:”. De no contar con ella, cambie la ruta del ejemplo a C:\AdSQL\Project\_Data2.

CREATE DATABASE Projects

ON

PRIMARY

(NAME =ProjectPrimary,

FILENAME=' C:\ AdSQL\Projects\_Data\Projectprimary.mdf' ,

SIZE = 100MB,

MAXSIZE = 200),

FILEGROUP ProjectsFG

(NAME = ProjectData1,

FILENAME = 'E:\AdSQL\Projects\_Data2\ProjectData1.Ndf' ,

SIZE = 200 MB,

MAXSIZE = 1200,

FILEGROWTH =100),

FILEGROUP ProjectHistoryFG

( NAME = ProjectHistory1,

FILENAME=' E:\ AdSQL\Projects\_Data2\ProjectHistory1.Ndf' ,

SIZE = 100 MB,

MAXSIZE = 500,

FILEGROWTH = 50)

LOG ON

( NAME = Archlog1,

FILENAME = ' C:\Projects\_Data\Projectflata1.ldf ' ,

SIZE= 300 MB,

MAXSIZE = 800,

FILEGROWTH = 100)

Es posible añadir, eliminar o modificar las propiedades del archivo mediante el comando ALTER DATABASE. El siguiente ejemplo añade un nuevo archivo a la base de datos Projects. Lleve a cabo el siguiente ejercicio:

ALTER DATABASE Projects

ADD FILE

(NAME=ProjectsData4,

FILENAME= ' E:\Projects\_Data\ProjectData4 .ndf',

SIZE=100MB,

MAXSIZE=500MB,

FILEGROWTH=75MB) TO FILEGROUP ProjectsFG

Estas opciones de archivo pueden también configurarse desde SSMS.

Cómo configurar archivos de bases de datos con sistemas RAID.

Los sistemas RAID (en inglés, Redudant Array of Independent Diks o “conjunto redundante de discos independientes”) son sistemas de disco que proporcionan una mejor tolerancia a los fallos, una mayor capacidad de almacenamiento y un rendimiento mejor para el subsistema de disco, dependiendo de la configuración elegida. Si bien los sistemas de hardware RAID no son parte de la configuración de SQL Server, dichos sistemas afectan directamente el rendimiento de SQL Server. Existen varios niveles RAID, cada uno de los cuales emplea un algoritmo de tolerancia de fallos distinto. Los niveles RAID utilizados con más frecuencia son 0, 1, 5 y 10.

* RAID 0: Este nivel recibe también el nombre de disk striping o volumen dividido, ya que crea un sistema de archivo de disco que denominaremos conjunto dividido. RAID 0 ofrece un rendimiento ideal en operaciones de lectura y escritura, ya que distribuye estas operaciones entre todos los discos del conjunto. Sin embargo, RAID 0 no proporciona tolerancia hacia los fallos; si un disco falla, perderá el acceso a todos los datos del conjunto.
* RAID 1: También conocido como disk mirroring o configuración de discos en espejo, proporciona una copia redundante del disco seleccionado. El nivel RAID 1 mejora el rendimiento, pero puede empeorar el rendimiento de las operaciones de escritura.
* RAID 5: Es el nivel de RAID más utilizado. Este nivel distribuye los datos entre los discos del conjunto RAID igual que el nivel RAID 0, pero a su vez añade también información de paridad para proporcionar tolerancia a los fallos. La información de paridad se distribuye así entre todos los discos del conjunto. RAID 5 ofrece un mejor rendimiento que RAID 1. Sin embargo, si un disco falla, el rendimiento en operaciones de lectura disminuye.
* RAID 10: También RAID 1+0, incluye tanto la división sin paridad como la configuración de discos en espejo. RAID 10 ofrece una mejor disponibilidad y rendimiento que RAID 5, especialmente en aplicaciones de escritura intensiva.

La configuración RAID ideal para su base de datos dependerá de varios factores, entre los que se incluyen el rendimiento y la capacidad de recuperación necesarias. RAID 10 es el sistema recomendado para los archivos de registro de transacciones, archivo, e índices. Si tiene limitaciones de presupuesto, mantenga los archivos de registro en un sistema RAID 10 y almacene los archivos de datos e índice en un sistema RAID 5.

Prácticas recomendadas.

Para obtener el mejor rendimiento posible con los archivos de datos y registro, siga estas recomendaciones:

* Para evitar la contención de disco, no aloje los archivos de datos en la misma unidad que contiene los archivos del sistema operativo.
* Coloque los archivos de registro de transacciones en una unidad separada de la unidad de archivos de datos. Esta subdivisión le permitirá obtener el mejor rendimiento posible, dado que reduce el riesgo de contención del disco entre archivos de registro y archivos de datos.
* Si es posible, coloque la base de datos tempdb en una unidad distinta,preferiblemente en un sistema RAID 10 o RAID 1+0. En entornos en los que se dé un uso intensivo a las bases de datos tempdb, obtendrá un major rendimiento si coloca tempdb en una unidad separada que permita a SQL Server llevar a cabo operaciones tempdb en paralelo con las operaciones de la base de datos.

Práctica: Configurar archivos de bases de datos y grupos de archivos.

Esta práctica le mostrará cómo crear una base de datos que contenga varios archivos y grupos de archivo, para después configurar uno de los grupos como grupo predeterminado.

Nota: Para el siguiente ejercicio, cree una carpeta llamada C:\AdSQL y dentro de ella dos carpetas, Sales\_Data y Sales\_Data2. Si cuenta con una unidad de disco externa, cree Project\_Data2 en esa unidad. En el ejemplo asumiremos que existe una unidad externa “E:”. De no contar con ella, cambie la ruta del ejemplo a C:\AdSQL\Sales\_Data2.

1. Abra SQL Server Management Studio (SSMS).

2. Conecte a la instancia de SQL Server utilizando el modo de autenticación de Windows haciendo clic en Conectar.

3. Haga clic en Nueva consulta.

4. Escriba la primera parte de un comando CREATE DATABASE para crear una base de datos llamada Sales; esta base de datos tendrá tres grupos de archivo:

CREATE DATABASE Sales

ON

5. Escriba la primera parte del código que creará el grupo de archivos primarios para contener el archivo SalesPrimary, como se muestra a continuación:

PRIMARY

( NAME = SalesPrimary,

FILENAME = 'C:\AdSQL\Sales\_Data\Salesprimary.mdf' ,

SIZE = 50MB,

MAXSIZE= 200,

FILEGROWTH =20),

6. Escriba la parte del código que definirá el segundo grupo de archivos, SalesFG, que almacenará los datos contenidos en los archivos SalesData1 y SalesData2:

FILEGROUP SalesFG

( NAME = SalesData1,

FILENAME= ' E: \Sales\_Data\SalesData1.mdf',

SIZE = 200MB,

MAXSIZE = 800,

FILEGROWTH = 100),

( NAME = SalesData2,

FILENAME = 'E: \Sales\_Data\SalesData2.mdf' ,

SIZE = 400MB,

MAXSIZE = 1200,

FILEGROWTH = 300),

7. Añada el siguiente comando para crear el tercer grupo de archivos, SalesHistoryFG, que almacenará información de historia del archive SalesHistory1:

FILEGROUP SalesHistoryFG

( NAME SalesHistory1,

FILENAME = ' E: \Sales\_Data\SalesHistory1.mdf' ,

SIZE =100MB,

MAXSIZE = 500,

FILEGROWTH = 50)

8. Escriba el código para añadir un archivo de registro llamado SalesLog:

LOG ON

(NAME = Archlog1,

FILENAME = ' E:\Sales\_Data\SalesHistory1.mdf' ,

SIZE = 300MB,

MAXSIZE = 800,

FILEGROWTH = 100)

9. Ejecute el comando completo CREATE DATABASE:

CREATE DATABASE Sales

ON

PRIMARY

( NAME = SalesPrimary,

FILENAME = 'D:\SalesData\Salesprimary.mdf' ,

SIZE = 50MB,

MAXSIZE= 200,

FILEGROWTH =20),

FILEGROUP SalesFG

( NAME = SalesData1,

FILENAME= ' E:\Sales\_Data\SalesData1.mdf' ,

SIZE = 200MB,

MAXSIZE = 800,

FILEGROWTH = 100),

( NAME = SalesData2,

FILENAME = ' C:\Sales\_Data\SalesData2.mdf' ,

SIZE = 400MB,

MAXSIZE = 1200,

FILEGROWTH = 300),

FILEGROUP SalesHistoryFG

( NAME SalesHistory1,

FILENAME = ' E: \Sales\_Data\SalesHistory1.mdf' ,

SIZE =100MB,

MAXSIZE = 500,

FILEGROWTH = 50)

LOG ON

(NAME = Archlog1,

FILENAME = ' E:\Sales\_Data\SalesHistory1.ldf' ,

SIZE = 300MB,

MAXSIZE = 800,

FILEGROWTH = 100)

CREATE DATABASE Sales

ON

PRIMARY

( NAME = SalesPrimary,

FILENAME = 'C:\AdSQL\Sales\_Data\Salesprimary.mdf' ,

SIZE = 50MB,

MAXSIZE= 200,

FILEGROWTH =20),

FILEGROUP SalesFG

( NAME = SalesData1,

FILENAME= 'D:\Sales\_Data\SalesData1.mdf',

SIZE = 200MB,

MAXSIZE = 800,

FILEGROWTH = 100),

( NAME = SalesData2,

FILENAME = 'D:\Sales\_Data\SalesData2.mdf',

SIZE = 400MB,

MAXSIZE = 1200,

FILEGROWTH = 300),

FILEGROUP SalesHistoryFG

(NAME = SalesHistory1,

FILENAME = 'C:\Sales\_Data\SalesHistory1.mdf',

SIZE = 100MB,

MAXSIZE = 500,

FILEGROWTH = 50)

LOG ON

(NAME = Archlog1,

FILENAME = 'C:\Sales\_Data\SalesHistory1.ldf',

SIZE = 300MB,

MAXSIZE = 800,

FILEGROWTH = 100)

10. Utilice el siguiente comando ALTER DATABASE para configurar el grupo de archivos SalesFG como grupo predeterminado para la base de datos. Todos los objetos creados en la base de datos despues de realizar este cambio quedarán almacenados por defecto en SalesFG:

ALTER DATABASE Sales

MODIFY FILEGROUP SalesFG DEFAULT

**Tema 2: Especificar un escenario de recuperación.**

Un escenario de recuperación es una opción de configuración de la base de datos que controla el modo de registro de las transacciones, si se ha realizado una copia de seguridad de las mismas, y cuáles son las opciones de recuperación para la base de datos. El modelo de recuperación elegido afectará tanto a la recuperación de datos como al rendimiento de la propia base de datos, basándose en los registros que el modelo de recuperación haya llevado o no a cabo.

Resumen de modelos de recuperación

SQL Server 2012 ofrece tres modelos de recuperación de bases de datos distintos: completa, simple y de registro masivo. Estos tres modelos determinan el funcionamiento de SQL Server con respecto al archivo de transacciones y selecciona las operaciones registradas; determina también si el registro se ha visto truncado. Damos la denominación de “truncar” el registro de transacciones al proceso de eliminar transacciones. A continuación se definirán cada uno de los modelos de recuperación disponibles:

* En el modelo completo, el motor de la base de datos registra todas las operaciones en el registro de transacciones. El registro nunca se trunca. Este modelo le permite recuperar una base de datos hasta el momento del fallo (o hasta un punto anterior en el tiempo si utiliza SQL Server 2012 Enterprise Edition).
* En el modelo simple, el motor de la base de datos registra de forma minima casi todas las operaciones y trunca el registro de transacciones después de cada punto de comprobación. En este modelo, no se pueden hacer copias de seguridad ni es posible recuperar el registro de transacciones. Es más: tampoco podrá restaurar páginas individuales de datos.

*El modelo de recuperación simple no es adecuado para aquellas bases de datos en las que se ha producido una pérdida reciente de datos que han de ser recuperados obligatoriamente.*

* En el modelo de registro masivo, el motor de la base de datos registra de forma mínima las operaciones de tipo masivo, tales como SELECT INTO y BULK INSERT. En este modelo de recuperación, cuando una copia de seguridad de registro contiene cualquier operación de tipo masivo, es possible restaurar la base de datos hasta el final de la copia de seguridad de registro, pero no hasta un punto concreto en el tiempo. Este modelo está diseñado para ser utilizado únicamente en operaciones masivas.

Configuración de modelos de recuperación.

Puede ver el modelo de recuperación especificado para una base de datos concreta en la página de Propiedades de la base de datos en SSMS o ejecutando la consulta sys.databases en la vista del catálogo, tal y como se muestra a continuación:

SELECT name, recovery\_model\_desc FROM sys.databases

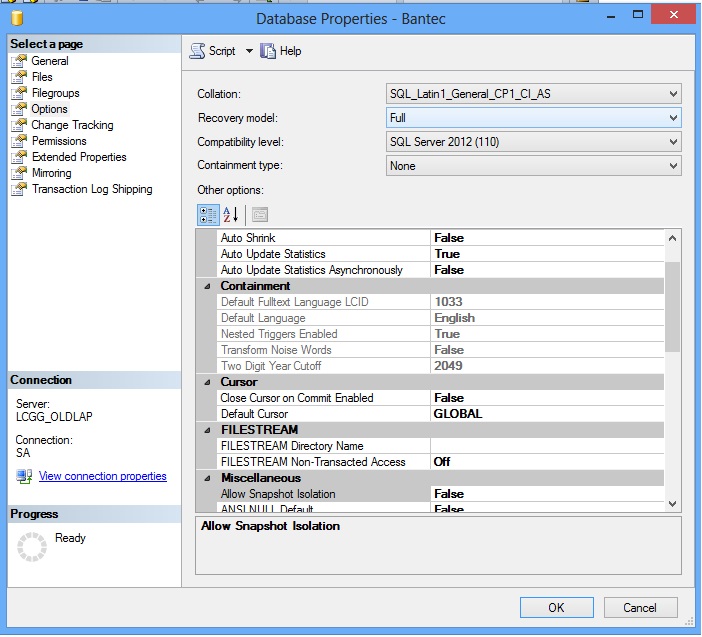
Para configurar el modelo de recuperación para una base de datos, puede ir a la página de Propiedades de la base de datos en SSMS o utilizar el comando ALTER DATABASE.

En SSMS, es posible cambiar el modelo de recuperación siguiendo estos pasos:

1. Expanda la pestaña Bases de datos debajo del Explorador de objetos en SSMS

2. Haga clic con el botón derecho en la base de datos para la que desea establecer un modelo de recuperación y elija Propiedades. Seleccione la página de Opciones.

3. Puede cambiar el modo de recuperación desde la pestana desplegable Modelo de recuperación, como se muestra en la figura 1.



**Figura 1**. Cambiar el modelo de recuperación en SSMS.

La sintaxis básica para configurar el modelo de recuperación utilizando ALTER DATABASE es la siguiente:

ALTER DATABASE <database name>

SET RECOVERY FULL | SIMPLE | BULK\_LOGGED

Como ya se ha apuntado con anterioridad, el modelo completo de recuperación es el indicado para una base de datos en un entorno de producción, ya que es el que proporciona la configuración de mayores posibilidades de recuperación. Si se importan datos periódicamente utilizando un mecanismo masivo, es posible cambiar temporalmente el modelo de recuperación de su base de datos a un modelo de registro masivo para obtener un mejor rendimiento en la operación masiva. Cuando el proceso de importación llegue a su fin, haga que su base de datos vuelva al modelo completo.

Práctica: Cambiar el modelo de recuperación de una base de datos.

Esta práctica le mostrará cómo cambiar el modelo de recuperación de una base de datos a un modelo de registro masivo para obtener un buen rendimiento en operaciones masivas, y posteriormente volver al modelo completo.

1. Establezca el modelo de registro masivo como modelo de recuperación para la base de datos AdventureWorks ejecutando el siguiente comando ALTER DATABASE (antes de modificar el modo, haga una copia de seguridad).

*-- Antes de crear la copia de seguridad, deberá crear una carpeta C:\AdSQL\Backup al nivel del SO.*

BACKUP DATABASE AdventureWorks TO DISK =

' C:\AdSQL\Backup\AdventureWorks.Bak'

GO

*- - Cambie el modelo de recuperación a modo de registro masivo*

ALTER DATABASE AdventureWorks

SET RECOVERY BULK\_LOGGED

2. Escriba y ejecute el siguiente comando ALTER DATABASE para volver a cambiar el modelo de recuperación a completo tras habar realizado las operaciones masivas necesarias; haga otra copia de seguridad de la base de datos para tener una copia de los datos que acaba de cargar:

ALTER DATABASE AdventureWorks

SET RECOVERY FULL

*-- Realice una copia de seguridad de toda la base de datos*

BACKUP DATABASE AdventureWorks TO DISK= 'C:\AdSQL\Backup\AdventureWorks.Bak'

**Tema 3: Configuración de los elementos principales de la seguridad del servidor.**

SQL Server 2012 ofrece un modelo de seguridad completo que le ayudará a impedir el acceso no autorizado a recursos de datos importantes. Este modelo se basa en una serie de permisos que se otorgan a los elementos principales de la base de datos (individuos, grupos y procesos que puedan utilizar recursos de SQL Server).

SQL Server 2012 se encarga de la autenticación de los permisos para todas las conexiones de usuario, de manera que todas las conexiones tengan que especificar tanto un modo como unas credenciales de autenticación. Podrá elegir entre dos modos de autenticación diferentes (modo de autenticación de Windows y modo mixto de autenticación) que controlan el modo de conexión utilizado por los usuarios para conectar a SQL Server. Puede crear dos tipos de inicio de sesión en SQL Server (Windows o SQL Server) que le permitirán administrar el acceso a la instancia SQL Server.

Para ayudarle a administrar los inicios de sesión de los elementos principales con privilegios administrativos en SQL Server, puede ordenar estos inicios de sesión en funciones fijas de servidor. Tanto el modo de autenticación como los inicios de sesión constituyen el primer nìvel de seguridad en SQL Server, de modo que deberá estudiar detenidamente cuál es la opción más segura a la hora de configurar su entorno.

Elegir entre diferentes modos de autenticación.

SQL Server 2012 le ofrece dos modos de autenticación para acceder a los recursos de la base de datos: el modo Windows y el modo mixto de autenticación:

* Autenticación Windows: Si configura SQL Server para utilizar autentición de Windows, únicamente los usuarios autenticados por Windows podrán tener acceso a la instancia SQL Server. Deberá añadir un inicio de sesion de Windows para cada usuario del sistema o grupo que necesite accede a la instancia. Este es el modo de autenticación predeterminado y recomendado ya que con él podrá beneficiarse de las políticas de seguridad centralizadas de su dominio Active Directory.

* Modo de autenticación mixto: El modo de autenticación mixto permite a los usuarios acceder a la instancia de SQL utilizando inicios de sesión tanto de Windows como de SQL Server (ninguno de los cuales está asignadoa un usuario concreto del sistema operativo). Este modo de autenticación se utiliza cuando se quiere conceder acceso a usuarios que no utilicen Windows (por ejemplo, cuando los usuarios de otro sistema operativo necesitan acceder a SQL Server).

Puede cambiar el modo de autenticación utilizando las Propiedades del servidorde SSMS. Para ello, habrá de seguir estos pasos:

1. En SSMS, haga clic con el botón derecho sobre su servidor y elija Propidades.

2. Seleccione la página de Seguridad.

3. Debajo de Autenticación de servidor, seleccione el modo de autenticación que desea utilizar en su servidor. Puede seleccionar el Modo de autentición de Windows, o el Modo de autenticación de Windows y SQL Server.

4. Haga clic en Aceptar para guardar los cambios.

5. Haga clic en Aceptar para cerrar el cuadro de diálogo que le avisará de que algunos cambios en la configuración no surtirán efecto hasta que reinicie el SQL Server.

6. Para reiniciar su servidor, haga clic con el botón derecho en su servidor debajo del Explorador de objetos y elija Reiniciar.

Configuración de inicios de session en SQL Server.

Los inicios de sesión son aquellos elementos principales del servidor que permiten a los usuarios acceder a SQL Server. Puede crear inicios de sesión de forma gráfica en SSMS o mediante un comando CREATE LOGIN.

La sintaxis básica CREATE LOGIN para crear un inicio de sesión en Windows es:

CREATE LOGIN [Domain\User] FROM WINDOWS

La sintaxis para crear un inicio de sesión de SQL Server es:

CREATE LOGIN login\_name WITH PASSWORD=‘password’

Para inicios de sesión de SQL Server, puede especificar cuál de las siguientes opciones prefiere en el momento de crear el inicio de sesión:

* MUST\_CHANGE: El inicio de sesión deberá cambiar la contraseña en el siguiente inicio de sesión.
* CHECK\_EXPIRATION: SQL Server comprobará la política de caducidad de Windows para el inicio de sesión de SQL Server.
* CHECK\_POLICY: SQL Server aplicará la política local de contraseñas de Windows a todos los inicios de sesión de SQL Server.

En el siguiente ejemplo, va a crear un inicio de sesión de SQL Server y exigir una comprobación de la política de contraseña y de caducidad de contraseña:

CREATE LOGIN secureSQL WITH PASSWORD=‘ Ty%6tSfS$g23’,

CHECK\_EXPIRATION = ON,

CHECK\_POLICY=ON

*Política de contraseñas*

*Para conseguir un entorno SQL Server seguro, deberá utilizar las opciones disponibles para la comprobación de la política de caducidad de Windows para inicios de sesión de SQL Server y aplicarles la política de contraseñas de Windows.*

Si necesita cambiar alguna propiedad del inicio de sesión, puede utilizar el comando ALTER LOGIN. El siguiente ejemplo le mostrará cómo cambiar la contraseña para un inicio de sesión de SQL Server:

ALTER LOGIN login\_name WITH PASSWORD=‘password’

Cuando necesite eliminar un inicio de sesión, puede utilizar el comando DROP LOGIN:

DROP LOGIN login\_name

O puede utilizar la siguiente sintaxis:

DROP LOGIN [Domain\User]

Para obtener información de inicio de sesión de SQL Server tal como opciones de estado o de inicio de sesión, puede ejecutar la consulta sys.sql\_logins en la vista del catálogo.

*Eliminar logines*

*No es posible eliminar un inicio de sesión que contenga un elemento de seguridad, objeto a nivel de servidor, o un trabajo del Agente de SQL Server. Deberá deshabilitar los inicios de sesión antes de eliminarlos, y eliminarlos únicamente cuando esté completamente seguro de que la acción no afectará a su entorno.*

*Además, si el inicio de sesión ha sido asignado a un usuario de la base de datos y se elimina, SQL Server no eliminará al usuario de forma automática,de modo que tendrá un usuario huérfano.*

Normalmente, los ABD necesitan administrar excepciones a la hora de habilitar accesos a un grupo de Windows. Por ejemplo, es probable que deba habilitar accesos para todos los miembros de un grupo determinado de Windows, salvo uno. Para realizar esta tarea, deberá crear un inicio de sesión de Windows para el grupo de Windows y posteriormente denegar el acceso al usuario concreto que no debe tener acceso.

En el siguiente ejemplo se muestra la sintaxis básica para llevar a cabo esta operación:

CREATE LOGIN [domain\_name\group\_name] FROM WINDOWS

DENY CONNECT SQL TO [domain\_name\User\_name]

Administrar funciones fijas de servidor

SQL Server le ofrece un conjunto de funciones fijas de servidor, tales como sysadmin y securityadmin, que podrá utilizar para asignar y administrar privilegios de administración para los inicios de sesión, añadiendo inicios de sesión en calidad de miembros de estas funciones. La tabla 2 describe las funciones fijas del servidor para SQL Server 2012.

**Tabla 2.** Funciones fijas de servidor de SQL Server.

Función fija del servidor Los miembros pueden...

sysadmin Realizar cualquier actividad en SQL Server. Los permisos de esta función engloban los permisos de todas las demás funciones fijas del servidor.

serveradmin Configurar ajustes a nivel de servidor.

setupadmin Añadir y eliminar servidores enlazados y ejecutar algunos procedimientos de almacenamiento del sistema, como por ejemplo sp\_serveroption.

securityadmin Administrar los inicios de sesión del servidor.

Processadmin Administrar los procesos ejecutándose en una instancia de SQL Server.

dbcreator Crear y alterar bases de datos.

diskadmin Administrar archivos de disco.

bulkadmin Ejecutar el comando BULK INSERT.

Para obtener más información acerca de la función fija del servidor, puede ejecutar la consulta sys.server\_role\_members en la vista del catálogo, que le devolverá una fila por cada miembro de función del servidor.

La sintaxis básica para añadir un inicio de sesión a una función fija de servidor es:

EXECUTE sp\_addsrvrolemember login\_name, fixed\_server\_role

Puede utilizar el procedimiento almacenado sp\_dropsrvrolemember para eliminar el inicio de sesión de la función fija del servidor.

De forma alternativa, también puede utilizar SSMS para añadir o eliminar inicios de sesión de funciones fijas del servidor. Puede llevar a cabo estas tareas mostrando las propiedades bien de los inicios de sesión, bien de la función del servidor.

Práctica: Seleccionar un modo de autenticación y crear un inicio de sesión.

Estás prácticas le permitirán cambiar el modo de autenticación de su servidor a un Modo mixto y crear un inicio de sesión de SQL Server. Reforzará la política de contraseña y de caducidad de contraseñas para ese inicio de sesión y añadirá el inicio de sesión a la función fija del servidor sysadmin.

Práctica 1: Cambiar el modo de autenticación.

En esta práctica, cambiaremos el modo de autenticación a Modo mixto.

1. En SSMS, haga clic con el botón derecho en su servidor y elija Propiedades.

2. Seleccione la página de Seguridad. Debajo de Autenticación de servidor, seleccione Modo de autenticación de Windows y de SQL Server.

3. Haga clic con el botón derecho en su servidor y elija Reiniciar para que el cambio tenga efecto.

Práctica 2: Añadir inicio de sesión en SQL Server.

En esta práctica, añadiremos un nuevo inicio de sesión en SQL Server y estableceremos restricciones de caducidad y de comprobación. Posteriormente se añadirá el inicio de sesión a la función fija del servidor sysadmin.

1. En el explorador de objetos, expanda la carpeta Seguridad, haga clic con el botón derecho en Inicio de sesión y elija Nuevo inicio de sesión. Aparecerá un nuevo cuadro de diálogo Inicio de sesión>Nuevo.

2. En la caja de texto Nombre de inicio de sesión, escriba sqlLogin.

3. Seleccione la opción de Autenticación de SQL Server; en las cajas de texto de Contraseña y Confirmar contraseña, escriba Pa$$w0rd.

4. Desactive la casilla El usuario debe cambiar la contraseña en el próximo inicio de sesión.

5. Para añadir el inicio de sesión a la función fija de servidor sysadmin, seleccione la página Funciones del servidor(server roles). Active la casilla sysadmin y haga clic en Aceptar. Intente acceder con esta cuenta. No será posible, veamos el sig. tema.

**Tema 4: Configuración de los asegurables de la base de datos.**

Ya hemos estudiado cómo los elementos principales de seguridad del servidor son las entidades que solicitan el acceso a los recursos de la base de datos. Los asegurables (securables) son, en cambio, las entidades que permiten o no a los elementos principales acceder a dichos recursos. En el nivel más alto de elementos que deben ser asegurados se encuentran los servidores y las bases de datos, pero también pueden establecerse permisos a niveles inferiores. Este tema se ocupará de los asegurables al nivel de la base de datos.

Tras haber configurado el modo de autenticación y de crear inicios de sesión para los elementos principales, necesita habilitar los permisos de acceso adecuados para la base de datos. Para llevar a cabo esta acción, deberá vincular cada uno de los inicios de sesión de la base de datos que deben acceder a la misma a un usuario de la base de datos. Para una administración más fácil y rápida, puede añadir los usuarios de la base de datos como miembros de funciones de la misma.

Administración de los usuarios de la base de datos.

Para establecer inicios de sesión de acceso a una base de datos, necesita crear un usuario de la base de datos para cada uno de los inicios de sesión necesarios para acceder a la misma. Deberá crear el usuario en la base de datos a la que dicho usuario deba acceder. La sintaxis básica para crear un usuario de base de datos es:

CREATE USER user\_name FOR LOGIN login\_name

Si no especifica un nombre de inicio de sesión, SQL Server intentará crear un usuario vinculado a un inicio de sesión con el mismo nombre. Puede utilizar el comando ALTER USER para modificar las propiedades del usuario y el comando DROP USER para eliminar a los usuarios de la base de datos. Puede también utilizar SSMS para crear y administrar los usuarios de la base de datos; puede administrarlos desde Inicios de sesión bajo la pestaña de Seguridad, o en Usuarios debajo de cada una de las pestañas de Bases de datos.

Cuando un inicio de sesión no tiene ningún usuario vinculado al mismo que intente acceder a la base de datos, SQL Server buscará un usuario Invitado a la base de datos. SQL Server crea un usuario Invitado en cada base de datos. El usuario Invitado no está autorizado, de forma predeterminada, a conectar con la base de datos. Puede permitir que este usuario se conecte activando esta cuenta de usuario Invitado como se explica a continuación:

GRANT CONNNECT TO Guest

Puede volver a denegar el acceso al invitado ejecutando el siguiente comando:

REVOKE CONNECT TO Guest

Administración de usuarios huérfanos.

El término “usuarios huérfanos” hace referencia a aquellos usuarios de la base de datos que no están vinculados a ningún inicio de sesión en la instancia SQL Server. En SQL Server 2012, un usuario se convertirá en usuario huérfano cuando se elimine el inicio de sesión que se le ha asignado. Para obtener más información sobre usuarios huérfanos, ejecute el siguiente comando:

USE AdventureWorks;

GO

EXECUTE sp\_change\_users\_login @Action=' Report' ;

*Eliminación de usuarios de la base de datos.*

*El motor de la base de datos no le permitirá eliminar usuarios de la base de datos si poseen un esquema que contenga objetos. Deberá transferir el esquema a otro usuario o función antes de eliminar a este usuario de la base de datos.*

Administración de las funciones de la base de datos.

En caso de tener muchos usuarios de la base de datos, los procesos de creación, modificación y eliminación de usuarios, así como el proceso de asegurar que todos los usuarios tienen los permisos adecuados, puede resultar una labor tediosa que llevará mucho tiempo. Para ayudarle en la administración de estas tareas, cada uno de los usuarios de base de datos le ofrece un conjunto de funciones fijas que podrá utilizar para agrupar a usuarios similares. La tabla 3 describe estas funciones.

**Tabla 3.** Funciones fijas de bases de datos SQL Server.

Función fija de base Permisos a nivel de base de datos  
de datos

db\_acces sadmin Concedido: ALTER ANY USER, CREATE SCHEMA

dbaccessadmin Concedido con a opción GRANT: CONNECT

dbbackupoperator Concedido: BACKUP DATABASE, BACKUP LOG,  
CHECK- POINT

dbdatareader Concedido: SELECT

cìbdatawriter Concedido: DELETE, INSERT, UPDATE

dbddladmin Concedido: ALTER ANY ASSEMBLY, ALTER ANY

SIMMETRIC KEY, ALTER ANY CERTIFICATE,  
ALTER ANY CONTRACT, ALTER ANY DATABASE  
DOL TRIGGER, ALTER ANY DATABASE EVENT,  
NOTIFICATION, ALTER .ANY DATASPACE, ALTER ANY FULLTECT CATALOG, ALTER ANY MESSAGE TYPE, ALTER ANY REMOTE SERVICE BINDING, ALTER ANY ROUTE, ALTER ANY SCHEM, ALTER ANY SERVICE, ALTER ANY SYMMETRIC KEY, CHECKPOINT, CREATE AGGREGATE, CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE QUEUE, CREATE RULE, CREATE SYNONYM, CREATE TABLE, CREATE TYPE, CREATE VIEW,  
CREATE XML SCHEMA COLLECTION, REFERENCES

db\_denydatareader Denegado: SELECT

db\_denydatawriter Denegado: SELECT, INSERT, UPDATE

db owner Concedido con la opción GRANT: CONTROL

db\_securityadmin Concedido: ALTER ANY APPLICATION ROLE, ALTER ANY ROLE, CREATE SCHEMA, VIEW DEFINITION

*Administración de las funciones de miembros de la base de datos.*

*Los miembros de las funciones db\_owner y db\_securityadmin pueden llevar la administración de miembros de funciones fijas de la base de datos, pero sólo miembros de la función db\_owner pueden añadir miembros a esta función.*

Puede también crear sus propias funciones de base de datos para agrupar a los usuarios de la base de datos que tengan las mismas necesidades de acceso y asignar permisos a un grupo en lugar de asignar permisos de forma individual a cada usuario. Por ejemplo, puede agrupar a usuarios que sean miembros del departamento de Contabilidad en una función de bases de datos denominada Accounting, de modo que pueda asignar permisos únicamente a esa función de la base de datos y hacer que los permisos sean aplicables a todos los miembros de ese grupo.

La sintaxis básica para crear una función de la base de datos es

CREATE ROLE role name

Puede modificar las propiedades de la función utilizando el comando ALTER ROLE, y eliminarlas mediante el comando DROP ROLE. Puede también administrar las funciones de la base de datos utilizando SSMS partiendo de la pestaña Seguridad situada debajo de cada base de datos.

Para añadir un usuario de la base de datos a una función, utilice el procedimiento almacenado sp\_addrolemember con la siguiente sintaxis:

EXECUTE sp\_addrolemember role\_name, user\_name

De forma alternativa, también puede añadir un usuario de la base de datos mediante SSMS, modificando las propiedades de usuario de la base de datos o las propiedades de la función.

Puede anidar las funciones de la base datos de modo que pueda añadir funciones de la misma a otras funciones. Imagine, por ejemplo, que desea agrupar a todos los encargados del departamento de Contabilidad en una función de la base de datos llamada AccountingMgr. Podría conceder a esa función los permisos concedidos a la función Accounting, anidando AccountingMgr dentro de Accounting y posteriormente añadiendo los permisos extra a la función AccountingMgr. Para obtener información sobre los miembros de una función de la base de datos, puede ejecutar la consulta ***sys.datablase\_role\_members*,** que le mostrará una fila por cada miembro de la función.

Esquemas de administración.

SQL Server 2012 pone en práctica el concepto ANSI de esquemas, esto es, colecciones de objetos de la base de datos (tales como tablas, vistas, procedimientos almacenados y desencadenadores o triggers) que conforman un único espacio. La principal ventaja de los esquemas en SQL Server 2012 es que los esquemas y los usuarios son ahora entidades separadas. El nombre de usuario ya no forma parte del nombre del objeto, tal y como ocurría en anteriores versiones de SQL Server, de modo que ahora es posible eliminar usuarios o cambiar el nombre de un usuario sin tener que realizar cambios en la aplicación. Cada esquema pertenece a un usuario o función, pero en caso de que necesite eliminar a dicho usuario o función, simplemente deberá transferir la propiedad del esquema del usuario o función que va a ser eliminado a un nuevo usuario o función. La sintaxis básica para crear un esquema es:

CREATE SCHEMA schema\_name AUTHORIZATION owner

Para modificar un esquema, puede utilizar el comando ALTER SCHEMA; para eliminar un esquema, utilice el comando DROP SCHEMA. Puede también realizar estas tareas desde SSMS. Para obtener información sobre los esquemas, ejecute la consulta ***sys.schemas***.

Además, puede asignar un esquema predeterminado a cada usuario de la base de datos. Este esquema por defecto se emplea cuando el usuario no especifica el nombre del esquema a la hora de acceder a un objeto. Por ejemplo, si el usuario Pedro tiene un esquema predeterminado de RecursosHumanos y quiere acceder a la tabla de Empleado sin especificar ningún esquema, puede simplemente especificar Empleado en lugar de tener que especificar ***RecursosHumanos.Empleado.***

Puede asignar un esquema predeterminado mediante los comandos CREATE USER o ALTER USER. También puede asignar un esquema predeterminado a través de SSMS, en las propiedades de usuario.

Práctica: Configuración de los asegurables del servidor.

En esta práctica, aprenderá a configurar los asegurables del servidor para la base de datos Adventureworks. Creará un inicio de sesión y un usuario de base de datos para el usuario Peter. Peter necesita tener acceso a los objetos del esquema HumanResources en AdventureWorks.

1. Utilice el comando CREATE LOGIN para crear un inicio de sesión de SQL Server y un nombre de usuario llamado Peter que tenga acceso a la base de datos Adventureworks:

CREATE LOGIN Peter WITH PASSWORD=' Pa$$w0rd'

GO

USE AdventureWorks

GO

CREATE USER Peter FROM LOGIN Peter

2. Conceda a Peter el permiso SELECT para los objetos de la base de datos con

el siguiente comando (preste atención al elemento de sintaxis:: para especificar un nombre de esquema):

GRANT SELECT ON SCHEMA:: [HumanResources]TO [Peter]

3. Haga clic en Nueva Consulta. Haga clic derecho en la zona de consultas(área de escritura de scripts) y elija Conexión>Cambiar Conexión. Establezca una conexión utilizando el inicio de sesión “Peter” y la contraseña “Pa$$w0rd”.

4. Ejecute la siguiente consulta para comprobar el acceso de Peter a SQL Server:

USE Adventureworks

GO

SELECT \* FROM Employee

5. Obtendrá un mensaje de error de Objeto no válido, lo cual quiere decir que el usuario Peter no tiene los permisos adecuados para acceder a la tabla Employee. Deberá resolver este problema ejecutando el siguiente comando ALTER USER para designar HumanResources como esquema predeterminado para Peter, de modo que Peter pueda seleccionar la tabla Employee directamente sin tener que utilizar el nombre de esquema HumanRe sources para calificar la tabla nombrada:

ALTER USER Peter WITH DEFAULT\_SCHEMA=HumanResources

6. Ejecute nuevamente la consulta del paso 4. Ahora debería obtener un resultado válido.

**Tema 5: Creación de índices.**

Hemos visto cómo la creación de tablas constituye el primer paso en la construcción de una buena base de datos. Posteriormente, es necesario añadir datos a dichas tablas. Sin embargo, si esos datos nunca van a ser recuperados, estaremos perdiendo inútilmente espacio de almacenamiento. SQL Server no necesita que las tablas estén indexadas para recuperar datos: puede, simplemente explorar una tabla para encontrar la información buscada. No obstante, la mayor parte de organizaciones almacenan cantidades masivas de datos en una tabla y necesitan poder recuperarlos de forma instantánea. Para permitir que los datos puedan recuperarse con rapidez sin que el rendimiento empeore a medida que los usuarios van añadiendo filas a una tabla, es necesario utilizar índices.

Los índices no constituyen una novedad, y tampoco son un concepto exclusivo de una base de datos. De hecho, los utilizamos todos los días. Por ejemplo: en un libro, usted encontrará un índice impreso. Si quisiera leer sobre índices de texto para preparar ese tema concreto de su examen, podría encontrar la información de dos maneras: podría abrir el libro, empezar en la página 1 y hojear cada página hasta encontrar la información. O bien, puede dirigirse al índice situado al final del libro, localizar el tema y dirigirse directamente a las páginas correspondientes al mismo. Encontrará la información de igual manera, pero la utilización del índice es una manera más eficiente de hacerlo.

Vamos a explicar cómo SQL Server construye y utiliza índices para asegurar una recuperación de datos rápida y una mayor estabilidad en el rendimiento

*Caso de uso: Índices en el mercado (Michael Hotek)*

*Hace ya algunos años, cuando SQL Server 6.5 llevaba ya una temporada en el mercado, comencé un proyecto para una nueva compañía en Chicago. La compañía quería ayudar a la gente a encontrar un piso que se ajustara a los criterios de búsqueda de sus clientes. Uno de los empleados había oído hablar de un lenguaje de programación llamado Visual Basic que les permitía crear el tipo de aplicación que necesitaban para administrar los cientos de complejos residenciales de la zona. Crearon la aplicación, la probaron y entraron en la fase de producción. Cuatro meses más tarde, el negocio no dejaba de crecer y la compañía abrió vanas oficinas en diferentes partes del país.*

*Y entonces, la compañía empezó a tener problemas. La búsqueda de pisos mediante la aplicación de SQL Server cada vez llevaba más tiempo. Muchos empleados llegaron a tal punto de frustración que comenzaron a elaborar sus propios archivos en papel. El desarrollador había revisado el código y no podía localizar el problema, de modo que la compañía me llamó para examinar la parte correspondiente al uso de SQL Server en la aplicación.*

*Lo primero que hice fue preguntar al desarrollador si había revisado los índices de las tablas en SQL Server. Encontré la respuesta al bajo rendimiento de la aplicación cuando el desarrollador me preguntó qué era un índice. Tardé una hora en llegar a la oficina central y 15 minutos en resolver el problema de rendimiento, implemente añadiendo varios índices clave. Pasé el resto del día indexando otras tablas para evitar problemas en el futuro, y explicando a la persona encargada del desarrollo qué era un índice, por qué era útil y cómo decidir qué elementos debían ser indexados…*

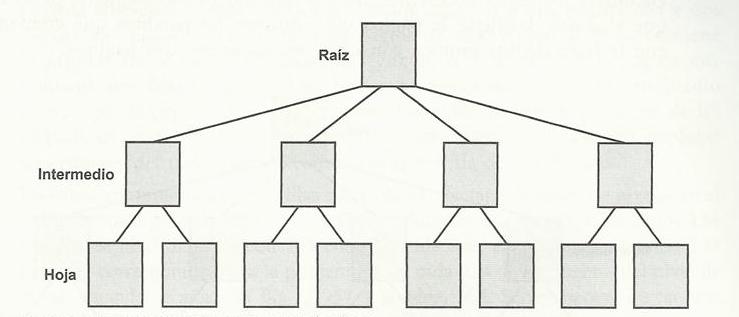
**La estructura de índices.**

Un índice resulta útil solamente si nos ayuda a localizar datos rápidamente, independientemente de la cantidad de datos que tengamos almacenados. Eche un vistazo al índice de cualquier libro: contiene solamente una pequeña muestra de todas las palabras del libro, ofreciendo un modo compacto de buscar la información. Si el índice se organizara en función de las páginas en las que aparece una palabra determinada, tendría que leer muchos párrafos y páginas hasta encontrar la información deseada. En lugar de eso, el índice se organiza alfabéticamente, lo cual significa que usted puede dirigirse a un lugar específico del índice para encontrar lo que busca. También le permite escanear hasta encontrar la palabra que busca, tras lo cual sabe que ya no debe buscar más allá. Un índice en SQL Server se organiza de forma parecida. Veremos ahora cómo SQL Server utiliza una estructura de árbol-B para construir índices, que le ofrece un método de recuperación rápida de datos aún con tablas extremadamente grandes.

**La estructura de árbol.**

SQL Server utiliza una estructura de árbol-B para construir y mantener índices. La ilustración en la figura 2 muestra una estructura de árbol-B.

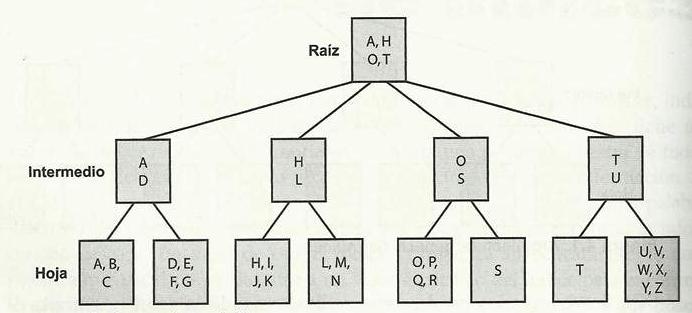
Un árbol-B está compuesto de un nodo raíz que contiene una única página de datos, ninguno o más niveles intermedios con páginas adicionales y un nivel o capa de hojas. Las páginas de la capa de hojas contienen entradas agrupadas por orden que se corresponden con los datos indexados. EI número de filas indexadas de una página viene determinado por el espacio de almacenamiento requerido por las columnas definidas en el índice. Por ejemplo: un índice definido sobre una columna de enteros de 4 bytes tendrá cinco veces más valores por página que un índice definido sobre una columna char(60) que quiera 60 bytes de almacenamiento por página.



**Figura 2.** Arquitectura general de índices.

SQL Server crea los niveles intermedios tomando la primera entrada de cada página de la capa de hojas y almacena las entradas en una página con un puntero a nivel de página de capa de hojas. La página raíz se construye de la misma manera.

Al construir un índice de este modo, SQL Server puede realizar búsquedas en tablas con billones de filas de datos con tanta rapidez como si tuvieran tan sólo unos cientos de filas. Veamos la estructura del árbol de la figura 3 para ver cómo una consulta utiliza un índice para localizar datos con rapidez:



**Figura 3**. Construcción de índices.

Si por ejemplo estuviéramos buscando el término “SQL Server”, la consulta exploraría la página raíz. Encontraría el valor O y el valor T. Dado que S viene antes de T, la consulta sabe que necesita mirar en la página O para encontrar los datos necesarios. La consulta se trasladaría entonces al nivel intermedio hacia el que apunta O. Nótese que con esta operación hemos eliminado directamente tres cuartas partes de las posibles páginas de búsqueda  
explorando un subconjunto muy pequeño de valores. Posteriormente, la consulta exploraría la página del nivel intermedio y encontraría el valor S; se dirigiría hacia la página hacia la que apunta esta entrada. Hasta ahora, la consulta ha explorado exactamente dos páginas del índice para encontrar los datos necesarios. Nótese que independientemente de la letra que elijamos, localizar la página que contiene las palabras que comienzan con la letra elegida implica consultar, exactamente, dos páginas.

Este comportamiento explica por qué la estructura del índice es una estructura de árbol-B. Cada una de las búsquedas realizadas transita por el número de niveles en el índice para localizar los datos que nos interesan.

E número de niveles en un índice, así como el número de páginas dentro cada nivel, viene determinado por un simple cálculo matemático. Una página de datos en SQL Server ocupa 8,192 bytes y puede almacenar hasta 8,060 bytes de información.

Si construimos un índice sobre una columna char(60), cada fila de la tabla requeriría 60 bytes de almacenamiento, lo cual supone ocupar también 60 bytes de almacenamiento por fila dentro del índice.

Si únicamente tenemos 100 filas de datos en la tabla, necesitaríamos 6,000 bytes de almacenamiento. Dado que todas las entradas cabrían en una página de datos, el índice tendría una única página que sería a su vez la página de la capa de hojas. De hecho, podríamos almacenar 134 filas en la tabla y aún así designar una única página para el índice.

Tan pronto como añadiéramos la fila número 135, sin embargo, las entradas dejarían de caber en una única página, de modo que SQL Server crea dos páginas adicionales. Esta operación crea un índice con una página raíz y dos páginas de capa de hojas. La primera página de nivel o capa de hojas contiene la primera mitad de las entradas; la segunda, la otra mitad; y la página raíz contiene dos filas de datos. Este índice no necesitaría un nivel intermedio puesto que la página raíz pude contener todos los valores al principio de las páginas de nivel de hojas. En este punto, una consulta necesitaría explorar dos páginas del índice para localizar cualquier fila de la columna.

Podemos continuar añadiendo filas a la tabla sin afectar al número de niveles en el índice hasta llegar a la cifra de 17,957 filas. Llegado este punto, tendríamos 134 páginas de nivel de hojas con 134 entradas cada una. La página raíz tendría 134 entradas correspondientes a la primera fila de cada una de las páginas del nivel de hojas. Cuando se añade la fila 17,957 a la tabla, SQL Server necesita crear una nueva página para el índice al nivel de capa de hojas, pero la página raíz no puede tener más de 135 entradas porque se excedería de los 8.060 bytes permitidos por página. De manera que SQL Server añade un nivel intermedio que contiene dos páginas. La primera contiene la entrada inicial de la primera mitad de las páginas del nivel de hojas, y la segunda contiene la entrada inicial de la segunda mitad. La página raíz contendría ahora dos filas, correspondientes al valor inicial de cada una de las dos páginas intermedias.

La próxima vez que SQL tenga que introducir un nuevo nivel intermedio sería al añadir la fila número 2, 406,105 a la tabla.

Como puede verse, este tipo de estructura permite que SQL Server localice muy rápidamente las filas relativas a la consulta, aún en tablas extremadamente largas.

En este ejemplo, encontrar una fila en una tabla con cerca de 2.5 millones de filas requeriría que SQL explorase únicamente tres páginas de datos. La tabla podría seguir creciendo hasta tener más de 300 millones de filas antes de que SQL Server tuviera que leer 4 páginas. Tenga en cuenta que este ejemplo utiliza una columna char(60). Si crease un índice con una columna int que requiriera únicamente 4 bytes de almacenamiento, SQL Server tendría que leer una única página para localizar la fila hasta que se introdujera la fila número 2,016. Podrían añadirse algo más de 4 millones de filas a la tabla y seguir necesitando únicamente la lectura de dos páginas para encontrar una fila. Habría que llegar hasta los 8 mil millones de filas antes de que SQL Server necesitara leer tres páginas para encontrar la fila en cuestión.

**Tema 6: Crear índices agrupados.**

El primer tipo de índice que debe crearse en una tabla es un índice agrupado. Como regla general, toda tabla debe tener un índice agrupado, y cada tabla podrá tener únicamente un índice de este tipo. En esta lección, aprenderá a crear un índice agrupado utilizando la instrucción CREATE INDEX de Transact-SQL,incluyendo las opciones que pueden especificarse para este comando. Por último, aprenderá a deshabilitar y rehabilitar un índice agrupado.

Implementación de índices agrupados.

Las columnas definidas para un índice agrupado se denominan clave de agrupamiento o clustering key. Un índice agrupado hace que SQL Server ordene los datos de una tabla de acuerdo con la clave de agrupamiento. Dado que la tabla únicamente puede ordenarse de una forma, únicamente podrá crearse una clave de agrupamiento por tabla.

Además, la capa o nivel de hojas de un índice agrupado lo constituyen los propios datos de la tabla. Cuando se alcanza este nivel dentro de un índice agrupado, SQL Server no tiene que utilizar ningún puntero para acceder a los datos de la tabla puesto que ya ha llegado a los datos en sí.

*Ordenación de los datos.*

*Uno de los errores más comunes consiste en pensar que un índice agrupado hace que los datos se ordenen de forma física en una tabla. Pero esto no es del todo cierto: un índice agrupado hace que, tanto las filas de una tabla como las páginas de datos de una lista doblemente vinculada que almacena los datos de la tabla, se ordenen en función de la clave de agrupamiento. Sin embargo, este orden sigue siendo ilógico. Las filas de la tabla pueden almacenarse en las fuentes del disco sin ordenar. Si un índice agrupado hiciera que los datos se ordenaran de forma física en el disco, crearía un nivel de actividad prohibitivo en el mismo.*

Como normal general, cada tabla deberá contar con un índice agrupado, que a su vez debería ser la clave primaria de la misma.

*Selección de índices agrupados.*

*Es posible sorprenderse de sobremanera al leer que el índice agrupado debe ser también la clave primaria. Hemos dicho, no obstante, que esta es la norma “general”, no un absoluto. Ser clave primaria no es siempre la elección ideal en el caso de un índice agrupado. Sin embargo, carecemos del espacio suficiente para poder explicar todas las permutaciones y factores a la hora de seleccionar la clave de agrupamiento perfecta, y aún cuando dicho espacio existiera, terminaríamos aplicando la misma norma general que ya hemos apuntado. Agrupar la clave primaria es siempre una opción preferible a no tener ningún índice agrupado.*

Utilice la instrucción CREATE. . . INDEX de Transact-SQL para crear un índice agrupado. La sintaxis general para este comando es la siguiente:

CREATE [ UNIQUE 1 [ CLUSTERED NONCLUSTERED J INDEX index name

ON <object> (column t ASC DESC ) [ ,.. . n ])

[INCLUDE ( column\_name [ , . . .n) ) ]

[ WITH ( <relational\_index\_option> ( . . . n ] ) ]

[ ON ( partition\_scheme\_name ( column\_name )

| filegroup\_name

| default

]

][ ;]

Las opciones CLUSTERED y NONCLUSTERED designan el tipo de índice que se creará. Después de especificar que quiere crear un índice agrupado, necesita dar un nombre a dicho índice. Todo índice ha de tener un nombre que se ajuste a las normas de los identificadores del objeto.

A continuación, utilice una cláusula ON para especificar el objeto frente al cual ha de crearse el índice. Puede crear un índice bien en una tabla, bien en una vista. Después de especificar la tabla o vista frente a la que ha de crearse el índice, especifique entre paréntesis las columnas sobre las que se creará el índice. Las palabras clave ASC y DESC especifican si el orden ha de ser ascendente o descendiente. Puede también utilizar la cláusula ON para delimitar el espacio de almacenamiento sobre el cual ha de colocarse el índice. Puede especificarse bien un grupo de archivos o un esquema de partición para el índice. Si no se especifica una localización y la tabla o vista no está particionada, SQL Server creará el índice en el mismo grupo de archivos de la tabla o vista.

La siguiente parte del comando CREATE INDEX le permite delimitar las opciones de índices relacionales. Si bien una explicación detallada de cada una de estas opciones va más allá del propósito del libro, la tabla 4.1. describe las diferentes opciones relacionales que pueden utilizarse para un índice.

**Tabla 4.** Opciones de índices relacionales.

Opción Descripción

PAD\_INDEX Especifica el factor de relleno del índice. Si está en ON, esta opción aplica el porcentaje de espacio libre especificado por la opción FILLFACTOR a las páginas del nivel intermedio del índice. Si está en OFF (opción predeterminada) o si no se especifica ningún FILLFACTOR, las páginas del nivel intermedio se rellenarán hasta prácticamente el máximo de su capacidad, dejando espacio suficiente para al menos una fila del tamaño máximo que el índice pueda albergar.

FILLFACTOR Delimita un porcentaje (entre O y 100) que indica el nivel hasta el cual el motor de la base de datos debe llenar la capa de hojas de cada página indexada durante la creación o reconstrucción del índice.

SORT\_IN\_TEMPDB Especifica silos resultados de la ordenación temporal deben almacenarse en la base de datos tempdb. La opción predeterminada es OFF, lo cual significa que los resultados intermedios se almacenan en la misma base de datos que el índice.

IGNORE\_DUP\_KEY Especifica la respuesta de error a los valores de clave de duplicado en una operación de inserción de filas múltiples sobre un índice único agrupado o no agrupado. La opción predeterminada es OFF, lo que significa que se produce un mensaje de error y la transacción INSERT es devuelta al principio en su totalidad. Cuando la opción elegida es ON, se emite un mensaje de aviso, y solamente aquellas filas que violen el índice único darán un mensaje de error.

STATISTICS\_ NO Especifica si las estadísticas de la distribución se computarán s  
Si la opción es OFF, la actualización predeterminada y utomática de las estadísticas queda abilitada. Si es ON, las estadísticas fuera de fecha no se computan de nuevo.

DROP\_EXISTING Si la opción activada es ON, se especifica que el índice nombrado y preexistente, bien sea agrupado o no agrupado, se elimine y sea reconstruido. La opción predeterminada es OFF.

ONLINE Si la opción activada es ON, especifica que las tablas e índices asociados están disponibles para consultas y modificación de datos durante la operación de indexado. La opción predeterminada es OFF.

ALLOW\_ROW\_LOCKS Si la opción activada es ON (opción predeterminada) se permiten los bloqueos de fila.

ALLOW\_PAGE\_LOCKS Si la opción activada es ON (opción predeterminada) se permiten los bloqueos de página.

MAXDOP Reemplaza la opción de configuración max degree of parallelism solamente durante la duración de la operación. MAXDOP limita el número de procesadores que pueden utilizarse en la ejecución de un plan en paralelo, hasta un máximo de 64 procesadores (la ejecuciónde índices en paralelo únicamente es posible en SQL Server 2012 Enterprise Edition).

De todas estas opciones, examinaremos con mayor detalle la opción ONLINE. Tal y como se señala en la tabla, esta opción permite especificar si SQL Server debe crear índices en línea o fuera de línea. La opción por defecto es ONLINE OFF. Si un índice agrupado se construye offline, SQL Server bloquea la tabla y los usuarios no pueden seleccionar ni modificar los datos. Si lo que se construye offline es un índice no agrupado, entonces SQL Server adquiere un bloqueo de tabla compartido, que permite declaraciones SELECT pero no la modificación de los datos.

Cuando se especifica la opción ONLINE ON durante la creación del índice, las consultas SELECT y las declaraciones de modificación de datos pueden acceder a la tabla o vista subyacentes. Cuando SQL Server crea un índice en línea, utiliza la funcionalidad de versión de filas para asegurar que el índice pueda construirse sin entrar en conflicto con otras operaciones de la tabla. La creación de índices online está únicamente disponible en la versión de SQL Server 2012 Enterprise Edition.

Deshabilitar un índice.

Puede deshabilitar un índice utilizando la declaración ALTER INDEX de Transact-SQL:

ALTER INDEX { index\_name | ALL }

ON <object>

DISABLE [ ; ]

Al deshabilitar un índice, la definición del mismo permanece en el catálogo del sistema, pero SQL Server deja de utilizarla. SQL Server no mantiene el índice en forma de datos en los cambios de tabla, y el índice no puede ser utilizado en consultas. Si se deshabilita un índice agrupado, toda la tabla es inaccesible.

Para volver a habilitar un índice, deberá primero eliminarlo y posteriormente recrearlo para regenerar y repoblar la estructura del árbol-B. Puede hacerlo mediante la instrucción ALTER INDEX utilizando la cláusula REBUILD:

ALTER INDEX { index\_name | ALL }

ON <object>

REBUILD [ ; ]

Práctica: Crear un índice agrupado.

Esta práctica le ayudará a crear un índice agrupado. Posteriormente lo deshabilitará y volverá a rehabilitarlo.

1. Inicie SSMS, conecte con su instancia(con su usuario administrador) y abra una nueva ventana de consulta.

2. Cambie el contexto a la base de datos AdventureWorks.

3. Cree un índice agrupado en la columna PostTime de la tabla DatabaseLog mediante la ejecución del siguiente comando:

CREATE CLUSTERED INDEX ci\_postdate

ON dbo.DatabaseLog(PostTime);

4.Ejecute la siguiente consulta para verificar que los datos pueden ser extraídos de la tabla:

SELECT \* from dbo.DatabaseLog;

5. Deshabilite el índice ejecutando la siguiente instrucción:

ALTER INDEX ci\_postdate ON dbo.DatabaseLog DISABLE;

6. Verifique que no se puede acceder a la tabla:

SELECT \* from dbo.DatabaseLog;

7. Vuelva a habilitar el índice agrupado y verifique que tiene acceso a la tabla

mediante la instrucción:

ALTER INDEX ci\_postdate ON dbo.DatabaseLog REBUILD;

GO

SELECT \* from dbo.DatabaseLog;

**Tema 7: Crear índices no agrupados.**

Después de construir un índice agrupado, puede crear índices no agrupados en la misma tabla. Frente a los índices agrupados, un índice no agrupado no exige un orden determinado de los datos de la tabla. Además, puede crear múltiplesíndices no agrupados para conseguir una devolución de resultados más eficazbasada en los tipos de consulta más frecuentes que puede ejecutar en la base dedatos. En esta lección, aprenderá a crear índices no agrupados, así como a construir un índice cubierto que pueda satisfacer una consulta por sí mismo. También aprenderá la importancia de equilibrar el número de índices creados con el gasto en recursos que requiere su mantenimiento.

**Implementar un índice no agrupado.**

Dado que un índice no agrupado no impone ningún tipo de orden dentro de una tabla, es posible crear hasta 249 índices de este tipo en una sola tabla. Los índices no agrupados, al igual que los agrupados, crean una estructura de árbol-B. Sin embargo, frente a los índices agrupados, la capa de hojas del índice en un índice no agrupado contiene un puntero hacia los datos en lugar de los datos en sí.

El puntero puede hacer referencia a uno de dos elementos: si la tabla tiene un

índice agrupado, el puntero apuntará hacia la clave de índice agrupada. Si por el

contrario la tabla carece de un índice agrupado, el punto apunta a un identificador

relativo (RID), que es una referencia a la localización física de los datos dentro de la página de datos.

Cuando el puntero hace referencia a un índice no agrupado, la consulta transita a través de la estructura de árbol-B del índice. Cuando la consulta lleva a la capa de hojas, utiliza el puntero para encontrar la clave de agrupamiento. Entonces, la consulta recorre el índice agrupado para llegar a la fila de datos. Si no existe un índice agrupado en la tabla, el puntero devuelve un RID, que hace que SQL Server explore un mapa de locación interno para localizar la página referenciada por el RID para poder así devolver los datos solicitados.

Puede utilizar la instrucción CREATE. . . INDEX para crear un índice no agrupado, al igual que se hace para crear un índice agrupado, con la salvedad de que habrá de especificar la palabra NONCLUSTERED.

**Crear un índice cubierto.**

Un índice contiene todos los valores de una columna o columnas que lo definen. SQL Server almacena estos datos de forma ordenada en páginas medianteuna lista doblemente enlazada, de manera que un índice es, en lo esencial, unarepresentación en miniatura de una tabla.

Esta estructura puede tener efectos interesantes sobre algunas consultas. Si la consulta necesita devolver datos únicamente de columnas con índice, no necesita acceder directamente a las páginas de datos en sí. Transitando el índice, puede localizar todos los datos necesarios.

Por ejemplo, imaginemos que está utilizando la tabla Customer de Adventureworks para encontrar los nombres de todos los clientes que tienen una línea de crédito de más de 10,000 euros. SQL Server exploraría la tabla para localizartodas las filas con un valor mayor de 10,000 en la columna CreditLine, lo cual resultaría altamente ineficaz. Si creáramos un índice en la columna CreditLine, SQL utilizaría el índice para localizar todas las filas que se ajustaran a este criterio de búsqueda. Posteriormente, transitaría por la clave primaria, porque está agrupada, para devolver los nombres de los clientes. Sin embargo, si creásemos un índice no agrupado con dos columnas (Credit Line y Customer Name), SQL Server no tendría acceso al índice agrupado para localizar las filas de datos. Al utilizar el índice no agrupado para localizar todas las filas en las que la línea de crédito fuera superior a 10,000 euros, SQL Server localizaría también los nombres de los clientes.

Los índices que SQL Server utiliza para realizar una consulta sin acceder a la tabla se denominan índices cubiertos.

De forma aún más interesante, SQL Server puede utilizar más de un índice para una determinada consulta. En el ejemplo anterior, podíamos crear índices no agrupados en la línea de crédito y el nombre de cliente, que SQL Server utiliza después conjuntamente para realizar la consulta.

*Selección del índice*

*SQL Server determina si debe utilizar un índice examinando únicamente la primera columna definida en el mismo. Por ejemplo, si definimos un índice en FirstName, LastName y una consulta estuviera buscando en el campo LastName, este índice no se utilizaría para ejecutar la consulta.*

**Equilibrar el mantenimiento del índice.**

¿Por qué no, entonces, crear docenas o incluso cientos de índices en una tabla? A primera vista, y sabiendo lo útiles que pueden llegar a resultar los índices, no parece una mala idea. Sin embargo, recuerde que un índice se construye. Los valores de la columna sobre la que se construye el índice son los utilizados para construirlo. Y los valores dentro del propio índice son, a su vez, organizados en un orden determinado. Supongamos que se añade una nueva fila a la tabla: antes de que la operación llegue a su fin, el valor de esta nueva fila debe afladirse a la localización correcta dentro del propio índice.

Si solamente tenemos un índice en la tabla, una operación de escritura en la misma creará también una operación de escritura en el índice. Si el número de índices es 30, las operaciones de escritura en la tabla ocasionarán igualmente 30 operaciones de escritura en el indice.

Pero esto no es todo: si una página de índice en la capa de hojas no tiene espacio para el nuevo valor, SQL Server tiene que llevar a cabo una operación denomina da ruptura de pagina o page split. Durante esta operación, SQL Server designa una página vacía al índice y traslada la mitad de los valores de la página que se ha llenado a la nueva página. Si la ruptura de página hace que una de las páginas de indexado del nivel intermedio se desborde, se producirá una nueva ruptura también a ese nivel. Y si la nueva fila hace que la página raíz se desborde, SQL Server romperá la página raíz en un nuevo nivel intermedio, con lo cual se creará una nueva página raíz.

Como puede verse, los índices pueden mejorar el rendimiento en las consultas,pero cada índice que se crea degrada el rendimiento de todas las operaciones demanipulación de datos. En consecuencia, deberá considerar cuidadosamente elnúmero de índices que debe crear para que las operaciones se desarrollen de forma óptima. Como norma de aplicación general, si tiene cinco o más índices en una tabla diseñada para operaciones de tipo OLTP, probablemente necesite reevaluar por qué existen esos índices. Las tablas diseñadas para consultas relacionadas con operaciones de lectura o almacenamiento masivo de datos tienenpor lo general 10 o más índices porque no hemos de preocuparnos por el impacto de los mismos en las operaciones de escritura.

**Cuándo utilizar columnas incluidas.**

Además de tener en cuenta la degradación del rendimiento ocasionada por las operaciones de escritura, habrá de considerar que los índices tienen un límite máximo de 900 bytes. Este límite puede llegar a constituir todo un reto a la hora de construir índices cubiertos más complejos.

Una de las características de indexado de SQL Server 2012, las columnas incluidas, permite hacer frente a este reto. Las columnas incluidas forman parte del índice únicamente en el nivel de hoja del mismo. Los valores de las columnas incluidas no aparecen en los niveles raíz o intermedio del índice y no afectan a la limitación de tamaño de 900 bytes para el índice.

Práctica: Crear índices no agrupados.

En esta práctica, añadirá un índice no agrupado a algunas tablas.

1. Inicie SSMS, conecte con su instancia y abra una nueva consulta.

2. Dado que la mayoría de usuarios buscan a una persona por el campo ciudad, añada un índice no agrupado a la tabla Address en la columna City, como se muestra a continuación:

CREATE NONCLUSTERED INDEX idx\_CustomerAddress\_City ON

Person.Address (City);

**Tema 8: Crear particiones.**

Una de las funcionalidades de Microsoft SQL Server 2012 son las particiones. Las particiones permiten dividir una tabla en múltiples unidades de almacenamiento llamadas grupos de archivo, basándose en las especificaciones dadas por el usuario. Desde SQL Server 7.0, los administradores de bases de datos han sido capaces de separar tablas e incluso índices en grupos dearchivo específicos, de modo que, ¿qué tienen de especial las particiones?

Mediante las particiones, es posible colocar un subconjunto de una tabla o un índice en un grupo de archivos específico. Esto permite separar partes específicas de una tabla o índice en grupos de archivo individuales y administrar de forma efectiva el flujo de input/output (I/O) de las tablas volátiles. Además, y dado que las organizaciones cada día almacenan más y más datos durante periodos más largos de tiempo para poder analizarlos con posterioridad, las tablas son cada vez más grandes. Administrar tablas de tamaño masivo puede resultar una tarea muy compleja. Las particiones, no obstante, permiten la segregación de datos dentro de una tabla basada en la antigüedad de los datos, lo que permite a su vez realizar copias de seguridad de un subconjunto específico dentro de la tabla. La razón más importante para utilizar particiones es que permiten administrar con facilidad las rutinas de archivo y operaciones de carga de datos.

*Caso de uso : Mundo real (Michael Hotek)*

*Un día, recibí un mensaje instantáneo de uno de los educadores de nuestra compañía, Solid Quality Learning, que estaba trabajando con un cliente en una operación de almacenamiento masivo de datos y se encontraba con problemas a la hora de realizar operaciones de carga de datos habituales.*

*La tabla contenía millones de filas de datos. Cada día, las operaciones cargaban los conjuntos de datos más recientes en la tabla y eliminaban los datos de más de 90 días de antigüedad. Debido a que cada operación implicaba utilizar cientos de miles y millones de filas, cada operación resultaba compleja, pero podía ser resuelta utilizando procesos normales de inserción y borrado.*

*El principal reto lo constituía el hecho de que la tabla tenía un gran número de índices. El cliente no podía eliminar los índices para la operación de inserción de datos porque recrearlos habría supuesto que la base de datos permanecería inactiva durante varias horas (la compañía tampoco conocía la opción ONLINE para crear índices). Cargar los datos en la tabla con los índices en funcionamiento exigía varias horas, y gran parte de ese tiempo se dedicaba al mantenimiento de los mismos índices que causaban el bloqueo de la operación. El archivo de datos con más de 90 días de antigüedad daba lugar a problemas similares.*

*Afortunadamente, SQL Server 2012 ofrece un modo de cargar todos estos datos en la tabla sin incurrir en ningún consumo de tiempo y recursos excesivo durante el mantenimiento de los índices, y que permite además construir de forma incremental la porción del índice relativa a los nuevos datos introducidos.*

*SQL Server 2012 presenta la función partición de tablas. El uso de esta característica, independiente del número de filas a insertar, permitió a la compañía añadir datos a la tabla operativa en menos de un segundo, sin necesidad de que los índices fueran reconstruidos y sin ocasionar ningún problema. El mismo proceso pudo ser aplicado al archivo de datos, lo cual permitió a la compañía eliminar tantas filas como quisiera en menos de un segundo sin que los índices tuvieran que ser reconstruidos.*

*Las particiones y SQL Server Management Studio (SSMS).*

*Las particiones son una de las nuevas características de SQL Server 2012 que carecen de una interfaz gráfica de usuario en SSMS. Deberá realizar todas las operaciones relativas a las particiones escribiendo código.*

Cómo crear una función de partición.

Una función de partición es un objeto independiente de la base de datos que define los límites a la hora de particionar los datos. La creación de una función de partición es el primer paso a la hora de particionar una tabla, un índice, o una vista indizada. Estos son los únicos objetos que pueden particionarse, puesto que son los únicos que pueden almacenar datos en la base de datos.

Para crear una función de partición, utilice la instrucción de Transact-SQL CREATE PARTITION FUNCTION. La sintaxis general es como se muestra a continuación:

CREATE PARTITION FUNCTION partition\_function\_name ( input\_parameter\_type)

AS RANGE [ LEFT | RIGHT ]

FOR VALUES ( [ boundary\_value [ , . . . n ] ] ) [ ; ]

En primer lugar, dé un nombre a la partición que se adecúe a las normas de identificador de objetos. Después, especifique un tipo de datos para los parámetros de entrada. Puede utilizar cualquier tipo de datos excepto text, ntext, varchar(max), image, xml, timestamp, nvarchar(max), varbinary (max), tipos definidos por el usuario de Transact-SQL, y tipos de datos para CLR.

SQL Server 2012 únicamente permite crear funciones de rango. En la cláusula de la instrucción RANGE, deberá especificar un valor RIGHT o LEFT para especificar a qué partición pertenece el valor de límite de la misma. En la cláusula VALUES, deberá especificar la lista de los valores límite para la función de partición. Examine con detenimiento el siguiente ejemplo para entender el funcionamiento de esta instrucción:

CREATE PARTITION FUNCTION partfunc (int) AS

RANGE LEFT FOR VALUES (1000, 2000, 3000, 4000, 5000) ;

Esta instrucción crea una función de partición llamada partfunc que se aplica a valores de datos de tipo integer. Esto quiere decir que no puede aplicar la función a columnas que, o bien no estén definidas como tipo de datos integer, o que no hayan sido implícitamente convertidas a este tipo de datos.

La cláusula RANGE LEFT del ejemplo especifica que cada punto delimitador definido para la función reside en la partición izquierda. La cláusula VALUES define los puntos de límite de las particiones. Una función de partición señala siempre el rango completo de valores permitidos, de manera que todos los valores posibles están siempre definidos para una partición específica. La función de partición partfunc define seis particiones, tal y como muestra la tabla 6.1.

**Tabla 5**. Las seis particiones de la función de partición partfunc.

ID de partición Rango de valores

1 -infinito a 1,000

2 1,001 a 2,000

3 2,001 a 3,000

4 3,001 a 4,000

5 4,001 a 5,000

6 5,001 a +infinito

Si ha definido la función de partición como RANGE RIGHT en lugar de RANGE LEFT, los valores se corresponden con los mostrados en la tabla 6.

**Tabla 6**. Valores de la partición si la función de partición se define como RANGE RIGHT.

ID de partición Rango de valores

1 -infinito a 999

2 1,000 a 1,999

3 2,000 a 2,999

4 3,000 a 3,999

5 4,000 a 4,999

6 5,000 a +infinito

Nótese que una función de partición:

* No especifica una tabla, índice o vista indizada.
* No especifica un grupo de archivos.
* No referencia datos físicos de ninguna clase.

El esquema de partición y la declaración de creación de tablas o índices utilizan un objeto independiente de la base de datos para implementar la partición.

Práctica: Crear una función de partición.

En esta práctica, aprenderá a crear una función de partición que podrá utilizar en las prácticas siguientes. Para simplificar esta tarea, utilizaremos una base de datos de prueba.

1. Inicie SSMS, conecte con su instancia y abra una nueva ventana de consulta.

2. Si aún no ha creado un directorio llamado C:\AdSQL\test, créelo ahora.

3. Cree una nueva base de datos llamada partitiontest con la siguiente sintaxis:

USE master

GO

CREATE DATABASE partitiontest

ON PRIMARY

( NAME = db\_dat,

FILENAME = ' c:\AdSql\test\db.mdf' ,

SIZE= 3MB),

FILEGROUP FG1

( NAME= FG1\_dat,

FILENAME = 'c:\AdSql\test\FG1.ndf' ,

SIZE = 2MB),

FILEGROUP FG2

( NAME = FG2\_dat,

FILENAME 'c:\AdSql\test\FG2.ndf' ,

SIZE = 2MB),

FILEGROUP FG3

( NAME = FG3\_dat,

FILENAME='c:\AdSql\test\FG3.ndf' ,

SIZE = 2MB),

FILEGROUP FG4

( NAME = FG4\_dat,

FILENAME = 'c:\AdSql\test\FG4.ndf' ,

SIZE = 2MB)

LOG ON

( NAME = db\_log,

FILENAME = 'c:\AdSql\test\log.ndf' ,

SIZE = 2MB,

FILEGROWTH = 10% );

GO

USE partitiontest

GO

4. Cree una función de partición con la siguiente instrucción:

CREATE PARTITION FUNCTION partfunc (int) AS

RANGE LEFT FOR VALUES (1000, 2000)

GO

5. Compruebe los resultados de ejecutar esta instrucción mediante la ejecución de la siguiente consulta:

SELECT \* FROM sys.partition\_range\_values;

**Tema 9: Crear un esquema de partición.**

El segundo paso a la hora de particionar una tabla, índice o vista indizada consiste en crear un esquema de partición. Un esquema de partición define las estructuras físicas de almacenamiento, o grupos de archivo, que se utilizarán en una función de partición concreta.

**Cómo crear un esquema de partición.**

Puede utilizar la instrucción de Transact-SQL CREATE PARTITION SCHEME para crear un esquema de partición. La sintaxis general para este tipo de instrucción se muestra a continuación:

CREATE PARTITION SCHEME partition\_scheme name

AS PARTITION partition funcion name

TO ( { file\_group\_name I ( PRIMARY ] ) } [ , .. . n ] ) [ ; ]

Empezaremos dando un nombre al esquema de la partición siguiendo las reglas

de identificadores de objetos. Posteriormente utilizaremos la instrucción PARTITION para especificar el nombre de la función de partición que se vinculará a este esquema.

En la cláusula TO de la instrucción, especificaremos la lista de grupos de archivo

que definen el almacenamiento en disco para cualquier dato que utilicemos en el

esquema de partición. Cualquier grupo de archivos que especifiquemos en esta

cláusula tendrá que haber sido añadido con anterioridad a la base de datos, deberá tener al menos un archivo asignado al mismo, y no podrá estar marcado como de sólo lectura.

El siguiente ejemplo muestra cómo utilizar la instrucción para crear un esquema de partición llamado partscheme:

CREATE PARTITION SCHEME part scheme AS

PARTITION partfunc TO

([FG1], [F02], (F03], (FG4], [FG5], [FG6])  
GO

Fijese en que aún no hemos especificado una tabla, vista, o vista indizada ni referencia a ningún otro objeto de la base de datos, a excepción de la función de partición. Un esquema de partición simplemente especifica un nombre para una estructura física de almacenamiento.

¿Cómo funciona este esquema de partición con la función de partición estudiada anteriormente? Recuerde que la función de partición partfunc tenía seis particiones. Basándose en la definición del esquema de partición, SQL Server almacena todos los valores que residan en la Partición 1 en FG1, los valores de la partición 2 en FG2, y así sucesivamente. De manera que mediante un diseño cuidadoso de las funciones y esquemas de partición, podremos determinar con exactitud qué conjunto de datos dentro de una tabla, índice o vista indizada habrá de residir dentro de un grupo de archivos concreto.

Práctica: Crear un esquema de partición.

1. Inicie SSMS, conecte con su instancia y abra una nueva ventana de consulta.

2. Cree un esquema de partición mediante la ejecución de la siguiente instrucción:

USE partitiontest

CREATE PARTITION SCHEME partscheme AS

PARTITION partfunc TO

([FG1], [FG2] , [FG3]) ;

3.Vea los resultados mediante la ejecución de la siguiente consulta:

SELECT \* FROM sys.partition\_schemes;

**Tema 10: Particionar tablas e índices.**

Después de crear una función de partición y un esquema de partición, estamos preparados para particionar una tabla, un índice o una vista indexada.

**Crear una tabla, índice o vista indizada particionada.**

Para crear una tabla o índice particionado, deberá utilizar la misma sintaxis empleada en la creación de tablas e índices. Particionar una vista indizada consiste simplemente en particionar el índice de la vista. La instrucción de Transact-SQL CREATE TABLE es como sigue:

CREATE TABLE

[ database name . [ schema\_name ] . [ schema name . ] table name

( { <column definition> | <computed column definition> }

[ <table constraint> ] [ , . . .n ] )

[ ON { partition\_scheme name ( partition\_col umn name ) | filegroup

| “default” } ]

[ {TEXTIMAGEON { filegroup | “default” } ] [ ; ]

A su vez, la sintaxis general para la instrucción de Transaqt-SQL CREATE INDEX es:

CREATE [ UNIQUE ] ( CLUSTERED | NONCLUSTERED J INDEX index\_name

ON <object> ( column [ ASC | DESC ] [ , . . . 3 ])

[ INCLUDE ( column\_name [ . . n ] ) ]

[ WITH ( <relational\_index\_option> [ , . . . n ] ) ]

[ ON { partition\_scheme\_name ( column\_name )

| filegroup\_name | default ) ] [ ; ]

La cláusula más importante para la partición en cada instrucción es ON. Para particionar una tabla o índice, en lugar de especificar un grupo de archivos para dicha cláusula, deberá particionar un esquema de partición.

El siguiente ejemplo de código muestra las declaraciones CREATE PARTITION FUNCTION, CREATE PARTITION SCHEME y CREATE TABLE que utilizaríamos para particionar la tabla CustomerAddress:

CREATE PARTITION FUNCTION partfunc (int) AS

RANGE LEFT FOR VALUES (1000, 2000, 3000, 4000, 5000);

GO

CREATE PARTITION SCHEME partscheme AS

PARTITION partfunc TO

([FG1] , [FG2] , [FG3] , [FG4) , [FG5) , [FG6])

GO

CREATE TABLE dbo.Customer Address

(CustomerAddresslD int IDENTITY(1,1) PRIMARY KEY CLUSTERED

AddressTypeID tinyint NOT NULL,

PrimaryAddressFlag bit NOT NULL,

AddressLine1 varchar(30) NOT NULL,

AddressLine2 varchar(30) NULL,

AddressLine3 varchar(30) NULL,

City varchar(50) NOT NULL,

StateProvinceID int NULL,

PostalCode char(10) NULL,

ON partscheme(CustomerAddressID);

Este ejemplo...:

* ... crea una función de partición llamada partfunc.
* ... crea un esquema de partición llamado partscheme.
* ... crea la tabla dbo.CustomerAddress con la definición especificada.
* ... almacena la tabla en el esquema de partición partscheme.
* .... utiliza la columna CustomerAddresslD de la tabla para decidir qué filas de la tabla empiezan dentro de una partición dada.

A medida que se van añadiendo datos a la tabla, cualquier fila con un CustomerAddressID de 1,000 o menos entrará dentro de la partición 1 y se almacenará en FG1, las direcciones con un ID de entre 1,001 y 2,000 en la partición 2 y así sucesivamente.

**Indices particionados y columnas incluidas.**

El siguiente ejemplo muestra cómo crear un índice particionado en la misma tabla:

CREATE NONCLUSTERED INDEX idx\_CustomerAddress\_City ON

dbo. CustomerAddress (City)

ON partscheme (CustomerAddressID);

Lo interesante de la sintaxis anterior es que el índice está definido en la columna City. Sin embargo, el índice está particionado en la columna CustomerAddressID, que no existe en la definición del índice. La partición aprovecha las características de las columnas incluidas estudiadas anteriormente. Mediante las columnas incluidas, cualquier columna que forma parte del índice agrupado es automáticamente migrada a un índice creado en esa tabla. Esto nos permite particionar los índices igual que se particiona la tabla. Mas adelante veremos por qué esto es importante.

Al crear este índice, SQL Server almacena la porción del índice que se corresponde con un CustomerAddressID igual a o menor de 1,000 en FG1, almacena la porción que va de 1,001 a 2,000 en FG2 y así sucesivamente.

**Particionar una tabla o índice ya existentes.**

Es posible particionar un índice o una tabla ya existentes sin tener que eliminar los y volverlos a crear. En lugar de esto, si eliminamos un índice agrupado y se recrea en otro grupo de archivos, SQL Server mueve todos los contenidos de una tabla al mismo grupo de archivos en el que está el índice agrupado. Podemos utilizar este proceso para particionar una tabla o un índice que ya existen mediante el siguiente procedimiento:

1. Crear una función de partición.

2. Crear un esquema de partición

3. Eliminar el índice agrupado existente.

4. Recrear el índice agrupado sobre el esquema de la partición.

Mediante este proceso, la tabla se particiona de forma automática según el esquema de partición en el que esté alojado el índice, utilizando la función de partición vinculada a dicho esquema. La clave de agrupamiento se migra de forma automática a cada uno de los índices no agrupados como columna incluida, y cada uno de los índices no agrupados se particionan de igual forma a como lo hace la tabla.

Los índices no agrupados pueden particionarse utilizando una función y esquema de partición diferentes a los utilizados para la tabla. Sin embargo, esto no puede hacerse con un índice agrupado.

Práctica: Crear una tabla particionada.

En esta práctica crearemos una tabla de particiones sobre el mismo esquema de partición creado anteriormente.

1. Inicie SSMS, conecte con la instancia, abra una nueva ventana de consultas y cambie el contexto a la base de datos partitiontest.

2. Cree una tabla de prueba de partición mediante la ejecución del siguiente código, que especifica el esquema de partición partscheme en la cláusula ON:

USE PARTITIONTEST

CREATE TABLE dbo.t1 (

id INT

, v CHAR(1000) DEFAULT 'aaaa'

CONSTRAINT ci\_ti\_id PRIMARY KEY CLUSTERED (id))

ON partscheme(id);

3. Puede ver los resultados ejecutando la siguiente consulta:

SELECT \* FROM sys.partitions WHERE object\_id = OBJECT\_ID('dbo.t1');

4. A continuación, añada algunas filas de datos a la tabla mediante el siguiente lote:

SET NOCOUNT ON

DECLARE @i INT

SET @i=10

WHILE @i <=3000

BEGIN

INSERT dbo.t1 (id) SELECT @i

SET @i= @i +10

END

GO

5. Compruebe los datos en la tabla ejecutando la siguiente consulta:

SELECT \* from dbo.t1

6. Comprueba el estado de las particiones mediante la siguiente consulta:

SELECT \* FROM sys.partitions

WHERE object\_id = OBJECT\_ID('dbo.t1')

**Tema 11: Ejecutar consultas sobre las particiones.**

Una de las características más interesantes de las particiones es que es un proceso que permanece totalmente oculto para el desarrollador. Los datos de una tabla se recuperan y modifican exactamente igual tanto en tablas particionadas como no particionadas. Sin embargo, podemos ejecutar una consulta sobre los elementos internos de una partición para determinar sus elementos estructurales o para restringir un conjunto de resultados a una partición concreta. Esta lección le mostrará cómo utilizar la función $ PARTITION para determinar el número de partición que le correspondería a un valor determinado, o restringir una consulta a una partición concreta.

**Cómo ejecutar consultas sobre particiones.**

Puede ejecutar consultas sobre particiones utilizando la función especial $PARTITION. La sintaxis general para esta función es:

[ database\_name. ] $PARTITION.partition\_function\_name(expression)

La función $PARTITION devuelve un número de partición basado en los valores de la columna para una función de la partición determinada. Los usos más comunes para esta función son:

* Determinar el número de partición que correspondería a un determinado valor.
* Limitar una consulta a una partición específica.

La siguiente consulta SELECT muestra cómo utilizar la función $PARTITION para determinar el número de partición para un valor determinado:

SELECT $partition.partfunc (2784) as [PartitionNum];

El siguiente código muestra cómo utilizar $PARTITION para restringir una consulta a una partición específica, en este caso, a la partición número 3:

SELECT \* FROM dbo.CustomerAddress

WHERE $partition.partfunc (CustomerAddressID) = 3

Práctica: Ejecutar consultas sobre una partición.

En esta práctica utilizaremos la función $PARTITION para que nos devuelva el número de la partición para un valor determinado, el número de filas en cada partición con datos y todas las filas dentro de una partición concreta.

1. Abra SSMS, conecte con su instancia, abra una nueva ventana de consultas y cambie el contexto a la base de datos partitiontest.

2. Obtenga el número de filas para cada partición mediante la siguiente consulta:

SELECT $partition.partfunc(id) AS [PartitionNum], count (\*) [NumRows]

FROM dbo.t1 GROUP BY $partition.partfunc(id)

ORDER BY $partition.partfunc(id);

3. Averigue el número de partición al que correspondería el valor 4,000 mediante la siguiente consulta:

SELECT $partition.partfunc(4000) as [PartitionNum];

4. Devuelva todas las filas en la partición número 2 ejecutando la siguiente consulta:

SELECT \* FROM dbo.t1

WHERE $partition.partfunc (id) =2

5. A continuación, añada datos a la tabla y observe los resultados ejecutando el siguiente lote:

SET NOCOUNT ON

DECLARE @i INT, @max INT

SELECT @max=MAX(id) + 10 FROM dbo.t1

SET @i= @max

WHILE @i<= @max + 3000 - 10

BEGIN

INSERT dbo.t1 (id) SELECT @i

SET @i=@i\*10

END

GO

*--Cuál es la distribución de datos en la tabla ahora?*

SELECT $partition.partfunc(id) AS [PartitionNum], count (\*) [NumRows]

FROM dbo.t1 GROUP BY $partition.partfunc(id)

ORDER BY $partition.partfunc(id)

GO

Distribución uniforme de los datos

SQL Server no distribuye los datos de forma uniforme entre las particiones. La distribución uniforme de datos en los ejemplos iniciales resultó ser una coincidencia debida al modo de inserción de los mismos. Los datos se particionan únicamente en los puntos delimitadores. Si una partición determinada contiene más datos que otra, es porque un número mayor de sus filas coincide con el rango definido por la partición. Si desea equilibrar la distribución de datos entre las particiones, deberá hacerlo de forma manual.

**Tema 12: Hacer copias de seguridad.**

Una de las tareas más importantes que debe realizar cualquier administrador de

una base de datos consiste en mantener al menos una copia de una base de datos

operativa en caso de que ocurra algún desastre. El hecho de que la creación de una

copia de seguridad sea una operación frecuente no debe restarle importancia. Este

capítulo explica la importancia de las copias de seguridad de la base de datos para

nuestra estrategia de recuperación. Una estrategia de recuperación define el modo

de recuperación de la base de datos, al tiempo que cumple con las reglas de negocio relativas al tiempo que la base de datos puede estar no operativa y el nivel máximo de pérdida de datos aceptable. La tarea de hacer una copia de seguridad de la base de datos carece de sentido si no contamos con dicha estrategia. Este capítulo describe las diferentes opciones que Microsoft SQL Server ofrece para hacer copias de seguridad de la base de datos. Se explica también cómo restaurar las copias de seguridad para recuperar datos hasta un punto determinado en el tiempo, y aprenderemos a trasladar bases de datos mediante las opciones copia de seguridad/restaurar, separar/adjuntar o el Asistente para copia de la base de datos.

**Herramientas de restauración y copia de seguridad en SSMS.**

Si bien es posible acceder a la mayor parte de ellas a través de la interfaz gráfica de SQL Server Management Studio (SSMS), lo cierto es que las explicaciones en pantalla no resultan muy útiles a la hora de comprender el funcionamiento de estas operaciones. Además, algunas opciones y procesos de restauración no se pueden realizar mediante la interfaz gráfica de usuario (GUI).

***Caso de uso : Mundo real2 (Michael Hotek).***

*En cierta ocasión, mientras me disponía a tomarme unos días de vacaciones, recibí una llamada de un cliente desesperado. La organización para la que trabajaba se enfrentaba a un problema de difícil solución: uno de tos discos había fallado y la persona encargada de instalar la nueva unidad había elegido la unidad incorrecta. El sistema al completo dejó de funcionar y era imposible recuperar el conjunto de discos (RAID) de la instalación. La organización creía que yo era la única persona capaz de solucionar el problema, dado que durante casi dos décadas me ha dedicado a la recuperación de desastres en cientos de compañías y organizaciones de todo el mundo con bastante éxito. La mayor parte de tos proyectos en los que he trabajado implicaban la recuperación de sistemas de producción que yo no había visto con anterioridad. Los daños tenían frecuentemente su origen en las causas más variadas que puedan imaginarse: inundaciones, incendios, tornados, huracanes, tsunamis, rayos, terremotos, inmersión en agua, explosivos, balas, cualquier fallo de hardware imaginable, violaciones de la seguridad y evidentemente errores de los usuarios.*

*Tras entrar en la red virtual privada (VPN) del sistema y mantener varias reuniones, conseguimos reconstruir todo el sistema mediante una combinación de copias de seguridad, extracciones de datos a otros sistemas y la utilización del producto de Lumigent Technologies, Log Explorer. Fuimos incapaces de recuperar una pequeña cantidad de datos dañados durante una operación de recuperación fallida, pero la organización pudo reconstruir los datos manualmente.*

*Justo cuando estaba terminando mi trabajo, recibí un correo electrónico solicitando mi ayuda desde otra organización. Su proveedor había quemado el disco y con él todos los datos. Tampoco existía, como es de suponer, ninguna copia de seguridad de la base de datos. Así que volví a hacerme la misma pregunta de siempre: ¿cuándo aprenderán? La parte más importante de la implementación de una base de datos es, sin duda alguna, contar con una estrategia de copias de seguridad y restauración funcional y que haya sido probada, y poner esa estrategia en funcionamiento. Afortunadamente, no recibo llamadas de este tipo a diario, pero ciertamente, me habría gustado poder disfrutar de un día de descanso lejos de un ordenador...*

**Crear una copia de la base de datos.**

Uno de los aspectos críticos en el mantenimiento de una base de datos es mantener un duplicado de los datos que podamos recuperar en caso de que éstos se pierdan. SQL Server 2012 ofrece una serie de características que nos permiten llevar a cabo estas tareas. El método más común a la hora de mantener copias duplicadas de los datos consiste en utilizar las funciones de copia de seguridad de SQL Server. Algunas de ellas resultarán familiares a quienes hayan trabajado con versiones anteriores de SQL. Server. Además, SQL Server 2012 va más allá, al permitir una mayor flexibilidad a la hora de realizar copias de seguridad. Esta lección explica las diferentes opciones disponibles dentro del motor de copia de seguridad de SQL Server.

Las estructuras de registro y de la base de datos

*Para comprender la arquitectura de las copias de seguridad y de restauración de una base de datos, deberá familiarizarse con la estructura básica de las bases de datos, grupos de archivos, extensiones, páginas de datos y registros de transacción de SQL Server.*

*Permisos para la copia de seguridad*

*Frente a lo que ocurría en versiones anteriores de SQL Server, SQL Server 2012 pretende reforzar la seguridad mediante la ¡mplementación del principio de privilegios mínimos, es decir: la utilización de un conjunto mínimo de permisos necesarios para realizar cualquier operación. Así, deberá saber qué permisos son necesarios para ejecutar una copia de seguridad. Independiente de cómo se inicien, las copias de seguridad se ejecutarán dentro del motor de Ja base de datos bajo el contexto de seguridad de la cuenta de servicio de SQL Server. Debe conceder los permisos necesarios a esta cuenta para leer y escribir en cualquier directorio que utilice en la creación de la copia de seguridad; en caso contrario, las copias de seguridad darán ‘ un mensaje de error, ya que carecerán de los permisos suficientes. Para conceder a un usuario los permisos necesarios para crear una copia de seguridad de Ja base de datos, añada al usuario como miembro de la función* ***dbobackupoperator****. Esta función únicamente puede realizar copias de seguridad, registros o control dentro de la base de datos. No se permite ningún otro tipo de acceso.*

**Copias de seguridad completas.**

La creación de una copia de seguridad completa tiene como propósito capturar todos los datos almacenados dentro de la base de datos. El motor de copia de seguridad lleva a cabo esta tarea mediante la extracción de todas las extensiones de la base de datos dedicados a un objeto. Posteriormente, podrá utilizar una copia de seguridad completa para recrear la base de datos. Este método está siempre a su disposición, independientemente del modelo de recuperación que haya configurado para la base de datos.

*La granularidad de una copia de seguridad.*

*Seguramente habrá leído o leerá en numerosos libros, que una copia de seguridad de SQL realiza una copia de una página de datos y no de una extensión.Esto no es del todo correcto. SQL Server no dedica una única página de datos a un objeto que necesite espacio, sino que dedica una extensión completa. El motor de copia de seguridad sigue el mismo principio: extrae todas las páginas dedicadas a un objeto y, dado que la dedicación se produce extensión por extensión, el motor de copia de seguridad hace, en efecto, una copia de seguridad de todas las extensiones que SQL Server haya dedicado a objetos, independientemente de si SQL Server ha escrito datos o no en todas las páginas dentro de la extensión.*

*Frente a lo que ocurría en versiones anteriores de SQL Server, SQL Server 2012 pretende reforzar la seguridad mediante la implementación del principio de privilegios mínimos, es decir: la utilización de un conjunto mínimo de permisos necesarios para realizar cualquier operación. Así, deberá saber qué permisos son necesarios para ejecutar una copia de seguridad. Independiente de cómo se inicien, las copias de seguridad se ejecutarán dentro del motor de la base de datos bajo el contexto de seguridad de la cuenta de servicio de SQL Server. Debe conceder los permisos necesarios a esta cuenta para leer y escribir en cualquier directorio que utilice en la creación de la copia de seguridad; en caso contrario, las copias de seguridad darán un mensaje de error, ya que carecerán de ¡os permisos suficientes. Para conceder a un usuario los permisos necesarios para crear una copia de seguridad de la base de datos, añada al usuario como miembro de la función dbobackupoperator. Esta función únicamente puede realizar copias de seguridad, registros o control dentro de la base de datos. No se permite ningún otro tipo de acceso.*

**Copias de seguridad completas.**

La creación de una copia de seguridad completa tiene como propósito capturar todos los datos almacenados dentro de la base de datos. El motor de copia de seguridad lleva a cabo esta tarea mediante la extracción de todas las extensiones de la base de datos dedicados a un objeto. Posteriormente, podrá utilizar una copia de seguridad completa para recrear la base de datos. Este método está siempre a su disposición, independientemente del modelo de recuperación que haya configurado para la base de datos.

*La granularidad de una copia de seguridad.*

*Seguramente habrá leído o leerá en numerosos libros, que una copia de seguridad de SQL realiza una copia de una página de datos y no de una extensión. Esto no es del todo correcto. SQL Server no dedica una única página de datos a un objeto que necesite espacio, sino que dedica una extensión completa. El motor de copia de seguridad sigue el mismo principio: extrae todas las páginas dedicadas a un objeto y, dado que Ja dedicación se produce extensión por extensión, el motor de copia de seguridad hace, en efecto, una copia de seguridad de todas las extensiones que SQL Server haya dedicado a objetos, independientemente de si SQL Server ha escrito datos o no en todas las páginas dentro de la extensión.*

El motor de copia de seguridad está configurado para realizar una copia de seguridad tan rápidamente como sea posible utilizando una cantidad mínima de recursos. Al iniciar una copia de seguridad, el motor de copia de seguridad escribe las páginas en el dispositivo de copia de seguridad sin prestar atención a la ordenación. Dado que la ordenación no es necesaria en la copia de seguridad, SQL Server puede abrir múltiples hilos para escribir datos con la mayor rapidez permitida por el medio de escritura empleado. El único factor que limita la velocidad de la copia de seguridad es la velocidad a la que los datos pueden ser copiados en el dispositivo.

Dado que una copia de seguridad no se realiza de forma instantánea y puede tener lugar mientras los usuarios están conectados con la base de datos al tiempo que ejecutan consultas, es posible que se produzca una inconsistencia lógica en la base de datos. Si una página de datos se ha escrito en el dispositivo de copia y posteriormente se ha modificado, la restauración de esta copia de seguridad haría que la base de datos entrase en un estado de inconsistencia.

No obstante, SQL no permite que esto ocurra, ya que aplica los siguientes pasos durante el proceso de copia de seguridad completa:

1. Bloquea la base de datos, así como todas las transacciones.

2. Coloca una marca en el registro de transacciones.

3. Libera el bloqueo de la base de datos.

4. Crea una copia de seguridad de todas las páginas de la base de datos.

5. Bloquea la base de datos y todas las transacciones.

6. Coloca una marca en el registro de transacciones.

7. Libera el bloqueo de la base de datos.

8. Extrae todas las transacciones que se han producido entre las dos marcas de registros y las anexa a la copia de seguridad.

Este proceso garantiza que la base de datos sea totalmente coherente en el momento de la finalización de la copia de seguridad.

La instrucción básica para realizar una copia de seguridad de la base de datos es:

BACKUP DATABASE <database name> TO DISK = ‘<directory>\filenarne>‘

WITH INIT

Utilice la cláusula TO de la instrucción BACKUP DATABASE para especificar el dispositivo en el que se guardará la copia de seguridad; puede tratarse de un dispositivo de copia de seguridad lógico creado para tal efecto, o puede especificarse una ruta explícita a DISK o TAPE. La cláusula WITH tiene más de una docena de parámetros; todos ellos son opcionales. El parámetro INIT (el más común) ordena a SQL Server sobrescribir cualquier elemento del dispositivo de copia que pueda existir antes de empezar la operación de guardado de copia de seguridad.

**Copias de seguridad diferenciales.**

Una copia de seguridad diferencial captura todas las extensiones que hayan cambiado desde la última copia de seguridad completa de la base de datos. El principal propósito de una copia de seguridad diferencial es el de reducir el número de copias de seguridad del registro de transacciones que deberán ser restauradas. Utilice la copia de seguridad diferencial junto a una copia de seguridad completa. Si esta última no existe, no podrá crear una copia diferencial. Al igual que ocurre con las copias completas, puede realizar una copia diferencial de una base de datos independientemente del modelo derecuperación especificado para la misma.

Observe que una copia diferencial no es una copia incremental. Una copia de

seguridad incremental captura cualquier cambio realizado desde la copia de seguridad incremental anterior. En consecuencia, la restauración de una copia de

seguridad incremental necesita de todas las demás copias de seguridad incrementales realizadas con anterioridad. Una copia de seguridad diferencial captura siempre todas las extensiones que han cambiado desde la última copia de

seguridad completa, de modo que cada copia de seguridad diferencial contiene todo aquello que contenga toda copia de seguridad diferencial previa. Por ejemplo, imagine que realizamos una copia de segundad completa a medianoche y que

a lo largo del día se van creando copias de seguridad diferenciales cada cuatro horas. La copia de seguridad diferencial de las 04:00 contendría todas las extensiones que han sido modificadas desde la medianoche. L.a copia diferencial de las 08:00 contendrfa, a su vez, todas las extensiones que han sido modificadas desde la medianoche. Y por último, la copia diferencial del mediodía contendría a su vez todas las extensiones modificadas desde la medianoche.

Para determinar que extensiones deben guardarse mediante una copia de seguridad diferencial, SQL Server mantiene un mapa de extensiones. Un mapa de extensiones no es más que una página de datos dentro de la base de datos; cada bit dentro de la página representa una extensión. Cuando SQLServer modifica una extensión, cambia el bit correspondiente de la extensión de O a 1. Al crear una copia de seguridad completa, SQL Server restaura todos los bits a O. De este modo, SQL Server deberá interrogar únicamente a esta página para decidir qué extensiones debe guardar. Dado que las bases de datos pueden tener un tamaño ilimitado y las páginas de datos únicamente pueden ocupar 8 KB, SQL Server crear una de estas páginas cada 8,192 extensiones dedicadas a los objetos de la base de datos. Así, una única página puede cubrir miles de páginas de datos.

La instrucción más sencilla para realizar una copia de seguridad dìferencial es:

BACK DATABASE <database name> TO DISK ‘<directory>\fileflame>‘ WITH DIFFERENTIAL

La instrucción es prácticamente idéntica a la utilizada en una copia de seguridad completa, salvo que requiere el uso del parámetro DIFFERENTIAL. Las demás opciones funcionan igual que en una copia de seguridad completa.

**Copias de seguridad del registro de transacciones.**

Puede realizar copias de seguridad del registro de transacciones únicamente para aquellas bases de datos que hayan sido configuradas en el modelo de recuperación completo o de recuperación por medio de registros de operaciones masivas, y en las que aún no se haya producido una transacción minimarnente registrada. Las copias de seguridad del registro de transacciones únicamente se permiten después de haber realizado una copia de seguridad completa. Una copia de seguridad del registro de transacciones contiene únicamente un subconjunto de datos y requiere tener también al menos una copia de seguridad completa que nos permita recuperar la base de datos.

Una copia de seguridad de registro copia el registro activo. Se inicia en el número de secuencia de registro (LSN, del inglés Log Sequence Number) en el que se cornpletara la copia de seguridad de registro anterior. SQL Server realiza posteriormente una copia de seguridad de todas las transacciones posteriores hasta que el proceso de copia tope con una transacción abierta, momento en el que el proceso de copia de seguridad del registro finaliza. Todo número de secuencia de registro que haya sido copiado puede ser eliminado del registro de transacciones, lo cual permite al sistema reutilizar el espacio de registro.

**Registro de transacción, replicaciones y copia exacta o mirroring de la base de datos.**

Si está implementando una replicación transaccional o un proceso de copia exacta de la base de datos (mirroring), existe un requisito adicional sobre el registro de transacciones. Ambas características garantizan que los datos se transmitan, y en consecuencia deben asegurarse de que la transmisión de datos se ha producido correctamente antes de que SQL Server elimine cualquier transacción del registro, independientemente de si ha sido copiada o no. Al utilizar estas características, únicamente podrá eliminar una transacción...:

1. ... si ha sido asignada con éxito a la base de datos de distribución.

2. ... si ha sido asignada con éxito a la base de datos de copia.

La forma más sencilla de ejecutar una copia de seguridad del registro de transacciones es mediante la siguiente declaración

BACKUP LOG <database name> TO DISK = ‘directory>\<filename>‘ WITH INIT

**Cómo Crear una copia de seguridad de archivo o grupo de archivos.**

Las copias de seguridad de grupo de archivos ofrecen una estrategia de copia de seguridad alternativa a las copias de seguridad completas. En lugar de realizar una copia de seguridad de la base de datos al completo, puede realizar una copia de seguridad de los grupos de archivo de la misma. El punto de partida para una estrategia de copia de seguridad de grupo de archivos deberá incluir una copia de todos los grupos de archivo dentro de la base de datos que puedan ser posteriormente recreados.

Deberá seleccionar un método para la copia de seguridad de grupo de archivos cuando, por su tamaño, resulta poco práctica guardar o restaurar la basede datos completa.

Dado que una copia de seguridad de grupo de archivo nos permite realizar copias de partes de una base de datos, deberá configurar la base de datos en el modelo de recuperación completa o de recuperación por medio de registros de operaciones masivas, de manera que pueda crear una copia de seguridad de lectura/escritura. Para restaurar la base de datos, podrá entonces utilizar las copias de seguridad de grupo de archivos, diferencial o de transacción de registro.

La forma más sencilla de realizar una copia de seguridad de grupo de archivos es:

BACKUP DATABASE <database name> FILEGROUP ‘<f ilegroup name>‘ TO DISK =

‘<directory>\ filename>‘

También puede tornar una copia de seguridad diferencial sobre la base base de datos o de un grupo de archivos. La instrucción básica para una copia de seguridad de grupo de archivos diferencial es:

BACKUP DATABASE <database name> FILEGROUP = ‘<f ilegroup name>‘ TO DISK =‘<directory>\<fileflame>‘ WITH DIFFERENTIAL

**Cómo crear una copia de seguridad mediante medios reflejados.**

Cada copia de seguridad crea una única copia de los datos. Posteriormente, el administrador debe decidir si crear copias adicionales para proteger a la organización de fallos en los medios. Este proceso de duplicacio puede resultar tan tedioso como costoso en términos de tiempo; además, contar con una única copia de seguridad significa que contamos también con un único punto de error durante el proceso.

SQL Server 2012 cuenta una característica en la instrucción BACKUP. Durante la creación de la copia de seguridad, puede crear Copias adicionales de una copia de seguridad denominada “reflejos”. Para realizar esta operación, utilizaremos la siguiente cláusula opcional dentro de la instrucción BACKUP:

[[MIRROR TO <backup\_device> […,n]] [... next mirror]

Puede crear hasta cuatro reflejos diferentes; tres de ellos pueden especificarse dentro de la cláusula MIRROR TO. Una copia de seguridad mediante un conjunto de medios reflejado impone ciertas restricciones sobre el tipo de medio utilizado para su creación. Los medios utilizados para la creación de cada reflejo deben ser del mismo tipo y tener el mismo número de dispositivos. Además, deberán poseer propiedades similares. Por ejemplo, si estamos realizando una copia de seguridad en un disco, todos tos espejos deberán ser discos; si se trata de un volumen de cinta, todos los demás deberán serlo también.

*La división de datos dentro de las copias de seguridad.*

*Un Conjunto de medios contiene, por norma general, un único dispositivo (por ejemplo, una unidad de cinta o archivo) No obstante un conjunto de medios puede estar compuesto de hasta 4 dispositivos diferentes cuando el conjunto de medios se compone de dispositivos múltiples, el motor de la copia de seguridad genera un hilo por cada dispositivo físico, y escribe una porción de los datos dentro de ¡a copia de seguridad de cada dispositivo. Esto no es igual a la capacidad de división de datos existente en la tecnología RA1D, pero es similar a lo que ocurre en RAID O. Cada reflejo debe ser idéntico a los demás. Así, si especificamos un Conjunto de medios que contiene dos dispositivos de disco, cada reflejo deberá contener a su vez dos dispositivos de disco. Igualmente, si un Conjunto de medios contiene 64 dispositivos de cinta, el espejo deberá especificar 64 dispositivos de cinta.*

Cuando Utilizamos copias de seguridad mediante conjuntos de medios reflejados, SQL Server lee la página a partir de los archivos de datos una vez y posteriormente crea copias múltiples aI tiempo que escribe la página en el disco o cinta. Este proceso tiene como resultado la escritura de la misma página de datos en cada uno de los reflejos de forma simultánea Para asegurar un impacto mínimo mientras se utiliza esta propiedad, cada reflejo deberá tener un tipo de dispositivo idéntico con propiedades similares.

**Localización de las copias de seguridad.**

Uno de los errores más comunes consiste en pensar que los dispositivos que se utilizan para la copia de segundad han de estar físicamente unidos a la máquina en la que se aloja SQL Server. Las copias de segundad pueden enviarse a dispositivos unidos de forma local, y también pueden enviarse a una ruta UNC (Convención de Nomenclatura Universal). Al enviar las copias de seguridad a una ruta UNC. deberá tener en cuenta el impacto de la copia de segundad sobre el ancho de banda de la red.

El siguiente ejemplo creará una copia de segundad de la base de datos PUBS en un conjunto de medios de dos discos, y crea tres reflejos de la copia. La primera copia se crea en un disco unido de forma local, mientras que cada uno de los reflejos son un recurso de red al que se accede por medio a una ruta UNC.

BACKUP DATABASE PUBS TO DISK=‘C:DEMO\BACKUP\PUBS1B.BAK

MIRROR TO DISK=‘\\BAKSERVERl\BACKUP\PUBMIRROR1A.BACK’

DISK=‘ \\BAKSERVER1\BACKUP\PUBSMIRROR1.BAK’

MIRROR TO DISK=‘ \\BAKSERVER2 \BACKUP\PUBMIRROR2A.BACK’

DISK= ‘ \\BAKSERVER2\BACKUP\PUBSMIRROR2.BAK’

MIRROR TO DISK=‘ \\BAKSERVER3\BACKUP\PUBMIRROR3A.BACK’,

DISK=‘ \\BAKCSERVER1\BACKUP\PUBSMIRROR3.BAK’

WITH FORMAT

GO

**La cláusula FORMAT.**

La cláusula FORMAT es, normalmente , un parámetro opcional. Este parámetro escribe un nuevo encabezamiento de medios en cada conjunto de medios. sobrescribiendo el encabezamiento previo e invalidando cualquier otra copia de seguridad contenida en el medio. No obstante, la cláusula FORMAT es obligatoria en las copias de seguridad mediante un conjunto de medios reflejados.

**Copias de seguridad parciales.**

Es posible tener bases de datos en las que algunos grupos de archivo sean de sólo escritura mientras que otros sean de escritura. En versiones anteriores de SQL Server, una copia de seguridad completa capturaba todas las extensiones dedicadas en una base de datos, aún cuando el grupo de archivos estuviera marcado como de solo lectura, lo cual significaba que no se realizaba ningún cambio en los datos. En SQL Server 2012, existe un parámetro añadido a la instrucción BACKUP. La cláusula READ WRITE FILEGROUPS hace que el motor de la copia de seguridad salte todos aquellos archivos marcados como de sólo lectura, ahorrando tiempo y espacio en la copia de seguridad, ya que el motor únicamente debe localizar el conjunto de extensiones que pueden ser modificadas.

A continuación se muestra un ejemplo de ejecución de una copia de seguridad parcial:

BACKUP DATABASE PUBS READ\_WRITE\_FILEGROUPS TO

DISK= ‘C: \DEMO\BACKUP\PUBS1.BAK’

Práctica: Cómo crear una copia de seguridad de una base de datos mediante copias de seguridad completas, diferenciales, de transacción de registros y de grupo de archivo, diferencial de grupo de archivo y de registro de transacciones.

En esta práctica, crearemos copias de seguridad de la base de datos AventureWorks utilizando dos métodos diferentes: el completo/diferencial/de registro de transacciones y el de grupo de archivos/diferencial/de registro de transacciones.

Práctica 1: Crear una copia de seguridad mediante copia completa, diferencial y de registro de transacciones.

En esta práctica, crearemos una copia de seguridad de la base de datos AdventureWorks utilizando varias copias completas, diferenciales y de registro de transacciones.

1. Inicie SSMS, conecte con su instancia de SQL Server y abra una nueva ventana de consulta.

2. Cree un nuevo directorio llamado C:\AdSql\test2.

3. Ejecute la siguiente instrucción para crear una copia de seguridad completa de la base de datos:

Use MASTER

BACKUP DATABASE Adventureworks TO DISK = C:\AdSql\test2\AW.BAK'

4. Realice un cambio en la tabla Production. Product de la base de datos.(Ejemplo: update Production.Product SET StandarCost = StandarCost +1)

5. Ejecute la siguiente instrucción para crear una copia de seguridad del registro de transacciones y capturar el cambio realizado:

BACKUP LOG AdventureWorks TO DISK= ' C:\AdSql\test2\AW1.TRN'

6. Realice otro cambio en la tabla Production. Product.

7. Ejecute la siguiente instrucción para crear una copia de seguridad diferencial de la base de datos:

BACKUP DATABASE AdventureWorks TO DISK= 'C:\AdSql\test2\AWDIFF1.BAK' WITH DIFFERENTIAL

8. Realice otro cambio en la tabla **Production.Product**.

9. Ejecute la siguiente instrucción para crear una copia de seguridad completa de la base de datos de vuelta en la localización de disco especificada:

BACKUP LOG AdventureWorks TO DISK='C:\AdSql\test2\AW2.TRN'

Revise la carpeta C:\AdSql\test2\

Práctica 2: Crear una copia de seguridad de una base de datos mediante copias de seguridad de grupo de archivos, diferencial de grupo de archivos y de registro de transacciones.

En esta práctica, añadiremos un segundo grupo de archivos a la base de datos AdventureWorks y posteriormente crearemos una serie de copias de seguridad de grupo de archivos completa, diferencial de grupo de archivos y de registro de transacciones.

1. Inicie SSMS, conecte con su instancia SQL Server y abra una nueva ventana de consulta.

2. Cree un directorio llamado C:\AdSql\test3

3. Ejecute el siguiente lote de instrucciones para añadir el grupo de archivos FG1:

Use AdventureWorks

ALTER DATABASE [AdventureWorks]

ADD FILEGROUP [FG1]

Go

ALTER DATABASE [AdventureWorks]

ADD FILE

( NAME = AW1DATA

FILENAME ' C:\AdSql\test3\FG1.NDF',

SIZE = 5 MB)

TO FILEGROUP [FG1]

GO

*--Ahora creamos una tabla de prueba en el grupo de archivos*

CREATE TABLE dbo.t1 ( id INT, v CHAR (1000) DEFAULT 'bbbb',

) ON [FG1]

GO

4. Para crear una copia de seguridad completa de grupo de archivos, ejecute la siguiente instrucción:

BACKUP DATABASE Adventureworks FILEGROUP = 'PRIMARY' TO DISK = ' C:\AdSql\test3\AWPRI .BAK'

GO

BACKUP DATABASE AdventureWorics FILEGROUP = 'FG1' to disk =' C:\AdSql\test3\AWFG1.BAK'

GO

5. Inserte una fila de datos en la tabla dbo. t1.

6. Para crear una copia de seguridad del registro de transacciones de la base de datos, ejecute la siguiente instrucción:

BACKUP LOG AdventureWorks TO DISK = ' C:\AdSql\test3\AW3.TRN'

7. lnserte otra fila de datos en la tabla dbo. t1.

8. A continuación, cree una copia de seguridad diferencial de grupo de archivos mediante la siguiente instrucción:

BACKUP DATABASE Adventureworks FILEGROUP = 'FG1' TO DISK

' C:\AdSql\test3\FG1DIFF1.BAK' WITH DIFFERENTIAL

9. Inserte una nueva fila en la tabla dbo.t1.

10. Ejecute la siguiente instrucción para crear una nueva copia de seguridad de registro de transacciones que capture el último cambio realizado en los datos:

BACKUP LOG AdrentureWorks TO DISK = ' C:\AdSql\test3\AW4.TRN'

**Tema 13: Restaurar una base de datos.**

La habilidad para restaurar una copia de seguridad de una base de datos determina la rapidez con la que la base de datos puede responder a los requisitos de un negocio determinado después de haber sufrido daños. Esta lección explica todas las opciones disponibles en SQL Server 2012 para restaurar una base de datos de forma total o parcial. Puede utilizar la información presentada en esta lección para crear la base de un plan de recuperación en caso de desastre para su organización.

La planificación orientada a la recuperación.

La restauración de una base de datos es el concepto más importante que debe manejar. Este tema se explica en esta lección porque anteriormente era necesario explicar los diferentes tipos de copias de seguridad que pueden crearse para elaborar una estrategia de restauración. Esta estrategia es la que, en definitiva, determinará si una copia de seguridad tiene éxito o no. Como suele decirse en estos casos: si nunca ha restaurado una copia de seguridad, es como si no la tuviera.

**Restaurar una copia de seguridad completa.**

La mayor parte de operaciones de restauración se inician recreando una base de datos en un punto específico para posteriormente aplicar todas las copias de seguridad siguientes y actualizar la base de datos a un punto específico. El proceso comienza con la restauración de una copia de seguridad completa.

Tal y como se ha explicado anteriormente, una copia de seguridad completa contiene todos los contenidos de una base de datos al completo. Para reconstruirla, la operación de restauración debe colocar las páginas de vuelta en la base de datos en un orden secuencial. Este proceso asegura la completa coherencia de la base de datos, pero lleva más tiempo. Restaurar una copia de seguridad cornpleta requiere, por norma general, un 30 por 100 más de tiempo de lo que se ha tardado en crear la propia copia de seguridad.

Sobrescribir y mover una base de datos.

La restauración de una copia de seguridad completa sobrescribe la base de datos del mismo nombre, en caso de que ésta exista en la misma instancia. Si la base de datos no existe, la operación de restauración crea los archivos y grupos de archivos de Ja base de datos antes de restaurar las páginas. Dado que la creación de estos archivos desde cero puede suponer un consumo significativo de tiempo, no debe eliminar la base de datos antes de restaurarla si va a sobrescrib irla. Si va a realizar una copia de seguridad y restauración para mover la base de datos a un servidor diferente con una estructura de directorios diferente, o si la estructura de directorios ha sido modificada,puede utilizar la opción WITH MOVE para hacer que la operación de restauración cree los archivos subyacentes en una ruta diferente a la de la copia de seguridad original.

A continuación se muestra un ejemplo de la sintaxis para la restauración completa de una base de datos:

RESTORE DATABASE PUBS FROM DISK = 'C:\DEMO\BACKUP\PUBSFULL.BAK'

WITH REPLACE, STANDBY = 'C:\DEMO\BACKUP\PUBSSTANDBY.STN'

Esta instrucción utiliza el contenido del archivo PUBSFULL.BAK para la operación de restauración. La opción REPLACE hace que SQL Server sobrescriba labase de datos existente, de nombre PUBS. La opción STANDBY hace que la base de datos quede en estado de restauración: no se permiten más operaciones de escritura, pero los usuarios pueden conectarse a la base de datos y escribir declaraciones SELECT.

Otras dos cláusulas importantes en cualquier instrucción de restauración son WITH RECOVERY o WITH NORECOVERY.

Si la opción de restauración utiliza la opción WITH RECOVERY, la base de datos está en línea, el LSN avanza y la base de datos puede aceptar transacciones. No se permite ninguna operación de restauración más después de haber recuperado una base de datos mediante la opción WITH RECOVERY.

Cuando una operación de restauración utiliza la opción WITH NORECOVERY, el estado de la base de datos o del grupo de archivos sigue fijado en modo RESTORING. En este estado, es posible restaurar copias de seguridad adicionales, tales como las copias de seguridad diferenciales o del registro de transacciones, para aplicar los cambios que hayan ocurrido desde que se realizó la última copia de seguridad completa.

Las operaciones de restauración

Una operación de restauración puede constituir un paso único en el que se restaure una copia de seguridad completa, lo cual permite recuperar la base de datos y seguir adelante con las transacciones. No obstante, en la mayor parte de entomos de producción, una operación de restauración consiste en varios archivos de copias de seguridad múltiples que se van restaurando uno después del otro, de manera que la base de datos vaya progresivamente quedando en un estado determinado y poder así asegurar que se recupera el mayor número de datos posible. Para que esto ocurra, la instrucción RESTORE debe permitir que el usuario pueda especificar de forma explícita en qué momento se llevó a cabo la última operación de restauración, y la base de datos debe recuperarse y ser puesta en servicio.

**Restaurar una copia de seguridad diferencial.**

Para restaurar una copia de seguridad diferencial, primero ha de restaurar la copia de seguridad completa al tiempo que se asegura de que la base de datos no ha sido recuperada aún. Posteriormente, la copia de seguridad diferencial más reciente se aplica a la base de datos.

Restauración diferencial de grupo de archivos.

El proceso a seguir para restaurar una copia de seguridad diferencial de grupo de archivos es muy parecido a la restauración de una copia de seguridad diferencial: deberá ejecutar una restauración completa de grupo de archivos sin recuperar el grupo de archivos.

Examine el siguiente ejemplo, que muestra una secuencia de operaciones para realizar una copia de seguridad completa seguida de una copia de seguridad diferencial:

RESTORE DATABASE PUBS FRON DISK = 'C:\DEMO\BACKUP\PUBSFULL.BAK' WITH NORECOVERY

RESTORE DATABASE PUBS FROM DISK = ‘C:\DEMO\BACKUP\PUBSDIFF.BAK’ WITH RECOVERY

La primera instrucción restaura una copia de seguridad completa pero deja la base de datos sin recuperar. La segunda aplica la restauración y a continuación recupera la base de datos.

La sintaxis para restaurar una copia de seguridad completa es idéntica a la utilizada para restaurar una copia de seguridad diferencial. SQL Server toma las extensiones de la copia diferencial y las escribe en la base de datos.

A continuación se muestra un ejemplo de esta secuencia de operaciones para una copia de seguridad de grupo de archivo junto con una copia de seguridad diferencial de grupo de archivos:

RESTORE DATABASE Adventureworks FILEGROUP = “FG1” FROM DISK =

'C:\TEST\AWFG1. BAK' WITH NORECOVERY

RESTORE DATABASE Adventureworks FILEGROUP = “FG1” FROM DISK =

'C:\TEST\FG1DIFF1.BAK' WITH RECOVERY

Al restaurar una copia de seguridad diferencial para mover una restauración de grupo de archivos hacia adelante, no es necesario especificar el grupo de archivos al que se está aplicando el diferencial. SQL Server reconoce de forma automática los grupos de archivo que se encuentran en la fase RESTORING, así como todas las extensiones dentro de la copia de seguridad diferencial que pueden aplicarse al grupo de archivos. Todas aquellas extensiones que no se correspondan con un grupo de archivos en fase RESTORING son ignoradas.

**Restaurar una copia de seguridad del registro de transacciones.**

Puede utilizar las copias de seguridad del registro de transacciones para avanzar la base de datos hasta un punto en el tiempo. Por normal general, este punto es la última operación ejecutada contra la base de datos, pero es posible seleccionar un momento diferente. Los registros de transacciones pueden aplicarse a una copia de seguridad completa, o después de la restauración de una copia de seguridad diferencial.

Una copia de seguridad del registro de transacciones contiene una secuencia de transacciones identificada por un LSN. Las transacciones pueden nombrarse también de forma explícita mediante una marca en el registro de transacciones, que registra la hora exacta en la que se ejecuta una transacción, así como los cambios realizados en la misma.

Restaurar una base de datos a un punto específico.

Utilice la opción STOPAT para restaurar una base de datos a un LSN, transacción con nombre, o punto en el tiempo concreto. Esta característica permite que la base de datos sea restaurada de forma que no contenga todas las transacciones. Normalmente, elegimos esta opción para restaurar una base de datos que se ha corrompido para restaurar justo hasta el momento en que se ha producido la corrupción de datos. Puede también utilizar esta opción para recuperar una base de datos de la que se hayan eliminado datos de forma accidental. La base de datos se restaura hasta un punto en el tiempo justo antes de que se borraran los datos, pero puesto que este proceso ocasiona la pérdida de todas las transacciones que se hayan realizado después de este punto, deberá utilizarlo con precaución.

Cadenas del registro de transacciones.

Cuando se crea una base de datos, el LSN comienza en 1 y se va incrementando hasta el infinito. Mientras la base de datos no cambie a un modelo de recuperación simple o no se ejecute la instrucción BACKUP LOG.. . WITH TRUNCATE ONLY, las copias de seguridad del registro de transacciones ejecutadas contra la base de datos forman una cadena continua hasta el momento de la creación de la base de datos.

La cadena del registro de transacciones cruza todas las copias de seguridad completas, diferenciales y de grupo de archivos que se hayan realizado. Mientras conserve todas las copias de seguridad y las subsiguientes copias de seguridad del registro de transacciones, podrá recuperar una base de datos hasta un punto en el tiempo iniciando cualquier copia de seguridad completa y posteriormente aplicando todas las copias de seguridad del registro de transacciones subsiguientes a la misma.

En casos extremos, se han llegado a recuperar bases de datos mediante la restauración de una copia de seguridad completa creada años antes y posteriormente restaurando las copias de seguridad del registro de transacciones

creadas en los años posteriores.

A continuación se muestran dos secuencias de restauración diferentes:

-- Secuencia para restaurar usando un respaldo completo, diferencial y transaccional.Primero el completo

RESTORE DATABASE AdventureWorks FILEGROUP = 'FG1' FROM DISK =

'C:\TEST\AWFG1. BAK’ WITH NORECOVERY

--Diferencial

RESTORE DATABASE AdventureWorks FROM DISK = 'C:\TEST\FG1DFF1.BAK'

WITH NO RECOVERY

- -Transaccional

RESTORE LOG Adventureworks FROM DISK = 'C:\TEST\AW2.TRN' WITH RECOVERY

-- Secuencia de restauracion usando respaldos completes y de archivos transaccionales multiples.

--Completo

RESTORE DATABASE AdventureWorks FILEGROUP = 'FG1' FROM DISK =

'C:\TEST\AWFG1.BAK' WITH NORECOVERY

- -Transaction log

RESTORE LOG AdventureWorks FROM DISK = 'C:\TEST\AWl.TRN' WITH NORECOVERY

RESTORE LOG AdventureWorks FROM DISK 'C:\TEST\AW2.TRN' WITH RECOVERY

Recuperar una base de datos hasta un punto en el tiempo en caso de desastre.

La recuperación de bases de datos sin que se produzca pérdida de datos resultaría una tarea mucho más fácil silos problemas únicamente ocurrieran una vez que la copia de seguridad se ha creado y antes de que la aplicación lleve a cabo transacciones adicionales. Pero este no suele ser el caso. En cualquier desastre, tendremos siempre transacciones en el registro para las que no existe una copia de seguridad. Por esta razón, el primer paso a seguir en toda operación de recuperación es el de escribir una instrucción BACKUP LOG al final. Mediante este proceso, capturaremos todas las transacciones para las que no exista una copia de

seguridad; tradicionalmente, conocemos este proceso como copia de seguridad de la cola del registro. Dado que es posible emitir una instrucción BACKUP LOG contra una base de datos aún cuando todos los archivos de datos (incluido el archivo primario) ya no está disponible, la única excusa para no realizar una copia de seguridad de la cola del registro es no hacer la copia cuando el registro de transacciones ha dejado de existir.

La copia de seguridad de la cola del registro se convierte entonces en el registro de transacciones final que aplicaremos en el proceso de restauración, permitiendo que la base de datos pueda ser recuperada sin que se hayaperdido ningún dato.

**La restauración parcial.**

SQL Server 2012 permite restaurar una base de datos de forma parcial mientras el resto de la base de datos sigue siendo accesible. Mientras la consulta que está siendo ejecutada no solicite los datos dentro del grupo o grupo de archivos que se están restaurando, los usuarios no tienen por qué saber que la operación se está llevando a cabo.

Para realizar una restauración parcial, aprovecharemos el hecho de que cada grupo de archivos (salvo el grupo de archivos primario) tiene un estado independiente del estado en el que se encuentre la base de datos. Las restauraciones parciales siempre se llevan a cabo utilizando copias de seguridad de grupos de archivos.

Restaurar la granularidad.

Dependiendo de cómo se haya construido ¡a base de datos, la restauración de un grupo de archivos puede afectar a múltiples tablas, a una única tabla, o en el caso de las particiones, a una porción de una tabla.

Después de que los grupos de archivo se hayan restaurado a la base de datos, puede aplicar las copias de seguridad diferenciales yio del registro de transacciones para actualizar la base de datos de acuerdo con los demás grupos de archivo.

No es posible recuperar una porción de una base de datos a un punto específico en el tiempo, porque todos los grupos de archivo dentro de la base de datos deben avanzar hasta el LSN actual para permitir una operación de escritura en un grupo de archivos determinado.

**Restaurar una página corrupta.**

Si bien no es frecuente, puede darse el caso de que una o más páginas dentro de una tabla se corrompan. En versiones anteriores de SQL Server, esta corrupción daba lugar a un error grave que podía causar la desconexión de toda la base de datos. Arreglar el error dependía de la página que se hubiera corrompido. Si la corrupción de la página se había producido en una página del índice, el índice podía eliminarse y volverse a crear. Sin embargo, si la corrupción ocurría en los datos, era necesario restaurar una copia de seguridad, lo que hacía que la base de datos estuviera desconectada durante todo el proceso de restauración.

SQL Server 2012 ofrece un método alternativo para llevar a cabo este proceso. Se trata de la opción PAGE\_VERIFY\_CHECKSUM. Tras habilitar este modo de verificación en la base de datos, cualquier página que se corrompa es registrada y puesta en cuarentena, en un proceso conocido como “cuarentena de página corrupta”. Para llevar a acabo la verificación, ejecute la siguiente instrucción:

ALTER DATABASE <database name> SET PAGE\_VERIFY CHECKSUM

Esta opción está desactivada de forma predeterminada, ya que consume recursos para operaciones de lectura y escritura en cualquier página de la base de datos. Si se activa, cada vez que sea necesario escribir o leer datos de una página, SQL Server calculará la suma total para esa página. Si la operaciÓn de suma de comprobación no coincide con la suma de comprobación almacenada con anterioridad en la página, eso quiere decir que la página está corrupta. Esta desconexión hace que se produzca un mensaje de error y que la transacción que ha encontrado la página corrupta se revierta. Posteriormente, la página se registra en la tabla **suspect\_pages** de la base de datos **msdb**.

Para arreglar este problema, puede restaurar una página individual a partir de una copia de seguridad. A continuación se muestra un ejemplo de un conjunto de instrucciones que permiten restaurar una página corrupta:

--Respaldar el fin del archivo de transacciones.

BACKUP LOG PUBS TO DISK='C:\HA\DEMO\BACKUP\PUBS1.TRN' WITH INIT

GO

--Respaldar una pagina corrupta de un respaldo reciente

--Nota: Este comando require que no haya usuarios conectados a la base de datos

USE MASTER

GO

RESTORE DATABASE PUBS PAGE = '1:88'

WITH RECOVERY

GO

- -Se aplican archivos de transacciones adicionales para mover la pagina hacia adelante

-- Aplicamos el fin del archivo de transacciones para traer a la base de datos a un cierto punto en el tiempo

USE MASTER

GO RESTORE LOG PUBS

FROM DISK = 'C:\HA\DEMO\BACKUP\PUBS1.TRN'

WITH RECOVERY

GO

**Restaurar con errores de medios.**

Uno de los problemas más difíciles de superar durante la restauración es el de encontrarnos con medios que han resultado dañados. En anteriores versiones de SQL Server, un medio dañado solía empeorar una situación ya de por sí negativa. El daño que sufre un medio de copia de seguridad no suele detectarse antes de que se inicie la copia; cuando la restauración da comienzo, borra todo lo que existiera con anterioridad en la base de datos. Si la operación de restauración se interrumpiera, tendríamos una base de datos invalidada por completo.

Desgraciadamente, esto es lo que ocurre cuando los medios de copia de seguridad están dañados.

SQL Server 2012 ofrece una opción para la instrucción RESTORE, que permite a SQL Server saltar los sectores dañados de un medio y completar la operación de restauración. La opción WITH CONTINUE\_AFTER\_ERROR salta los sectores dañados y permite la restauración de todas las partes que puedenleerse en un medio concreto.

Si bien la operación de restauración puede completarse, no está garantizado que la base de datos esté operativa, ni tan siquiera que contenga datos. Cuando se completa una operación RESTORE en la que se han producido errores en los medios, la base de datos queda en modo de emergencia: puede conectarse con la base datos y ejecutar declaraciones SELECT, pero no pueden hacerse cambios sobre los datos. Si decidimos que la base de datos está intacta y operativa, podemos cambiar el estado para permitir operaciones normales. En el peor de los casos, cualquier dato intacto podrá ser extraído de la base de datos. Esta, evidentemente, no es una solución ideal, pero es mejor que nada.

**Validar una copia de seguridad.**

Tras haber creado varias copias de seguridad, ¿cómo sabemos cuáles de esas copias son utilizables? La única forma de garantizar que una copia de seguridad es utilizable es restaurarla y verificar todos los datos. El proceso puede consumir una gran cantidad de tiempo y no suele ser práctico. No obstante, SQL Server nos permite verificar la integridad de una copia de seguridad. Si bien este método no es igual que restaurar una base de datos, nos ofrece la posibilidad de comprobar con gran exactitud la integridad de una copia de datos.

Utilice la siguiente instrucción para verificar la integridad de una copia de seguridad:

RESTORE VERIFYONLY FROM <backup\_device> ( , . . . n )

Al ejecutar esta instrucción, SQL Server comprueba el encabezamiento del medio para asegurarse de que está intacto. Posteriormente, verifica la suma de verificación de la copia de seguridad, lee las cadenas internas de la página, y vuelve a calcular la suma de verificación de la copia de seguridad para comparar.

Se llevan a cabo entonces una serie de comprobaciones para asegurarse de que la copia de seguridad está intacta. No obstante, SQL Server no comprueba la estructura de datos de la copia de seguridad.

Versiones anteriores y verificación de la copia de seguridad.

Las versiones anteriores de SQL Server contaban también con una instrucción RESTORE VERIFYONLY que comprobaba el encabezamiento del medio y posteriormente devolvía un mensaje de confirmación de éxito o error de la transacción. Siempre que el encabezamiento estuviera intacto, se obtenía un mensaje de confirmación de éxito. A la hora de la verdad, este comportamiento hacía de la instrucción algo inservible en versiones anteriores, puesto que la instrucción no realizaba ninguna comprobación útil. Así, solía recomendarse no utilizar esta instrucción. Pero ahora, SQL Server 2012 realiza todas las comprobaciones necesarias, de modo que cada vez que realice una copia de seguridad, debería ejecutar esta instrucción.

Práctica: Restaurar la base de datos AdventureWorks.

Utilizando las copias de seguridad anteriores, restauraremos la base de datos AdventureWorks a un momento actual en el tiempo.

1. Inicie SSMS, conecte con su instancia SQL Server y abra una nueva ventana de consultas.

2. Cambie el contexto a la base de datos master.

3. Haga una copia de seguridad de la cola del registro mediante la siguiente instrucción:

BACKUP LOG AdventureWorks TO DISK = 'C:\ADSQL\TEST2\AWTAIL.TRW'

4. Ejecute las siguientes instrucciones RESTORE para restaurar la base de datos AdventureWorks.

RESTORE DATABASE AdventureWorks FROM DISK = 'C:\ADSQL\TEST2\AW.BAK' WITH NORECOVERY

RESTORE DATABASE AdventureWorks FROM DISK = 'C:\ ADSQL\TEST4\AWDIFF1.BAK'

WITH NORECOVERY

RESTORE LOG AdventureWorks FROM DISK = 'C:\ ADSQL\TEST2\AW2.TRN' WITH NORECOVERY

RESTORE LOG AdventureWorks FROM DISK = 'C:\ ADSQL\TEST2\AWTAIL.TRN' WITH RECOVERY

5. Si ya ha realizado los ejercicios anteriores, la base de datos AdventureWorks debería tener un único grupo de archivos y la tabla dbo. t1 no existirá.

6. Si aún no ha realizado los ejercicios de la copia de seguridad anteriores, deberá verificar que la base de datos Adventure Works contiene todos los cambios realizados.

**Tema 14: Mover una base de datos.**

En ocasiones, las bases de datos deben ser trasladadas, bien dentro del mismo servidor, bien entre servidores. SQL Server ofrece tres mecanismos diferentes para llevar a cabo esta tarea. El primero, consistente en crear una copia de seguridad de la base de datos para su restauración posterior, lo que se ha explicado anteriormente. Ahora veremos los dos métodos restantes: el método de separar/adjuntar y la utilización del Asistente de Copia de la Base de datos, que permite utilizar el método de separar/adjuntar o el Programa de Administración de Objetos (SMO) de SQL.

**Mover una base de datos mediante el método separar/adjuntar.**

Puede desmontar una base de datos de SQL Server separándola. Este proceso elimina todas las entradas de las tablas del sistema para la base de datos, haciendo que la base de datos no esté accesible en la instancia de SQL Server. Aún cuando la base de datos sea inaccesible, los archivos que contienen los objetos y datos de la misma siguen existiendo en el sistema operativo, en la misma localización en la que fueron creados. Después de separarlos, es posible copiar estos archivos a un lugar determinado de la red, puesto que SQL Server ya no necesita acceder a ellos.

Para que la base de datos pueda volver a ser accesible, únicamente deberá adjuntarla. Este proceso añade una entrada en las tablas del sistema para la base de datos; posteriormente, SQL Server habilitará el acceso a la base de datos.

Utilice la siguiente instrucción para realizar una operación de separación:

EXEC sp\_detach\_db 'AdventureWorks', 'true'

Utilice la siguiente instrucción para adjuntar una base de datos:

CREATE DATABASE AdventureWorks ON

(FILENAME = 'C:\TEST5\AdventureWorks\_Data .mdf’)

(FILENAME 'C:\TEST5\AdventureWorks\_Log.ldf’)

FOR ATTACH

Rendimiento del método separar/adjuntar.

La operación de separación únicamente requiere que SQL cierre los archivos y elimine una entrada en las tablas del sistema; la operación de adjuntar únicamente requiere que SQL abra tos archivos y cree una entrada en las tablas del sistema. Cada operación necesita, a su vez, de entre 1 y 2 segundos para ser completada.

**Utilizar el Asistente de Copia de la Base de Datos.**

Asegúrese de que ha instalado SQL Server Integration Services. El Asistente para copiar bases de datos se ejecuta dentro de SQL Server Integration Services (SSIS). Para utilizar este asistente, deberá tener SSIS instalado. La cuenta proxy bajo la cual se ejecuta el paquete deberá ser también miembro de la función **sysadmin**, tanto en la instancia de origen como en la de destino.

El Asistente para la copia de bases de datos permite copiar todos los objetos dentro de una base de datos a otra instancia, o a otra base de datos dentro de la misma instancia. Este proceso copia todos los objetos, usuarios, esquemas y permisos de la base de datos, creando un duplicado exacto de la misma. Deberá copiar los objetos de nivel de servidor por separado.

Para acceder al asistente, haga clic con el botón derecho en una base de datos y seleccione Tareas>Copiar base de datos. Cuando aparezca la pantalla de bien venida, haga clic en Siguiente. Seleccione el servidor de origen desde el que se copiará la base de datos y haga clic en Siguiente. Seleccione el servidor de destino y haga clic en Siguiente.

Seleccione el método separar/adjuntar, o SMO. Si selecciona el primer método, SSIS separará la base de datos, la copiará en el destino indicado y posteriormente la adjuntará. Este proceso es exactamente el mismo que se ha descrito con anterioridad en este capítulo. Si selecciona SMO, la base de datos permanecerá en línea y será accesible para los usuarios, al tiempo que los API de escritura se utilizarán para generar scripts que recreen todos los objetos en el destino. Se moverán todos los datos. La copia de una base de datos mediante SMO es un proceso considerablemente más lento que los métodos adjuntar/separar o hacer copia de seguridad/restaurar.

Haga clic en Siguiente para seleccionar las bases de datos que desea mover ocopia. Si especifica la opción “mover”, la base de datos se creará en el destino y posteriormente será eliminada en el origen. Si especifica la opción “copiar”, la base de datos se creará tanto en origen como en destino. Haga clic en Siguiente para desplegar la página de Configuración de la base de datos de destino.

Puede especificar el nombre de una base de datos de destino junto con la localización de los archivos. Si la base de datos ya existe, podrá especificar que la copia elimine o sobrescriba la base de datos existente. Haga clic en Siguiente para especificar las opciones del paquete SSIS, tales como el nombre del paquete y las opciones de registros. Haga clic en Siguiente para especificar si desea que el paquete se ejecute de forma inmediata, o si prefiere que se ejecute a posterior, así como para especificar en qué cuenta proxy deberá hacerlo. Haga clic en Siguiente para verificar las opciones seleccionadas. Para completar el asistente, haga clic en Finalizar.

Práctica: Utilizar el método separar/adjuntar para mover una base de datos.

En este ejercicio, separaremos la base de datos AdventureWorks, la copiaremos a una nueva localización y después la adjuntaremos utilizando SSMS.

Es posible que los nombres de archivo de su base de datos Adventureworks varíen. También es posible que haya añadido grupos de archivo a la base de datos en los ejercicios anteriores. En esta práctica, se utilizarán nombres de archivo genéricos: no olvide realizar los ajustes necesarios para trabajar en su entorno.

1. Inicie SSMS y conecte con la instancia de SQL Server.

2. Asegúrese de que no se ha creado ninguna Conexión con la base de datos Adventure Works.

3. Dentro del Explorador de objetos, haga clic con el botón derecho en la base de datos AdventureWorks y seleccione Tareas>Separar.

4. Haga clic en Aceptar.

5. Abra el explorador de Windows y copie los archivos .mdf, .ndf y .ldf de AdventureWorks en el directorio c:\AdSQL\test5 creado anteriormente.

6. Haga clic con el botón derecho en el nodo del explorador de objetos y seleccione Adjuntar.

7. Haga clic en Añadir y seleccione el archivo .mdfAdvetureworks. Haga clic en Aceptar.

8. Verifique que la base de datos Adventure Works aparece en el explorador de objetos y que puede acceder a la base de datos, leerlos y escribir datos en la base de datos.