Microsoft SQL SERVER 2008 Manual de Programación

Audiencia

Este curso está diseñado para profesionales de IT que quieren convertirse en expertos en SQL Server 2008 para implementar y administrar bases de datos. También para programadores que desarrollan en otros productos y quieren conocer como implementar bases de datos SQL Server.

Pre-Requisitos

Tener conocimientos básicos de bases de datos relacionales, lenguaje SQL y XML.

Nota: Si usted no está seguro de sus conocimientos lo invitamos a tomar los cursos gratuitos dentro de Microsoft Virtual Academy (MVA).

Al finalizar el curso

Después de completar este curso los alumnos serán capaces de:

- Escribir sentencias básicas usando el lenguaje SQL
- Crear y trabajar con bases de datos
- Crear tablas y tipos de datos
- Planear, crear y optimizar índices
- Implementar integridad de datos
- Usar características de XML incluidas en SQL Server
- Implementar vistas
- Implementar procedimientos almacenados
- Implementar funciones
- Implementar código administrado dentro de la base de datos
- Manejar transacciones y bloqueos
- Instalar SQL Server 2008 y Configurarlo
- Planear y Generar Copias de Seguridad.
- Restaurar Copias de Seguridad
- Configurar la seguridad de SQL Server 2008
- Cifrar datos
- Monitorear bases de datos SQL Server 2008
- Crear Planes de Mantenimiento
- Trabajar con el Agente de SQL Server para crear trabajos, programaciones, alertas, etc.
- Implementar Replicación
- Implementar bases de datos de alta disponibilidad

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

Índice:

Módulo 1

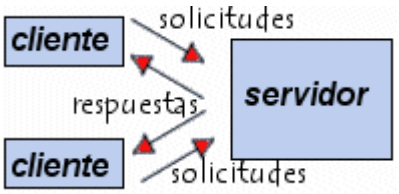
Introducción a Bases de Datos

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Conceptos de Bases de Datos

Definición
Es un conjunto de información relacionada que se encuentra agrupada o estructurada. Un archivo por sí mismo no constituye una base de datos, sino más bien la forma en que está organizada la información es la que da origen a la base de datos.
Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulan ese conjunto de datos.

Arquitectura Cliente/Servidor
Los sistemas cliente/servidor están construidos de tal modo que la base de datos puede residir en un equipo central, llamado servidor y ser compartida entre varios usuarios. Los usuarios tienen acceso al servidor a través de una aplicación de cliente o de servidor:
En los sistemas cliente/servidor grandes, miles de usuarios pueden estar conectados con una instalación de SQL Server al mismo tiempo.
SQL Server tiene una protección completa para dichos entornos, con barreras de seguridad que impiden problemas como tener varios usuarios intentando actualizar el mismo elemento de datos a la vez. SQL Server también asigna eficazmente los recursos disponibles entre los distintos usuarios, como la memoria, el ancho de banda de la red y la E/S de disco.



Modelos

- **Bases de datos jerárquicas**: Éstas son bases de datos que, como su nombre indica, almacenan su información en una estructura jerárquica. En este modelo los datos se organizan en una forma similar a un árbol (visto al revés), en donde un nodo padre de información puede tener varios hijos. El nodo que no tiene padres es llamado raíz, y a los nodos que no tienen hijos se los conoce como hojas. Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos permitiendo crear estructuras estables y de gran rendimiento. Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos.
- **Base de datos de red:** Éste es un modelo ligeramente distinto del jerárquico; su diferencia fundamental es la modificación del concepto de nodo: se permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico). Fue una gran mejora con respecto al modelo jerárquico, ya que ofrecía una solución eficiente al problema de redundancia de datos; pero, aun así, la dificultad que significa administrar la información en una base de datos de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales.
- **Base de datos relacional**: Éste es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Este es el tipo de **Microsoft SQL Server**.

- **Bases de datos orientadas a objetos**: Este modelo, bastante reciente, y propio de los modelos informáticos orientados a objetos, trata de almacenar en la base de datos los objetos completos (estado y comportamiento).
- **Bases de datos documentales**: Permiten la indexación a texto completo, y en líneas generales realizar búsquedas más potentes.
- **Base de datos deductivas**: Un sistema de base de datos deductivas, es un sistema de base de datos pero con la diferencia que permite hacer deducciones a través de inferencias. Se basa principalmente en reglas y hechos que son almacenados en la base de datos. También las bases de datos deductivas son llamadas base de datos lógicas, a raíz de que se basan en lógica matemática.
- **Bases de datos distribuidas**: La base de datos está almacenada en varias computadoras conectadas en red. Surgen debido a la existencia física de organismos descentralizados. Esto les da la capacidad de unir las bases de datos de cada localidad y acceder así, por ejemplo a distintas universidades, sucursales de tiendas, etcétera.

Sistema de Gestión de Base de Datos (SGBD)

Los sistemas de gestión de base de datos son un tipo de software muy específico, dedicado a servir de interfaz entre los datos, el usuario y las aplicaciones que la utilizan. El propósito general de los sistemas de gestión de base de datos es el de manejar de forma clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante, para un buen manejo de datos.

Existen distintos objetivos que deben cumplir los SGBD:

- **Abstracción de la información**. Los SGBD ahorran a los usuarios detalles acerca del almacenamiento físico de los datos. Da lo mismo si una base de datos ocupa uno o cientos de archivos, este hecho se hace transparente al usuario.
- **Independencia**. La independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.
- **Consistencia**. En aquellos casos en los que no se ha logrado eliminar la redundancia, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea. En los SGBD existen herramientas que facilitan la programación de este tipo de condiciones.
- **Seguridad**. La información almacenada en una base de datos puede llegar a tener un gran valor. Los SGBD deben garantizar que esta información se encuentra segura frente a usuarios malintencionados, que intenten leer información privilegiada; frente a ataques que deseen manipular o destruir la información; o simplemente ante las torpezas de algún usuario autorizado pero descuidado. Normalmente, los SGBD disponen de un complejo sistema de permisos a usuarios y grupos de usuarios, que permiten otorgar diversas categorías de permisos.
- **Integridad**. Se trata de adoptar las medidas necesarias para garantizar la validez de los datos almacenados. Es decir, se trata de proteger los datos ante fallos de hardware, datos introducidos por usuarios descuidados, o cualquier otra circunstancia capaz de corromper la información almacenada. Los SGBD proveen mecanismos para garantizar la recuperación de la base de datos hasta un estado consistente conocido en forma automática.
- **Respaldo**. Los SGBD deben proporcionar una forma eficiente de realizar copias de respaldo de la información almacenada en ellos, y de restaurar a partir de estas copias los datos que se hayan podido perder.
- **Control de la concurrencia**. En la mayoría de entornos, lo más habitual es que sean muchas las personas que acceden a una base de datos, bien para recuperar información, bien para almacenarla. Y es también frecuente que dichos accesos se

realicen de forma simultánea. Así pues, un SGBD debe controlar este acceso concurrente a la información, que podría derivar en inconsistencias.

- **Manejo de Transacciones**. Una Transacción es un programa que se ejecuta como una sola operación. Esto quiere decir que el estado luego de una ejecución en la que se produce una falla es el mismo que se obtendría si el programa no se hubiera ejecutado. Los SGBD proveen mecanismos para programar las modificaciones de los datos de una forma mucho más simple que si no se dispusiera de ellos.
- **Tiempo de respuesta**. Lógicamente, es deseable minimizar el tiempo que el SGBD tarda en darnos la información solicitada y en almacenar los cambios realizados.

Las **ventajas** de su uso son:

- Proveen facilidades para la manipulación de grandes volúmenes de datos. Simplificando la programación de chequeos de consistencia, manejando las políticas de respaldo adecuadas que garantizan que los cambios de la base serán siempre consistentes sin importar si hay errores en el disco, o hay muchos usuarios accediendo simultáneamente a los mismos datos, o se ejecutaron programas que no terminaros su trabajo correctamente, etc., permitiendo realizar modificaciones en la organización de los datos con un impacto mínimo en el código de los programas y permitiendo implementar un manejo centralizado de la seguridad de la información (acceso a usuarios autorizados), protección de información, de modificaciones, inclusiones, consulta.
- Las facilidades anteriores bajan drásticamente los tiempos de desarrollo y aumentan la calidad del sistema desarrollado si son bien explotados por los desarrolladores.
- Usualmente, proveen interfaces y lenguajes de consulta que simplifican la recuperación de los datos.

Las **desventajas** son:

- Generalmente es necesario disponer de una o más personas que administren la base de datos, en la misma forma en que suele ser necesario en instalaciones de cierto porte disponer de una o más personas que administren los sistemas operativos. Esto puede llegar a incrementar los costos de operación en una empresa. Sin embargo hay que balancear este aspecto con la calidad y confiabilidad del sistema que se obtiene.
- Si se tienen muy pocos datos que son usados por un único usuario por vez y no hay que realizar consultas complejas sobre los datos, entonces es posible que sea mejor usar una planilla de cálculo.

Bases de Datos OLTP (On Line Transactional Processing)

Los sistemas OLTP son bases de datos orientadas al procesamiento de transacciones. El proceso transaccional es típico de las bases de datos operacionales. El acceso a los datos está optimizado para tareas frecuentes de lectura y escritura

Bases de Datos OLAP (On Line Analytical Processing)

Los sistemas OLAP son bases de datos orientadas al procesamiento analítico. Este análisis suele implicar, generalmente, la lectura de grandes cantidades de datos para llegar a extraer algún tipo de información útil: tendencias de ventas, patrones de comportamiento de los consumidores, elaboración de informes complejos… etc. El acceso a los datos suele ser de sólo lectura. La acción más común es la consulta, con muy pocas inserciones, actualizaciones o eliminaciones

Objetos de la base de datos

- **Tablas**: En una base de datos la información se organiza en tablas, que son filas y columnas similares a las de los libros contables o a las de las hojas de cálculo. Cada fila de la tabla recibe también el nombre de registro y cada columna se denomina también campo.
- **Tipos de datos**: Los objetos que contienen datos tienen asociado un tipo de datos que define la clase de datos, por ejemplo, carácter, entero o binario, que puede contener el objeto.
- **Vistas**: Una vista es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla real, una vista consta de un conjunto de columnas y filas de datos con un nombre.
- **Índices**: Al igual que el índice de un libro, el índice de una base de datos permite encontrar rápidamente información específica en una tabla o vista indexada. Un índice contiene claves generadas a partir de una o varias columnas de la tabla o la vista y punteros que asignan la ubicación de almacenamiento de los datos.
- **Procedimientos Almacenados**: Conjunto de instrucciones escritas en lenguaje SQL para ser ejecutadas. Aceptan parámetros.
- **Funciones**: Rutinas que aceptan parámetros, realizan una acción, como un cálculo complejo, y devuelven el resultado de esa acción como un valor. Similares a las funciones de los lenguajes de programación

Transacciones

Una transacción es una secuencia de operaciones realizadas como una sola unidad lógica de trabajo. Una unidad lógica de trabajo debe exhibir cuatro propiedades, conocidas como propiedades de atomicidad, coherencia, aislamiento y durabilidad (ACID), para ser calificada como transacción.

- **Atomicidad**. Una transacción debe ser una unidad atómica de trabajo, tanto si se realizan todas sus modificaciones en los datos, como si no se realiza ninguna de ellas.
- **Coherencia**. Cuando finaliza, una transacción debe dejar todos los datos en un estado coherente. En una base de datos relacional, se deben aplicar todas las reglas a las modificaciones de la transacción para mantener la integridad de todos los datos. Todas las estructuras internas de datos, como índices de árbol B o listas doblemente vinculadas, deben estar correctas al final de la transacción.
- **Aislamiento**. Las modificaciones realizadas por transacciones simultáneas se deben aislar de las modificaciones llevadas a cabo por otras transacciones simultáneas. Una transacción reconoce los datos en el estado en que estaban antes de que otra transacción simultánea los modificara o después de que la segunda transacción haya concluido, pero no reconoce un estado intermedio.
- **Durabilidad**. Una vez concluida una transacción, sus efectos son permanentes en el sistema. Las modificaciones persisten aún en el caso de producirse un error del sistema.

2- Diseño de Base de Datos

El proceso de diseño de una base de datos se guía por algunos principios. El primero de ellos es que se debe evitar la información duplicada o, lo que es lo mismo, los datos redundantes, porque malgastan el espacio y aumentan la probabilidad de que se produzcan errores e incoherencias. El segundo principio es que es importante que la información sea correcta y completa. Si la base de datos contiene información incorrecta, los informes que recogen información de la base de datos contendrán también información incorrecta y, por lo tanto, las decisiones que se tome a partir de esos informes estarán mal fundamentadas.

El proceso de diseño

El proceso de diseño consta de los siguientes pasos:

- Determinar la finalidad de la base de datos.
- Buscar y organizar la información necesaria.
- Dividir la información en tablas.
- Convertir los elementos de información en columnas: Decidir qué información desea almacenar en cada tabla. Cada elemento se convertirá en un campo y se mostrará como una columna en la tabla.
- Especificar claves principales: La clave principal es una columna que se utiliza para identificar inequívocamente cada fila.
- Definir relaciones entre las tablas: Examinar cada tabla y decidir cómo se relacionan los datos de una tabla con las demás tablas. Agregue campos a las tablas o cree nuevas tablas para clarificar las relaciones según sea necesario.
- Ajustar el diseño: Analizar el diseño para detectar errores. Crear las tablas y agregar algunos registros con datos de ejemplo. Comprobar si puede obtener los resultados previstos de las tablas. Realizar los ajustes necesarios en el diseño.
- Aplicar las reglas de normalización: Aplicar reglas de normalización de los datos para comprobar si las tablas están estructuradas correctamente. Realizar los ajustes necesarios en las tablas.

Normalización

El proceso de normalización de bases de datos consiste en aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo entidad-relación al modelo relacional.

Las bases de datos relacionales se normalizan para:

- Evitar la redundancia de los datos.
- Evitar problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

Las normas son:

- **Primera forma normal**: Establece que en cada columna solo puede haber un valor y no una lista de valores o sea que las columnas repetidas deben eliminarse y colocarse en tablas separadas. Por ejemplo, en una tabla de Facturas no puede haber un campo “Artículo” en el que se incluya más de un artículo, para solucionar esto se deberá crear una tabla separada para especificar todos los articulos asociados a dichas facturas.
- **Segunda forma normal**: Establece que se cumpla la primera forma normal y además que cada columna que no sea clave dependa por completo de toda la clave principal y no sólo de parte de la clave. Esta regla se aplica cuando existe una clave principal formada por varias columnas. Suponga, por ejemplo, que existe una tabla con varias columnas de las cuales “Código de Pedido” y “Código de Producto” forman la clave

principal. En dicha tabla no podría incluirse el campo “Nombre de Producto” ya que el mismo depende solo de una parte de la clave principal.

- **Tercera forma normal**: Establece que se cumplan las dos primeras formas normales y además que cada columna que no sea clave dependa solo de la clave principal completa y no de columnas que no sean clave. O dicho de otra forma: cada columna que no sea clave debe depender de la clave principal y nada más que de la clave principal. Por ejemplo, considere una tabla de Facturas cuya clave principal es el “Número de Factura” y contiene un campo “Cliente”, no clave, que es el código del cliente asociado a esa factura. Esta tabla no podría contener el campo “Razón Social del Cliente” ya que este dato depende del campo “Cliente” que no es clave de la tabla.

Desnormalización

Una base de datos normalizada impide las dependencias funcionales de los datos para que el proceso de actualización de la base de datos sea fácil y eficiente. Sin embargo, la realización de consultas en la base de datos puede requerir la combinación de varias tablas para unir la información. A medida que el número de tablas combinadas crece, el tiempo de ejecución de la consulta aumenta considerablemente. Por este motivo, el uso de una base de datos normalizada no es siempre la mejor alternativa.

Una base de datos con la medida justa de desnormalización reduce el número de tablas que deben combinarse sin dificultar en exceso el proceso de actualización. Suele ser la solución acertada, sobre todo en base de datos grandes.

Por ejemplo en una aplicación de compra y ventas de productos los movimientos de los artículos se guardan en tablas detallando cada uno de sus movimientos, para el caso necesario de poder hacer un seguimiento de los mismos. Cada vez que se quiere consultar el stock de un determinado artículo habría que calcularlo en base a todos sus movimientos, lo cual podrá llevar a la aplicación a una respuesta muy lenta para conseguir este valor sobre todo en base de datos grandes. Guardando el stock actual en una tabla asociado a cada artículo y recalculando dicho valor con cada movimiento mostrar el stock al momento solo requerirá de la lectura de un registro.

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

Módulo 2

Introducción al Lenguaje SQL

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Introducción

Definición
El lenguaje de gestión de bases de datos más conocido en la actualidad es el SQL (Structured Query Language), que es un lenguaje estándar internacional, comúnmente aceptado por los fabricantes de generadores de bases de datos. SQL es un lenguaje de consulta para los sistemas de bases de datos relacionales, pero que no posee la potencia de los lenguajes de programación.

Transact-SQL
Transact-SQL es el lenguaje de programación que proporciona SQL Server para ampliar el lenguaje SQL con los elementos característicos de los lenguajes de programación: variables, sentencias de control de flujo, bucles, etc.
Cuando se desea realizar una aplicación completa para el manejo de una base de datos relacional, resulta necesario utilizar alguna herramienta que soporte la capacidad de consulta del SQL y la versatilidad de los lenguajes de programación tradicionales. Transact SQL es el lenguaje de programación que proporciona SQL Server para extender el SQL estándar con otro tipo de instrucciones.

Las sentencias de SQL se clasifican según su finalidad dando origen a 3 sub lenguajes:

- **DCL (Data Control Language)**: Lenguaje que incluye órdenes para manejar la seguridad de los datos y de la base de datos. Permite crear roles y establecer permisos. Sentencias: GRANT, REVOKE
- **DDL (Data Definition Language)**: Lenguaje que incluye órdenes para definir, modificar o borrar objetos de la base de datos. Sentencias: CREATE, DROP, ALTER
- **DML (Data Manipulation Language)**: Lenguaje de manipulación de datos que permite actualizar y recuperar los datos almacenados en la base de datos. Este tipo de instrucciones son las que se desarrollaran en este capítulo. Sentencias: SELECT, INSERT, UPDATE, DELETE

2- Lenguaje

Variables
Una variable local de Transact-SQL es un objeto que contiene un valor individual de datos de un tipo específico.
La instrucción **DECLARE** inicializa una variable de Transact-SQL. El nombre debe ser único y tener un único @ como primer carácter. Se debe asignar siempre el tipo de dato de la misma y su longitud si fuera necesaria.

Ejemplo:

```
DECLARE @Codigo int
```

El alcance de una variable es el conjunto de instrucciones Transact-SQL desde las que se puede hacer referencia a la variable. El alcance de una variable se extiende desde el punto en el que se declara hasta el final del lote o procedimiento almacenado en el que se ha declarado.
Para asignar un valor a una variable, use la instrucción **SET**. Éste es el método preferido para asignar un valor a una variable. También se puede asignar un valor a una variable si se hace referencia a ella en la lista de selección de una instrucción **SELECT**.

Ejemplo:

```
SET @Codigo = 1
```

Operadores
- Aritméticos: +, - ,* ,/ ,% (módulo)
- Lógicos: And, Or, Not
- Comparativos: =, <>, >, >=, <, <=
- Concatenación: +

Comentarios
- Para comentario de línea: --
- Para comentario de bloque: /* */

```
-- Comentario de línea
/*
   Comentario de Bloque
*/
```

Control de Flujo
- **BEGIN / END**: Las instrucciones BEGIN y END se usan para agrupar varias instrucciones Transact-SQL en un bloque lógico. Use las instrucciones BEGIN y END en cualquier parte cuando una instrucción de control de flujo deba ejecutar un bloque con dos o más instrucciones Transact-SQL.
- **IF / ELSE**: La instrucción IF se usa para probar una condición.
- **RETURN**: La instrucción RETURN termina incondicionalmente una consulta, procedimiento almacenado o lote. Ninguna de las instrucciones de un procedimiento almacenado o lote que siga a la instrucción RETURN se ejecutará.
- **WHILE / BREAK** o **CONTINUE**: La instrucción WHILE repite una instrucción o bloque de instrucciones mientras la condición especificada siga siendo verdadera. Se suelen utilizar dos instrucciones de Transact-SQL con WHILE: BREAK o CONTINUE. La

instrucción BREAK sale del bucle WHILE más profundo, y la instrucción CONTINUE reinicia un bucle WHILE.

- **CASE**: La función CASE es una expresión especial de Transact-SQL que permite mostrar un valor alternativo dependiendo del valor de una columna o variable.

Ejemplos:

a.

```
IF (@MiError <> 0)
      BEGIN
      PRINT 'Se encontró un error. Los datos no fueron grabados'
      ROLLBACK
      END
ELSE
      BEGIN
      PRINT 'Sin Errores'
      COMMIT
      END
RETURN @MiError
```

b.

```
CASE Nombre
      WHEN 'Recursos Humanos' THEN 'RH'
      WHEN 'Finanzas' THEN 'FI'
      WHEN 'Servicios' THEN 'SE'
      WHEN 'Mantenimiento' THEN 'MA'
END AS Abreviatura
```

c.

```
DECLARE abc CURSOR FOR
SELECT * FROM Purchasing.ShipMethod;
OPEN abc;
FETCH NEXT FROM abc
WHILE (@@FETCH_STATUS = 0)
      FETCH NEXT FROM abc;
CLOSE abc;
DEALLOCATE abc;
```

3- Instrucción SELECT

Definición
Recupera filas de la base de datos y habilita la selección de una o varias filas o columnas de una o varias tablas. La sintaxis completa de la instrucción SELECT es compleja, aunque las cláusulas principales se pueden resumir del modo siguiente:

```
SELECT
[TOP expresión [PERCENT] [ WITH TIES ] ]
<lista_columnas> [ INTO NuevaTabla ]
[ FROM tabla ]
[ WHERE condición ]
[ GROUP BY expresión ]
[ HAVING condición ]
[ ORDER BY expresión [ ASC | DESC ] ]
```

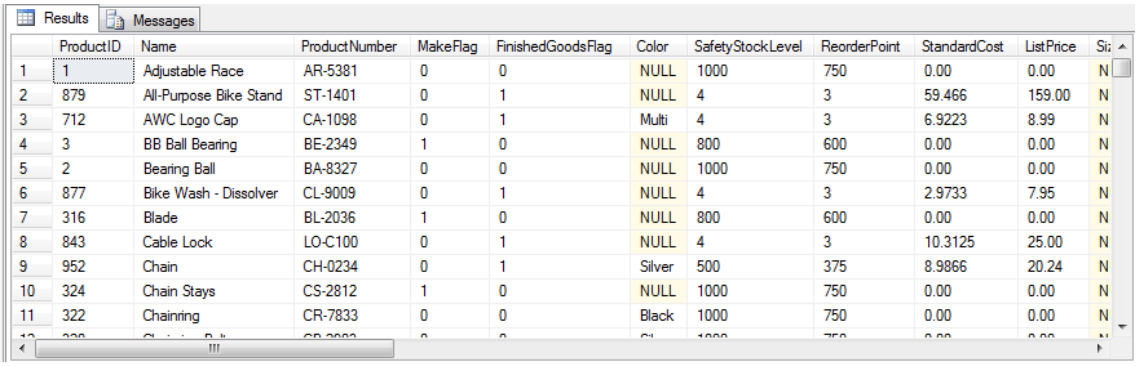
Descripción de las cláusulas

- **TOP**: Especifica que solo se devolverá el primer conjunto de filas del resultado de la consulta. La cantidad de registros puede ser un número o un porcentaje de los mismos. Puede usar una variable.
- **WITH TIES**: Especifica que las filas adicionales se devolverán del conjunto de resultados base con el mismo valor en las columnas ORDER BY que el que aparece en la última de las filas de TOP n (PERCENT). TOP...WITH TIES sólo se puede especificar en instrucciones SELECT y siempre que haya una cláusula ORDER BY especificada.
- **INTO**: Crea una nueva tabla e inserta en ella las filas resultantes de la consulta.
- **FROM**: Especifica las tablas, vistas, tablas derivadas y tablas combinadas que se utilizan en la instrucción. En la instrucción SELECT, la cláusula FROM es necesaria excepto cuando la lista de selección sólo contiene constantes, variables y expresiones aritméticas (sin nombres de columna).
- **WHERE**: Especifica la condición de búsqueda de las filas devueltas por la consulta.
- **GROUP BY**: Agrupa un conjunto de filas seleccionado en un conjunto de filas de resumen de acuerdo con los valores de una o más columnas o expresiones. Se devuelve una fila para cada grupo. Las funciones de agregado de la lista de <selección> de la cláusula SELECT proporcionan información sobre cada grupo en lugar de filas individuales.
- **HAVING**: Especifica una condición de búsqueda para un grupo o agregado. Normalmente, HAVING se utiliza en una cláusula GROUP BY. Cuando no se utiliza GROUP BY, HAVING se comporta como una cláusula WHERE.
- **ORDER BY**: Especifica el orden utilizado en las columnas devueltas en una instrucción SELECT.
- **UNION**: Combina los resultados de dos o más consultas en un solo conjunto de resultados que incluye todas las filas que pertenecen a las consultas de la unión. La operación UNION es distinta de la utilización de combinaciones de columnas de dos tablas.

Ejemplos

a. Devuelve todos los campos de la tabla Product ordenados por el campo Name

```
SELECT * FROM Production.Product ORDER BY Name ASC
```



	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice	Si
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0.00	0.00	N
2	879	All-Purpose Bike Stand	ST-1401	0	1	NULL	4	3	59.466	159.00	N
3	712	AWC Logo Cap	CA-1098	0	1	Multi	4	3	6.9223	8.99	N
4	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0.00	0.00	N
5	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0.00	0.00	N
6	877	Bike Wash - Dissolver	CL-9009	0	1	NULL	4	3	2.9733	7.95	N
7	316	Blade	BL-2036	1	0	NULL	800	600	0.00	0.00	N
8	843	Cable Lock	LO-C100	0	1	NULL	4	3	10.3125	25.00	N
9	952	Chain	CH-0234	0	1	Silver	500	375	8.9866	20.24	N
10	324	Chain Stays	CS-2812	1	0	NULL	1000	750	0.00	0.00	N
11	322	Chainring	CR-7833	0	0	Black	1000	750	0.00	0.00	N

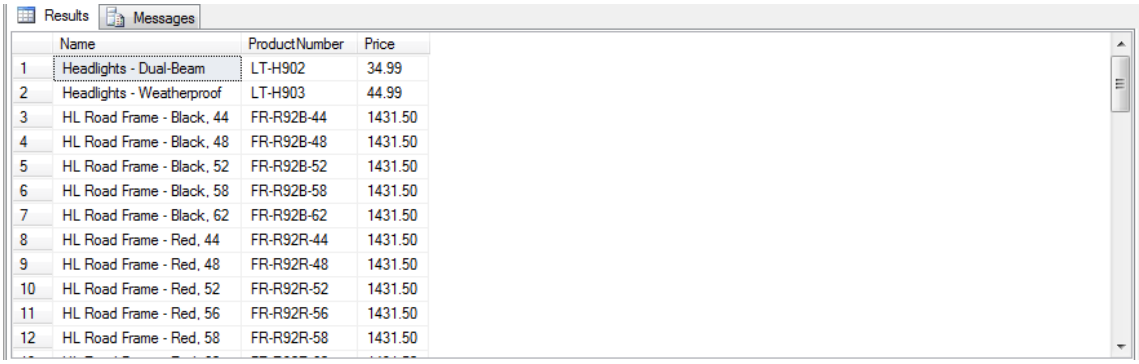
b. Devuelve solo los campos Name, ProductNumber, ListPrice de la misma tabla. El campo ListPrice es renombrado a Price

```
SELECT Name, ProductNumber, ListPrice AS Price
FROM Production.Product ORDER BY Name ASC
```

	Name	ProductNumber	Price
1	Adjustable Race	AR-5381	0.00
2	All-Purpose Bike Stand	ST-1401	159.00
3	AWC Logo Cap	CA-1098	8.99
4	BB Ball Bearing	BE-2349	0.00
5	Bearing Ball	BA-8327	0.00
6	Bike Wash - Dissolver	CL-9009	7.95
7	Blade	BL-2036	0.00
8	Cable Lock	LO-C100	25.00
9	Chain	CH-0234	20.24
10	Chain Stays	CS-2812	0.00
11	Chainring	CR-7833	0.00
12	Chainring Bolts	CB-2903	0.00

c. Sólo devuelve las filas de Product que tienen una línea de productos R y cuyo valor correspondiente a los días para fabricar es inferior a 4.

```
SELECT Name, ProductNumber, ListPrice AS Price
FROM Production.Product
WHERE ProductLine = 'R'  AND DaysToManufacture < 4
ORDER BY Name ASC
```



	Name	ProductNumber	Price
1	Headlights - Dual-Beam	LT-H902	34.99
2	Headlights - Weatherproof	LT-H903	44.99
3	HL Road Frame - Black, 44	FR-R92B-44	1431.50
4	HL Road Frame - Black, 48	FR-R92B-48	1431.50
5	HL Road Frame - Black, 52	FR-R92B-52	1431.50
6	HL Road Frame - Black, 58	FR-R92B-58	1431.50
7	HL Road Frame - Black, 62	FR-R92B-62	1431.50
8	HL Road Frame - Red, 44	FR-R92R-44	1431.50
9	HL Road Frame - Red, 48	FR-R92R-48	1431.50
10	HL Road Frame - Red, 52	FR-R92R-52	1431.50
11	HL Road Frame - Red, 56	FR-R92R-56	1431.50
12	HL Road Frame - Red, 58	FR-R92R-58	1431.50

d. Devuelve el total de cada SalesOrderID de la tabla SalesOrderDetail que exceda $100,000.00.

```
SELECT SalesOrderID, SUM(LineTotal) AS SubTotal
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
HAVING SUM(LineTotal) > 100000.00
ORDER BY SalesOrderID
```

Results | Messages

	SalesOrderID	SubTotal
1	43875	121761.939600
2	43884	115696.331324
3	44518	126198.336168
4	44528	108783.587200
5	44530	104958.806836
6	44795	104111.515642
7	46066	100378.907800
8	46067	101833.419700
9	46607	120182.184984
10	46616	150837.438737
11	46643	109253.425364
12	46645	101336.838234

Cláusula JOIN

Usando la cláusula JOIN se pueden devolver datos de dos o más tablas basándose en relaciones lógicas entre ellas. Indica como SQL Server debería usar los datos de una de las tablas para seleccionar registros en la otra.

- **INNER JOIN**: Combina registros de dos tablas siempre que haya valores coincidentes en un campo común. Puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Éste es el tipo más común de combinación.

  Ejemplo: Devuelven las ventas totales y los descuentos de todos los productos de la tabla Product que tuvieron movimientos. Para eso consulta la tabla SalesOrderDetail.

```
SELECT p.Name AS ProductName, NonDiscountSales=(OrderQty *
UnitPrice), Discounts = ((OrderQty * UnitPrice) *
UnitPriceDiscount)
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail sod
ON p.ProductID = sod.ProductID
ORDER BY ProductName DESC
```

Results | Messages

	ProductName	NonDiscountSales	Discounts
1	Women's Tights, S	359.952	0.00
2	Women's Tights, S	179.976	0.00
3	Women's Tights, S	404.946	0.00
4	Women's Tights, S	89.988	0.00
5	Women's Tights, S	449.94	0.00
6	Women's Tights, S	404.946	0.00
7	Women's Tights, S	478.4362	9.5687
8	Women's Tights, S	224.97	0.00
9	Women's Tights, S	179.976	0.00
10	Women's Tights, S	134.982	0.00
11	Women's Tights, S	179.976	0.00
12	Women's Tights, S	179.976	0.00

- **OUTER JOIN**: **LEFT**: El resultado de esta operación siempre contiene todos los registros de la tabla de la izquierda (la primera tabla que se menciona en la consulta), aun cuando no exista un registro correspondiente en la tabla de la derecha, para uno de la izquierda. **RIGHT**: Similar al anterior considerando que la tabla que se muestra completa es la que se especifica a la derecha. Los valores de la tabla completa que no tengan correspondencia en la segunda tabla presentarán sus valores en nulo.

  Ejemplo: Devuelven las ventas totales y los descuentos de todos los productos de la tabla Product, aunque algún producto no tenga ventas. Para eso consulta la tabla SalesOrderDetail. El resultado es similar al anterior. La diferencia es que este conjunto de registros contendrá la tabla Product completa haya tenido movimientos o no. En el caso de no tener movimientos los datos de ventas estarán en nulo.

```
SELECT p.Name AS ProductName, NonDiscountSales=(OrderQty *
UnitPrice), Discounts = ((OrderQty * UnitPrice) *
UnitPriceDiscount)
FROM Production.Product p
LEFT JOIN Sales.SalesOrderDetail sod
ON p.ProductID = sod.ProductID
ORDER BY ProductName DESC
```

- **CROSS JOIN:** Esta operación presenta los resultados de tabla izquierda y derecha aunque no tengan correspondencia en la otra tabla. La tabla combinada contendrá, entonces, todos los registros de ambas tablas y presentará valores nulos para registros sin pareja.

Ejemplo:

```
SELECT e.BusinessEntityID, c.FirstName, c.LastName,
e.BirthDate, e.Gender,  a.AddressLine1, a.City, a.PostalCode
FROM HumanResources.Employee e
INNER JOIN Person.BusinessEntityAddress ea
ON e.BusinessEntityID=ea.BusinessEntityID
JOIN Person.Address a ON ea.AddressId=a.AddressId
INNER JOIN Person.Person c ON e.BusinessEntityID=c.BusinessEntityID
WHERE e.Gender='M'
ORDER BY c.LastName, c.FirstName
```

Results | Messages

	BusinessEntityID	FirstName	LastName	BirthDate	Gender	AddressLine1	City	PostalCode
1	285	Syed	Abbas	1965-02-11	M	7484 Roundtree Drive	Bothell	98011
2	211	Hazem	Abolrous	1967-11-27	M	5050 Mt. Wilson Way	Kenmore	98028
3	121	Pilar	Ackerman	1962-10-11	M	5407 Cougar Way	Seattle	98104
4	67	Jay	Adams	1966-03-14	M	896 Southdale	Monroe	98272
5	270	François	Ajenstat	1965-06-17	M	1144 Paradise Ct.	Issaquah	98027
6	207	Greg	Alderson	1960-11-18	M	8684 Military East	Bellevue	98004
7	216	Sean	Alexander	1966-04-07	M	7985 Center Street	Renton	98055
8	227	Gary	Altman	1961-03-21	M	2598 Breck Court	Renton	98055
9	217	Zainal	Arifin	1966-03-02	M	342 San Simeon	Issaquah	98027
10	269	Dan	Bacon	1971-07-28	M	1921 Ranch Road	Issaquah	98027
11	41	Bryan	Baker	1963-09-28	M	2275 Valley Blvd.	Monroe	98272
12	262	David	Barber	1954-07-23	M	8967 Hamilton Ave.	Newport Hills	98006

4- Instrucción INSERT

Definición
Agrega una nueva fila o filas a una tabla o vista.

```
INSERT [ INTO ] objeto
{
   [ ( lista_columnas ) ]
   { VALUES ( ( { DEFAULT | NULL | expresión } [ ,...n ] ) [ ,...n ] )
   | tabla_derivada
   | sentencia_ejecutable
   | DEFAULT VALUES
   }
}
```

Descripción de las cláusulas

- **INTO**: Es una palabra clave opcional que se puede utilizar entre INSERT y la tabla de destino.
- **Lista_columnas**: Es una lista de una o más columnas en las que se insertarán los datos. Se debe incluir entre paréntesis y delimitar con comas.
- **VALUES**: Presenta la lista de valores de datos que se van a insertar. Debe haber un valor de datos por cada columna de la lista, si se especifica, o en la tabla. La lista de valores debe ir entre paréntesis. Los valores de la lista VALUES deberán estar en el mismo orden que la lista de columnas. La inserción de más de una fila de valores requiere que la lista VALUES esté en el mismo orden que las columnas de la tabla, para tener un valor en cada columna, o que lista especifique de forma explícita la columna en que la que se almacena cada uno de los valores entrantes. El número máximo de filas que se pueden insertar en una instrucción INSERT única es 1000. Para insertar más de 1000 filas, cree varias instrucciones INSERT, o realice una importación masiva de datos mediante la utilidad BCP o la instrucción BULK INSERT.
- **Tabla_derivada**: Es cualquier instrucción SELECT válida que devuelva filas con los datos que se van a cargar en la tabla.
- **Sentencia_ejecutable**: Es cualquier instrucción EXECUTE válida que devuelva datos con la instrucciones SELECT. Puede contener la llamada a un procedimiento almacenado.
- **DEFAULT VALUES**: Hace que la nueva fila contenga los valores predeterminados definidos para cada columna.

Consideraciones
Si una columna no se incluye en la lista de columnas, el Motor de base de datos debe ser capaz de proporcionar un valor basado en la definición de la columna; en caso contrario, no se puede cargar la fila. El Motor de base de datos proporciona automáticamente un valor para la columna si ésta:

- Tiene una propiedad IDENTITY. Se usa el valor de identidad incremental siguiente.
- Tiene un valor predeterminado. Se usa el valor predeterminado de la columna.
- Tiene un tipo de datos timestamp. Se utiliza el valor actual de marca de tiempo.
- Acepta valores NULL. Se usa un valor NULL.
- Es una columna calculada. Se utiliza el valor calculado.

Ejemplos

a. Inserta una fila en la tabla UnitMeasure. Dado que los valores para todas las columnas se suministran e incluyen en el mismo orden que las columnas de la tabla, no es necesario especificar los nombres de columna en la lista. Igualmente es una buena práctica declarar la lista siempre.

```
INSERT INTO Production.UnitMeasure
VALUES ('F2', 'Square Feet', GETDATE())
```

b. Mismo ejemplo que en la opción a, pero especificando la lista de columnas.

```
INSERT INTO Production.UnitMeasure
(Name, UnitMeasureCode, ModifiedDate)
VALUES ('Square Yards', 'Y2', GETDATE())
```

c. Inserta 5 filas en la tabla Departments

```
INSERT INTO dbo.Departments
VALUES (1, 'Human Resources', 'Margheim'),
(2, 'Sales', 'Byham'),
(3, 'Finance', 'Gill'),(4, 'Purchasing', 'Barber'),
(5, 'Manufacturing', 'Brewer')
```

d. Inserta el resultado de la instrucción SELECT en la tabla EmployeeSales

```
INSERT dbo.EmployeeSales
SELECT e.BusinessEntityID, c.LastName, sp.SalesYTD
FROM HumanResources.Employee AS e INNER JOIN Sales.SalesPerson AS sp
ON e.BusinessEntityID = sp.BusinessEntityID
INNER JOIN Person.Person AS c ON e.BusinessEntityID =
c.BusinessEntityID
WHERE e.BusinessEntityID LIKE '2%'
ORDER BY e.BusinessEntityID, c.LastName
```

Cláusula OUTPUT

Devuelve las filas insertadas como parte de la operación de inserción. Los resultados se pueden devolver a la aplicación de procesamiento o insertarse en una tabla o variable de tabla para su nuevo procesamiento.

```
DECLARE @MyTableVar table( ScrapReasonID smallint,
                           Name varchar(50),
                           ModifiedDate datetime);
INSERT Production.ScrapReason
   OUTPUT INSERTED.ScrapReasonID, INSERTED.Name, INSERTED.ModifiedDate
   INTO @MyTableVar
VALUES (N'Operator errors', GETDATE());

SELECT ScrapReasonID, Name, ModifiedDate FROM @MyTableVar;
```

5- Instrucción UPDATE

Definición
Cambia los datos de una tabla o vista.

```
UPDATE objeto
    SET
     { nombre columna = { expresión | DEFAULT | NULL }
       | @variable = expresión
       | @variable = columna = expresión
     } [ ,...n ]
  [ FROM{ <tabla> } [ ,...n ] ]
  [ WHERE { <condición> ]
```

Descripción de las cláusulas

- **SET**: Especifica la lista de nombres de variable o de columna que se van a actualizar.
- **Nombre Columna**: Es una columna que contiene los datos que se van a cambiar. Debe existir en la tabla o vista. Las columnas de identidad o columnas calculadas no se pueden actualizar.
- **Expresión**: Es una variable, un valor literal, una expresión o una instrucción de sub selección entre paréntesis que devuelve un solo valor. El valor devuelto por la expresión sustituye al valor existente en la columna o variable.
- **DEFAULT**: Especifica que el valor predeterminado definido para la columna debe reemplazar al valor existente en esa columna. Esta operación también puede utilizarse para cambiar la columna a nulo si no tiene asignado ningún valor predeterminado y se ha definido para aceptar valores nulos.
- **FROM <tabla>**: Especifica que se utiliza un origen de tabla, vista o tabla derivada para proporcionar los criterios de la operación de actualización. Si el objeto que se actualiza es el que se indica en la cláusula FROM y sólo hay una referencia al objeto en ella, puede especificarse o no un alias de objeto. Si el objeto que se actualiza aparece más de una vez en la cláusula FROM, una única referencia al objeto no debe especificar un alias de tabla. Todas las demás referencias al objeto de la cláusula FROM deben incluir un alias de objeto.
- **WHERE**: especifica las condiciones que limitan las filas que se actualizan. Su uso es importante ya que si no todos los registros de la tabla o vista reciben la modificación.

Ejemplos
a. Actualiza todos los registros de la tabla SalesPerson.

```
UPDATE Sales.SalesPerson
SET Bonus = 6000, CommissionPct = .10, SalesQuota = NULL
```

b. Actualiza solo los registros cuyo nombre empieza con Road-250 y son de color rojo. La modificación muestra cómo usar valores calculados.

```
UPDATE Production.Product
SET ListPrice = ListPrice * 2
WHERE Name LIKE N'Road-250%' AND Color = N'Red'
```

c. Modifica la columna SalesYTD de la tabla SalesPerson para reflejar las ventas más recientes registradas en la tabla SalesOrderHeader.

```
UPDATE Sales.SalesPerson SET SalesYTD = SalesYTD + SubTotal
FROM Sales.SalesPerson AS sp
INNER JOIN Sales.SalesOrderHeader AS so
ON sp.BusinessEntityID = so.SalesPersonID
AND so.OrderDate = (SELECT MAX(OrderDate)
FROM Sales.SalesOrderHeader
WHERE SalesPersonID = sp.BusinessEntityID);
```

Cláusula OUTPUT

```
DECLARE @MyTableVar table(
    EmpID int NOT NULL,
    OldVacationHours int,
    NewVacationHours int,
    ModifiedDate datetime);

UPDATE TOP (10) HumanResources.Employee
SET VacationHours = VacationHours * 1.25, ModifiedDate = GETDATE()
OUTPUT inserted.BusinessEntityID,
       deleted.VacationHours,
       inserted.VacationHours,
       inserted.ModifiedDate
INTO @MyTableVar;

SELECT EmpID, OldVacationHours, NewVacationHours, ModifiedDate
FROM @MyTableVar;
```

6- Instrucción DELETE

Definición
Quita filas de una tabla o vista.

```
DELETE [ FROM ] <objeto>
   [ FROM <tabla> [ ,...n ] ]
   [ WHERE { <condición> ]
```

Descripción de las cláusulas

- **FROM**: Palabra clave opcional que se puede utilizar entre la palabra clave DELETE y el destino (tabla, vista o rowset)
- **FROM <tabla>:** Especifica una cláusula FROM adicional. Esta extensión de Transact-SQL para DELETE permite especificar datos de <tabla> y eliminar las filas correspondientes de la tabla en la primera cláusula FROM. Se puede utilizar esta extensión, que especifica una combinación, en lugar de una sub consulta en la cláusula WHERE para identificar las filas que se van a quitar.
- **WHERE**: Especifica las condiciones utilizadas para limitar el número de filas que se van a eliminar. Si no se proporciona una cláusula WHERE, DELETE quita todas las filas de la tabla.

Ejemplos
a. Elimina todos los registros de la tabla SalesPersonQuotaHistory

```
DELETE FROM Sales.SalesPersonQuotaHistory
```

b. Elimina todas las filas de la tabla ProductCostHistory en las que el valor de la columna StandardCost es superior a 1000.00.

```
DELETE FROM Production.ProductCostHistory
WHERE StandardCost > 1000.00
```

c. Elimina las filas de la tabla SalesPersonQuotaHistory basándose en las ventas del año hasta la fecha almacenadas en la tabla SalesPerson

```
DELETE FROM Sales.SalesPersonQuotaHistory
FROM Sales.SalesPersonQuotaHistory AS spqh
INNER JOIN Sales.SalesPerson AS sp
ON spqh.BusinessEntityID= sp.BusinessEntityID
WHERE sp.SalesYTD > 2500000.00
```

Cláusula OUTPUT

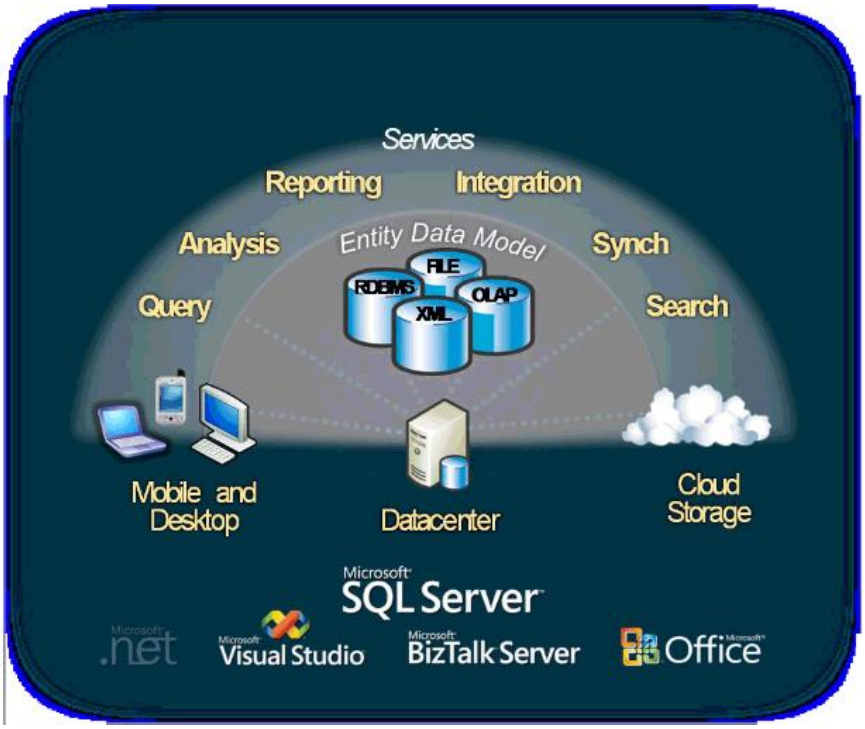
```
DECLARE @MyTableVar TABLE (
    Codigo INT, Razon VARCHAR(50)
);
DELETE FROM dbo.Clientes
OUTPUT DELETED.* INTO @MyTableVar

SELECT * FROM @MyTableVar
```

Módulo 3

Introducción a SQL Server 2008

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Servicios



Motor de la base de Datos
El motor de base de datos es el componente principal de SQL Server. Proporciona almacenaje, recuperación, y modificación de datos.

Analisys Service – Data Mining
Analysis Services otorga un gran alcance a la plataforma Business Intelligence para SQL Server, permitiendo poner en ejecución OLAP Data Warehouses y usar técnicas de Data Mining para analizar datos de negocio y tomar decisiones apropiadas.

SQL Server Integration Service (SSIS)
SQL Server Integration Services (antes **Data Transformation Services**) proporciona una solución comprensiva para la transferencia y transformación de datos entre fuentes de datos diversas.

Notification Services
Notification Services proporciona un Framework para el desarrollo de aplicaciones basadas en subscripciones a través de las cuales se notifica a los usuarios acerca de eventos.

Full-Text Search
Búsqueda Full-Text que permite indexar rápida y flexiblemente consultas de datos basadas en palabras.

Reporting Service
Reporting Services permite la creación de informes de datos de SQL Server. Los informes pueden ser diseñados usando Visual Studio .NET–based Report Designer y pueden ser accedidos usando un IIS Web site.

Replicación
La replicación permite copiar y distribuir datos y objetos de las bases de datos, de una base de datos o servidor a otro, y luego opcionalmente sincronizar entre las bases de datos para asegurar consistencia.

Service Broker
Service Broker habilita la creación de colas para comunicación de transacciones basadas en mensajes (message-based), para que sean confiables entre los servicios de software. Esto hace a SQL Server 2008 una mejor plataforma para soluciones basada en Servicios (service-based).

2- Algunas Novedades

Almacenamiento comprimido de tablas e índices
SQL Server 2008 admite la compresión de almacenamiento en disco tanto en formato de filas como de páginas para tablas, índices y vistas indizadas. La compresión de tablas e índices con particiones se puede configurar independientemente para cada partición.

Tablas anchas
Las tablas anchas son tablas que contienen uno o más conjuntos de columnas. Una tabla ancha puede contener hasta 30000 columnas, 1000 índices y 30000 estadísticas. Las tablas anchas usan las columnas dispersas para aumentar hasta 30.000 el número total de columnas permitidas.

Tipos de datos de fecha y hora
SQL Server 2008 introduce cuatro nuevos tipos de datos de fecha y hora. Estos tipos permiten a las aplicaciones tener tipos independientes para la fecha y la hora, un mayor intervalo de años, mayor precisión en las fracciones de segundo, y compatibilidad para desplazamiento de zona horaria.

Tipo de tabla definida por el usuario
Database Engine (Motor de base de datos) presenta un nuevo tipo de tabla definido por el usuario que permite representar estructuras de tabla para usarlas como parámetros en procedimientos almacenados y funciones, en un lote o en el cuerpo de un procedimiento almacenado o función. Puede crear restricciones UNIQUE y claves principales en tipos de tabla definidos por el usuario.

Parámetros con valores de tabla
Database Engine (Motor de base de datos) presenta un nuevo tipo de parámetro que puede hacer referencia a tipos de tabla definidos por el usuario. Los parámetros con valores de tabla pueden enviar múltiples filas de datos a una instrucción o rutina de SQL Server (tal como un procedimiento almacenado o una función) sin crear una tabla temporal.

Conmutación de la partición en tablas e índices con particiones
La partición de datos permite administrar y tener acceso a subconjuntos de los datos de forma rápida y eficaz, a la vez que mantiene la integridad de la totalidad de la colección de datos. Ahora puede utilizar la conmutación de partición para transferir rápida y eficazmente subconjuntos de datos conmutando una partición de una tabla a otra.

Índices filtrados y estadísticas
En SQL Server 2008, puede utilizar un predicado para crear índices filtrados y estadísticas en un subconjunto de filas de la tabla. Los índices filtrados y las estadísticas son especialmente adecuados para consultas que seleccionan de subconjuntos bien determinados de datos, tales como columnas con valores NULL principalmente, columnas con categorías heterogéneas de valores, y columnas con intervalos de valores distintos.

Cifrado de datos transparente
El cifrado del dato transparente (TDE) presenta una nueva opción de base de datos que cifra automáticamente los archivos de base de datos, sin necesidad de modificar ninguna aplicación. Esto evita que usuarios sin autorización tengan acceso a una base de datos, aunque obtengan los archivos de base de datos o de copia de seguridad de la base de datos.

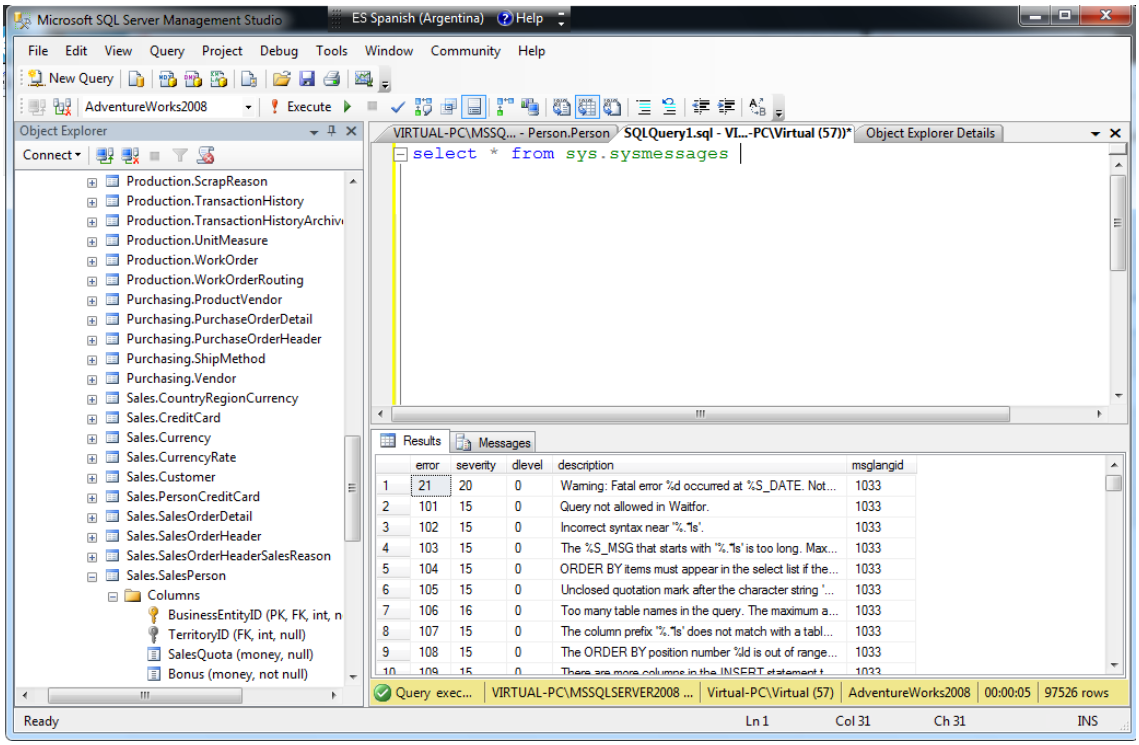
Auditoría
SQL Server Audit es una nueva característica de SQL Server 2008 que permite crear auditorías personalizadas de eventos Database Engine (Motor de base de datos). Utiliza

eventos extendidos para registrar la información para la auditoría y proporciona las herramientas y procesos necesarios para habilitar, almacenar y ver auditorías en diversos servidores y objetos de base de datos.

SQL Server Express
La capacidad de SQL Server Express para admitir bases de datos grandes se ha incrementado de 4 GB a 10 GB.

Datos Espaciales
SQL Server 2008 admite los tipos de datos **geometry** y **geography** para almacenar datos espaciales. **Geometry** representa los datos en un sistema de coordenadas euclidiano (plano). **Geography** almacena datos elípticos (tierra redonda), como las coordenadas de latitud y longitud del sistema GPS.

SQL Server Management Studio
SQL Server Management Studio es un entorno integrado para obtener acceso a todos los componentes de SQL Server, configurarlos, administrarlos y desarrollarlos. SQL Server Management Studio combina un amplio grupo de herramientas gráficas con una serie de editores de script para ofrecer acceso a SQL Server a programadores y administradores de todos los niveles de especialización.
SQL Server Management Studio combina las características del Administrador corporativo, el Analizador de consultas y Analysis Manager, herramientas incluidas en versiones anteriores de SQL Server, en un único entorno. Además, SQL Server Management Studio funciona con todos los componentes de SQL Server, como Reporting Services, Integration Services y SQL Server Compact 3.5. Los programadores obtienen una experiencia familiar y los administradores de bases de datos una única herramienta completa que combina herramientas gráficas fáciles de usar con funciones de script enriquecidos.



Módulo 4

Creación de Bases de Datos

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Bases de Datos

Introducción
Las bases de datos de SQL Server 2008 están formadas por un conjunto de tablas. Estas tablas contienen datos y otros objetos, como vistas, índices, procedimientos almacenados, funciones definidas por el usuario y desencadenadores, que se definen para permitir realizar actividades con los datos.

Diseño
Al diseñar la base de datos, independientemente de su tamaño y complejidad, se llevan a cabo los siguientes pasos básicos:

- Recopilar información.
- Identificar los objetos.
- Crear el modelo de objetos.
- Identificar los tipos de información para cada objeto.
- Identificar las relaciones entre los objetos.

Muchas aplicaciones pertenecen a una de estas dos categorías de aplicaciones de base de datos. Las características de estos tipos de aplicaciones tienen una influencia decisiva en las consideraciones del diseño de una base de datos.

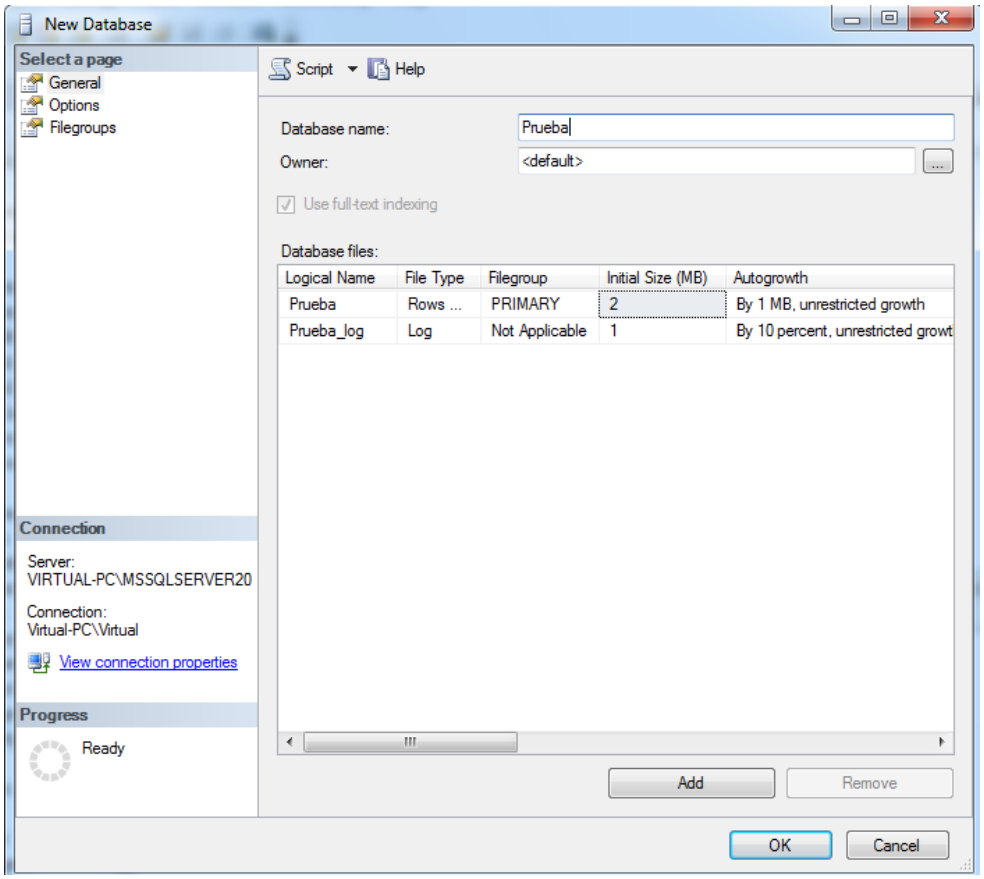
- **Proceso de transacciones en línea (OLTP, Online Transaction Processing)**: Datos que cambian con frecuencia. Estas aplicaciones cuentan normalmente con muchos usuarios que realizan transacciones al mismo tiempo que cambian datos en tiempo real. Consideraciones a tener en cuenta en este tipo de base de datos: alto grado de normalización, dosificación de índices, ubicación correcta de los datos y pocos datos históricos.
- **Ayuda a la toma de decisiones (OLAP, OnLine Analytical Processing)**: son óptimas para las consultas de datos que no impliquen cambios frecuentes en los mismos. Consideraciones a tener en cuenta en este tipo de base de datos: poca normalización, muchos índices y datos pre procesados.

Bases de Datos de Sistema

- **Master**: La base de datos master se compone de las tablas de sistema que realizan el seguimiento de la instalación del servidor y de todas las bases de datos que se creen posteriormente. Asimismo controla las asignaciones de archivos, los parámetros de configuración que afectan al sistema, las cuentas de inicio de sesión. Esta base de datos es crítica para el sistema, así que es bueno tener siempre una copia de seguridad actualizada.
- **Tempdb**: Es una base de datos temporal, fundamentalmente un espacio de trabajo, es diferente a las demás bases de datos, puesto que se regenera cada vez que arranca SQL Server. Se emplea para las tablas temporales creadas explícitamente por los usuarios, para las tablas de trabajo intermedias de SQL Server durante el procesamiento y la ordenación de las consultas.
- **Model**: Se utiliza como plantilla para todas las bases de datos creadas en un sistema. Cuando se ejecuta una instrucción **CREATE DATABASE**, la primera parte de la base de datos se crea copiando el contenido de la base de datos model, el resto de la nueva base de datos se llena con páginas vacías.
- **Msdb**: Es empleada por los servicios SQL Server Agent, Database Mail, Service Broker, log shipping, etc. para guardar información con respecto a tareas de automatización como por ejemplo copias de seguridad y tareas de duplicación, asimismo solución a problemas.

Creación
Se puede crear una base de datos usando la herramienta SQL Server Management Studio o con la sentencia **CREATE DATABASE**:

Creación de bases de datos en el SQL Server Management Studio:




```
CREATE DATABASE nombre_basedatos
[ ON
    [ < filespec > [ ,...n ] ]
    [ , < filegroup > [ ,...n ] ]
]
[ LOG ON { < filespec > [ ,...n ] } ]
[ COLLATE nombre_collation ]
 [ PRIMARY ]
( [ NAME = nombre_logico, ]
    FILENAME = 'nombre_fisico'
    [ , SIZE = tamaño ]
    [ , MAXSIZE = { tamaño_maximo | UNLIMITED } ]
    [ , FILEGROWTH = incremento_crecimiento ] ) [ ,...n ]
< filegroup > ::=
FILEGROUP filegroup_name < filespec > [ ,...n ]
```

Argumentos:

- **Nombre_BaseDatos**: Nombre lógico de la base de datos. Deben cumplir las reglas de los identificadores
- **ON**: Especifica la información sobre el archivo de datos
- **LOG ON**: Especifica la información sobre el archivo del registro de transacciones.
- **Collate**: Establece el juego de caracteres soportados.
- **Primary**: Especifica el grupo de archivos (**filegroup**) para este archivo. El grupo de archivo base del SQL Server se llama Primary.
- **FileName**: Nombre físico del archivo para el sistema operativo. Se incluye con la ruta completa donde será grabado.
- **Size**: Tamaño inicial de la base de datos. Para el archivo principal de datos debe ser mayor al tamaño de la base de datos de sistema **Model**, ya que la misma es copiada en el momento de la creación para inicializarla.
- **MaxSize**: Tamaño máximo para la base de datos. Si no se especifica la base de datos puede crecer hasta llenar el disco.
- **FileGrowth**: Especifica el incremento de crecimiento de la base de datos

Al crear una base de datos la base de datos de sistema **Model** es copiada para inicializar la nueva base de datos. El resto es llenado con páginas vacías

Ejemplo:

```
CREATE DATABASE Prueba
ON (NAME = 'Prueba_Data',
FILENAME = 'D:\DATA\Prueba.mdf',
SIZE = 20 MB,
FILEGROWTH = 0)
LOG ON (NAME = 'Prueba Log',
FILENAME = 'D:\DATA\Prueba_Log.ldf',
SIZE = 5 MB,
FILEGROWTH = 0)
```

Modificación y Borrado

Para modificar o borrar una tabla se puede usar la herramienta SQL Server Management Studio o las instrucciones **ALTER DATABASE / DROP DATABASE**.
Al modificar se puede agregar o sacar archivos o grupos de archivos, modificar atributos, cambiar el nombre o el tamaño de una base de datos. También permite establecer o modificar valores de las opciones.
Al borrar una base de datos ésta se borra físicamente del disco.

Opciones de Base de Datos

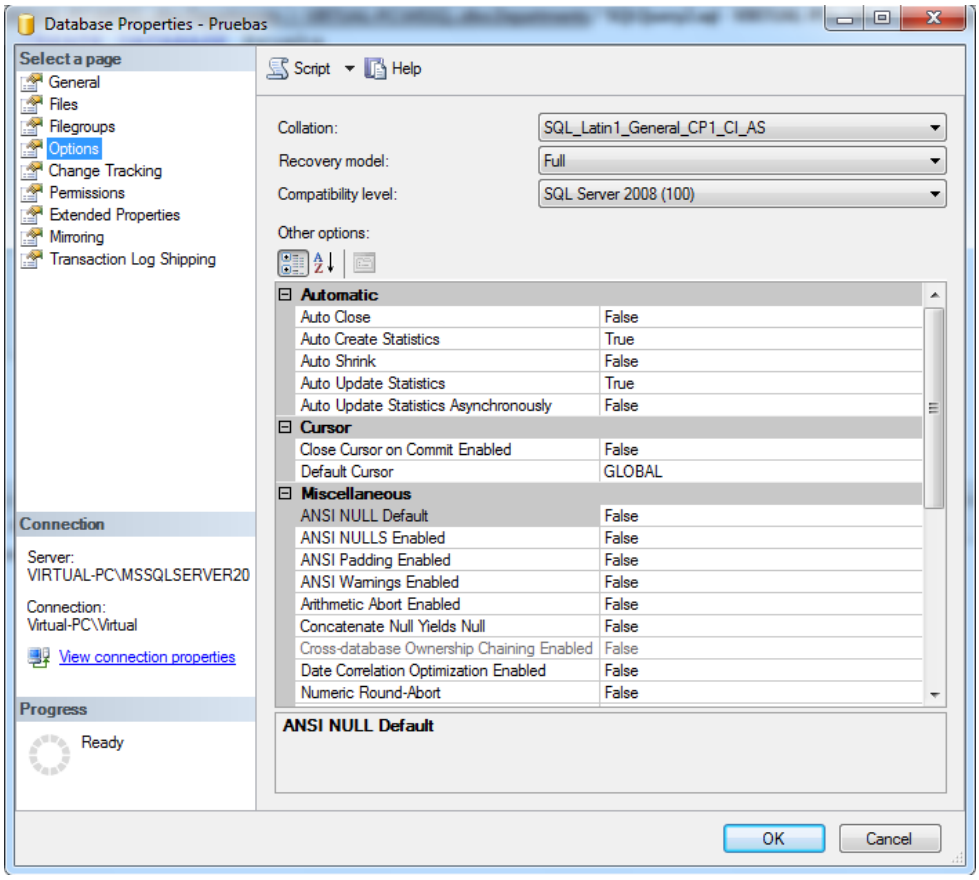
Una vez creada la base de datos, se pueden establecer distintos valores para sus opciones usando SQL Server Management Studio o la sentencia ALTER DATABASE. El cambio de opciones se hace para cada base de datos por separado. Si se desea modificar varias simultáneamente podría modificarse la base de datos de sistema **Model** y toda base de datos creada a continuación contendrá las modificaciones.

Alguna de las opciones que pueden establecerse son:

- **AUTO_CREATE_STATISTICS**: Crea estadísticas en forma automática necesarias para la optimización de consultas. El valor predeterminado es ON.
- **AUTO_UPDATE_STATISTICS**: Actualiza automáticamente las estadísticas que están desactualizadas. El valor predeterminado es ON.

- **AUTO_CLOSE**; Si está en ON cierra la base de datos automáticamente cuando el último usuario cierra su sesión. El valor predeterminado es OFF (excepto para la edición Express)
- **AUTO_SHRINK*:*** Si está en ON la base de datos se encoge automáticamente en forma periódica. El valor predeterminado es OFF.
- **READ_ONLY / READ_WRITE*:*** Controla si los usuarios pueden modificar los datos. El valor predeterminado es READ_WRITE.
- **SINGLE_USER / RESTRICTED_USER / MULTI_USER**: Controla que usuarios pueden conectarse a la base de datos. El valor predeterminado es MULTI_USER. Cuando se especifica SINGLE_USER, sólo se puede conectar un usuario a la base de datos en un momento dado. Todas las demás conexiones de usuario se desconectan. Cuando se especifica RESTRICTED_USER, sólo pueden conectarse a la base de datos los miembros de la función fija de base de datos ***db*_owner** y los de las funciones fijas de servidor **dbcreator** y **sysadmin**, pero no se limita la cantidad de miembros. Cuando se especifica MULTI_USER, se permite el acceso de todos los usuarios que cuenten con los permisos adecuados para conectarse a la base de datos.
- **OFFLINE / ONLINE / EMERGENCY**: Cuando se especifica OFFLINE, la base de datos se cierra sin problemas y se marca como sin conexión. Cuando se especifica ONLINE (predeterminado), la base de datos está abierta y disponible para su utilización. Cuando se especifica EMERGENCY, la base de datos se marca como READ_ONLY, se deshabilita el registro y se limita el acceso a los miembros de la función fija de servidor **sysadmin**.
- **RECOVERY MODEL: FULL / SIMPLE / BULK_LOGGED**: El valor predeterminado es FULL. Provee un modelo de recuperación completo ante fallas. BULK_LOGGED no usa el registro de transacciones para ese tipo de movimiento. SIMPLE recupera la base de datos solo desde el último backup completo o diferencial.
- **PAGE_VERIFY**: Permite detectar entradas de E/S incompletas. CHECKSUM: guarda un valor calculado en la cabecera de la página basado en su contenido. Este valor es recalculado y comparado con los datos de la página para controlarlas. Es el valor predeterminado. TORN_PAGE_DETECTION: guarda un bit específico por cada sector de 512 bytes dentro de la cabecera de la página. Este bit se usa para el control.
- **SQL ANSI_NULL_DEFAULT**: Permite al usuario controlar el uso predeterminado del valor nulo de una columna al crear o modificar una tabla. El valor predeterminado es OFF o sea NOT NULL.
- **ANSI_NULLS**; Cuando está en ON, todas las comparaciones con nulos devuelven nulos. Si está en OFF solo devuelve nulo si ambos valores son nulos. El valor predeterminado es OFF
- **QUOTED_IDENTIFIER**: Cuando se especifica ON, se pueden utilizar comillas dobles para encerrar los identificadores delimitados. Cuando se especifica OFF, los identificadores no pueden ir entre comillas y deben adaptarse a todas las reglas de Transact-SQL que se aplican a los identificadores (Usar [] para delimitar identificadores).

Modificación de opciones usando el SQL Server Management Studio



Ejemplos de ALTER DATABASE:

```
ALTER DATABASE AdventureWorks
SET READ_ONLY

ALTER DATABASE AdventureWorks
SET RECOVERY FULL, PAGE_VERIFY CHECKSUM
```

Modelo de Recuperación

- **Completo (Full):** Requiere copias de seguridad de registros. No se pierde trabajo si un archivo de datos se pierde o resulta dañado. Se puede recuperar hasta cualquier momento, por ejemplo, antes del error de aplicación o usuario.
- **Simple**: Sin copias de seguridad de registros. Recupera automáticamente el espacio de registro para mantener al mínimo los requisitos de espacio, eliminando, en esencia, la necesidad de administrar el espacio del registro de transacciones.
- **Por medio de registros de operaciones masivas (BulkCopy)**: Requiere copias de seguridad de registros. Complemento del modelo de recuperación completa que permite operaciones de copia masiva de alto rendimiento. Reduce el uso del espacio de registro mediante el registro mínimo de la mayoría de las operaciones masivas.

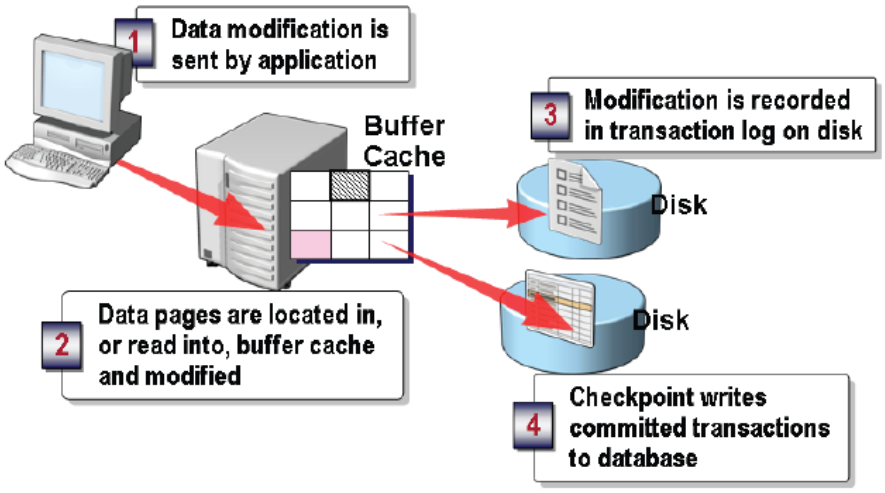
Intercalación (Collation)
La configuración de intercalación de SQL Server 2008 depende del tipo de instalación. Normalmente, se debe elegir una intercalación de SQL Server que admita la configuración regional del sistema de Windows que más se suela usar en la organización.
Las intercalaciones de SQL Server 2008 controlan lo siguiente:

- La página de códigos que se utiliza para almacenar datos no Unicode en SQL Server.
- Las reglas que rigen la forma en que SQL Server ordena y compara los caracteres que se almacenan en tipos de datos no Unicode.

Arquitectura física del registro de transacciones (Transaction Log)
El registro de transacciones se utiliza para garantizar la integridad de los datos de la base de datos y para la recuperación de datos. Comprender la arquitectura física puede mejorar la eficacia en la administración de registros de transacciones.
El registro de transacciones de una base de datos está asignado a uno o varios archivos físicos. Conceptualmente, el archivo de registro es una cadena de entradas de registro. El registro de transacciones es un archivo de registro circular. Las nuevas entradas del registro se agregan al final del registro lógico y se expanden hacia el final del archivo físico.
SQL Server guarda cada transacción usando el registro de transacciones para mantener la consistencia de la base de datos y ayudar a recuperarlos en el caso de fallas de la misma. Las transacciones son grabadas primero en el registro de transacciones y automáticamente pasadas a los datos.

El proceso es el siguiente:

- La aplicación envía los datos
- Cuando la modificación es ejecutada, las páginas afectadas son cargadas en memoria, si las mismas no hubieran sido cargadas anteriormente.
- Cada modificación es guardada en el registro de transacciones.
- El proceso de punto de comprobación (**CheckPoint**) graba toda la transacción en la base de datos.



Puntos de comprobación (CheckPoint)
El Motor de base de datos de SQL Server genera puntos de comprobación automáticos. El intervalo entre puntos de comprobación automáticos se basa en el espacio del registro utilizado

y en el tiempo transcurrido desde el último punto de comprobación. El intervalo de tiempo entre los puntos de comprobación automáticos puede ser muy variable y largo si se realizan pocas modificaciones en la base de datos. Los puntos de comprobación automáticos también se pueden producir con frecuencia si se modifican muchos datos.

Compresión de Datos

SQL Server 2008 admite la compresión de filas y páginas tanto para tablas como para índices. La compresión de datos se puede configurar para los objetos de base de datos siguientes:

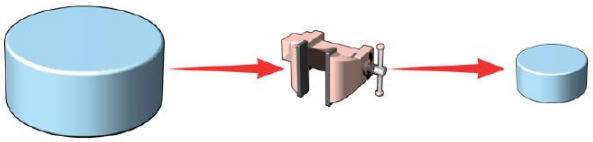
- Una tabla entera que está almacenada como un montón.
- Una tabla entera que está almacenada como un índice clúster.
- Un índice no clúster entero.
- Una vista indizada entera.
- Para tablas e índices con particiones, la opción de compresión se puede configurar para cada partición y las diferentes particiones de un objeto no tienen por qué tener la misma configuración de compresión.

La configuración de compresión de una tabla no se aplica automáticamente a sus índices no clúster. Cada índice se debe establecer individualmente. La compresión no está disponible para las tablas del sistema. Las tablas y los índices se pueden comprimir cuando se crean utilizando las instrucciones **CREATE TABLE** y **CREATE INDEX**. Para cambiar el estado de compresión de una tabla, índice o partición, utilice las instrucciones **ALTER TABLE** o **ALTER INDEX**.

La compresión de fila permite almacenar tipos de longitud fija en el almacenamiento de longitud variable. La compresión de página minimiza el almacenamiento de datos redundantes en las páginas de almacenamiento.

La sobrecarga de la CPU en la compresión de fila es menor que para página, pero la compresión de página puede proporcionar una mejor compresión.

```
ALTER TABLE <table_name>
REBUILD PARTITION = 1 WITH (DATA_COMPRESSION =  <option>)
```



Información sobre Base de datos

Se puede consultar información sobre una base de datos usando SQL Server Management Studio o consultando las vistas de sistemas, funciones de sistema o procedimientos almacenados de sistema que provee SQL Server.

SQL Server Management Studio

- **Explorador de Objetos**: Herramienta gráfica para ubicar Server, bases de datos, y sus objetos
- **Ventana de Propiedades**: La información mostrada varía según el objeto seleccionado
- **Reportes**: Se incluyen varios reportes de consulta de información

Vista del Catálogo

Las vistas de catálogo devuelven información utilizada por el Motor de base de datos de SQL Server. Se recomienda utilizar las vistas de catálogo porque son la interfaz más general para los metadatos del catálogo y proporcionan el método más eficaz para obtener, transformar y presentar formas personalizadas de esta información. Todos los metadatos del catálogo disponibles para el usuario se exponen mediante las vistas de catálogo.
Las vistas de catálogo de SQL Server se han organizado en varias categorías. Algunas de ella son:

- Vistas de catálogo de archivos y bases de datos: Por ejemplo:
  - **sys.databases** que devuelve un registro por cada base de datos.
  - **sys.database_files** que devuelve un registro por cada archivo de una base de datos.
- Objetos: Por ejemplo:
  - **sys.tables** que devuelve un registro por cada tabla de una base de datos.
  - **sys.views** que devuelve un registro por cada vista de una base de datos.
  - **sys.columns** que devuelve un registro por cada columna de un objeto.
- Seguridad: Por ejemplo:
  - **sys.database_permissions** que devuelve un registro por cada permiso definido en una base de datos.
  - **sys.database_role_member** que devuelve un registro por cada miembro de un rol de una base de datos.

Funciones del Sistema

Las siguientes funciones escalares devuelven información acerca de la base de datos y de los objetos de la misma. Solo se mencionan algunas de ellas:

- **DB_ID**: Devuelve el número de identificación (Id.) de esa base de datos.

```
SELECT DB_ID('master')                    Devuelve 1
```

- **DB_NAME**: Devuelve el nombre de la base de datos.

```
SELECT DB_NAME(1)                         Devuelve master
```

- **FILE_ID**: Devuelve el número de identificación del archivo (Id.) del nombre de archivo lógico dado de la base de datos actual.

```
SELECT FILE_ID('AdventureWorks_Data')     Devuelve 1
```

- **FILE_NAME**: Devuelve el nombre del archivo lógico dado de la base de datos actual.

```
SELECT FILE_NAME(1)                       Devuelve AdventureWorks_Data
```

Procedimientos Almacenados de Sistema

Estos son algunos de los procedimientos almacenados de sistema que permiten consultar información sobre base de datos:

- **Sp_Databases**: Lista las bases de datos disponibles de un Server

```
EXEC Sp_Databases
```

- **Sp_HelpDB**: Información sobre las bases de datos de un servidor
- **Sp_Help**: Presenta información acerca de un objeto de base de datos (cualquier objeto de la vista de compatibilidad **sys.sysobjects**), un tipo de datos definido por el usuario o un tipo de datos. Puede ejecutarse sin parámetros, entonces muestra la información sobre todos los objetos o puede pasarse como parámetro el nombre del objeto a consultar.

```
EXEC sp_help
EXEC sp_help 'Production.Product'
```

2- Grupos de Archivos (FileGroups)

Definición
SQL Server asigna una base de datos a un conjunto de archivos del sistema operativo. Los datos y la información del registro nunca se mezclan en el mismo archivo, y cada archivo sólo es utilizado por una base de datos. Los grupos de archivos se denominan colecciones con nombre de archivos que se utilizan como ayuda en tareas de colocación de datos y administrativas, como las operaciones de copias de seguridad y restauración.

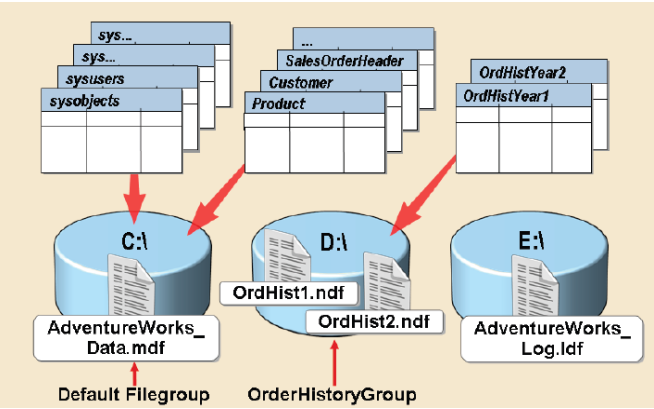
Archivos de base de datos
Las bases de datos de SQL Server utilizan tres tipos de archivos:

- **Archivos de datos principales**: El archivo de datos principal es el punto de partida de la base de datos y apunta a los otros archivos de la base de datos. Cada base de datos tiene un archivo de datos principal. La extensión recomendada para los nombres de archivos de datos principales es .**mdf**.
- **Archivos de datos secundarios**: Los archivos de datos secundarios son todos los archivos de datos menos el archivo de datos principal. Puede que algunas bases de datos no tengan archivos de datos secundarios, mientras que otras pueden tener varios archivos de datos secundarios. La extensión de nombre de archivo recomendada para los archivos de datos secundarios es .**ndf**.
- **Archivos de registro**: Los archivos de registro almacenan toda la información de registro que se utiliza para recuperar la base de datos. Como mínimo, tiene que haber un archivo de registro por cada base de datos, aunque puede haber varios. La extensión de nombre de archivo recomendada para los archivos de registro es .**ldf**.

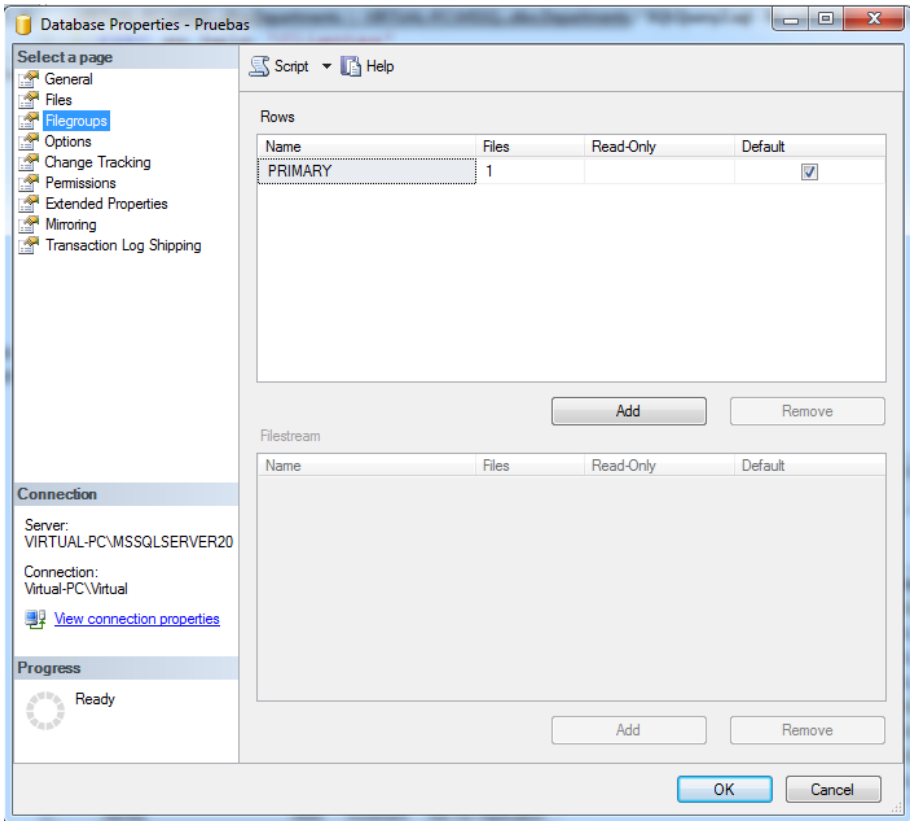
SQL Server no exige las extensiones de nombre de archivo .mdf, .ndf y .ldf, pero estas extensiones ayudan a identificar las distintas clases de archivos y su uso.

Grupos de Archivos
Los objetos y archivos de una base de datos se pueden agrupar en grupos de archivos con fines de asignación y administración. Hay dos tipos de grupos de archivos:

- **Principal**: El grupo de archivos principal contiene el archivo de datos principal y los demás archivos asignados específicamente a otro grupo de archivos. Todas las páginas de las tablas del sistema están asignadas al grupo de archivos principal. (**PRIMARY**)
- **Definidos por el usuario**: Los grupos de archivos definidos por el usuario son los grupos de archivos especificados mediante la palabra clave **FILEGROUP** en la instrucción CREATE DATABASE o ALTER DATABASE. Ningún archivo puede pertenecer a más de un grupo de archivos.



Creación



Ejemplo:

```
CREATE DATABASE [AdventureWorks] ON PRIMARY
      (NAME = 'AdventureWorks_Data', FILENAME =
      'C:\AdventureWorks_Data.mdf' ),
FILEGROUP [OrderHistoryGroup]
      ( NAME = N'OrdHist1', FILENAME = N'D:\OrdHist1.ndf' ),
      ( NAME = N'OrdHist2', FILENAME = N'D:\OrdHist2.ndf' )
LOG ON
( NAME = 'AdventureWorks_log', FILENAME = 'E:\AdventureWorks_log.ldf')
```

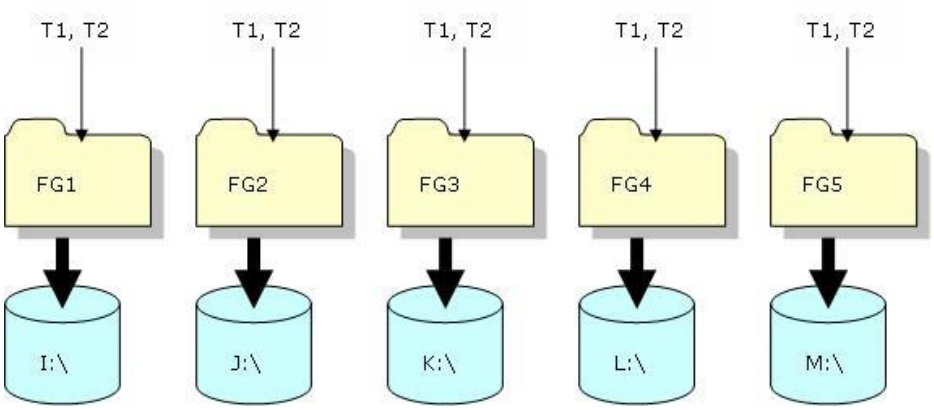
Mejoras en el rendimiento

El uso de archivos y grupos de archivos permite mejorar el rendimiento de la base de datos al permitir crear la base de datos en varios discos, varios controladores de disco o sistemas RAID (matriz redundante de discos independientes). Por ejemplo, si su equipo tiene cuatro discos, puede crear una base de datos que contenga tres archivos de datos y un archivo de registro, y mantener un archivo en cada disco. Cuando se produce un acceso a los datos, hay cuatro cabezales de lectura/escritura que pueden tener acceso en paralelo a los datos a la vez, con lo que se aceleran las operaciones de la base de datos.

Consideraciones para la creación y organización de grupos de archivos

- Separar datos de solo lectura de los modificables

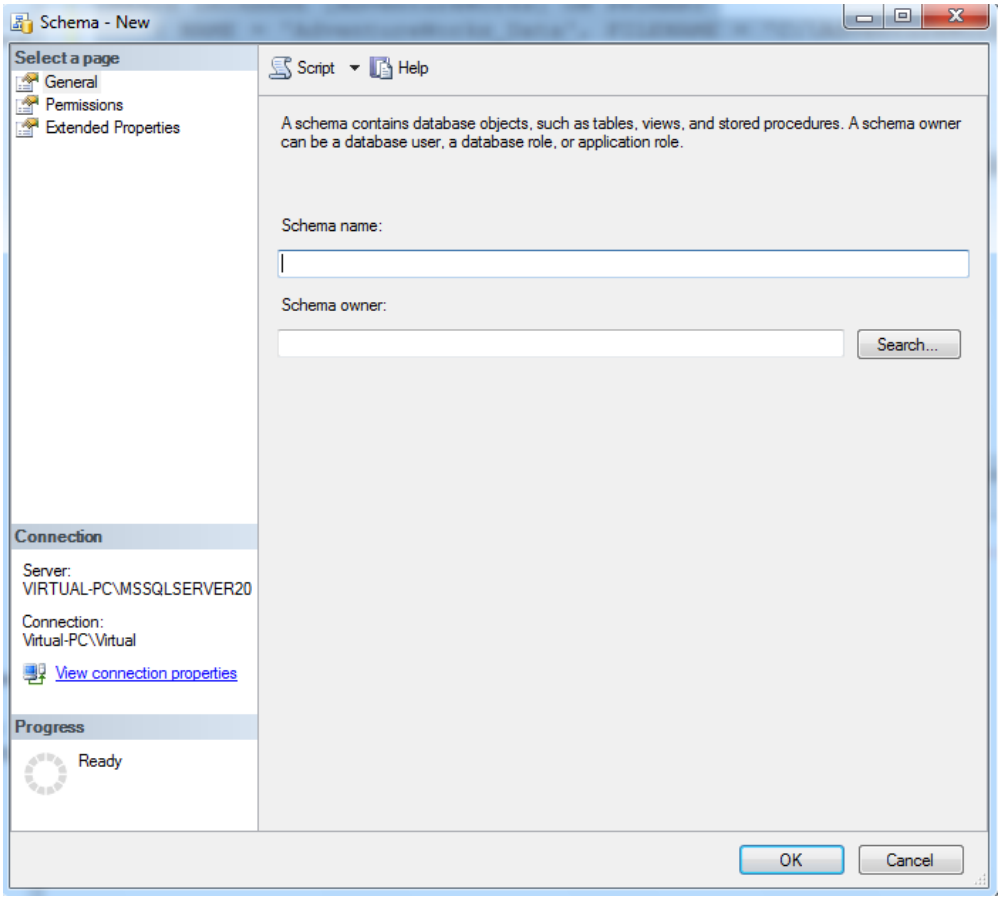
- Separar índices de sus tablas, mejora la performance
- Hacer copias de seguridad por archivo en vez de hacer copias de toda la base de datos
- Tablas e índices con el mismo tipo de mantenimiento deberían estar en el mismo grupo de archivos
- Cambiar siempre el grupo de archivos predeterminado de manera de no incluir automáticamente las tablas de usuario en **Primary**
- Organizar tablas con particiones en múltiples grupos de archivos. Es una buena manera de separar datos con distintos niveles de acceso y no trabajar con tablas excesivamente grandes.



Tablas con particiones en 5 grupos de archivos diferentes.

3- Esquemas (Schemas)

Definición
El comportamiento de los esquemas cambió a partir de SQL Server 2005. Los esquemas ya no son equivalentes a los usuarios de la base de datos; cada esquema ahora es un espacio de nombres (namespace) distinto que existe de forma independientemente del usuario de base de datos que lo creó. Es decir, un esquema simplemente es un contenedor de objetos. Cualquier usuario puede ser propietario de un esquema, y esta propiedad es transferible.
Todas las bases de datos contienen un esquema llamado **dbo**. Es el esquema predeterminado para todos los usuarios cuando no se define explícitamente el esquema
Se pueden crear esquemas usando la herramienta SQL Server Management Studio o la instrucción **CREATE SCHEMA**.



Ejemplo:

```
CREATE SCHEMA Ventas
```

Para modificar o borrar un esquema use las instrucciones ALTER/DROP SCHEMA.

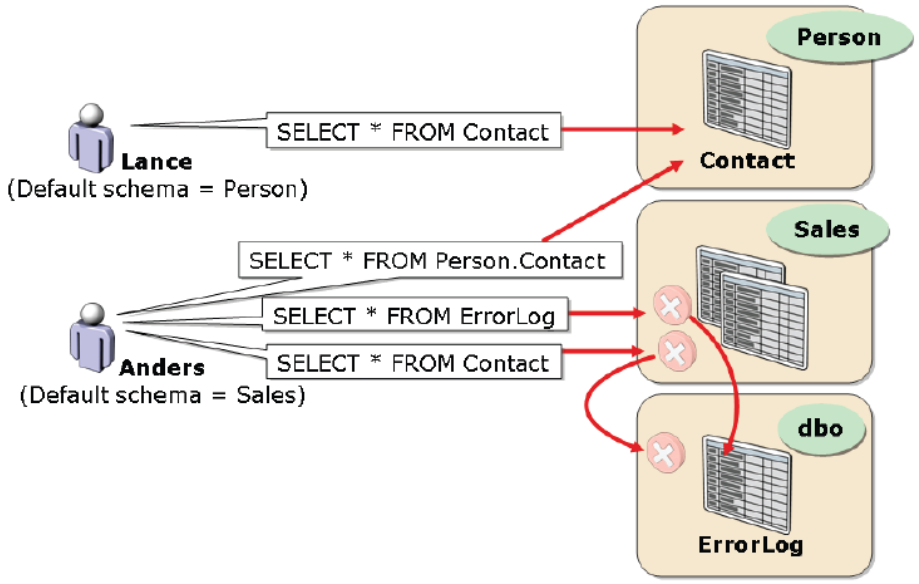
Ventajas

- Mayor flexibilidad para organizar la base de datos usando espacio de nombres, ya que de esta manera los objetos no dependen del usuario que lo creo.

- Manejo de permisos mejorada, ya que los permisos pueden ser asignados a los esquemas y no directamente a cada objeto.
- Al dar de baja un usuario no es necesario renombrar los objetos que le pertenecían.

Resolución de nombres
Cuando una base de datos contiene múltiples esquemas los nombres de los objetos pueden confundirse ya que se podría tener el mismo nombre de objeto en dos esquemas diferentes.
Por ello es bueno asignarle a cada usuario un esquema predeterminado de manera de controlar mejor el nombre de los objetos cuando no se usa el nombre completo. Para resolver los nombres de los objetos incompletos SQL Server 2008 busca primero en el esquema predeterminado asociado al usuario, si no lo encuentra intenta usando **dbo**.
Para asignar un esquema predeterminado a un usuario se usa ALTER USER

```
ALTER USER Maria WITH DEFAULT_SCHEMA = Ventas
```



4- Instantáneas de Base de Datos (Database Snapshot)

Definición

Una instantánea de base de datos es una vista estática de sólo lectura de una base de datos denominada base de datos de origen. Pueden existir varias instantáneas en una base de datos de origen. Una instantánea de base de datos es coherente en cuanto a las transacciones con la base de datos de origen tal como existía en el momento de la creación de la instantánea. Una instantánea se mantiene hasta que el propietario de la base de datos la quita explícitamente.

Las instantáneas de base de datos están disponibles a partir de SQL Server 2005 Enterprise Edition y versiones posteriores. Todos los modelos de recuperación admiten instantáneas de base de datos.

Las instantáneas de la base de datos dependen de la base de datos de origen. Deben residir en la misma instancia de servidor que la base de datos. Además, si la base de datos deja de estar disponible por alguna razón, todas sus instantáneas también dejan de estar disponibles.

Las instantáneas se pueden utilizar para crear informes. Además, en el caso de que se produzca un error de usuario en una base de datos de origen, ésta se puede revertir al estado en que se encontraba cuando se creó la instantánea. La pérdida de datos se limita a las actualizaciones de la base de datos efectuadas desde la creación de la instantánea. Asimismo, la creación de una instantánea de la base de datos puede ser útil inmediatamente antes de realizar un cambio importante en una base de datos, como cambiar el esquema o la estructura de una tabla.

Funcionamiento de las instantáneas de la base de datos

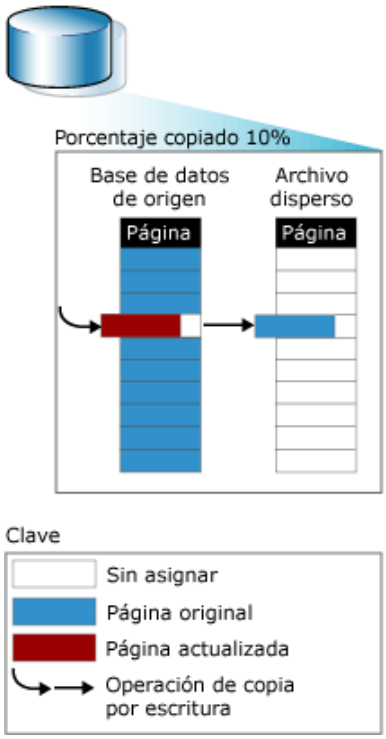
Antes de modificar por primera vez una página de la base de datos de origen, la página original se copia de la base de datos de origen a la instantánea. Este proceso se denomina operación "copiar al escribir". La instantánea almacena la página original y conserva los registros de datos en el estado en que se encontraban cuando se creó la instantánea. Las actualizaciones sucesivas de los registros de una página modificada no afectan al contenido de la instantánea. El mismo proceso se repite para cada página que se modifica por primera vez. De esta forma, la instantánea conserva las páginas originales de todos los registros de datos que se han modificado alguna vez desde que se realizó la instantánea.

Un usuario accede a una página copiada sobre la instantánea solo si ésta sufrió modificaciones desde la creación de la misma. En el caso de no ser así es redireccionado a la página correspondiente sobre la base de datos de origen. Esto ocurre en modo transparente para el usuario.

Debido a que las instantáneas de la base de datos no tienen almacenamiento redundante, no protegen frente a errores de disco u otro tipo de daños. La realización periódica de copias de seguridad y las pruebas del plan de restauración son operaciones fundamentales para la protección de una base de datos.

Para almacenar las páginas originales copiadas, la instantánea utiliza uno o varios archivos dispersos. Inicialmente, un archivo disperso es básicamente un archivo vacío que no contiene datos de usuario y al que todavía no se ha asignado espacio en el disco para datos de usuario. A medida que se actualizan páginas en la base de datos de origen, el tamaño del archivo aumenta. Cuando se realiza una instantánea, el archivo disperso ocupa poco espacio en el disco. Sin embargo, al actualizar la base de datos con el tiempo, el archivo disperso puede aumentar hasta alcanzar un tamaño considerable.

En la siguiente ilustración se muestra una operación "copiar al escribir". En el diagrama de la instantánea, los rectángulos de color gris claro representan espacio potencial en un archivo disperso que todavía está sin asignar. Al recibir la primera actualización de una página en la base de datos de origen, el Motor de base de datos escribe en el archivo y el sistema operativo asigna espacio en los archivos dispersos de la instantánea y copia ahí la página original. A continuación, el Motor de base de datos actualiza la página en la base de datos de origen.



Creación

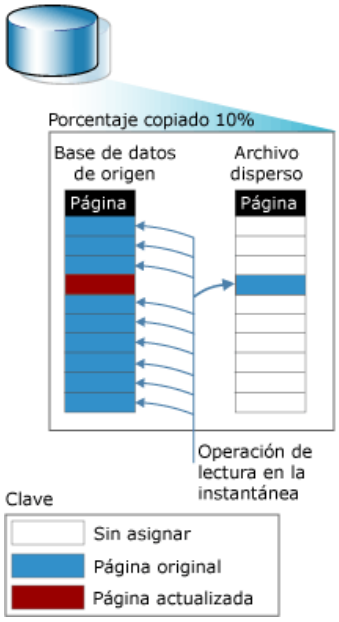
```
CREATE DATABASE AdventureWorks_ds ON
      (NAME= AdventureWorks_Data, FILENAME =
         'D:\Data\AdventureWorks_ds.ss' )
      AS SNAPSHOT OF AdventureWorks
```

Lectura sobre una instantánea de la base de datos

Para el usuario, la instantánea de la base de datos no parece cambiar nunca, ya que las operaciones de lectura en una instantánea siempre tienen acceso a las páginas de datos originales, con independencia de dónde residan.

Si la página no se ha actualizado todavía en la base de datos de origen, la operación de lectura en la instantánea lee la página original de la base de datos de origen. Tras actualizarse una página, la operación de lectura en la instantánea sigue teniendo acceso a la página original, que ahora se encuentra almacenada en un archivo disperso.

En la siguiente ilustración se muestra una operación de lectura en la instantánea que tiene acceso a una página después de actualizarse en la base de datos de origen. La operación de lectura lee la página original en el archivo disperso de la instantánea.



Tamaño de la instantánea de la base de datos

Si la base de datos de origen es bastante grande y está preocupado por el uso del espacio de disco, en algún momento deberá reemplazar una instantánea antigua por una nueva. La duración idónea de una instantánea depende de su índice de crecimiento y el espacio de disco disponible para los archivos dispersos. El espacio de disco que requiere una instantánea depende de cuántas páginas diferentes de la base de datos de origen se actualicen durante la existencia de la instantánea. En consecuencia, si la mayoría de las actualizaciones se realizan en un subconjunto pequeño de páginas que se actualizan repetidamente, el índice de crecimiento disminuirá con el tiempo y los requisitos de espacio de la instantánea seguirán siendo relativamente pequeños. Por el contrario, si todas las páginas originales acaban actualizándose al menos una vez, la instantánea aumentará hasta igualar el tamaño de la base de datos de origen. Si el espacio del disco empieza a agotarse, las instantáneas compiten entre sí por dicho espacio. Si la unidad de disco se llena, se producirán errores en las operaciones de escritura en todas las instantáneas.

Módulo 5

Creación de Tipos de Datos Y Tablas

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Tipos de Datos

Definición
En SQL Server, cada columna, variable local, expresión y parámetro tiene un tipo de datos relacionado. Un tipo de datos es un atributo que especifica el tipo de datos que el objeto puede contener: datos de enteros, datos de caracteres, datos de moneda, datos de fecha y hora, cadenas binarias, etc.
SQL Server proporciona un conjunto de tipos de datos del sistema que define todos los tipos de datos que pueden utilizarse con SQL Server. También puede definir sus propios tipos de datos en Transact-SQL o Microsoft .NET Framework.

Tipo de Datos SQL Server

Categoría	Tipo de Dato	Bytes	Comentarios
Enteros	Tinyint	1	Un número entero entre 0 y 255
	Smallint	2	Un entero corto entre – 32.768 y 32.767.
	Int	4	Un entero largo entre – 2.147.483.648 y 2.147.483.647.
	BigInt	8	
Numéricos exactos	Decimal, Numeric	2 – 17	Generalmente se usa Decimal
Numéricos aproximados	Float, Real	8 - 15	Tratar de evitarlos. Complica el uso de la cláusula Where
Moneda	SmallMoney	4	
	Money	8	
Fecha	SmallDatetime	4	Intervalo 01-01-1900 a 06-06-2079
	Datetime	8	Intervalo 01-01-1753 a 31-12-9999
	Datetime2	6 - 8	Intervalo 01-01-0001 a 31-12-9999 (nuevo)
	Datetimeoffset	10	Define una fecha que se combina con una hora del día con reconocimiento de zona horaria y basada en un reloj de 24 horas. (nuevo) Intervalo 01-01-0001 a 31-12-9999
	Date	3	Solo Fecha (nuevo) Intervalo 01-01-0001 a 31-12-9999
	Time	5	Solo Hora. No distingue zona horaria. (nuevo)
Caracteres	Char	0 – 8000	Hasta 8000 caracteres. Usar cuando se conoce la cantidad exacta de caracteres.
	VarChar	0 – 8000	Hasta 8000 caracteres. Usar cuando la cantidad de caracteres es variable.
	Varchar(Max)	0 – 2GB	No se guarda en el registro. Se aconseja este tipo por sobre Text. Permite trabajar como un varchar, sin la necesidad de punteros
	Text	0 – 2GB	No se guarda en el registro. No debería usarse, usar varchar(max)
Caracteres Unicote	nChar, nVarchar	0 – 8000	Hasta 4000 caracteres. Idem char, varchar.
	nVarchar(Max)	0 – 2GB	Nuevo. Idem varchar(max)
	nText	0 – 2GB	No se guarda en el registro. No debería usarse,

			usar nvarchar(max)
Binarios	Binary, Varbinary	0 – 8000	Permiten almacenar cualquier tipo de datos.
	Varbinary(Max)	0 – 2GB	Nuevo. No se guarda en el registro.
Imagen	Image	0 – 2GB	No se guarda en el registro. No debería usarse, usar varbinary(max)
GUID	uniqueIdentifier	16	Se inicializa con la función NEWID() o NEWSEQUENTIALID(). Se utiliza mucho durante la replicación.
XML	Xml	0 – 2GB	Almacena datos en formato XML
Especiales	Bit	1	Tipo de datos entero que puede aceptar los valores 1, 0 o nulo.
	Cursor	0 - 8	Producen un conjunto completo de resultados
	Timestamp	8	Suele utilizarse como mecanismo para marcar la versión de las filas de la tabla
	Hierarchyid	variable	Representa una posición en una jerarquía de árbol

Consideraciones en la elección de tipos de datos

- **Columnas de longitud fija y variable*:*** El diseño de las tablas le permite comprender las ventajas e inconvenientes del uso de columnas de longitud fija y de longitud variable. Las columnas de longitud variable reducen el tamaño de la base de datos porque solamente ocupan el espacio necesario para almacenar el valor real. Las columnas de longitud fija siempre ocupan el espacio máximo definido por el esquema, aunque el valor real esté vacío. El inconveniente de las columnas de longitud variable radica en que algunas operaciones no son igual de eficaces que en las columnas de longitud fija. Por ejemplo, si una columna de longitud variable es inicialmente pequeña y crece considerablemente después de una actualización, es posible que el registro deba reubicarse. Además, si se realizan actualizaciones con frecuencia, las páginas de datos se fragmentan más con el paso del tiempo. Por lo tanto, es recomendable el uso de las columnas de longitud fija cuando la longitud de los datos no varía demasiado y cuando se realizan actualizaciones con frecuencia.
- **Mantener registros chicos:** El número de filas que una página puede contener depende del tamaño de cada fila. Una página podrá contener más filas si éstas son pequeñas. Así pues, una sola operación de disco realizada en una tabla con filas compactas recuperará más filas y, de este modo, la operación será más efectiva. Asimismo, la caché del motor de almacenamiento tiene capacidad para más filas, lo que permite aumentar potencialmente la tasa de visitas. Las filas compactas también contribuyen a reducir el espacio desaprovechado en las páginas de datos. Esto es más característico de las filas grandes.
- **Bit*:*** SQL Server optimiza el almacenamiento de las columnas de tipo bit. Si una tabla contiene 8 columnas o menos de tipo bit, éstas se almacenan como 1 byte. Si hay entre 9 y 16 columnas de tipo bit, se almacenan como 2 bytes, y así sucesivamente.
- **Decimal, Numeric*:*** Al trabajar con estos tipos de dato es necesario establecer dos valores: **Precisión** (El número total máximo de dígitos decimales que se puede almacenar, tanto a la izquierda como a la derecha del separador decimal. La precisión debe ser un valor comprendido entre 1 y la precisión máxima de 38. La precisión predeterminada es 18) y **escala** (el número máximo de dígitos decimales que se puede almacenar a la derecha del separador decimal. La escala debe ser un valor comprendido entre 0 y la precisión. Sólo es posible especificar la escala si se ha especificado la precisión. La escala predeterminada es 0. Los tamaños de almacenamiento máximo varían, según la precisión.

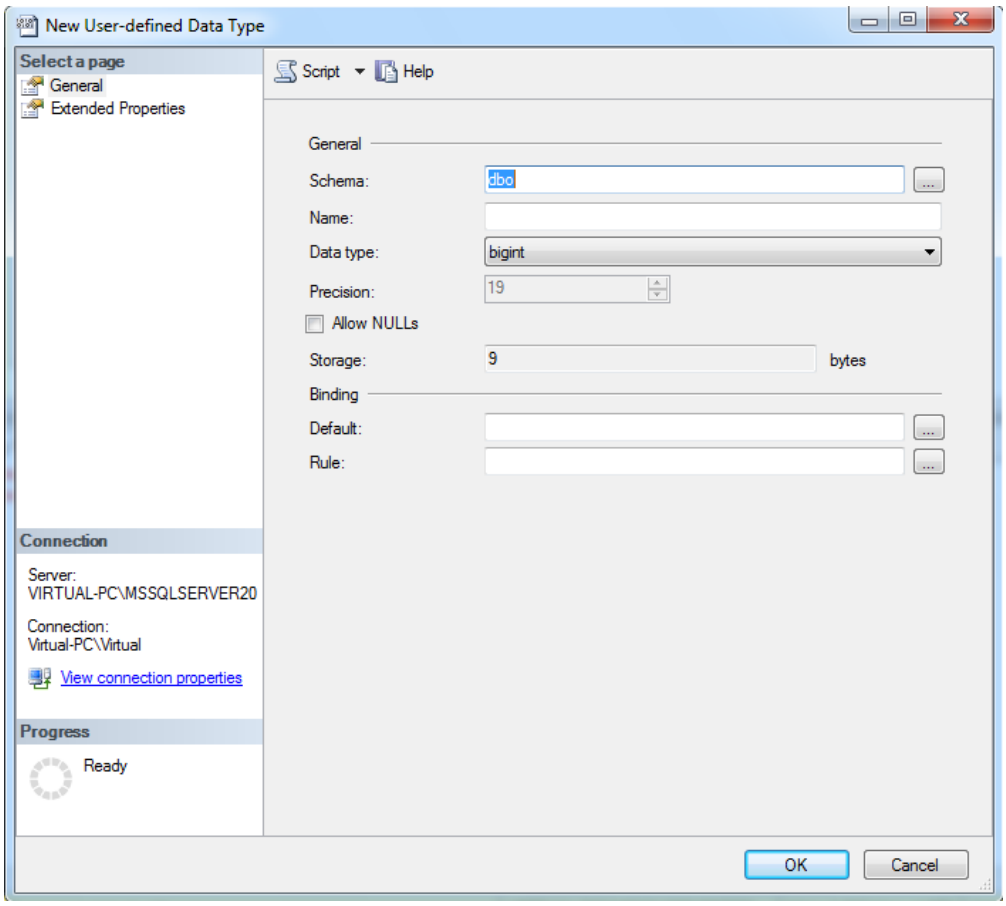
Tipos de Datos de Alias

Los tipos de alias se basan en los tipos de datos del sistema de SQL Server. Se pueden utilizar cuando varias tablas deben almacenar el mismo tipo de datos en una columna y desea asegurarse de que dichas columnas tienen exactamente el mismo tipo de datos, longitud y nulabilidad. Las variables de tabla no admiten los tipos de alias.

Cuando cree un tipo de datos de alias, debe suministrar los parámetros siguientes:

- Nombre
- Tipo de datos del sistema en el que se basa el nuevo tipo de datos.
- Nulabilidad (si el tipo de datos permite o no valores NULL).

Si se crea un tipo de alias en la base de datos **Model**, existirá en todas las nuevas bases de datos definidas por el usuario. Sin embargo, si el tipo de datos se crea en una base de datos definida por el usuario, sólo existirá en esa base de datos.




```
CREATE TYPE dbo.Codigo
FROM char(2)
NULL
```

Los argumentos mínimos a especificar son:

- **Nombre del Tipo**: Es el nombre del tipo de dato. Debe cumplir las reglas de los identificadores. La especificación del nombre del propietario es opcional.

- **Tipo_Dato**: Es el tipo de datos suministrado por SQL Server en el que se basa el tipo de datos de alias.
- **NULL | NOT NULL**: Especifica si el tipo puede contener un valor nulo. Si no se especifica, el valor predeterminado es NULL.

NULL o NOT NULL
NULL indica que el valor es desconocido. Un valor NULL no es lo mismo que un valor cero o vacío. No hay dos valores NULL que sean iguales. La comparación entre dos valores NULL, o entre un valor NULL y cualquier otro valor, tiene un resultado desconocido porque el valor de cada NULL es desconocido.
Si es probable que haya valores NULL almacenados en los datos y no desea que aparezcan valores NULL en los datos, debería crear consultas e instrucciones de modificación de datos que quiten los valores NULL o los transformen en algún otro valor. Por ejemplo, puede usar **ISNULL** para convertir un valor nulo en otro valor.

```
SELECT ISNULL(title,'') FROM Person.Person
```

Conversión de Tipos de Datos
CAST y **Convert** convierten una expresión de un tipo de datos a otro en SQL Server 2008. Convert permite mostrar fechas en diferentes formatos.

Ejemplos:

```
CAST(ListPrice AS varchar(12))
SELECT CONVERT(varchar(8),SYSDATETIME(),112)
```

Tipo de Dato Table
Es un tipo de datos especial que se puede utilizar para almacenar un conjunto de resultados para su procesamiento posterior. table se utiliza principalmente para el almacenamiento temporal de un conjunto de registros.
Se puede declarar como variables de tipo table, definiendo el esquema de la tabla en la declaración.

```
DECLARE @BalanceList TABLE (CustomerID int,
CurrentBalance decimal(18,2));
```

También se puede crear un tipo de dato definido por el usuario de tipo **table**. Estos pueden ser usados como parámetros o como variables.

```
CREATE TYPE dbo.CustomerBalance
AS TABLE (CustomerID int, CurrentBalance decimal(18,2));
GO

DECLARE @BalanceList dbo.CustomerBalance;
```

Luego podrian llenarse

```
INSERT INTO @BalanceList
VALUES (1,10), (2,14.20), (3,18.23)
```

Los parámetros con valores de tabla se pueden usar en modificaciones de datos basados en conjuntos que afectan varias filas mediante la ejecución de una sola instrucción. Por ejemplo, puede seleccionar todas las filas de un parámetro con valores de tabla e insertarlas en una

tabla de base de datos o crear una instrucción de actualización mediante la combinación de un parámetro con valores de tabla y la tabla que desee actualizar.

Ejemplo

```
UPDATE dbo.Categories
SET Categories.CategoryName = ec.CategoryName
FROM dbo.Categories INNER JOIN @tvpEditedCategories AS ec
ON dbo.Categories.CategoryID = ec.CategoryID;

INSERT INTO dbo.Categories (CategoryID, CategoryName)
SELECT nc.CategoryID, nc.CategoryName FROM @tvpNewCategories AS nc;
```

Existen varias limitaciones en los parámetros con valores de tabla:

- No puede pasar parámetros con valores de tabla a funciones definidas por el usuario.
- Los parámetros con valores de tabla sólo se pueden indizar para admitir restricciones UNIQUE o PRIMARY KEY. SQL Server no mantiene estadísticas de parámetros con valores de tabla.
- Los parámetros con valores de tabla son de sólo lectura en el código Transact-SQL. No puede actualizar los valores de columna de las filas de un parámetro con valores de tabla ni insertar ni eliminar filas. Para modificar los datos que se pasan a un procedimiento almacenado o a una instrucción con parámetros de un parámetro con valores de tabla, debe insertar los datos en una tabla temporal o en una variable de tabla.
- No puede usar instrucciones ALTER TABLE para modificar el diseño de los parámetros con valores de tabla.

2- Tablas

Definición

Tras diseñar una base de datos, puede crear las tablas que almacenarán los datos en la base de datos. Normalmente, los datos se almacenan en tablas permanentes; no obstante, también se pueden crear tablas temporales. Las tablas se almacenan en los archivos de base de datos hasta que se eliminan, y están disponibles para cualquier usuario que cuente con los permisos necesarios.

Puede definir hasta 1.024 columnas por tabla. Los nombres de las tablas y de las columnas deben seguir las reglas de los identificadores; tienen que ser únicos dentro de una tabla específica, pero puede utilizar el mismo nombre de columna en distintas tablas de la misma base de datos. También debe definir un tipo de datos para cada columna.

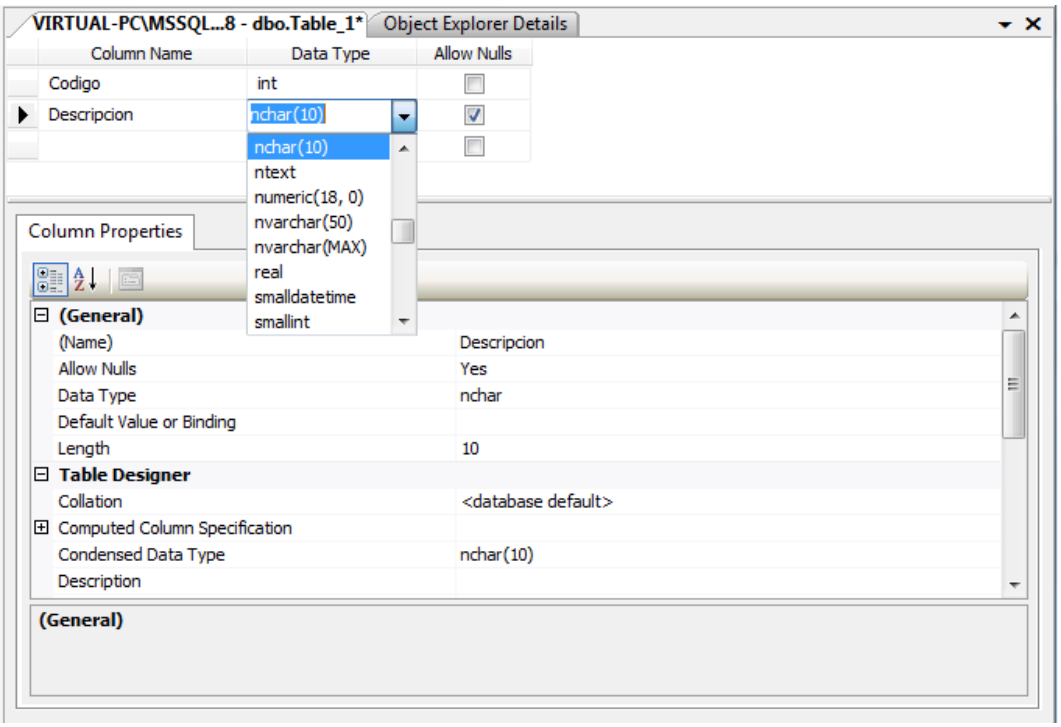
Aunque los nombres de las tablas tienen que ser únicos para cada esquema de una base de datos, puede crear varias tablas con el mismo nombre si especifica distintos esquemas para cada una de ellas.

Las tablas temporales son similares a las permanentes, salvo por el hecho de que las tablas temporales se almacenan en **Tempdb** y se eliminan automáticamente cuando ya no se utilizan.

Hay dos tipos de tablas temporales: locales y globales. Se diferencian entre sí por los nombres, la visibilidad y la disponibilidad. Las tablas temporales locales tienen como primer carácter de sus nombres un solo signo de número (#); sólo son visibles para el usuario de la conexión actual y se eliminan cuando el usuario se desconecta de la instancia de SQL Server. Las tablas temporales globales presentan dos signos de número (##) antes del nombre; son visibles para cualquier usuario después de su creación y se eliminan cuando todos los usuarios que hacen referencia a la tabla se desconectan de la instancia de SQL Server.

Creación

Se puede crear una tabla usando la herramienta SQL Server Management Studio o con la sentencia SQL CREATE TABLE.



Ejemplo:

```
CREATE TABLE Production.Pedidos
(NumPedido int identity NOT NULL,
Fecha datetime NOT NULL,
CodCliente int NOT NULL,
Notas nvarchar(200) NULL)
```

Atributos de las Columnas

- **Nombre**: Nombre de la columna. Debe ser único
- **Tipo de Dato**: Tipo de dato de la columna. Obligatorio
- **Nulabilidad**: Si soporta nulos o no. Si no se especifica se toma el valor predeterminado en NULL ANSI.
- **Longitud**: Algunos tipos de datos tienen una longitud predeterminada, en otros hay que ingresarla
- **Collation**: (Intercalación) Juego de caracteres aceptados en esta columna. Por defecto toma la asociada a la base de datos. Cada intercalación trabaja diferente lo cual puede generar distintos ordenamiento o diferencias en las comparaciones
- **Columnas Calculadas**: No se guardan físicamente. Las columnas calculadas se calculan a partir de una fórmula que puede utilizar otras columnas de la misma tabla. La fórmula puede utilizar columnas no calculada, constantes, funciones, y/o cualquier combinación de estos elementos conectados mediante uno o más operadores.
- **Identity**: Contiene valores secuencial autogenerados para cada registro de la tabla. Es usualmente usada como clave primaria. Su uso mejora la performance, simplifican la programación y generan claves primarias (Primary Keys) chicas.

Propiedad Identity

Crea una columna de identidad en una tabla. Típicamente asociado a un valor entero, provee una generación de valores secuenciales en forma automática. Debe especificar tanto el valor de inicialización como el incremento, o bien ninguno de los dos. Si no se especifica ninguno, el valor predeterminado es (1,1).
Normalmente no puede ingresarse valores en un campo definido como Identity, pero puede usarse **SET IDENTITY_INSERT ON** para permitir ingresar algún valor en estas columnas.
No asegura que el valor sea único.
La función @@**IDENTITY** devuelve el último valor de identidad insertado. **Scope_Identity** devuelve el último valor de identidad insertado en una columna de identidad en el mismo ámbito. Un ámbito es un módulo: un procedimiento almacenado, desencadenador, función o lote.

Columnas Calculadas

Las columnas calculadas se calculan a partir de una expresión que puede utilizar otras columnas de la misma tabla. La expresión puede ser un nombre de columna no calculada, una constante, una función, y cualquier combinación de estos elementos conectados mediante uno o más operadores. La expresión no puede ser una subconsulta.
Si no se especifica lo contrario, las columnas calculadas son columnas virtuales no almacenadas físicamente en la tabla. Sus valores se vuelven a calcular cada vez que se utilizan en una consulta. Utilice la palabra clave **PERSISTED** en las instrucciones CREATE TABLE y ALTER TABLE para almacenar físicamente las columnas calculadas de la tabla.

Ejemplo:

```
CREATE TABLE [dbo].[Table_2](
      [Registro] [int] NOT NULL,
      [Valor1] [int] NOT NULL,
      [Valor2] [int] NULL,
      [Total]  AS ([Valor1]+[Valor2])
) ON [PRIMARY]
```

Modificación y Borrado

Para modificar o borrar una tabla se puede usar la herramienta SQL Server Management Studio o las instrucciones **ALTER TABLE** / **DROP TABLE**.

Al modificar se puede agregar o sacar columnas, cambiar su tipo de dato, modificar restricciones (constraints), habilitar o deshabilitar desencadenadores (triggers), etc.

Al borrar una tabla se borran todos sus datos, índices, desencadenadores (triggers), restricciones (constraints) y permisos. Las vistas y procedimientos almacenados que referencian a dicha tabla no se borran automáticamente sino que deben ser borradas o modificadas manualmente.

Como se organizan los datos

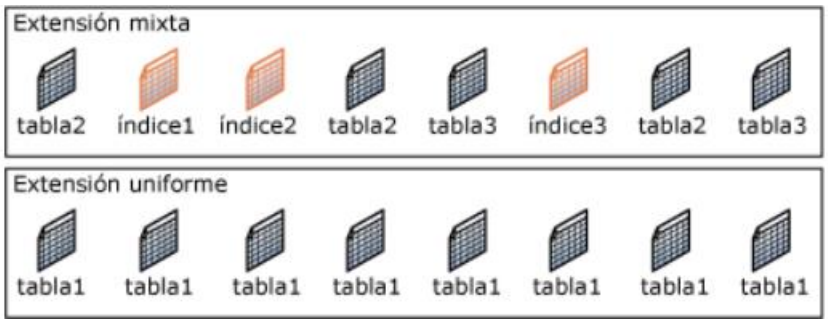
Los datos de una base de datos SQL Server están organizados en **páginas**. Las páginas de un archivo de datos de SQL Server están numeradas secuencialmente, comenzando por cero (0) para la primera página del archivo. Cada archivo de una base de datos tiene un número de identificador único. Para identificar de forma única una página de una base de datos, se requiere el identificador del archivo y el número de la página.

En SQL Server, el tamaño de página es de 8 KB. Esto significa que las bases de datos de SQL Server tienen 128 páginas por megabyte. Cada página empieza con un encabezado de 96 bytes, que se utiliza para almacenar la información del sistema acerca de la página. Esta información incluye el número de página, el tipo de página, el espacio disponible en la página y el Id. de unidad de asignación del objeto propietario de la página.

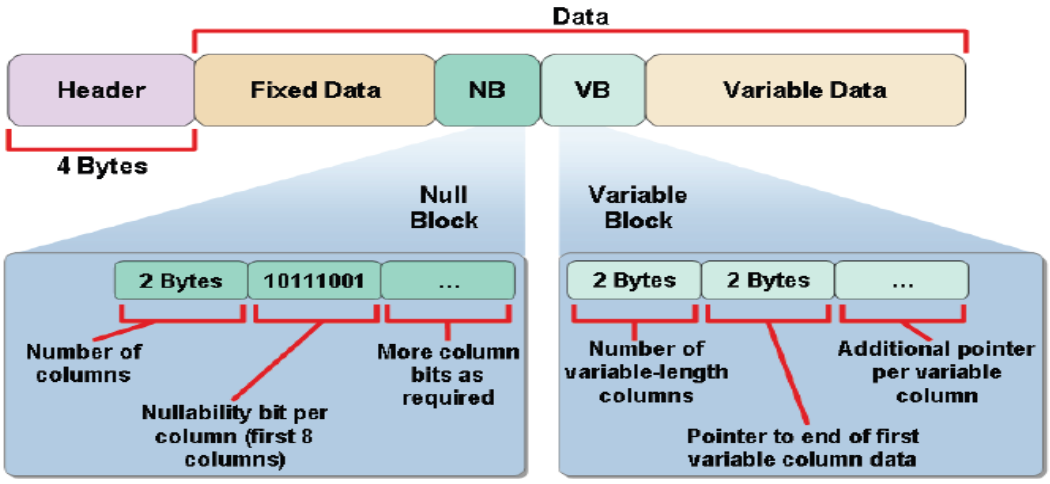
Las **extensiones** (extents) son una colección de ocho páginas físicamente contiguas; se utilizan para administrar las páginas de forma eficaz. Todas las páginas se almacenen en extensiones. Constan de ocho páginas contiguas físicamente, es decir 64 KB. Esto significa que las bases de datos de SQL Server tienen 16 extensiones por megabyte. Para hacer que la asignación de espacio sea eficaz, SQL Server no asigna extensiones completas a tablas con pequeñas cantidades de datos.

SQL Server tiene dos tipos de extensiones:

- Las extensiones uniformes son propiedad de un único objeto; sólo el objeto propietario puede utilizar las ocho páginas de la extensión.
- Las extensiones mixtas, que pueden estar compartidas por hasta ocho objetos. Cada una de las 8 páginas de la extensión puede ser propiedad de un objeto diferente.



Un **registro** contiene un encabezado y una porción para los datos. El encabezado contiene información sobre las columnas del registro, como el puntero a la terminación de la zona de datos fijos o si existen datos de longitud variable en el registro.

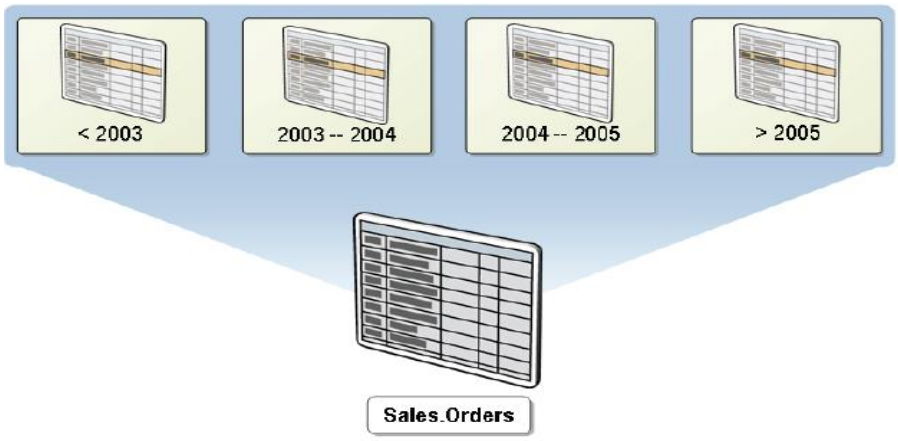


La porción de datos contiene varios elementos:

- **Datos Fijos**: Los datos de longitud fija se guardan en el registro antes que los datos de longitud variable.
- **Bloque Nulo (NB)**: Su longitud es variable. 2 bytes para guardar el número de columnas seguidas en un bitmap nulo indicando si cada una de esas columnas está en nulo o no.
- **Bloque Variable (VB)**: Consiste en 2 bytes que describen cuantas columnas de longitud variable existen en el registro, más 2 bytes por columna que marca el puntero al final de cada una de estas variables. Es omitido si no hay columnas de longitud variable.
- **Datos Variables**: Estas columnas se guardan en la página al final. Si no existen datos de este tipo, no se ocupa espacio en el registro.

3- Tablas con Particiones

Definición
Una tabla con particiones es una tabla cuyos datos están separados horizontalmente en múltiples ubicaciones físicas. Este corte se basa en rango de valores de una sus columnas. El manejo de la ubicación física se hace mediante grupos de archivos (filegroups). Por ejemplo se puede crear una tabla de ventas y después partirla en diferentes grupos de archivos basándose en la fecha de la venta, poniendo las ventas del año en curso en una partición y las ventas de años anteriores en otro grupo de archivos. Esto permite mejorar la performance manteniendo una sola tabla.



Ventajas
La principal razón para implementar este tipo de tablas es poder manejar fácilmente distintos grupos de datos manteniendo una sola tabla. Los beneficios del uso de estas tablas son:

- Permitir manejar copias de seguridad separadas. Al separar por ejemplo los datos en datos en curso e históricos el manejo de copias de seguridad va a ser muy diferente.
- Controlar el tamaño de la tabla. Permite trabajar con tablas más chicas manejando las necesidades de acceso a cada dato.
- Se pueden partir los índices también permitiendo reorganizarlos, optimizarlos y reconstruirlos más rápidamente.
- Búsquedas y uniones más rápidas al trabajar con un árbol de índices más chico.
- Reducción de bloqueos, ya que el bloqueo se frena a nivel de la partición.

Funciones de Partición
Una función de partición especifica cómo se divide la tabla o el índice. La función asigna el dominio a un conjunto de particiones. Para crear una función de partición debe especificar el número de particiones, la columna de partición y el intervalo de valores de columnas de partición para cada partición. Tenga en cuenta que al especificar la columna de partición solamente puede especificar una columna.
Hay que tener en cuenta algunas reglas para armar las particiones. Las columnas calculadas que participan en una función de partición deben marcarse explícitamente como PERSISTED. Todos los tipos de datos válidos para el uso en columnas de índice pueden utilizarse como una columna de partición con la excepción de timestamp. Los tipos de datos ntext, text, image, xml, varchar(max), nvarchar(max) o varbinary(max) no se pueden especificar. Las columnas de tipos de datos de alias y de tipo definido por el usuario CLR (Common Language Runtime) .NET Framework de Microsoft no se pueden especificar.

Las particiones se crean con la sentencia CREATE PARTITION FUNCTION

```
CREATE PARTITION FUNCTION nombre_función_partición (tipo_data_parámetro)
AS RANGE [ LEFT | RIGHT ]
FOR VALUES ( [ valores_límites [ ,...n ] ] )
```

Argumentos:

- **nombre_función_partición*:*** Es el nombre de la función de partición. Debe cumplir las reglas de los identificadores. La especificación del nombre del propietario es opcional.
- **tipo_data_parámetro*:*** Es el tipo de datos de la columna utilizada para la partición. La columna en sí, conocida como columna de partición, se especifica en la instrucción CREATE TABLE o CREATE INDEX.
- **Valores_límites**: Especifica los valores de límite para cada partición de una tabla. Si está vacío, la función de partición asigna la tabla o el índice completos a una sola partición. Sólo es posible utilizar una columna de partición, especificada en una instrucción CREATE TABLE o CREATE INDEX. Es una expresión constante que puede hacer referencia a variables. Éstas incluyen variables o funciones de tipo definido por el usuario y funciones definidas por el usuario. No puede hacer referencia a expresiones de Transact-SQL.

Ejemplo: Este código muestra una función de partición llamada pf_FechaOrden que arma cuatro particiones. La primera para fechas anteriores a Enero de 2010, la segunda para las fechas comprendidas entre Enero del 2010 y fines de Diciembre del 2010, la tercera para fechas comprendidas entre Enero del 2011 y fines de Diciembre del 2011 y la última para las fechas posteriores a Enero del 2012.

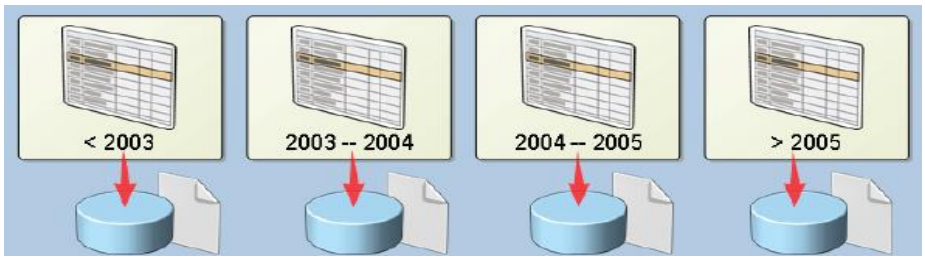
```
CREATE PARTITION FUNCTION pf_FechaOrden (datetime)
AS RANGE RIGHT
FOR VALUES ('01/01/2010', '01/01/2011', '01/01/2012')
```

Esquemas de particiones

Un esquema de particiones asigna las particiones creadas por una función de partición a un conjunto de grupos de archivos (filegroups) que el usuario ha definido. Al crear un esquema de particiones se definen los grupos de archivos en los que se asignan las particiones de tabla basándose en los parámetros de la función de partición. Debe especificar suficientes grupos de archivos para albergar todas las particiones. Puede especificar que todas las particiones se asignen a un grupo de archivos diferente, que algunas particiones se asignen a un solo grupo de archivos o que todas las particiones se asignen a un único grupo de archivos. También puede especificar grupos de archivos adicionales "no asignados" si posteriormente desea agregar más particiones. En este caso, SQL Server marca uno de los grupos de archivos con la propiedad NEXT USED. Esto significa que el grupo de archivos albergará la siguiente partición que se agregue.
Un esquema de particiones sólo puede utilizar una función de partición. Sin embargo, una función de partición puede participar en más de un esquema de particiones.

Ejemplo: Crea un esquema de particiones llamado ps_FechaOrden para la función pf_FechaOrden. Cada una de las 4 particiones es asignada a un grupo de archivos diferentes por posición a los grupos de archivo fg1, fg2, fg3 y fg4. Fg5 es un ejemplo de asignación de grupo de archivo para usos posteriores.

```
CREATE PARTITION SCHEME ps_FechaOrden
AS PARTITION pf_FechaOrden
TO (fg1, fg2, fg3, fg4, fg5)
```

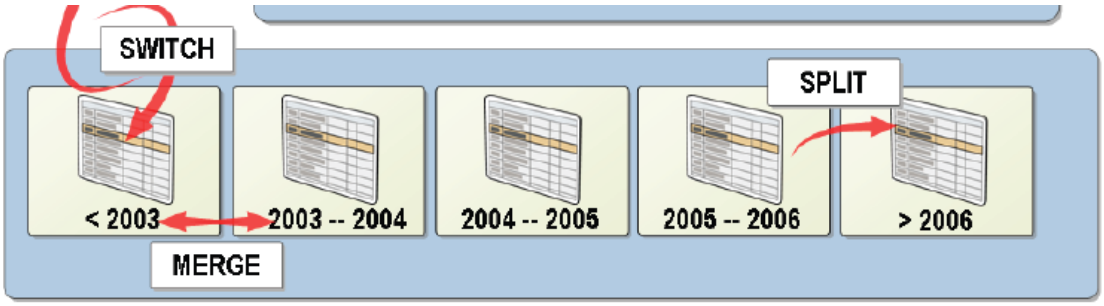


Crear Tabla con Particiones

Ejemplo:

```
CREATE TABLE [dbo].[OrderDetails](
      [OrderID] [int] NOT NULL,
      [LineNumber] [smallint] NOT NULL,
      [ProductID] [int] NULL,
      [UnitPrice] [money] NULL,
      [OrderQty] [smallint] NULL,
      [ReceivedQty] [float] NULL,
      [RejectedQty] [float] NULL,
      [OrderDate] [datetime] NOT NULL,
      [DueDate] [datetime] NULL,
      [ModifiedDate] [datetime] NOT NULL
            CONSTRAINT [OrderDetailsModifiedDateDFLT]
            DEFAULT (getdate()),
      [LineTotal]  AS (([UnitPrice]*[OrderQty])),
      [StockedQty]  AS (([ReceivedQty]-[RejectedQty]))
      ) ON ps_FechaOrden (OrderDate)
```

Que operaciones pueden realizarse sobre una tabla con particiones



- **SWITCH**: Permite transferir subconjuntos de datos rápida. Puede asignar una tabla como partición a una tabla con particiones existente, puede cambiar una partición de una tabla con particiones a otra, puede volver a asignar una partición para crear una sola tabla.

```
      ALTER TABLE dbo.PartitionedTransactions
      SWITCH PARTITION 1
            TO dbo.TransactionArchive
```

- **MERGE**: Quita una partición y mezcla cualquier valor que exista en la partición en una de las particiones restantes. RANGE (valor límite) debe ser un “valor límite” existente, en el que se mezclan los valores de la partición quitada. El grupo de archivos que originalmente contenía la partición con ese valor límite se quita del esquema de partición, a menos que lo utilice una partición restante o que esté marcado con la propiedad NEXT USED.

```
ALTER PARTITION FUNCTION pf_FechaOrden()
MERGE RANGE ('01/01/2010')
```

- **SPLIT**: Agrega una partición a la función de partición. El argumento “valor límite” determina el intervalo de la nueva partición y debe diferir de los intervalos de límites existentes de la función de partición.

```
ALTER PARTITION FUNCTION pf_FechaOrden()
SPLIT RANGE ('01/01/2012')
```


BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

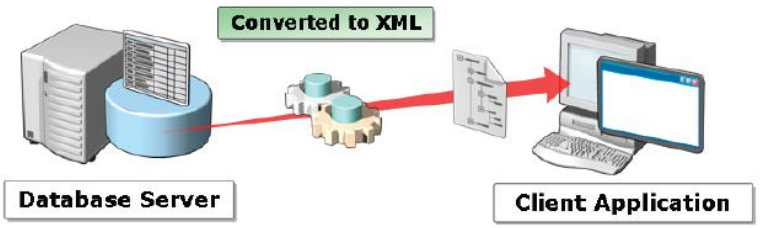
Módulo 6

Usando XML

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Uso de FOR XML

Trabajar con datos en formato XML es hoy un requerimiento de muchas aplicaciones. Se presentan muchos casos en los cuales los desarrolladores deben transformar datos entre el formato XML y el formato relacional, también se presenta la necesidad de guardar o manipular datos en ese XML. Para la recuperación de datos en formato XML dentro de SQL Server 2008 se usa la cláusula XML FOR.



Cláusula FOR XML
Permite ejecutar consultas SQL para obtener resultados en formato XML en lugar de conjuntos de filas estándar. Estas consultas pueden ejecutarse directamente o desde procedimientos almacenados y funciones definidas por el usuario. Para obtener los resultados directamente, utilice primero la cláusula FOR XML de la instrucción SELECT. A continuación, dentro de la cláusula FOR XML, especifique un modo de XML: RAW, AUTO, EXPLICIT o PATH.

```
FOR XML
{  { RAW [ ( 'Nombre_Elemento' ) ] | AUTO }
[
<CommonDirectives>
[ , { XMLDATA | XMLSCHEMA [ ( 'TargetNameSpaceURI' ) ] } ]
[ , ELEMENTS [ XSINIL | ABSENT ]
]
| EXPLICIT
[
<Directivas>
[ , XMLDATA ]
]
| PATH [ ( 'Nombre_Elemento' ) ]
[
<Directivas>
[ , ELEMENTS [ XSINIL | ABSENT ] ]
]}
<Directivas> ::=
[ , BINARY BASE64 ]
[ , TYPE ]
[ , ROOT [ ( 'Nombre_Root' ) ] ]
```

Argumentos

- **RAW**: genera un único elemento <row> por cada fila del conjunto de filas devuelto por la instrucción SELECT. Para generar una jerarquía XML se pueden escribir consultas FOR XML anidadas

- **AUTO**: devuelve los resultados de la consulta como elementos XML anidados. Genera jerarquías sencillas.
- **EXPLICIT**: concede un mayor control de la forma del XML. Es posible mezclar atributos y elementos con total libertad para decidir la forma del XML.
- **PATH**: proporciona la flexibilidad del modo EXPLICIT de una manera más sencilla.
- **ELEMENTS**: Devuelve las columnas como elementos y no como atributos para las modos RAW, AUTO y PATH.
- **BINARY BASE64**: Devuelve los datos binarios como binarios en base 64.
- **ROOT**: Agrega un elemento de primer nivel en el XML resultante. Las devoluciones de SQL Server 2008 en formato XML por defecto no crean este primer elemento, el cual es parte de los requisitos de un documento XML bien formado. Se puede especificar el nombre para este elemento.
- **TYPE**: Devuelve la consulta como tipo de dato **xml**.
- **XMLDATA**: Devuelve un esquema XDR.
- **XMLSCHEMA**: Devuelve un esquema XSD.

Consultas en Modo RAW
El XML devuelto cumple con las siguientes características:

- Cada registro es representado con un elemento llamado **<row>** u otro nombre que se proporciona de modo opcional
- Las columnas se representan como atributos con el mismo nombre excepto que se especifique la cláusula **ELEMENTS**
- Soporta consultas que usan GROUP BY.

Ejemplos:

```
SELECT Cust.CustomerID CustID, SalesOrderID
FROM Sales.Customer Cust INNER JOIN Sales.SalesOrderHeader [Order]
ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML RAW
```

El resultado es un XML fragmentado con un elemento llamado <row> por registro y los campos como atributos.

<row CustID="1" SalesOrderID="43860"/>
<row CustID="1" SalesOrderID="44501"/>
<row CustID="1" SalesOrderID="45283"/>
.....

Si agrega la cláusula ELEMENTS los atributos del primer ejemplo pasan a ser elementos:

```
SELECT Cust.CustomerID CustID, SalesOrderID
FROM Sales.Customer Cust INNER JOIN Sales.SalesOrderHeader [Order]
ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML RAW, ELEMENTS
```

El resultado es:

<row><CustID>1</CustID><SalesOrderID>43860</SalesOrderID></row>
<row><CustID>1</CustID><SalesOrderID>44501</SalesOrderID></row>

.....

Si se agrega la cláusula ROOT se consigue un documento bien formato. En el ejemplo también se le da un nombre a los elementos usando la cláusula RAW.

```
SELECT Cust.CustomerID CustID, SalesOrderID
FROM Sales.Customer Cust INNER JOIN Sales.SalesOrderHeader [Order]
ON Cust.CustomerID = [Order].CustomerID
ORDER BY Cust.CustomerID
FOR XML RAW('Orden'), ROOT('Ordenes')
```

El resultado es:

<Ordenes>
<Orden CustID="1" SalesOrderID="43860"/>
<Orden CustID="1" SalesOrderID="44501"/>
<Orden CustID="1" SalesOrderID="45283"/>
....
</Ordenes>

Consultas en Modo AUTO

El modo AUTO devuelve los resultados de la consulta como elementos XML anidados. Esto no ofrece un gran control sobre la forma del XML generado a partir del resultado de una consulta. Las consultas en modo AUTO son útiles si desea generar jerarquías sencillas. El XML devuelto cumple con las siguientes características:

- Cara registro es representado como un elemento con el mismo nombre de la tabla
- Las columnas se representan como atributos con el mismo nombre excepto que se especifique la cláusula ELEMENTS
- Cada JOIN de la consulta es un elemento anidado. Para que el resultado este bien organizado es aconsejable usar la cláusula ORDER BY para devolver la información en el orden correcto
- No soporta consultas que incluyan la cláusula GROUP BY, excepto que los datos devueltos sean de una consulta a una vista que la usa.

Ejemplos:

```
SELECT Cust.CustomerID, OrderHeader.CustomerID,
OrderHeader.SalesOrderID, OrderHeader.Status
FROM Sales.Customer Cust INNER JOIN Sales.SalesOrderHeader OrderHeader
ON Cust.CustomerID = OrderHeader.CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO
```

El resultado es:

<Cust CustomerID="1">
<OrderHeader CustomerID="1" SalesOrderID="43860" Status="5"/>
<OrderHeader CustomerID="1" SalesOrderID="44501" Status="5"/>
.....
</Cust>
<Cust CustomerID="5">
<OrderHeader CustomerID="5" SalesOrderID="65307" Status="5"/>
<OrderHeader CustomerID="5" SalesOrderID="71890" Status="5

.....
</Cust>

Si agrega la cláusula ELEMENTS los atributos del primer ejemplo pasan a ser elementos:

```
SELECT Cust.CustomerID, OrderHeader.CustomerID,
OrderHeader.SalesOrderID, OrderHeader.Status
FROM Sales.Customer Cust INNER JOIN Sales.SalesOrderHeader OrderHeader
ON Cust.CustomerID = OrderHeader.CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO, ELEMENTS
```

El resultado es:

```
<Cust>
        <CustomerID>1</CustomerID>
                <OrderHeader>
                        <CustomerID>1</CustomerID>
                        <SalesOrderID>43860</SalesOrderID>
                        <Status>5</Status>
                </OrderHeader>
                <OrderHeader>
                        <CustomerID>1</CustomerID>
                        <SalesOrderID>44501</SalesOrderID>
                        <Status>5</Status>
                </OrderHeader>
        ......
</Cust>
......
```

Si agrega la cláusula ROOT:

```
SELECT Cust.CustomerID, OrderHeader.CustomerID,
OrderHeader.SalesOrderID, OrderHeader.Status
FROM Sales.Customer Cust INNER JOIN Sales.SalesOrderHeader OrderHeader
ON Cust.CustomerID = OrderHeader.CustomerID
ORDER BY Cust.CustomerID
FOR XML AUTO, ELEMENTS, ROOT('Ordenes')
```

El resultado es:

```
<Ordenes>
        <Cust>
                <CustomerID>1</CustomerID>
                        <OrderHeader>
                                <CustomerID>1</CustomerID>
                                <SalesOrderID>43860</SalesOrderID>
                                <Status>5</Status>
                        </OrderHeader>
                        <OrderHeader>
                                <CustomerID>1</CustomerID>
                                <SalesOrderID>44501</SalesOrderID>
                                <Status>5</Status>
                        </OrderHeader>
                ......
```

```
        </Cust>
        <Cust>
                .............
        </Cust>
</Ordenes>
```

Consultas en Modo EXPLICIT

Los modos RAW y AUTO no proporcionan demasiado control sobre la forma del XML generado a partir del resultado de una consulta. Sin embargo, el modo EXPLICIT ofrece la máxima flexibilidad para generar el XML que se desee a partir del resultado de una consulta.
La consulta en modo EXPLICIT debe escribirse de una determinada manera para poder especificar explícitamente la información adicional sobre el XML requerido, como el anidamiento esperado en el XML, como parte de la propia consulta. Dependiendo del XML que se solicite, la escritura de consultas en modo EXPLICIT puede resultar complicada. Tal vez, una alternativa más sencilla que escribir consultas en modo EXPLICIT sea usar el modo PATH con anidamiento.

En primer lugar, la consulta debe crear las dos columnas de metadatos siguientes:

- La primera columna debe proporcionar el número de etiqueta, el tipo de entero, del elemento actual, y el nombre de la columna debe ser **Tag**. La consulta debe proporcionar un número de etiqueta único para cada elemento que se vaya a construir a partir del conjunto de filas.
- La segunda columna debe proporcionar un número de etiqueta del elemento primario, y el nombre de la columna debe ser **Parent**.
- De este modo, las columnas **Tag** y **Parent** ofrecen información sobre la jerarquía.

Los valores de estas columnas de metadatos, junto con la información de los nombres de columna, se usan para generar el XML deseado. Tenga en cuenta que la consulta debe proporcionar los nombres de columna de una manera determinada. Observe también que un valor 0 ó NULL en la columna **Parent** indica que el elemento correspondiente no tiene uno primario. El elemento se agrega al XML como elemento de nivel superior.
Al escribir consultas en modo EXPLICIT, los nombres de columna del conjunto de filas resultante se deben especificar con este formato:

Nombre Elemento ! Tag ! Nombre Atributo ! Directiva

- **Nombre Elemento**: Es el identificador genérico resultante del elemento. Por ejemplo, si se especifica Customers , se genera el elemento <Customers>.
- **Tag**: Es un valor de etiqueta único asignado a un elemento.
- **Nombre Atributo**: Proporciona el nombre del atributo que se va a crear.
- **Directiva**: Es opcional y se puede usar para proporcionar información adicional para generar el XML. Por ejemplo la directiva **Element** generar un elemento en vez de un atributo.

Para comprender cómo se procesa la tabla universal generada por una consulta para obtener el XML resultante, suponga que ha escrito una consulta que genera esta tabla universal:

Tag	Parent	CustomeriD!1!Cid	CustomeriD!1!Name	Order!2!Id	Order!2!Date	OrderDetail!3!id!Id	OrderDetail!3!id!Ref
1	Nulo	C1	Maria	Nulo	Nulo	Nulo	Nulo
2	1	C1	Nulo	1	1/8/2008	Nulo	Nulo

3	2	C1	Nulo	1	Nulo	OD1	P1
3	2	C1	Nulo	1	Nulo	OD2	P2
2	1	C1	Nulo	2	2/8/2008	Nulo	Nulo

Las dos primeras columnas son **Tag** y **Parent**. Estos valores determinan la jerarquía. Los nombres de columna se han especificado de una manera determinada, como se describe más adelante en este tema.
Al generar el XML a partir de esta tabla universal, se crea una partición vertical de los datos de la tabla en grupos de columnas. La agrupación se determina en función del valor de **Tag** y los nombres de columna. Al crear el XML, la lógica de procesamiento selecciona un grupo de columnas para cada fila y construye un elemento. En este ejemplo, se observa lo siguiente:

- Para el valor 1 de la columna **Tag** en la primera fila, las columnas cuyos nombres incluyen el mismo número de etiqueta, Customer!1!cid y Customer!1!name, forman un grupo. Estas columnas se utilizan para procesar la fila, y se observa que la forma del elemento generado es <Customer id=... name=...>. El formato del nombre de columna se describe más adelante en este tema.
- Para las filas con valor 2 en la columna **Tag**, las columnas Order!2!id y Order!2!date forman un grupo que se usa después para construir elementos, <Order id=... date=... />.
- Para las filas con valor 3 en la columna **Tag**, las columnas OrderDetail!3!id!id y OrderDetail!3!pid!idref forman un grupo. Cada una de estas filas genera un elemento, <OrderDetail id=... pid=...>, a partir de estas columnas.

Tenga en cuenta que, al generar la jerarquía en XML, las filas se procesan por orden. La jerarquía XML se determina como se indica a continuación:
La primera fila especifica el valor 1 en **Tag** y el valor NULL en **Parent**. Por lo tanto, el elemento correspondiente, <Customer>, se agrega al XML como elemento de nivel superior. La segunda fila identifica el valor 2 en **Tag** y el valor 1 en **Parent**. Por lo tanto, el elemento, <Order>, se agrega como elemento secundario de <Customer>. Las dos filas siguientes identifican el valor 3 en **Tag** y el valor 2 en **Parent**. Por lo tanto, los dos elementos <OrderDetail> se agregan como elementos secundarios de <Order>.

El resultado es el siguiente:

```
<Customer cid="C1" name="Janine">
        <Order id="O1" date="1/20/1996">
                <OrderDetail id="OD1" pid="P1"/>
                <OrderDetail id="OD2" pid="P2"/>
        </Order>
        <Order id="O2" date="3/29/1997">
</Customer>
```

En resumen, los valores de las columnas de metadatos **Tag** y **Parent**, la información proporcionada en los nombres de columna y el orden correcto de las filas generan el XML deseado al utilizar el modo EXPLICIT.

Ejemplo:

```
SELECT 1    as Tag, NULL as Parent,
      e.BusinessEntityID as [Employee!1!EmpID],
      NULL as [Name!2!FName],
      NULL as [Name!2!LName]
FROM HumanResources.Employee E INNER JOIN Person.Person  C
```

```
ON  E.BusinessEntityID = C.BusinessEntityID
UNION ALL
SELECT 2 as Tag, 1 as Parent,
            e.BusinessEntityID,
            FirstName,
            LastName
FROM HumanResources.Employee E INNER JOIN Person.Person C
ON  E.BusinessEntityID = C.BusinessEntityID
ORDER BY [Employee!1!EmpID],[Name!2!FName]
FOR XML EXPLICIT
```

El resultado es:

```
<Employee EmpID="1">
        <Name FName="Guy" LName="Gilbert"/>
</Employee>
<Employee EmpID="2">
        <Name FName="Kevin" LName="Brown"/>
</Employee>
....
```

Consultas en Modo PATH

El modo PATH produce XML personalizados usando **XPath** para organizar elementos y atributos. Esto permite generar XML complejos sin las complejidad del modo EXPLICIT. Tenga en cuenta las siguientes características de XPath:

- Los nodos XML en un árbol están expresados como rutas, separados por barras (/)
- Los atributos están indicados usando una arroba (@)
- Las rutas relativas pueden ser indicadas con un punto (.) para representar el nodo en curso o con doble punto (..) para representar los nodos padres

Ejemplo:

```
SELECT EmployeeID "@EmpID",
FirstName "EmpName/First",LastName "EmpName/Last"
FROM Person.Contact INNER JOIN HumanResources.Employee
ON Person.Contact.ContactID = Employee.ContactID
FOR XML PATH('Empleados')
```

El resultado es:

```
<Empleados EmpId=”1”>
        <EmpName>
                <First>Guy</First>
                <Last>Gilbert</Last>
        </EmpName>
</Empleados>
<Empleados EmpID="2">
        <EmpName>
                <First>Kevin</First>
                <Last>Brown</Last>
        </EmpName>
</Empleados>
.....
```

XML Anidado
Permite representar relaciones Padre/Hijo como jerarquías XML. Hay varias maneras de generar XML anidado usando la cláusula FOR XML:

- Generando consultas con JOIN y el modo AUTO.
- Especificando **TYPE** en un subconsulta para generar un tipo de dato xml.
- Combinando tablas universales usando UNION ALL en el modo EXPLICIT

Como usar los modos AUTO y EXPLICIT ya fueron desarrollados en este módulo.

Ejemplo uso de **TYPE**:

```
SELECT Name as CategoryName,
(SELECT Name as SubCategoryName
FROM Production.ProductSubCategory SubCategory
WHERE SubCategory.ProductCategoryID=Category.ProductCategoryID
FOR XML AUTO, TYPE, ELEMENTS)
FROM Production.ProductCategory Category
FOR XML AUTO
```

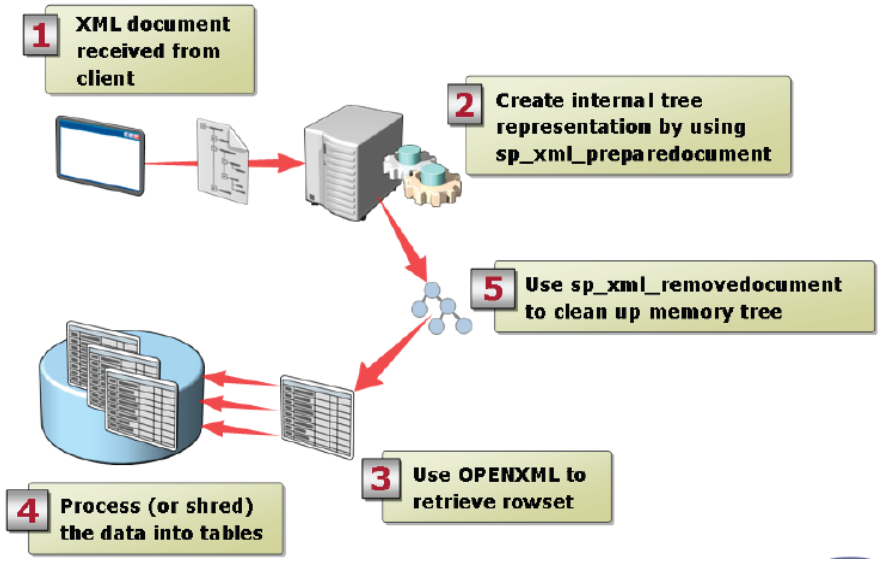
El resultado es:

```
<Category CategoryName="Accessories">
        <SubCategory>
                <SubCategoryName>Bike Racks</SubCategoryName>
        </SubCategory>
        <SubCategory>
                <SubCategoryName>Bike Stands</SubCategoryName>
        </SubCategory>
</Category>
```

2- Usando OPENXML

Definición
OPENXML permite procesar un documento XML y generar un conjunto de registro. El proceso consiste en varios pasos:

- Recibir el documento XML: Por ejemplo como parámetro de un procedimiento almacenado.
- Generar una representación interna en forma árbol: Usar el procedimiento almacenado de sistema **sp_xml_preparedocument** para parsear el documento y transformarlo en una estructura de árbol en memoria. Este árbol es similar a la representación **DOM** (Document Object Model). El documento debe estar bien formado para poder generar este árbol
- Generar un conjunto de registros desde el árbol: Usar la función **OPENXML** para generar el conjunto de registros en memoria. Usar **XPath** para especificar los nodos del árbol a ser convertidos en conjunto de registros
- Procesar los datos desde el conjunto de registros de la misma manera que se usan los otros conjuntos de registros.
- Destruir el árbol interno usando el procedimiento almacenado de sistemas **sp_xml_removedocument**



Procedimiento Almacenado sp_xml_preparedocument
Lee el texto del XML proporcionado como entrada, analiza el texto y proporciona el documento analizado en un estado apto para su uso. Este documento analizado es una representación en árbol de varios nodos en el documento XML: elementos, atributos, texto, comentarios, etc.
Tiene tres parámetros:

- **Xmltext**: Texto original en formato XML para ser procesado. Puede aceptar los tipos de dato nchar, varchar, nvarchar, text, ntext, o xml.
- **Hdoc**: Es el identificador del documento recién creado. Es un número entero.
- **xpath_namespaces*:*** (Opcional) Especifica las declaraciones de espacio de nombres que se utilizan en las expresiones XPath de fila y columna de OPENXML.

Ejemplo:

```
CREATE PROCEDURE ProcessOrder @doc xml
AS
DECLARE @hdoc integer
EXEC sp_xml_preparedocument @hdoc OUTPUT, @doc
```

Procedimiento Almacenado sp_xml_removedocument
Quita la representación interna del documento XML especificado por el identificador del documento y lo convierte en no válido.

```
EXEC sp_xml_removedocument @hdoc
```

Sintaxis OPENXML

OPENXML(idoc int [in] , patron_registro nvarchar [in] , [flags byte [in]])
[WITH (Declaracion_Esquema | Nombre_Tabla)]

Argumentos

- **Idoc**: Es el identificador del documento de la representación interna de un documento XML. La representación interna de un documento XML se crea llamando al procedimiento almacenado **sp_xml_preparedocument**.
- **patron_registro**: Es el patrón **XPath** utilizado para identificar los nodos (en el documento XML cuyo identificador se pasa en el parámetro idoc) que se van a procesar como filas.
- **Flags**: Indica la asignación que debe utilizarse entre los datos XML y el conjunto de filas relacional, y cómo debe llenarse la columna de desbordamiento; flags es un parámetro de entrada opcional, y puede tomar uno de los siguientes valores:
  - 0: Usa el mapa predeterminado
  - 1: Devuelve atributos
  - 2: Devuelve elementos
  - 8: Devuelve atributos y elementos
- **Declaracion_Esquema**: Es la definición de esquema de la forma del conjunto de registro que va a generar:
  - **Nombre Columna**: Es el nombre de columna en el conjunto de filas.
  - **Tipo de dato de la Columna**: Es el tipo de datos SQL Server de la columna en el conjunto de filas. Si los tipos de columna son distintos del tipo de datos xml subyacente del atributo, se producirá una conversión de tipos.
  - **Patrón de la Columna**: Es un parámetro opcional, un patrón XPath general que describe la forma de asignar los nodos XML a las columnas. Si no se especifica el patrón, se realiza la asignación predeterminada por el argumento **flags**
  - **MetaPropiedad**: Es una de las metapropiedades que proporciona OPENXML. Si se especifica MetaProperty, la columna contiene información que proporciona la metapropiedad. Las metapropiedades permiten extraer información (como información sobre el espacio de nombres y la posición relativa) acerca de nodos XML. Esto proporciona más información que la que se puede ver en la representación de texto.
- **Nombre_Tabla**: Es el nombre de tabla que puede proporcionarse (en lugar de la declaracion_esquema) si ya existe una tabla con el esquema deseado y no se requiere patrones de columna.

Ejemplo:
Teniendo el siguiente documento XML:

```
<Customer CustomerID="1" CustomerType="S">
        <Order SalesOrderID="43860" Status="5"
        OrderDate="2001-08-01T00:00:00">
                <OrderDetail ProductID="761" Quantity="2"/>
                <OrderDetail ProductID="770" Quantity="1"/>
        </Order>
</Customer>
```

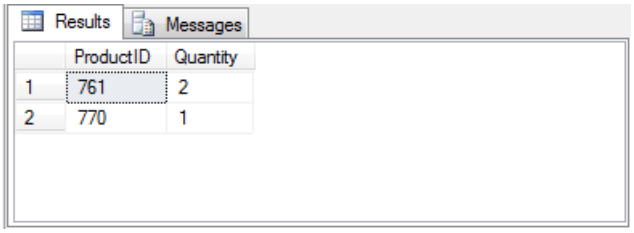
Aplicamos la siguiente consulta:

```
SELECT * FROM OPENXML (@idoc, '/Customer/Order/OrderDetail')
WITH (ProductID int, Quantity int)
```

El ejemplo completo sería:

```
Declare @doc xml
set @doc='<Customer CustomerID="1" CustomerType="S">
            <Order SalesOrderID="43860" Status="5"
                  OrderDate="2001-08-01T00:00:00">
                  <OrderDetail ProductID="761" Quantity="2"/>
                  <OrderDetail ProductID="770" Quantity="1"/>
            </Order>
          </Customer>'
Declare @h int
EXEC sp_xml_preparedocument @h OUTPUT, @doc
SELECT * FROM OPENXML (@h, '/Customer/Order/OrderDetail')
WITH (ProductID int, Quantity int)
EXEC sp_xml_removedocument @h
```

El resultado es:



Trabajando con Espacios de Nombre XML (Namespaces)

Muchos documentos XML utilizan espacios de nombres para ayudar a las aplicaciones a reconocer elementos y evitar confusiones en elementos que tienen el mismo nombre del elemento pero este se usa para distintos propósitos.

Supongamos estos dos documentos XML:

```
<student>
        <id>3235329</id>
        <name>Jeff Smith</name>
        <language>C#</language>
        <rating>9.5</rating>
</student>

<student>
        <id>534-22-5252</id>
```

```
        <name>Jill Smith</name>
        <language>Spanish</language>
        <rating>3.2</rating>
</student>
```

Obviamente cualquier persona puede detectar las diferencias entre ambos documentos, pero una aplicación no. El elemento <Language> esta usado con un significado diferente para cada estudiante teniendo en cuenta el tipo de estudiante, si es un estudiante de sistemas se refiere al lenguaje de programación, si es un estudiante estándar se refiere a su idioma. La solución para este tipo de situaciones es usar espacios de nombre XML.

Ejemplo:

```
<d:student xmlns:d='http://www.develop.com/student'
xmlns:i='urn:schemas-develop-com:identifiers'
xmlns:p='urn:schemas-develop-com:programming-languages'>
      <i:id>3235329</i:id>
      <name>Jeff Smith</name>
      <p:language>C#</p:language>
      <d:rating>9.5</d:rating>
</d:student>
```

De esta manera se podría estar diferenciando el elemento <language> como idioma o como lenguaje de programación, ya que esa información es provista por el espacio de nombre. Es posible asociar más de un espacio de nombre a un documento.

3- Usando el tipo de dato xml - XQuery

Tipo de dato XML
El tipo de datos **xml** es un tipo de dato nativo desde el SQL Server 2005 que permite contener documentos XML. Su tamaño máximo llega a 2GB. La posibilidad de guardar información en formato XML dentro de la base de datos tiene varias ventajas para las aplicaciones:

- Información estructurada y semi estructurada están guardadas en la misma ubicación haciendo más fácil su administración.
- Aprovechar las ventajas de un almacenamiento de datos optimizado y que trabaja en un entorno de consultas.

Un documento XML guardado en una columna, parámetro o variable de tipo xml puede ser tratado como si fueran el documento original, excepto que los espacios en blanco, prefijos de espacios de nombre, orden de los atributos y las declaraciones XML no son retenidas. SQL Server provee la siguiente funcionalidad para este tipo de dato:

- **Indexado**: Las columnas de este tipo de dato pueden ser indexada y también soporta índices de texto (fulltext index).
- Consulta de datos XQuery-based: Se pueden usar para extraer datos del XML usando expresiones **XQuery**.
- Expresiones **XQuery**.
- Modificación de datos XQuery-based
- **TYPED** xml: Un XML con tipo es un XML que está asociado a un esquema. Permite validar el documento contra su esquema.

XQuery
Se utiliza para realizar consultas sobre el tipo de datos xml. XQuery se basa en el lenguaje para consultas **XPath** existente, con un incremento de la compatibilidad para lograr una mejor iteración, mejores resultados de la ordenación y la posibilidad de generar el XML necesario. XQuery opera según el modelo de datos XQuery. Se trata de una abstracción de documentos XML, y los resultados de XQuery pueden tener tipo o no tenerlo.

Sentencias FLWOR
FLWOR es el acrónimo de **FOR**, **LET**, **WHERE**, **ORDER BY** y **RETURN**. Una instrucción FLWOR está formada por:

- Una o varias cláusulas **FOR** que enlazan una o varias variables de iteración a secuencias de entrada.
- Una cláusula **LET** opcional. Esta cláusula asigna un valor a la variable para una iteración concreta. La expresión asignada puede ser una expresión XQuery, como una expresión XPath, y puede devolver una secuencia de nodos o una secuencia de valores atómicos. Las secuencias de valores atómicos se pueden construir con literales o con funciones constructoras.
- Una variable de iteración. Esta variable puede tener una aserción de tipo opcional mediante la palabra clave **as**.
- Una cláusula opcional **WHERE**. Esta cláusula aplica un predicado de filtro en la iteración.
- Una cláusula opcional **ORDER BY**.
- Una expresión **RETURN**. Construye el resultado de la instrucción FLWOR.

Ejemplo:

```
DECLARE @x xml
```

```
SET @x='<ManuInstructions ProductModelID="1"
ProductModelName="SomeBike" >
      <Location LocationID="L1" >
            <Step>Manu step 1 at Loc 1</Step>
            <Step>Manu step 2 at Loc 1</Step>
            <Step>Manu step 3 at Loc 1</Step>
      </Location>
      <Location LocationID="L2" >
            <Step>Manu step 1 at Loc 2</Step>
            <Step>Manu step 2 at Loc 2</Step>
            <Step>Manu step 3 at Loc 2</Step>
      </Location>
      </ManuInstructions>'
SELECT @x.query('
      for  $step in /ManuInstructions/Location[1]/Step
      return string($step)')
```

El resultado es:

Manu step 1 at Loc 1
Manu step 2 at Loc 1
Manu step 3 at Loc 1

Método Query
Se usa para extraer XML de un campo o variable de tipo de dato xml.

Ejemplo:

```
DECLARE @Doc xml
SET @Doc = '<Root>
      <ProductDescription ProductID="1" ProductName="Road Bike">
            <Features>
                  <Warranty>1 year parts and labor</Warranty>
                  <Maintenance>3 year parts and labor extended
                        maintenance is available</Maintenance>
            </Features>
      </ProductDescription>
      </Root>'
SELECT @Doc.query('/Root/ProductDescription/Features')
```

El resultado es:

<Features>
<Warranty>1 year parts and labor</Warranty>
<Maintenance>3 year parts and labor extended maintenance is available</Maintenance>
</Features>

Método Value
Este método se utiliza para extraer un valor de una instancia XML almacenada en una columna, parámetro o variable de tipo xml. De esta manera, se pueden especificar consultas SELECT que combinen o comparen datos XML con datos de columnas que no son XML. Se debe especificar una expresión XQuery que identifica un solo nodo.

Ejemplo:

```
DECLARE @Doc xml
DECLARE @Prod int
SET @Doc = '<Root>
      <ProductDescription ProductID="1" ProductName="Road Bike">
            <Features>
                  <Warranty>1 year parts and labor</Warranty>
                  <Maintenance>3 year parts and labor extended
                      maintenance is available</Maintenance>
            </Features>
      </ProductDescription>
      </Root>'
SET @Prod =  @Doc.value('(/Root/ProductDescription/@ProductID)[1]',
'int' )
SELECT @Prod
```

El resultado es: 1

Método Exist
Determinado si un determinado existe o no en un documento XML.

Ejemplo:

```
DECLARE @x xml
DECLARE @f bit
SET @x = '<root Somedate = "2002-01-01Z"/>'
SET @f = @x.exist('/root[(@Somedate cast as xs:date?) eq
xs:date("2002-01-01Z")]')
SELECT @f
```

El resultado es: Verdadero.

Métodos Modify
Permite modificar el contenido de un documento XML. Utilice este método para modificar el contenido de una columna o variable de tipo xml, agregar o borrar nodos. Usa tres extensiones del lenguaje XQuery: **Insert**, **Delete** y **Replace**

Ejemplo:

```
DECLARE @Doc xml
SET @Doc = '<Root>
            <ProductDescription ProductID="1" ProductName="Road Bike">
                        <Features>
                        </Features>
            </ProductDescription>
      </Root>'
SELECT @Doc
-- Agrega
SET @Doc.modify('
insert element Maintenance {"3 year parts and labor extended
maintenance is available"} as first
```

```
into (/Root/ProductDescription/Features)[1]')
SELECT @Doc
```

El resultado es final es:

```
<Root>
	<ProductDescription ProductID="1" ProductName="Road Bike">
		<Features>
			<Maintenance>3 year parts and labor extended maintenance is
			available</Maintenance>
		</Features>
	</ProductDescription>
</Root>
```

Método Nodes
Convierte el XML en un conjunto de registros. Permite identificar nodos que se asignarán a una fila nueva.

Ejemplo:

```
DECLARE @xmlOrder xml
SET @xmlOrder = '<?xml version="1.0" ?>
	<Order OrderID="1000" OrderDate="2005-06-04">
		<LineItem ProductID="1" Price="2.99" Quantity="3" />
		<LineItem ProductID="2" Price="3.99" Quantity="1" />
	</Order>'
SELECT nCol.value('@ProductID', 'integer') ProductID,
nCol.value('@Quantity', 'integer') Quantity
FROM @xmlOrder.nodes('/Order/LineItem') AS nTable(nCol)
```

El resultado es:

Módulo 7

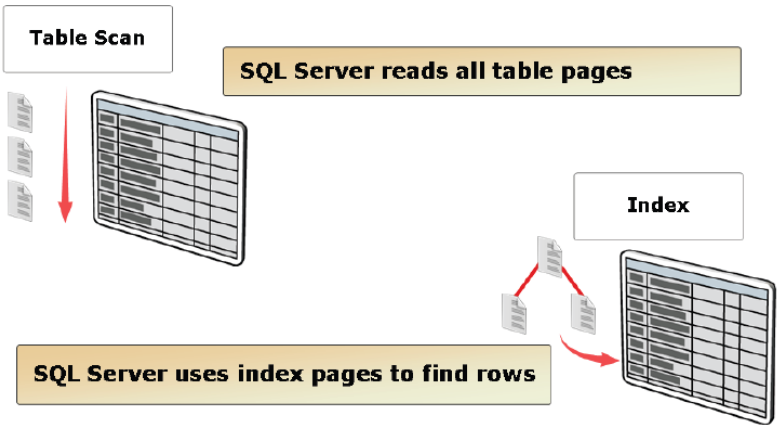
Índices

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Planeando Índices

Como se accede a los datos
SQL Server 2008 accede a los datos de dos maneras:

- Recorriendo la tabla completa página a página desde el principio de la tabla extrayendo los registros que cumplen con el criterio de la consulta
- Usando índices. Cuando usa un índice primero recorre la estructura de árbol del índice para encontrar los registros que cumplen con los criterios de la consulta



Los índices bien diseñados pueden reducir las operaciones de E/S de disco y consumir menos recursos del sistema, con lo que mejoran el rendimiento de la consulta. Los índices pueden ser útiles para varias consultas que contienen instrucciones SELECT, UPDATE o DELETE. Cuando se ejecuta una consulta, el optimizador de consultas evalúa cada método disponible para recuperar datos y selecciona el método más eficiente. El método puede ser un recorrido de la tabla o puede ser recorrer uno o más índices si existen.
Al realizar un recorrido de la tabla, el optimizador de consultas leerá todas las filas de la tabla y extraerá las filas que cumplen con los criterios de la consulta. Un recorrido de la tabla genera muchas operaciones de E/S de disco y puede consumir recursos. No obstante, puede ser el método más eficaz si, por ejemplo, el conjunto de resultados de la consulta es un porcentaje elevado de filas de la tabla.
Cuando el optimizador de consultas utiliza un índice, busca en las columnas de clave de índice, busca la ubicación de almacenamiento de las filas que necesita la consulta y extrae las filas coincidentes de esa ubicación. Generalmente, la búsqueda del índice es mucho más rápida que la búsqueda de la tabla porque, a diferencia de la tabla, un índice frecuentemente contiene muy pocas columnas por fila y las filas están ordenadas.
El optimizador de consultas normalmente selecciona el método más eficaz cuando ejecuta consultas. No obstante, si no hay índices disponibles, el optimizador de consultas debe utilizar un recorrido de la tabla. Su tarea es diseñar y crear los índices más apropiados para su entorno de forma que el optimizador de consultas tenga una selección de índices eficaces entre los que elegir.

Índice clúster (clustered)
Un índice clúster guarda los datos de la tabla en el orden basado en la clave de dicho índice. Solo puede haber un índice clúster por tabla.
En SQL Server, los índices se organizan como árboles B (b-tree). Las páginas de un árbol B de índice se llaman nodos del índice. El nodo superior del árbol B se llama **nodo raíz**. El nivel inferior de los nodos del índice se denomina **nodos hoja**. Los niveles del índice entre el nodo

raíz y los nodos hoja se conocen en conjunto como niveles intermedios. En un índice clúster, los nodos hoja contienen las páginas de datos de la tabla subyacente. El nodo raíz y los nodos intermedios incluyen páginas de índice que contienen filas de índice. Cada fila de índice contiene un valor clave y un puntero a una página de nivel intermedio en el árbol B, o bien a una fila de datos del nivel hoja del índice. Las páginas de cada nivel del índice se vinculan en una lista con vínculos dobles.

Los índices clúster tienen una fila en **sys.partitions**, con **index_id** = 1 para cada partición utilizada por el índice. De forma predeterminada, un índice clúster tiene una sola partición.
Las páginas de la cadena de datos y las filas que contienen se ordenan según el valor de la clave de índice clúster. Todas las inserciones se hacen en el punto en el que el valor de clave de la fila insertada quede dentro de la secuencia de orden entre las filas existentes.

En un índice clúster, la columna **root_page** de **sys.system_internals_allocation_units** apunta al nivel superior del índice clúster para una partición específica. SQL Server baja en el índice para buscar la fila correspondiente a una clave de índice clúster. Para buscar un intervalo de claves, SQL Server se desplaza por el índice hasta encontrar el valor de clave inicial del intervalo y después recorre las páginas de datos mediante los punteros anterior y siguiente. Para buscar la primera página de la cadena de páginas de datos, SQL Server sigue los punteros situados más a la izquierda desde el nodo raíz del índice.

```
SELECT * FROM sys.partitions ORDER BY [object_id]
```

Results | Messages

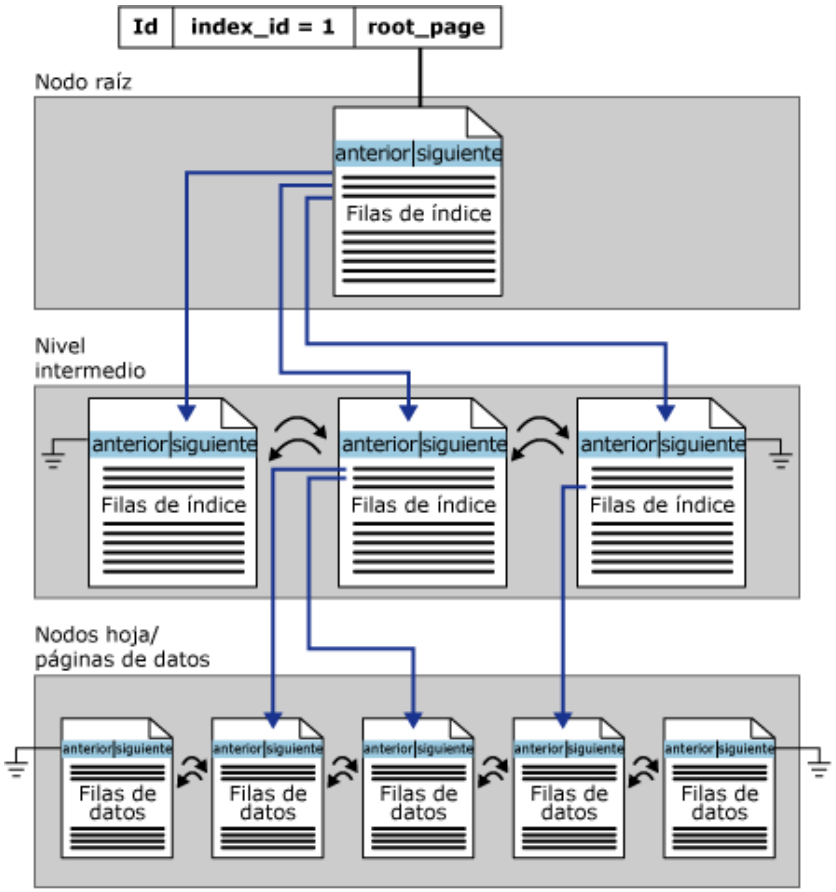
	partition_id	object_id	index_id	partition_number	hobt_id	rows	filestream_filegroup_id	data_compression	data_compression_desc
1	196608	3	1	1	196608	822	0	0	NONE
2	327680	5	1	1	327680	116	0	0	NONE
3	458752	7	1	1	458752	128	0	0	NONE
4	562949953880064	7	2	1	562949953880064	128	0	0	NONE
5	524288	8	0	1	524288	2	0	0	NONE
6	844424931246080	17	3	1	844424931246080	0	0	0	NONE
7	562949954535424	17	2	1	562949954535424	0	0	0	NONE

```
SELECT * FROM sys.system_internals_allocation_units
```

Results | Messages

	allocation_unit_id	type	type_desc	container_id	filegroup_id	total_pages	used_pages	data_pages	first_page	root_page	first_iam_page
1	196608	1	IN_ROW_DATA	196608	1	16	13	11	0x130000000100	0x3A0000000100	0x450000000100
2	327680	1	IN_ROW_DATA	327680	1	2	2	1	0x120000000100	0x120000000100	0xC20000000100
3	458752	1	IN_ROW_DATA	458752	1	5	5	3	0x100000000100	0x670100000100	0x110000000100
4	524288	1	IN_ROW_DATA	524288	1	2	2	1	0x200000000100	0x200000000100	0x0D0000000100
5	7705425149952	1	IN_ROW_DATA	7705425149952	1	0	0	0	0x000000000000	0x000000000000	0x000000000000

En esta ilustración se muestra la estructura de un índice clúster en una sola partición.



Es un índice muy eficiente para el soporte de consultas cuando se buscan rangos de valores sobre el primer campo, para resolver ORDER BY, GROUP BY o la cláusula JOIN sobre ese campo o devolver conjuntos de registros muy grandes.

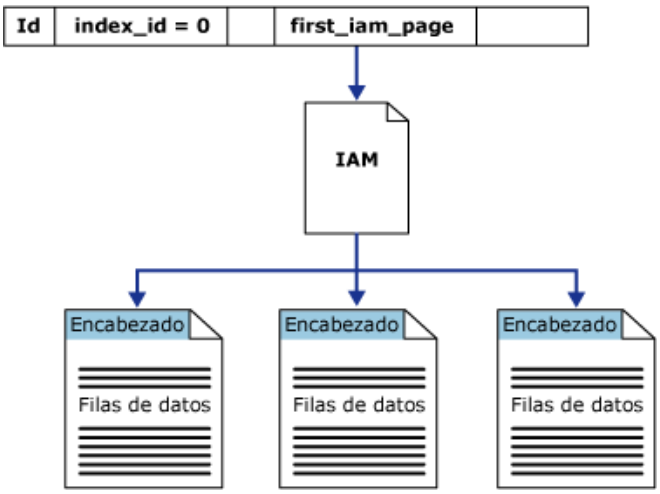
Se debe tener en cuenta:

- La clave debe ser lo más chica posible de manera de mantener el árbol B con la menor cantidad de niveles. Si se usa una clave compuesta elija la menor cantidad de campos posible. Recuerde que los índices no clúster usan esta clave para su organización lo cual hace que si la clave es muy grande también afecte a los índices no clúster.
- Elegir claves únicas o con muy pocos valores repetidos. Tener en cuenta el uso sobre columnas marcadas **Identity**, ya que asegura valores únicos.
- Elegir una columna/s que se use frecuentemente en los **ORDER BY**.
- Elegir una columna/s que se usan con frecuencia en los **WHERE**.
- No elegir datos que cambien con frecuencia, ya que para mantener el orden si la clave cambia el registro debe moverse para ubicarse donde le corresponde según su nuevo valor

Cuando se crea una clave primaria (**Primary Key**) en una tabla se crea un índice que por defecto es clúster.

Montón (Heap)
Un montón es una tabla que no tiene un índice clúster. Los montones tienen una fila en **sys.partitions**, con **index_id** = 0 para cada partición que el montón utiliza. De forma predeterminada, un montón contiene una sola partición. Cuando un montón tiene varias particiones, cada partición incluye una estructura de montón que contiene los datos de esa partición específica.
La columna **first_iam_page** en la vista de sistema **sys.system_internals_allocation_units** señala la primera página **IAM** de la cadena de páginas **IAM** que administra el espacio asignado a la pila en una partición específica. SQL Server utiliza las páginas IAM para desplazarse por el montón. Las páginas de datos y las filas que se encuentran en ellas no están en ningún orden concreto y no están vinculadas. La única conexión lógica entre las páginas de datos es la información registrada en las páginas IAM.
Los recorridos de tablas o las lecturas secuenciales de un montón se hacen recorriendo las páginas IAM para buscar las extensiones que almacenan las páginas de dicho montón.

La siguiente ilustración muestra cómo el Motor de base de datos de SQL Server utiliza las páginas IAM para recuperar las filas de datos en un solo montón de partición.



El montón se usa por defecto cuando no se define un índice clúster. Considerar su uso cuando la sobrecarga del mantenimiento de un índice clúster es mayor a su beneficio, para tablas con muy pocos registros o con datos muy duplicados.

Índices no Clúster (Non Clustered)
Los índices no clúster tienen la misma estructura de árbol B que los índices clúster, excepto por las siguientes diferencias importantes:

- Las filas de datos de la tabla subyacente no están ordenadas ni almacenadas basándose en sus claves no agrupadas.
- La capa de hoja de un índice no clúster está compuesta por páginas de índices, en lugar de páginas de datos.

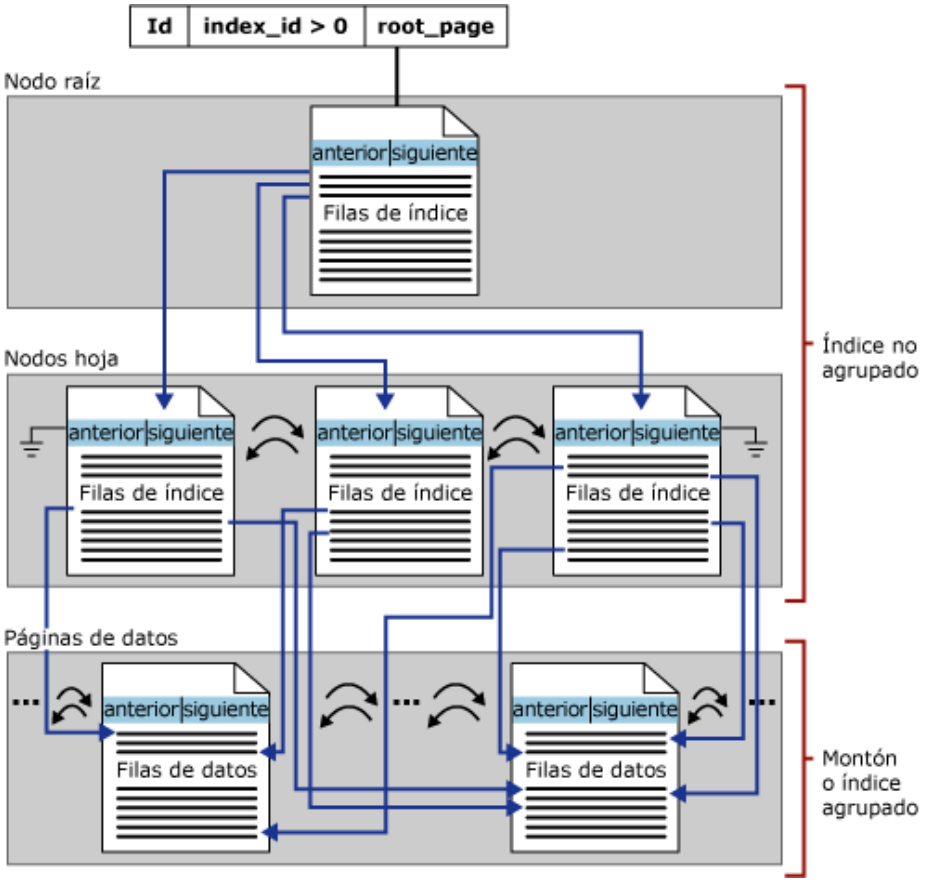
Los índices no clúster se pueden definir en una tabla o vista con un índice clúster o un montón. Cada fila del índice no clúster contiene un valor de clave no agrupada y un localizador de fila. Este localizador apunta a la fila de datos del índice clúster o el montón que contiene el valor de clave. Al crear un índice, si no se especifica su tipo es no clúster.

Los localizadores de filas de las filas de índices no clúster pueden ser:

- Si la tabla es un montón, lo que significa que no tiene ningún índice clúster, el localizador de fila es un puntero a la fila. El puntero se genera a partir del identificador (Id.) de archivo, el número de página y el número de la fila dentro de la página. El puntero completo se conoce como Id. de fila (RID).
- Si la tabla tiene un índice clúster o si el índice está en una vista indexada, el localizador de fila es la clave del índice clúster para la fila. Si el índice clúster no es un índice único, SQL Server hace que las claves duplicadas sean únicas agregando un valor generado internamente denominado valor de unicidad. Este valor de cuatro bytes no es visible para los usuarios

Los índices no clúster tienen una fila en **sys.partitions**, con **index_id** > 1 para cada partición utilizada por el índice. De forma predeterminada, un índice no clúster tiene una sola partición. Cuando un índice no clúster tiene varias particiones, cada una tiene una estructura de árbol B que contiene las filas de índice de esa partición específica.

En la siguiente ilustración se muestra la estructura de un índice no clúster en una sola partición.



Los índices no clúster son útiles cuando se requiere múltiples formas de acceder a los datos. Se recomienda su uso cuando:

- Se quiere mejorar la performance de consultas que usan JOIN o GROUP BY.
- Se quiere mejorar la performance de consultas con columnas que se usan frecuentemente en la cláusula WHERE con valores exactos.

- La tabla tiene pocas modificaciones y es muy grande. Cuantos más índices contenga este tipo de tabla, más eficientes serán sus búsquedas y al no haber muchas modificaciones el mantenimiento del índice no afecta a la performance.
- Las consultas no suelen devolver conjuntos de registros muy grandes.
- Aplicarlos generalmente en columna/s que tienen pocos duplicados.

Consideraciones a tener en cuenta cuando se crean índices no clúster:

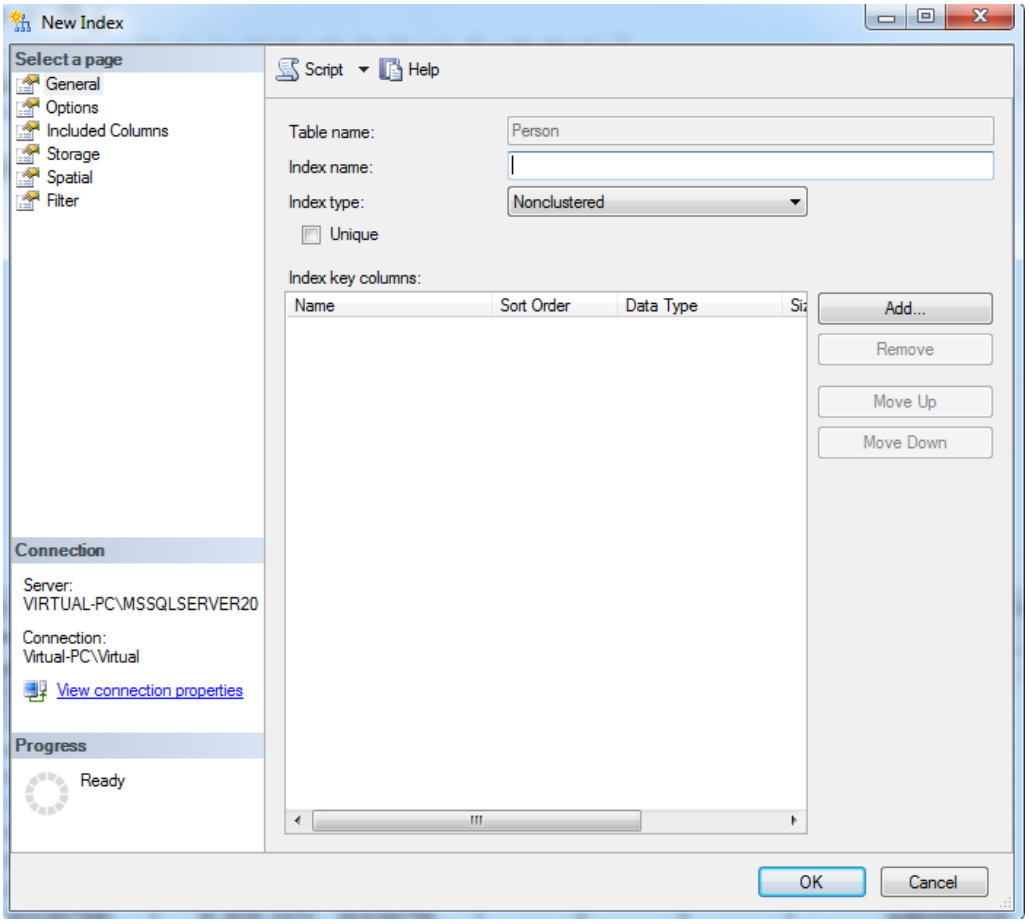
- Crear el índice clúster antes que los no clúster. SQL Server automáticamente reconstruye los índices no clúster cuando el índice clúster es creado, borrado o su clave es modificada.
- Evitar crear índices no clúster en un ambiente OLTP. Muchos índices afectan la performance de los INSERT, DELETE y UPDATE ya que además de modificar los datos debe modificar los índices.

Importante: Las columnas con los tipos de datos ntext, text, varchar(max), nvarchar(max), varbinary(max), xml, o image no pueden ser seleccionadas como columnas para índices.

2- Creando Índices

Creación
Se puede crear un índice usando el SQL Server Management Studio o usando la instrucción **CREATE INDEX** del Transact-SQL. Se puede crear un índice antes de que la tabla posea datos. Los índices relacionales se pueden crear en tablas o vistas.

Creación de índices en el SQL Server Management Studio



CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX nombre_índice ON { tabla | vista } (columna [ASC | DESC] [,...n])
INCLUDE (columna [,...n]
[WHERE <filter_predicate>])
[WITH
[PAD_INDEX = { ON | OFF }]
[[,] FILLFACTOR = fillfactor]
[[,] IGNORE_DUP_KEY = { ON | OFF }]
[[,] ONLINE = { ON | OFF }]
[[,] ALLOW_ROW_LOCKS = { ON | OFF }]
[[,] ALLOW_PAGE_LOCKS = { ON | OFF }]]
[ON {esquema_partición (columna) | filegroup | “default” }]

Argumentos

- **UNIQUE**: Crea un índice único en una tabla o vista. Un índice único es aquel en el que no se permite que dos filas tengan el mismo valor de clave del índice. El índice clúster de una vista debe ser único. No admite la creación de un índice único sobre columnas que ya contengan valores duplicados, independientemente de si se ha establecido o no IGNORE_DUP_KEY en ON.
- **CLUSTERED*:*** Crea un índice en el que el orden lógico de los valores de clave determina el orden físico de las filas correspondientes de la tabla. El nivel inferior, u hoja, de un índice clúster contiene las filas de datos en sí de la tabla.
- **NONCLUSTERED**: Crea un índice que especifica el orden lógico de una tabla. Con un índice no clúster, el orden físico de las filas de datos es independiente del orden indexado. Cada tabla puede tener hasta 249 índices clúster. Para las vistas indexadas, sólo se pueden crear índices no clúster en una vista que ya tenga definido un índice clúster único. Es el valor predeterminado.
- **Nombre_índice**: Es el nombre del índice. Los nombres de índice deben ser únicos en una tabla o vista, pero no es necesario que sean únicos en una base de datos. Debe cumplir las reglas de los identificadores. La especificación del nombre del propietario es opcional.
- **Columna**: Es la columna o columnas en las que se basa el índice. Especifique dos o más nombres de columna para crear un índice compuesto sobre los valores combinados de las columnas especificadas.
- **ASC | DESC**: Determina la dirección ascendente o descendente del orden de la columna de índice determinada. El valor predeterminado es ASC.
- **INCLUDE**: Especifica las columnas sin clave que se agregarán en el nivel hoja del índice no clúster.
- **WHERE**: Crea un índice filtrado especificando qué filas se van a incluir en el índice. El índice filtrado debe ser un índice no clúster en una tabla. Crea las estadísticas filtradas para las filas de datos en el índice filtrado.
- **ON esquema_partición**: Específica el esquema de partición que define los grupos de archivos a los que se asignarán las particiones de un índice con particiones.
- **ONLINE**: Especifica si las tablas subyacentes e índices asociados están disponibles para consultas y modificación de datos durante la operación de indexación. El valor predeterminado es OFF.
- **ALLOW_ROW_LOCKS**: Especifica si se permiten bloqueos de fila. El valor predeterminado es ON.
- **ALLOW_PAGE_LOCKS**: Especifica si se permiten bloqueos de página. El valor predeterminado es ON. Si ambos bloqueos están es OFF solo se pueden usar bloqueos de nivel tabla.

Ejemplo:

```
CREATE NONCLUSTERED INDEX [AK_Employee_LoginID]
ON [HumanResources].[Employee] ( [LoginID] ASC)
```

Modificación y Baja

Para modificar o borrar un índice se puede usar la herramienta SQL Server Management Studio o las instrucciones **ALTER INDEX** / **DROP INDEX**.

Al modificar un índice existente de una tabla o una vista se permite deshabilitarlo, regenerarlo o reorganizarlo, o modificar opciones en él. No es posible utilizar ALTER INDEX para volver a crear particiones en un índice o moverlo a un grupo de archivos distinto. No es posible utilizar esta instrucción para modificar la definición de índice, como por ejemplo para agregar o eliminar columnas o cambiar su orden. Utilice CREATE INDEX con la cláusula **DROP_EXISTING** para realizar estas operaciones.

Cuando se quita un índice clúster, se quita la definición del índice de los metadatos y las filas de datos que se almacenaron en el nivel hoja del índice clúster se almacenan en la tabla resultante no ordenada, o sea en un montón.

Cláusula DROP_EXISTING
Puede utilizar la cláusula DROP_EXISTING para volver a generar el índice, agregar o quitar columnas, modificar opciones, modificar el criterio de orden de las columnas o cambiar el grupo de archivos o el esquema de partición. No permite cambiar el tipo de índice.
DROP_EXISTING mejora el rendimiento cuando se vuelve a crear un índice clúster (con el mismo conjunto de claves o con uno distinto) en una tabla que también tiene índices no clúster. DROP_EXISTING reemplaza la ejecución de una instrucción DROP INDEX en el antiguo índice clúster seguida de la ejecución de una instrucción CREATE INDEX para el nuevo índice clúster. Los índices no clúster se vuelven a generar una vez, siempre que la definición de índice haya cambiado. La cláusula DROP_EXISTING no vuelve a generar los índices no clúster cuando la definición de índice posee los mismos nombres de índice, clave y columnas de partición, atributo de unicidad y criterio de orden que el índice original.

Índices Únicos
Un índice único garantiza que la clave de índice no contiene valores duplicados y, por tanto, cada fila de la tabla es en cierta forma única. Resulta conveniente especificar un índice único sólo si los propios datos se caracterizan por ser únicos. Si el índice es único y se intenta especificar el mismo valor en esa columna para más de un registro, se mostrará un mensaje de error que impedirá la entrada del valor duplicado.
Tanto los índices clúster como los no clúster pueden ser únicos. Siempre que los datos de la columna sean únicos, puede crear para la misma tabla un índice clúster único y varios índices no clúster no únicos.

Si se crea una restricción **PRIMARY KEY** o **UNIQUE**, se creará automáticamente un índice único en las columnas especificadas. No existen diferencias significativas entre la creación de una restricción UNIQUE y la creación de un índice único independiente de una restricción. La validación de los datos tiene lugar de la misma manera y el optimizador de consultas no establece diferencias entre un índice único creado por una restricción y uno creado manualmente.

Consideraciones
- Un índice único, una restricción **UNIQUE** o **PRIMARY KEY** no se pueden crear si existen valores de clave duplicados en los datos.
- Si los datos son únicos y desea hacer cumplir la exclusividad, la creación de un índice único en lugar de un índice no único en la misma combinación de columnas proporciona información adicional para el optimizador de consultas que puede dar como resultado unos planes de ejecución más eficaces. En este caso se recomienda crear un índice único (preferiblemente mediante una restricción UNIQUE).

Índices Compuestos
Un índice compuesto incluye más de una columna en un clave. Se puede mejorar mucho la performance usando este tipo de índices, especialmente cuando se hacen búsquedas de información de muchas maneras diferentes. Sin embargo las claves muy grandes requieren mucho espacio de almacenamiento.

Los índices compuestos tienen los siguientes requerimientos y limitaciones:
- Soporta hasta 16 columnas.

- La suma total de la longitud de las columnas no puede superar los 900 bytes.
- La cláusula WHERE de una consulta debe referenciar la primera columna para que el optimizador de consultas use el índice. Los índices que cubren consultas no tienen ésta restricción.
- Todas las columnas debes pertenecer a la misma tabla excepto cuando el índice es creado sobre una vista, en cuyo caso las columnas deberán pertenecer a la misma vista.
- Un índice sobre (columna1, columna2) no es igual a otro índice (columna2, columna1).

Los índices que cubren una consulta son muy rápidos al reducir las operaciones de E/S ya que los datos pueden ser recuperados de las páginas del índice sin la necesidad de acceder a las páginas de datos.
Consideraciones:

- Definir como primera columna a la columna con menos duplicados.
- Usar estos índices para mejorar la performance y reducir la cantidad de índices que se crean sobre una tabla.

Índice con Columnas Incluidas

Puede ampliar la funcionalidad de índices no clúster agregando columnas sin clave en el nivel hoja del índice no clúster. Al incluir columnas sin clave, puede crear índices no clúster que abarcan más consultas. Esto se debe a que las columnas sin clave tienen las siguientes ventajas:

- Pueden ser tipos de datos que no están permitidos como columnas de clave de índice.
- El Motor de base de datos no las tiene en cuenta cuando calcula el número de columnas de clave de índice o el tamaño de las claves de índice.

Un índice con columnas sin clave incluidas puede mejorar significativamente el rendimiento de una consulta cuando todas las columnas de la consulta se incluyen como columnas de clave o columnas sin clave. Las mejoras en el rendimiento se consiguen porque el optimizador de consultas puede localizar todos los valores de las columnas del índice, sin tener acceso a los datos de la tabla o del índice clúster, lo que da como resultado menos operaciones de E/S de disco.

Cuando diseñe índices no clúster con columnas incluidas tenga en cuenta las siguientes directrices:

- Las columnas sin clave se definen en la cláusula **INCLUDE** de la instrucción CREATE INDEX.
- Las columnas sin clave sólo se pueden definir en índices no clúster.
- Se permiten todos los tipos de datos, excepto text, ntext e image.
- Las columnas calculadas que son deterministas y precisas o imprecisas pueden ser columnas incluidas.
- Al igual que con las columnas de clave, las columnas calculadas derivadas de los tipos de datos image, ntext y text pueden ser columnas incluidas siempre que se permita el tipo de datos de la columna calculada como columna de índice sin clave.
- Los nombres de columna no se pueden especificar en la lista **INCLUDE** y en la lista de columnas de clave.

Ejemplo:

```
CREATE UNIQUE NONCLUSTERED INDEX [AK_Employee_LoginID]
ON [HumanResources].[Employee] ( [LoginID] ASC)
INCLUDE (BusinessEntityID, [NationalIDNumber])
```

Índices sobre Columnas Calculadas
Se pueden crear índices sobre columnas calculadas cuando:

- Las columnas calculadas son deterministas y precisas. Una expresión determinista es cuando siempre devuelve el mismo valor ante los mismos parámetros. Precisa es que no incluye datos con tipo de dato float o real .
- ANSI_NULLS está en ON al momento de ejecutar el CREATE TABLE.
- La columna calculada no evalúa tipo de datos text, ntext o image
- La conexión en donde el índice fue creado y todas las conexiones que modificaron sus valores debe tener las siguientes opciones en ON: ANSI_NULLS, ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL, QUOTED_IDENTIFIER, ARITHABORT. Y la opción NUMERIC_ROUNDABORT en OFF.

Índices con Particiones
Similar al concepto de tablas con particiones se pueden crear índices clúster y no clúster con particiones en donde las páginas de índices estén separadas horizontalmente en múltiples ubicaciones físicas. Estos índices mejoran la performance de los índices muy grandes.
Aunque los índices con particiones pueden implementarse independientemente de sus tablas base, por lo general tiene sentido diseñar una tabla con particiones y, a continuación, crear un índice en la tabla. Al hacerlo, SQL Server crea particiones en el índice con el mismo esquema de partición y columna de partición que la tabla. En consecuencia, en el índice se crean particiones básicamente de la misma forma que en la tabla. De este modo, el índice se **alinea** con la tabla. Cuando una tabla y sus índices están alineados, SQL Server puede dividir las particiones de forma rápida y eficaz al mismo tiempo que mantiene la estructura de la partición tanto en la tabla como en sus índices.
SQL Server no alinea el índice con la tabla si especifica un esquema de partición diferente o un grupo de archivos independiente en el que colocar el índice durante la creación.

Índices Filtrados
Nuevos de la versión SQL Server 2008.
Un índice filtrado es un índice no clúster optimizado, especialmente indicado para atender consultas que realizan selecciones a partir un subconjunto bien definido de datos. Utiliza un predicado de filtro para indizar una parte de las filas de la tabla. Un índice filtrado bien diseñado puede mejorar el rendimiento de las consultas, reducir los costos de mantenimiento y de almacenamiento del índice en relación con los índices de tabla completa.
Para diseñar índices filtrados efectivos, es importante entender qué consultas utiliza la aplicación y cómo se relacionan con los subconjuntos de datos. Algunos ejemplos de datos que tienen subconjuntos bien definidos son las columnas con una mayoría de valores NULL, las columnas con categorías de valores heterogéneas y las columnas con intervalos de valores diferenciados.

Ejemplo:

```
CREATE NONCLUSTERED INDEX FIBillOfMaterialsWithEndDate
ON Production.BillOfMaterials (ComponentID, StartDate)
WHERE EndDate IS NOT NULL ;
```

Índices con Espacio Libre
Las páginas de índices con espacio libre pueden mejorar la performance de las operaciones de actualización de datos. Si un nuevo registro debe ser ingresado y no hay espacio libre en la página que le corresponde, una nueva página debe ser creada y el contenido de la página

debe distribuirse en ambas páginas. Esto causa un efecto sobre la performance si sucede con mucha frecuencia.
SQL Server ofrece dos opciones que permiten controlar la cantidad de espacio libre de las páginas:

- **FILLFACTOR**: Utilice la opción FILLFACTOR (factor de relleno) para especificar cuánto puede llenar cada página cuando crea un nuevo índice con datos existentes. El factor de relleno se utiliza sólo cuando se crea o se vuelve a generar un índice. Las páginas no se mantienen en ningún grado específico de relleno. El valor predeterminado es 0; los valores válidos están comprendidos entre el 0 y el 100. Cuando el FILLFACTOR es 0 ó 100, el nivel hoja se llena al máximo (aconsejado en un ambiento OLAP). Si se especifica un valor de factor de relleno de por ejemplo 80, significa que el 20 por ciento de cada página de nivel hoja se dejará vacío para proporcionar espacio para la expansión del índice a medida que se agreguen datos a la tabla subyacente. El espacio vacío se reserva entre las filas de índice de cada página en lugar de al final de la página. Modificar al valor de FILLFACTOR está aconsejado en ambientes OLTP. No aplicar FILLFACTOR sobre índices con claves sobre columnas **Identity** ya que estás no generan división de páginas.
- **PAD_INDEX**: Si es ON el porcentaje de espacio disponible especificado por FILLFACTOR se aplica a páginas de nivel intermedio del índice. Si es OFF o no se especifica FILLFACTOR las páginas de nivel intermedio se llenan casi al máximo de su capacidad y dejan espacio suficiente para al menos una fila del tamaño máximo que puede tener el índice, considerando el conjunto de claves incluidas en las páginas de nivel intermedio. PAD_INDEX utiliza el mismo porcentaje especificado por FILLFACTOR. Si el porcentaje especificado para FILLFACTOR no es lo suficientemente grande como para admitir una fila, el Motor de base de datos anula internamente el porcentaje para permitir el valor mínimo. El número de filas de una página de nivel intermedio del índice no es nunca inferior a dos, independientemente de lo bajo que sea el valor de FILLFACTOR.

Ejemplo:

```
CREATE UNIQUE NONCLUSTERED INDEX [AK_Employee_LoginID]
ON [HumanResources].[Employee] ( [LoginID] ASC)
WITH ( FILLFACTOR = 65, PAD_INDEX = ON)
```

Obtener Información sobre Índices

Vista del Catálogo

- **sys.indexes**: Muestra el tipo de índice, grupo de archivo o partición y las opciones establecidas para ese índice.
- **sys.index_columns**: Columnas de la clave o incluidas y el tipo de orden (ASC o DESC)
- **sys.stats** : Información sobre estadísticas
- **sys.stats_columns*:*** Columnas asociadas a estadísticas

Funciones del Sistema

- **sys.dm_db_index_physical_stats*:*** Tamaño y fragmentación de los datos.

```
SELECT * FROM sys.dm_db_index_physical_stats
(DB_ID('AdventureWorks'), OBJECT_ID(N'Person.Address'), NULL,
NULL ,'DETAILED');
```

- **sys.dm_db_index_operational_stats*:*** Devuelve las entradas y salidas de bajo nivel actuales, el bloqueo, el bloqueo temporal y la actividad de método de acceso de cada partición de una tabla o índice de la base de datos.
- **sys.dm_db_index_usage_stats**: Devuelve recuentos de diferentes tipos de operaciones de índice y la hora en que se realizó por última vez cada uno de los tipos de operación.
- **INDEXKEY_PROPERTY*:*** Devuelve información acerca de la clave de índice. Devuelve NULL para los índices XML.

```
SELECT INDEXKEY_PROPERTY(OBJECT_ID('Production.Location', 'U'),
1,1,'ColumnId') AS [Column ID].
```

- **INDEXPROPERTY*:*** Devuelve el valor de propiedad del índice con nombre o las estadísticas de un número de identificación, nombre de índice o estadísticas y nombre de propiedad de una tabla especificada.
- **INDEX_COL**: Devuelve el nombre de la columna indexada

Procedimientos Almacenados de Sistema

- **Sp_HelpIndex*:***

```
EXEC sp_helpindex [Production.Product]
```

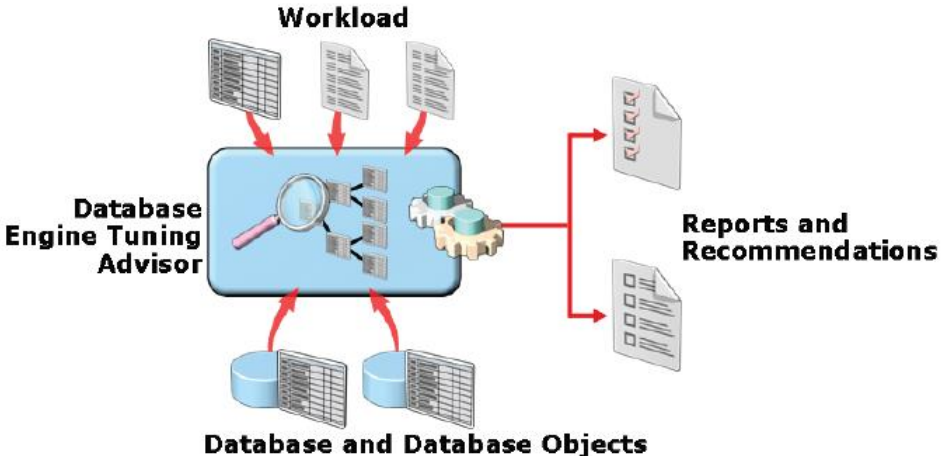
- **Sp_Help**:

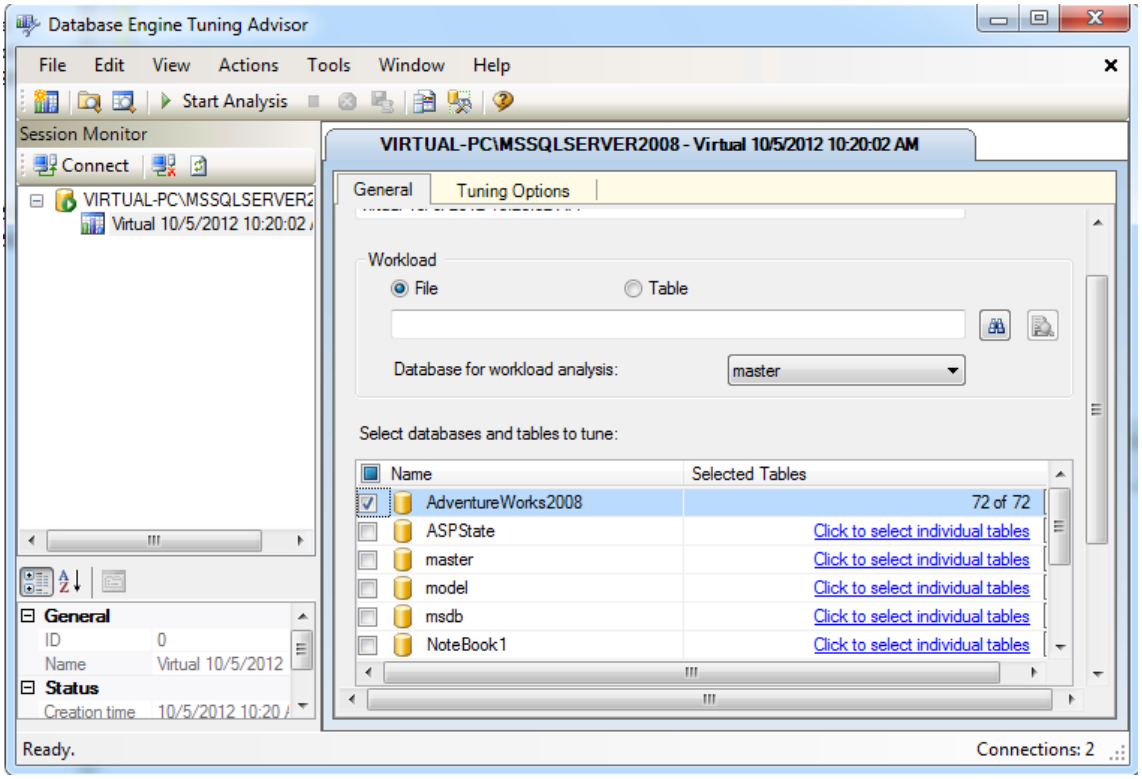
```
EXEC sp_help [Production.Product]
```

3- Optimizando Índices

Un factor clave para conseguir una E/S de disco mínima para todas las consultas de bases de datos es asegurarse de que se creen y se mantengan buenos índices. Una vez creados los índices éstos deben mantenerse para asegurarse que sigan trabajando en forma óptima. A medida que los datos son agregados, modificados o borrado producen fragmentación. Esta fragmentación puede ser buena o mala para la performance dependiendo de las necesidades del trabajo de la base de datos.

Asistente para la optimización de motor de base de datos
(Database Engine Tuning Advisor)





El Asistente para la optimización de motor de base de datos de Microsoft SQL Server ayuda a seleccionar y crear un conjunto óptimo de índices, vistas indexadas y particiones sin necesidad de conocer detalladamente la estructura de la base de datos ni el funcionamiento interno de Microsoft SQL Server.

El Asistente para la optimización de motor de base de datos analiza una carga de trabajo y la implementación física de una o más bases de datos. Una carga de trabajo es un conjunto de instrucciones Transact-SQL que se ejecuta en una o varias bases de datos que se desean optimizar. El Asistente para la optimización de motor de base de datos utiliza archivos de seguimiento, tablas de traza o scripts Transact-SQL como entrada de carga de trabajo al optimizar las bases de datos. Con el Editor de consultas de SQL Server Management Studio, se pueden crear cargas de trabajo de scripts Transact-SQL. Mediante la plantilla **Tuning** del Analizador de SQL Server se pueden crear cargas de trabajo de archivos y tablas de traza.
Tras analizar una carga de trabajo, el Asistente para la optimización de motor de base de datos puede recomendar que se agreguen, eliminen o modifiquen estructuras de diseño físico de las bases de datos. El asistente también puede recomendar qué estadísticas se deben recopilar para realizar copias de seguridad de las estructuras de diseño físico. Las estructuras de diseño físico incluyen índices clúster, índices no clúster, vistas indexadas y particiones.

El Asistente para la optimización de motor de base de datos puede:

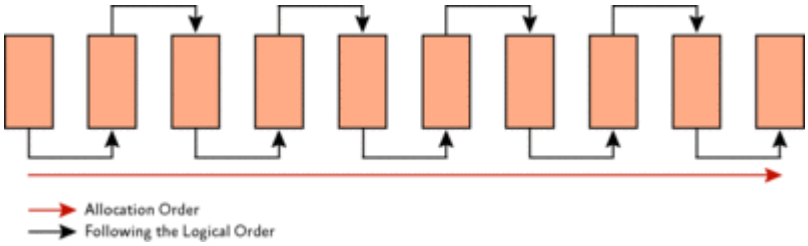
- Recomendar la mejor combinación de índices para las bases de datos mediante el uso del optimizador de consultas para analizar las consultas de una carga de trabajo.
- Recomendar particiones alineadas y no alineadas para las bases de datos a las que se hace referencia en una carga de trabajo.
- Recomendar vistas indexadas para las bases de datos a las que se hace referencia en una carga de trabajo.
- Analizar los efectos de los cambios propuestos en aspectos tales como el uso de índices, la distribución de consultas entre tablas y el rendimiento de las consultas de la carga de trabajo.
- Recomendar métodos para optimizar la base de datos con respecto a un pequeño conjunto de consultas problemáticas.
- Permitirle personalizar la recomendación mediante la especificación de opciones avanzadas como, por ejemplo, las restricciones de espacio en disco.
- Proporcionar informes que resuman los efectos de la implementación de las recomendaciones en una carga de trabajo concreta.
- Considerar alternativas en las que se ofrezcan posibles opciones de diseño en forma de configuraciones hipotéticas para que el Asistente para la optimización de motor de base de datos pueda evaluarlas.

Fragmentación de los índices
La fragmentación es consecuencia de los procesos de modificación de los datos (instrucciones INSERT, UPDATE y DELETE) efectuados en la tabla y en los índices definidos en la tabla. Como dichas modificaciones no suelen estar distribuidas de forma equilibrada entre las filas de la tabla y los índices, el llenado de cada página puede variar con el paso del tiempo. Para las consultas que recorren parcial o totalmente los índices de una tabla, este tipo de fragmentación puede producir lecturas de páginas adicionales. Esto impide el recorrido paralelo de los datos.
Existen 2 tipos de fragmentación:

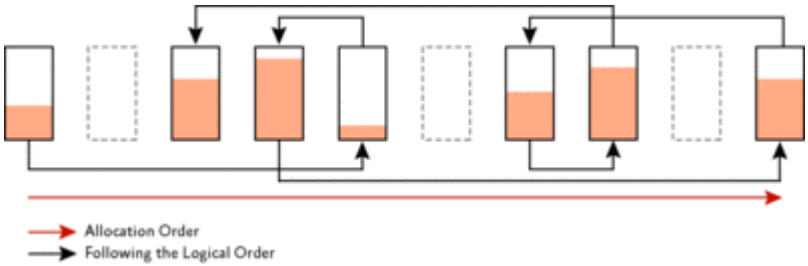
- **Interna**: Fragmentación dentro de páginas individuales de datos e índices, las cuales tienen espacios libres en ella generando la necesidad de más operaciones de E/S y más memoria para su lectura. Esto baja la performance en ambientes de lectura, pero en algunos casos puede beneficiar a las inserciones las cuales no requerirán dividir páginas con tanta frecuencia.

- **Externa**: Cuando el orden lógico de las páginas no es correcto, ya que las páginas no son contiguas. El acceso a los datos es mucho más lento por la necesidad de búsqueda de los datos.

La siguiente imagen muestra páginas de índices sin fragmentación:

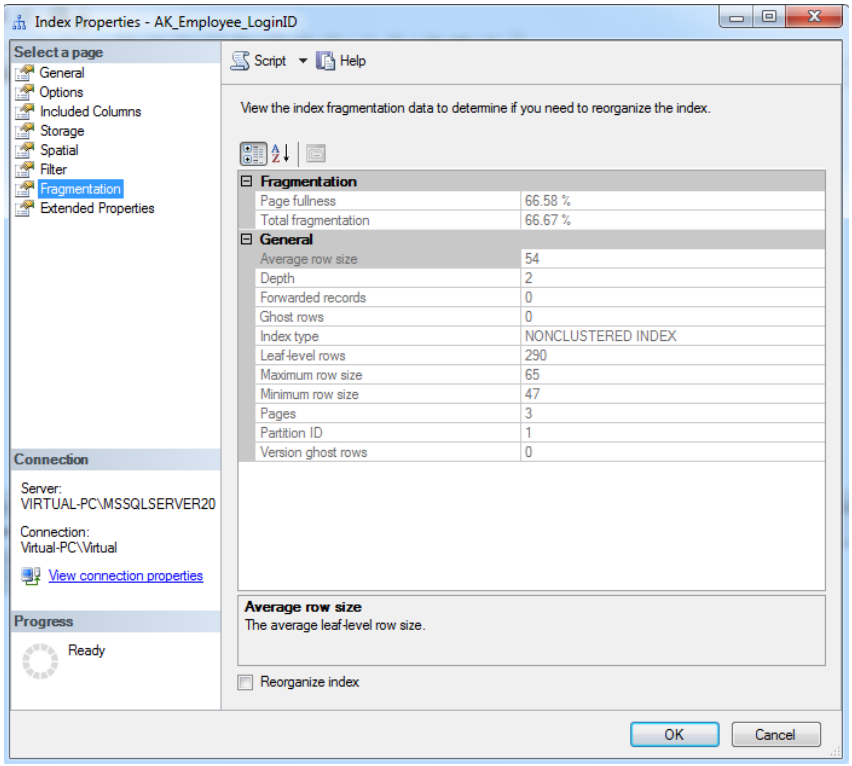


En cambio en esta imagen se ven las páginas con fragmentación interna y externa:



Para detectar la fragmentación se puede usar

- SQL Server Management Studio: Propiedades de un índice:



- **sys.dm_db_index_physical_stats*:*** Función de sistema
  sys.dm_db_index_physical_stats (
  { basedatos_id | NULL | 0 | DEFAULT }
  , { objecto_id | NULL | 0 | DEFAULT }
  , { indice_id | NULL | 0 | -1 | DEFAULT }
  , { particion_number | NULL | 0 | DEFAULT }
  , { modo | NULL | DEFAULT }
)

Ejemplo: Muestra la fragmentación de todos los índices de la tabla Product.

```
SELECT a.index id, name, avg fragmentation in percent
FROM sys.dm_db_index_physical_stats (DB_ID(N'AdventureWorks2008'),
OBJECT_ID(N'Production.Product'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b
ON a.object_id = b.object_id AND a.index_id = b.index_id
```

Results | Messages

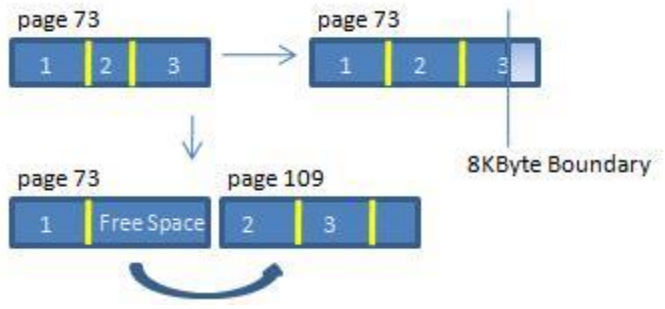
	index_id	name	avg_fragmentation_in_percent
1	1	PK_Product_ProductID	23.0769230769231
2	2	AK_Product_ProductNumber	50
3	3	AK_Product_Name	66.6666666666667
4	4	AK_Product_rowguid	50

Columna **avg_fragmentation_in_percent**: Porcentaje de fragmentación lógica (páginas de un índice que no funcionan correctamente).

Divisiones de páginas

Cuando se agrega una nueva fila a una página de índice completa, el Motor de base de datos mueve a una nueva página la mitad de las filas, aproximadamente, con el fin de que haya espacio para la nueva fila. Esta reorganización se denomina división de página. Una división de página genera espacio para los nuevos registros, pero puede tardar en realizarse y es una operación que consume muchos recursos. Además, puede causar fragmentación que, a su vez, causa más operaciones de E/S.

Esta división de páginas solo ocurre en índices, no ocurre en montones, en estos últimos simplemente busca un espacio libre durante la inserción.



Opciones para desfragmentar índices

Existen dos opciones: **reorganizar** y **reconstruir**. La decisión debería tomarse teniendo en cuenta el nivel de fragmentación:

- Si es menor al 30% reorganizar usando ALTER INDEX REORGANIZE
- Si es mayor al 30% es preferible reconstruir usando ALTER INDEX REBUILD.

Reorganizar
Para reorganizar uno o más índices, utilice la instrucción ALTER INDEX con la cláusula **REORGANIZE**.
La reorganización de un índice desfragmenta el nivel hoja de índices clúster y no clúster en tablas y vistas volviendo a ordenar físicamente las páginas en el nivel hoja de manera que coincidan con la ordenación lógica (de izquierda a derecha) de los nodos hoja. La ordenación de las páginas mejora el rendimiento de recorrido del índice. La reorganización también compacta las páginas del índice. Todas las páginas vacías que se crean como consecuencia de esta compactación se eliminan para proporcionar una mayor cantidad de espacio en el disco duro. La compactación se basa en el valor de factor de relleno de la vista de catálogo **sys.indexes**.

El proceso de reorganización utiliza una mínima cantidad de recursos del sistema. Además, la reorganización se realiza automáticamente en línea. El proceso no mantiene bloqueos durante mucho tiempo y, por lo tanto, no bloquea las consultas ni las actualizaciones en ejecución.
Reorganice un índice cuando éste no esté demasiado fragmentado. No obstante, si el índice está muy fragmentado, obtendrá mejores resultados si lo vuelve a generar.

Ejemplos:

```
ALTER INDEX AK_Product_Name ON Production.Product REORGANIZE
ALTER INDEX ALL ON Production.Product REORGANIZE
```

Reconstruir
Cuando se vuelve a generar un índice, se quita el índice anterior y se crea uno nuevo. Al hacerlo, se elimina la fragmentación, se recupera el espacio en el disco duro compactando páginas mediante la configuración especificada o existente del factor de relleno y se reordenan las filas de índices en las páginas contiguas (se asignan páginas nuevas según sea necesario). Esto puede mejorar el rendimiento del disco, reduciendo el número de lecturas de página necesarias para obtener los datos solicitados.

Para volver a generar índices, se pueden utilizar los siguientes métodos:
- ALTER INDEX con la cláusula **REBUILD**. Esta instrucción reemplaza a la instrucción DBCC DBREINDEX.
- CREATE INDEX con la cláusula DROP_EXISTING.

Ambos métodos generan la misma función pero tienen algunas diferencias. CREATE INDEX permite cambiar la clave y crear o modificar particiones mientras el ALTER INDEX no lo permite. Pero con ALTER INDEX se pueden reconstruir varios índices simultáneamente y si regenera el índice clúster se regeneran automáticamente los no clúster mientras que con CREATE INDEX no, excepto que modifique la definición de índice clúster.

Ejemplos:

```
ALTER INDEX AK_Product_Name ON Production.Product REBUILD
ALTER INDEX ALL ON Production.Product REBUILD WITH (FILLFACTOR = 80)
```

4- Índices XML

Índices XML
Las instancias XML se almacenan en las columnas de tipo xml como objetos binarios grandes (BLOB). Estas instancias XML pueden ser grandes, y la representación binaria almacenada de instancias de datos de tipo xml puede tener un tamaño de hasta 2 GB. Sin ningún índice, estos objetos binarios grandes se dividen en tiempo de ejecución para evaluar una consulta. Este proceso de división puede resultar lento.

Es posible crear índices XML en columnas del tipo de datos xml. Se indexan todas las etiquetas, los valores y las rutas de acceso de las instancias XML de la columna y se mejora el rendimiento de las consultas. Un índice XML puede afectar positivamente a una aplicación en estas situaciones:

- Las consultas en columnas XML son habituales en su carga de trabajo. Es preciso considerar el costo de mantenimiento del índice XML durante la modificación de datos.
- Los valores XML son relativamente grandes y las partes recuperadas son relativamente pequeñas. La generación del índice evita tener que analizar todo el conjunto de datos en tiempo de ejecución y favorece las búsquedas basadas en índices que permiten un procesamiento más eficiente de las consultas.

Los índices XML se dividen en las categorías siguientes:

- Índice XML **principal**: Incluye todas las etiquetas, los valores y las rutas de acceso de las instancias XML de una columna XML. Para crear un índice XML principal, la tabla que contiene la columna XML debe tener un índice clúster en la clave principal de la tabla. SQL Server utiliza esta clave principal para correlacionar las filas del índice XML principal con las filas de la tabla que contiene la columna XML. Es un índice clúster.
- Índice XML **secundario*:*** Debe existir un índice XML principal. Son índices no clúster. Existen tres tipos:
  - **Path**: Mejoran las consultas con expresiones de ruta de acceso.
  - **Property**: Mejoran las consultas que recuperan uno o varios valores de instancias XML individuales
  - **Value**: Mejoran las consultas que se basan en valores.

El primer índice de la columna de tipo xml debe ser el índice XML principal. Con el índice XML principal, se admiten los siguientes tipos de índices secundarios: PATH, VALUE y PROPERTY. Dependiendo del tipo de consulta, los índices secundarios pueden contribuir a mejorar el rendimiento.

Cuando cree un índice XML, tenga en cuenta lo siguiente:

- Para crear un índice XML principal, la tabla que contiene la columna XML que se va a indexar, debe tener un índice clúster en la clave principal.
- Si ya existe un índice XML, la clave principal agrupada de la tabla no puede modificarse. Antes de modificar la clave principal, deberá quitar todos los índices XML de la tabla.
- Un índice XML principal puede crearse en una sola columna de tipo xml. No es posible crear ningún otro índice con una columna de tipo XML como columna de clave. No obstante, puede incluir la columna de tipo xml en un índice que no sea XML. Cada columna de tipo xml de una tabla puede tener su propio índice XML principal. No obstante, sólo se admite un índice XML principal por cada columna de tipo xml.
- Los índices XML existen en el mismo espacio de nombres que los índices que no son XML. Por tanto, no puede tener un índice XML y otro que no lo sea en la misma tabla y con el mismo nombre.

- Las opciones IGNORE_DUP_KEY y ONLINE siempre se establecen en OFF para los índices XML. Puede especificar estas opciones con el valor OFF.
- La información de partición o grupo de archivos de la tabla de usuarios se aplica al índice XML. Los usuarios no pueden especificar dicha información por separado en un índice XML.
- La opción de índice DROP_EXISTING permite quitar un índice XML principal y crear uno nuevo o efectuar la misma operación con un índice XML secundario. No obstante, esta opción no puede quitar un índice XML secundario para crear un índice XML principal ni viceversa.
- Los nombres de índice XML principal tienen las mismas restricciones que los nombres de vista.

Ejemplos:

```
CREATE PRIMARY XML INDEX PXML_ProductModel_CatalogDescription
ON Production.ProductModel (CatalogDescription)

CREATE XML INDEX XMLPATH_ProductModel_CatalogDescription
ON Production.ProductModel(CatalogDescription)
USING XML INDEX PXML_ProductModel_CatalogDescription
FOR PATH

CREATE XML INDEX XMLPROPERTY_ProductModel_CatalogDescription
ON Production.ProductModel(CatalogDescription)
USING XML INDEX PXML_ProductModel_CatalogDescription
FOR PROPERTY

CREATE XML INDEX XMLVALUE_ProductModel_CatalogDescription
ON Production.ProductModel(CatalogDescription)
USING XML INDEX PXML_ProductModel_CatalogDescription
FOR VALUE
```

Módulo 8

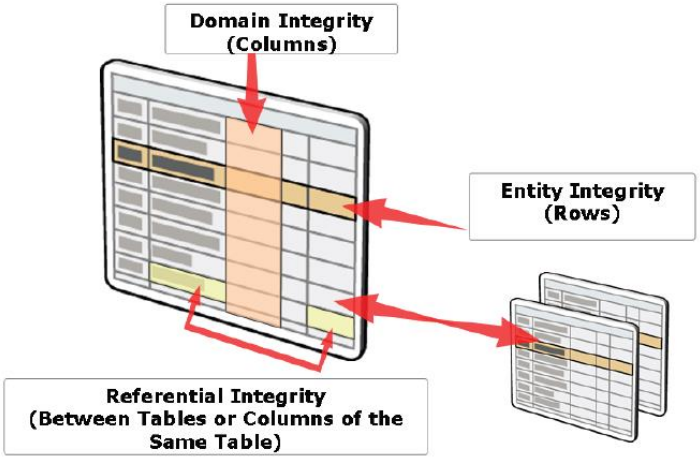
Integridad de Datos

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Integridad de Datos

Definición
La integridad de datos se refiere a los valores reales que se almacenan y se utilizan en las estructuras de datos de la aplicación. La aplicación debe ejercer un control deliberado sobre todos los procesos que utilicen los datos para garantizar la corrección permanente de la información.
Existen 3 tipos de integridad:

- **Dominio** (o columna): Valores válidos para las columnas y verificación de nulos.
- **Entidad** (o tabla): Requiere que todos los registros tengan un identificador único
- **Referencial**: Asegura las relaciones entre tablas



Opciones para reforzar la integridad de datos

- **Tipo de dato**: Controla el tipo de dato que puede guardarse en una columna. Es solo el primer paso.
- **Reglas y Valores Predeterminados** (Rules and Defaults): Las reglas definen valores aceptables para una columna y Defaults define valores predeterminado para las columnas cuando estás no reciben un valor específico. Son dos objetos independientes y pueden ser asociados a una o más columnas. No son compatibles con ANSI.
- **Restricciones** (Constraints): Define restricciones a los valores aceptables y es lo más recomendable para manejar la integridad de datos.
- **Desencadenadores** (Triggers): procedimientos almacenados especiales que se disparan ante la detección de eventos. Los eventos más comunes son INSERT, UPDATE y DELETE
- **Esquemas XML** (XML Schemas): Permite definir espacios de nombres, contenidos y estructuras para los tipos de datos xml.

2- Restricciones (Constraints)

Definición
Las restricciones son un método estándar ANSI de reforzar la integridad de datos. Existen distintos tipos de restricciones:

- PRIMARY KEY
- DEFAULT
- CHECK
- UNIQUE
- FOREIGN KEY

Se pueden crear restricciones usando **CREATE TABLA** o **ALTER TABLE** o usando el SQL Server Management Studio.

PRIMARY KEY
Una tabla suele tener una columna o una combinación de columnas cuyos valores identifican de forma única cada fila de la tabla. Estas columnas se denominan claves principales de la tabla y exigen la integridad de entidad de la tabla. Puede crear una clave principal mediante la definición de una restricción PRIMARY KEY cuando crea o modifica una tabla.

Consideraciones:

- Una tabla sólo puede tener una restricción PRIMARY KEY
- Ninguna columna a la que se aplique una restricción PRIMARY KEY puede aceptar valores nulos.
- Debido a que las restricciones PRIMARY KEY garantizan datos únicos, con frecuencia se definen en una columna de identidad (**Identity**)
- Si se define una restricción PRIMARY KEY para más de una columna, puede haber valores duplicados dentro de la misma columna, pero cada combinación de valores de todas las columnas de la definición de la restricción PRIMARY KEY debe ser única.

Cuando especifica una restricción PRIMARY KEY en una tabla, el Motor de base de datos exige la unicidad de los datos mediante la creación de un índice único para las columnas de la clave principal. Este índice también permite un acceso rápido a los datos cuando se utiliza la clave principal en las consultas.

Ejemplo:

```
CREATE TABLE [HumanResources].[Department](
[DepartmentID] [smallint] IDENTITY(1,1) NOT NULL,
[Name] [dbo].[Name] NOT NULL,
[GroupName] [dbo].[Name] NOT NULL,
[ModifiedDate] [datetime] NOT NULL,
CONSTRAINT [PK_Department_DepartmentID] PRIMARY KEY CLUSTERED
([DepartmentID] ASC )WITH (IGNORE_DUP_KEY = OFF)
ON [PRIMARY] )
```

DEFAULT
Cada columna de un registro debe contener un valor, aunque sea un valor nulo. Si la columna acepta valores nulos, puede cargar la fila con un valor nulo. Pero, dado que puede no resultar conveniente utilizar columnas que acepten valores nulos, una mejor solución podría ser establecer una definición DEFAULT o valor predeterminado para la columna siempre que sea necesario. Por ejemplo, es habitual especificar el valor cero como valor predeterminado para

las columnas numéricas, o N/D (no disponible) como valor predeterminado para las columnas de cadenas cuando no se especifica ningún valor.
Al cargar una fila en una tabla con una definición DEFAULT para una columna, se indica implícitamente al Motor de base de datos que cargue un valor predeterminado en la columna en la que no se haya especificado ningún valor. Si una columna no permite valores nulos y no tiene una definición DEFAULT, deberá especificar explícitamente un valor para la columna o el Motor de base de datos devolverá un error para indicar que la columna no permite valores nulos. También puede indicar explícitamente que se inserte un valor predeterminado para una columna mediante la cláusula DEFAULT VALUES de la instrucción INSERT.

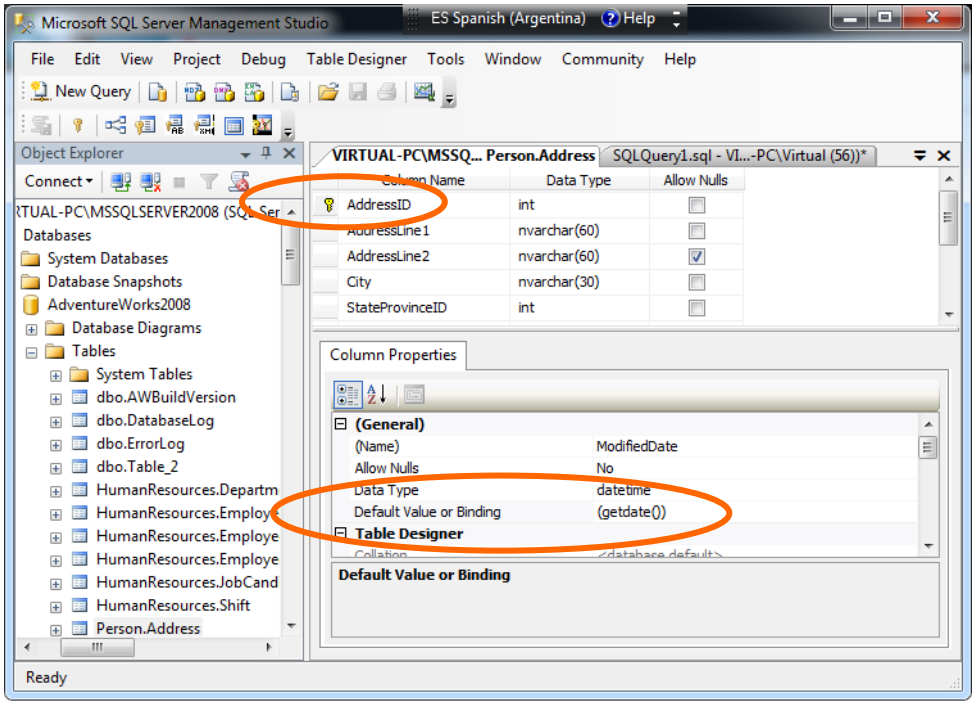
Consideraciones:

- Se aplica solo para la instrucción INSERT.
- Cada columna puede tener su propia restricción DEFAULT
- No se puede aplicar sobre columnas definidas **Identity**
- Acepta funciones y valores de sistema

Ejemplo:

```
CREATE TABLE [Production].[Location](
[LocationID] [smallint] IDENTITY(1,1) NOT NULL,
[Name] [dbo].[Name] NOT NULL,
[CostRate] [smallmoney] NOT NULL CONSTRAINT
[DF_Location_CostRate] DEFAULT ((0.00)),
[Availability] [decimal](8, 2) NOT NULL CONSTRAINT
[DF_Location_Availability] DEFAULT ((0.00)),
[ModifiedDate] [datetime] NOT NULL CONSTRAINT
[DF_Location_ModifiedDate]
DEFAULT (getdate()))
```

Vista de la Creación/Modificación de tablas en el SQL Server Management Studio. Permite marcar las restricciones PRIMARY KEY y DEFAULT.



CHECK
Las restricciones CHECK exigen la integridad del dominio mediante la limitación de los valores que puede aceptar una columna. Son similares a las restricciones FOREIGN KEY porque controlan los valores que se colocan en una columna. La diferencia reside en la forma en que determinan qué valores son válidos: las restricciones FOREIGN KEY obtienen la lista de valores válidos de otra tabla, mientras que las restricciones CHECK determinan los valores válidos a partir de una expresión lógica que se basa en datos de otra columna de la misma tabla. Por ejemplo, es posible limitar el intervalo de valores para una columna Porcentaje para que solo tome valores entre 0 y 100.
Puede crear una restricción CHECK con cualquier expresión lógica que devuelva TRUE (verdadero) o FALSE (falso) basándose en operadores lógicos.

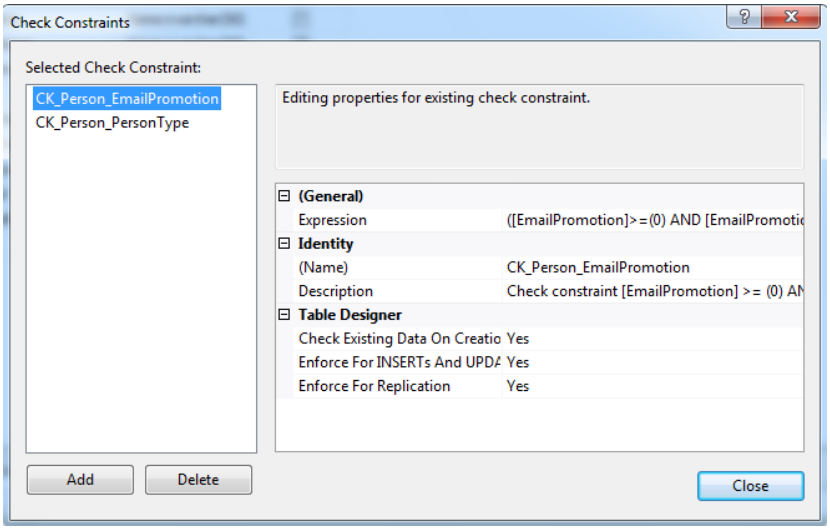
Consideraciones:

- Se verifica tanto en INSERT como en UPDATE
- No puede contener sub consultas
- Puede aplicar varias restricciones CHECK a una misma columna
- No puede ser aplicada a columnas con tipo de dato text, ntext o image.
- Si se aplica sobre una tabla que contiene datos, los mismos son verificados. Si algún registro no cumple con la condición la restricción no es aceptada.

Ejemplo:

```
ALTER TABLE [HumanResources].[EmployeeDepartmentHistory] WITH CHECK
ADD CONSTRAINT [CK EmployeeDepartmentHistory EndDate]
CHECK (([EndDate]>=[StartDate] OR [EndDate] IS NULL))
```

Restricción CHECK en el SQL Server Management Studio



UNIQUE
Puede utilizar restricciones UNIQUE para garantizar que no se escriben valores duplicados en columnas específicas que no forman parte de una clave principal. Tanto la restricción UNIQUE como la restricción PRIMARY KEY exigen la unicidad; sin embargo, debe utilizar la restricción UNIQUE y no PRIMARY KEY si desea exigir la unicidad de una columna o una combinación de columnas que no forman la clave principal.

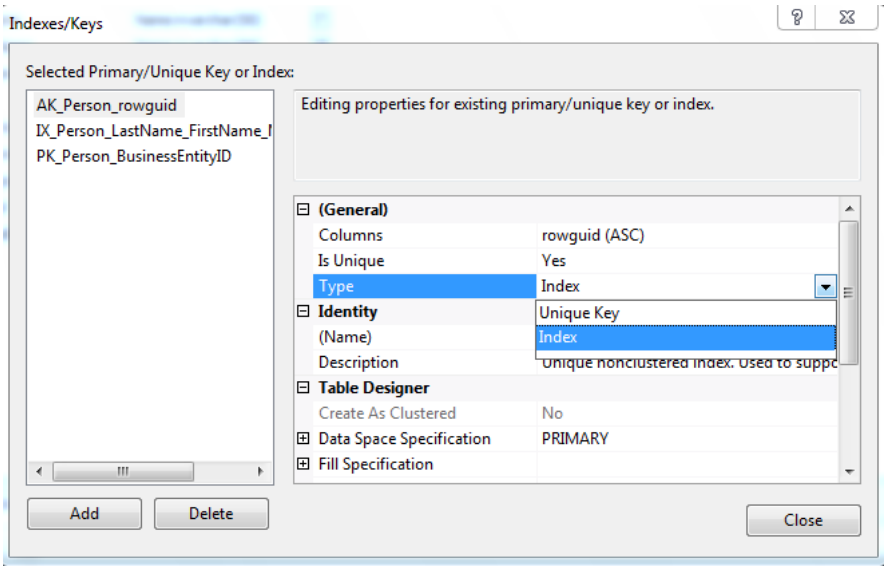
Consideraciones

- En una tabla se pueden definir varias restricciones UNIQUE, pero sólo una restricción PRIMARY KEY.
- Además, a diferencia de las restricciones PRIMARY KEY, las restricciones UNIQUE admiten valores nulos. Sin embargo, de la misma forma que cualquier valor incluido en una restricción UNIQUE, sólo se admite un valor nulo por columna.
- Es posible hacer referencia a una restricción UNIQUE con una restricción FOREIGN KEY.
- Crea un índice único para controlar la restricción.

Ejemplo:

```
CREATE TABLE [HumanResources].[Employee](
[EmployeeID] [int] IDENTITY(1,1) NOT NULL,
[NationalIDNumber] [nvarchar](15) NOT NULL UNIQUE NONCLUSTERED,
[ContactID] [int] NOT NULL,   …)
```

Restricción UNIQUE en el SQL Server Management Studio. Se crea a través del Índice.



FOREIGN KEY

Una clave externa (FK) es una columna o combinación de columnas que se utiliza para establecer y exigir un vínculo entre los datos de dos tablas. Puede crear una clave externa mediante la definición de una restricción FOREIGN KEY cuando cree o modifique una tabla.

En una referencia de clave externa, se crea un vínculo entre dos tablas cuando las columnas de una de ellas hacen referencia a las columnas de la otra que contienen el valor de clave principal. Esta columna se convierte en una clave externa para la segunda tabla.

No es necesario que una restricción FOREIGN KEY esté vinculada únicamente a una restricción PRIMARY KEY de otra tabla; también puede definirse para que haga referencia a las columnas de una restricción UNIQUE de otra tabla. Una restricción FOREIGN KEY puede contener valores nulos, pero si alguna columna de una restricción FOREIGN KEY compuesta contiene valores nulos, se omitirá la comprobación de los valores que componen la restricción FOREIGN KEY. Para asegurarse de que todos los valores de la restricción FOREIGN KEY compuesta se comprueben, especifique NOT NULL en todas las columnas que participan.

Consideraciones:

- La relación puede estar dada por una o varias columnas. La cantidad de columnas y los tipos de datos de la sentencia FOREIGN KEY debe coincidir con la cantidad de columnas y los tipos de datos especificados en la cláusula REFERENCE.
- No crea índices automáticamente.
- Puede referirse a columnas de la misma tabla.
- Un registro referenciado desde otra tabla no puede ser borrado ni modificado.

Ejemplo:

```
ALTER TABLE [Sales].[SalesOrderHeader] WITH CHECK ADD CONSTRAINT
[FK_SalesOrderHeader_Customer_CustomerID] FOREIGN KEY([CustomerID])
REFERENCES [Sales].[Customer] ([CustomerID])
```

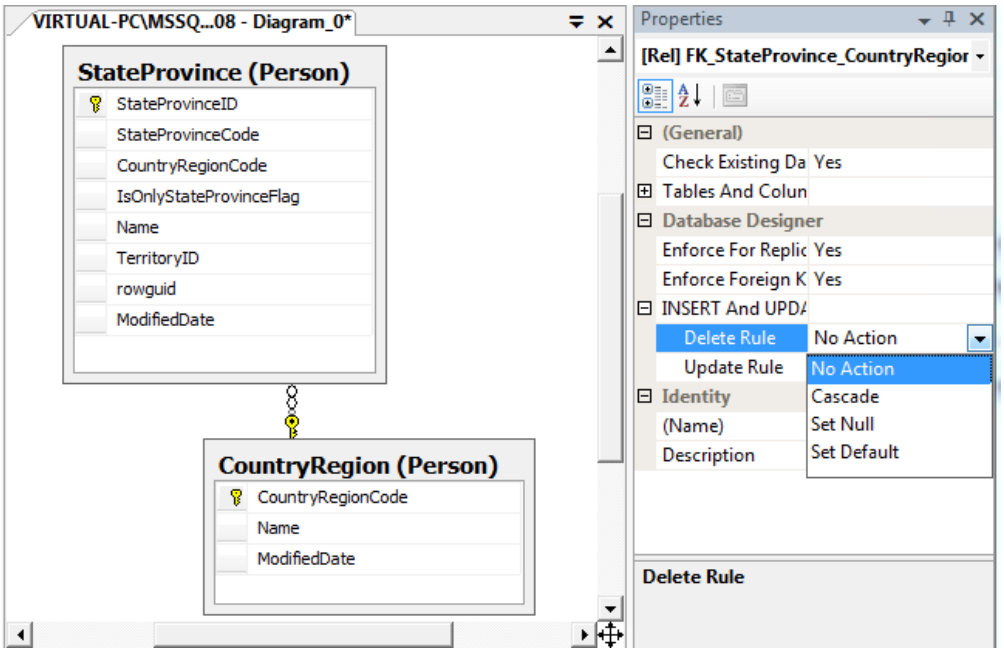
Restricciones de integridad referencial en cascada

Las restricciones de integridad referencial en cascada permiten definir las acciones que SQL Server lleva a cabo cuando un usuario intenta eliminar o actualizar una clave a la que apuntan las claves externas existentes.

Las cláusulas REFERENCES de las instrucciones CREATE TABLE y ALTER TABLE admiten las cláusulas ON DELETE y ON UPDATE. Ambas opciones admiten 4 valores:

- **NO ACTION**: Es el valor predeterminado si no se especifica ON DELETE u ON UPDATE. Si se intenta eliminar o actualizar una fila con una clave a la que hacen referencia las claves externas de las filas existentes en otras tablas, se produce un error y se revierte la instrucción DELETE o UPDATE.
- **CASCADE**: Si se intenta eliminar una fila con una clave a la que hacen referencia claves externas de filas existentes en otras tablas, todas las filas que contienen dichas claves externas también se eliminan. Si se intenta actualizar un valor de clave de una fila a cuyo valor de clave hacen referencia claves externas de filas existentes en otras tablas, también se actualizan todos los valores que conforman la clave externa al nuevo valor especificado para la clave.
- **SET NULL**: Si se intenta eliminar o actualizar una fila con una clave a la que hacen referencia las claves externas de las filas existentes de otras tablas, todos los valores que conforman la clave externa de las filas a las que se hace referencia se establecen en nulo. Todas las columnas de clave externa de la tabla de destino deben aceptar valores nulos para que esta restricción se ejecute.
- **SET DEFAULT**: Si se intenta eliminar o actualizar una fila con una clave a la que hacen referencia las claves externas de las filas existentes de otras tablas, todos los valores que conforman la clave externa de las filas a las que se hace referencia se establecen como predeterminados. Todas las columnas de clave externa de la tabla de destino deben tener una definición predeterminada para que esta restricción se ejecute. Si una columna acepta valores nulos y no se ha establecido ningún valor predeterminado explícito, NULL se convierte en el valor predeterminado implícito de la columna. Todos los valores distintos de nulo que se establecen debido a SET DEFAULT deben tener unos valores correspondientes en la tabla principal para mantener la validez de la restricción de la clave externa.

Vista de una FOREIGN KEY en un Diagrama de la Base de Datos en el SQL Server Management Studio.



Información sobre Restricciones

- Procedimientos almacenados de sistema:
  - **sp_HelpConstraint** devuelve información acerca de todas las restricciones de una tabla

```
EXEC sp_HelpConstraint 'production.product'
```

  - **sp_help**

- Vistas de Catálogo:
  - **sys.check_constraints** devuelve información sobre restricciones de tipo CHECK
  - **sys.foreign_keys** devuelve información sobre restricciones de tipo FK
  - **sys.default_constraints** devuelve información sobre restricciones de tipo DEFAULT

Consideraciones para el Uso de Restricciones

- Se debe especificar un nombre significativo cuando se crea una restricción, estos deben ser únicos dentro de la base de datos.
- Se pueden crear, modificar o borrar restricciones sin borrar ni recrear la tabla
- Se debe programar el control de errores dentro de la aplicación para cuando estas restricciones son violadas.
- Cuando se agrega o modifica una restricción FOREIGN KEY y CHECK sobre una tabla con datos existentes, estos son verificados. Se puede especificar que no ejecute este control.

Deshabilitación de Restricciones

Las restricciones FOREIGN KEY y CHECK pueden ser deshabilitadas. Las demás deben eliminarse y volver a crearse. Se puede decidir deshabilitar alguna de estas restricciones cuando se necesita ingresar un grupo de registros muy grandes y se desea mejorar la performance de este ingreso. Los datos van a ser verificados en las actualizaciones

posteriores. Para deshabilitar estas restricciones se usan las cláusulas WITH NO CHECK cuando se crea una nueva restricción y NOCHECK cuando se modifica una restricción existente.

Ejemplos:

```
ALTER TABLE [Sales].[SalesOrderHeader] WITH NOCHECK
ADD CONSTRAINT [FK_SalesOrderHeader_Customer_CustomerID]
FOREIGN KEY([CustomerID])
REFERENCES [Sales].[Customer] ([CustomerID])

ALTER TABLE [Sales].[SalesOrderHeader] NOCHECK
CONSTRAINT [FK_SalesOrderHeader_Customer_CustomerID]
```


BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

3- Desencadenadores (Triggers)

Definición
Los desencadenadores son una clase especial de procedimientos almacenados que son definidos para ser ejecutados automáticamente junto a un comando UPDATE, INSERT o DELETE sobre una tabla o una vista. Los desencadenadores son poderosas herramientas que pueden ser utilizados para aplicar las reglas de negocio de manera automática en el momento en que los datos son modificados. Pueden comprender el control lógico que realizan las restricciones, valores por defecto, y reglas de SQL Server (aún cuando es recomendable usar restricciones y valores predeterminados antes que desencadenadores en la medida que respondan a todas las necesidades de control de integridad de datos).

Consideraciones:

- El desencadenador y la sentencia que lo dispara son tratados como una sola transacción, que podría ser cancelada desde el desencadenador.
- Pueden manejar cambios en cascada, pero es preferible usar la integridad referencial en cascada.
- Puede definir restricciones más complejas que las soportadas por CHECK, puede referenciar columnas de otras tablas
- Puede evaluar el estado de la tabla antes y después de la modificación y ejecutar alguna tarea basándose en la diferencia.
- Permite varios desencadenadores del mismo tipo (INSERT, UPDATE, DELETE) sobre la misma tabla. Sólo se puede especificar el primer y último desencadenador AFTER para cada una de las operaciones INSERT, UPDATE y DELETE de una tabla. Si hay otros desencadenadores AFTER en la misma tabla, se ejecutan aleatoriamente.

Limitaciones:

- Un desencadenador se crea solamente en la base de datos actual; sin embargo, un desencadenador puede hacer referencia a objetos que están fuera de la base de datos actual.
- Si se especifica el nombre del esquema del desencadenador (para calificar el desencadenador), califique el nombre de la tabla de la misma forma.
- La misma acción del desencadenador puede definirse para más de una acción del usuario (por ejemplo, INSERT y UPDATE) en la misma instrucción CREATE TRIGGER.
- Los desencadenadores INSTEAD OF DELETE/UPDATE no pueden definirse en una tabla con una clave externa definida en cascada en la acción DELETE/UPDATE.
- En un desencadenador se puede especificar cualquier instrucción SET. La opción SET seleccionada permanece en efecto durante la ejecución del desencadenador y, después, vuelve a su configuración anterior.
- Cuando se activa un desencadenador, los resultados se devuelven a la aplicación que llama, exactamente igual que con los procedimientos almacenados. Para impedir que se devuelvan resultados a la aplicación debido a la activación de un desencadenador, no incluya las instrucciones SELECT que devuelven resultados ni las instrucciones que realizan una asignación variable en un desencadenador. Un desencadenador que incluya instrucciones SELECT que devuelven resultados al usuario o instrucciones que realizan asignaciones de variables requiere un tratamiento especial; estos resultados devueltos tendrían que escribirse en cada aplicación en la que se permiten modificaciones a la tabla del desencadenador. Si es preciso que existan asignaciones de variable en un desencadenador, utilice una instrucción SET NOCOUNT al principio del mismo para impedir la devolución de cualquier conjunto de resultados.
- Una instrucción TRUNCATE TABLE es de hecho una instrucción DELETE, pero no activa un desencadenador porque la operación no registra eliminaciones de filas

individuales. Sin embargo, sólo los usuarios con permisos para ejecutar una instrucción TRUNCATE TABLE tienen que ocuparse de cómo sortear un desencadenador de DELETE de esta manera.

Creación

```
CREATE TRIGGER [ nombre_esquema . ]nombre_trigger
ON { tabla | vista }
[ WITH <opciones_trigger_dml> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sentencias_sql [ ; ] [ ...n ] | EXTERNAL NAME <method specifier [ ; ] > }
```

Argumentos

- **nombre_esquema**: Es el nombre del esquema al que pertenece un desencadenador
- **Nombre_trigger**: Es el nombre del desencadenador. Debe cumplir las reglas de los identificadores. La especificación del nombre del propietario es opcional.
- **tabla | vista**: Es la tabla o vista en que se ejecuta el desencadenador
- **FOR | AFTER | INSTEAD OF**: Tipo de desencadenador
- **DELETE | INSERT | UPDATE**: Instrucción que dispara el desencadenador. Se debe especificar al menos una opción. En la definición del desencadenador se permite cualquier combinación de estas opciones, en cualquier orden.
- **WITH APPEND*:*** Especifica que debe agregarse un desencadenador adicional de un tipo existente.
- **NOT FOR REPLICATION*:*** Indica que el desencadenador no debe ejecutarse cuando un agente de replicación modifica la tabla involucrada en el mismo

Modificación y Borrado

Para modificar o borrar un desencadenador se deberán usar las instrucciones **ALTER TRIGGER** y **DROP TRIGGER**. Si se elimina una tabla todos sus desencadenadores se eliminan también.

Tipos de Desencadenadores

- **FOR | AFTER**: AFTER específica que el desencadenador se activa cuando todas las operaciones especificadas en la instrucción SQL desencadenadora se han ejecutado correctamente. AFTER es el valor predeterminado cuando sólo se especifica la palabra clave FOR. No se pueden definir en las vistas.
- **INSTEAD OF*:*** Especifica que se ejecuta el desencadenador en vez de la instrucción SQL desencadenadora. Se puede definir sobre vistas. Como máximo, se puede definir un desencadenador INSTEAD OF por cada instrucción INSERT, UPDATE o DELETE en cada tabla o vista. No obstante, en las vistas es posible definir otras vistas que tengan su propio desencadenador INSTEAD OF.

Desencadenadores vs. Restricciones

El principal beneficio de los desencadenadores es que pueden contener lógica compleja que usa código Transact-SQL. Sin embargo es preferible usar las restricciones para reforzar la integridad de datos. Se deberían usar los desencadenadores solo para funcionalidades no soportada por las restricciones.

Por ejemplo:

- Los desencadenadores pueden realizar cambios en cascada mediante tablas relacionadas de la base de datos; sin embargo, estos cambios pueden ejecutarse de manera más eficaz mediante restricciones de integridad referencial en cascada.
- Las restricciones sólo pueden comunicar la existencia de errores mediante mensajes de error estándar del sistema. Si la aplicación necesita mensajes personalizados y un control de errores más complejo, deberá utilizar un desencadenador.
- Los desencadenadores pueden impedir o revertir los cambios que infrinjan la integridad referencial y cancelar, de ese modo, cualquier intento de modificación de los datos. Ese tipo de desencadenador puede activarse cuando se cambia una clave externa y el nuevo valor no coincide con su clave principal. No obstante, para estos casos suelen utilizarse restricciones FOREIGN KEY.
- Si hay restricciones en la tabla de desencadenadores, se comprobarán después de la ejecución del desencadenador INSTEAD OF, pero antes de la ejecución del desencadenador AFTER. Si se infringen las restricciones, se revertirán las acciones del desencadenador INSTEAD OF y no se ejecutará el desencadenador AFTER.

Como trabaja el desencadenador de INSERT

Un desencadenador de INSERT se ejecuta después o en reemplazo de la instrucción INSERT de una tabla o vista en donde el mismo fue configurado.
Al ejecutarse la instrucción INSERT, si un desencadenador de INSERT existe, una nueva tabla temporaria residente en memoria llamada **Inserted** es creada. Esta tabla contiene una copia de los registros que fueron afectados en esta última operación. El desencadenador puede referenciar esta tabla en su código para comparar cambios, verificarlos o cualquier otra tarea.

Ejemplo:

```
CREATE TRIGGER [insrtWorkOrder] ON [Production].[WorkOrder]
AFTER INSERT AS
BEGIN
SET NOCOUNT ON;
INSERT INTO [Production].[TransactionHistory](
[ProductID],[ReferenceOrderID],[TransactionType],
[TransactionDate],[Quantity],[ActualCost])
SELECT inserted.[ProductID],inserted.[WorkOrderID]
,'W',GETDATE(),inserted.[OrderQty],0
FROM inserted;
END
```

Como trabaja el desencadenador de DELETE

Un desencadenador de DELETE se ejecuta después o en reemplazo de la instrucción DELETE de una tabla o vista en donde el mismo fue configurado.
Al ejecutarse la instrucción DELETE, si un desencadenador de DELETE existe, una nueva tabla temporaria residente en memoria llamada **Deleted** es creada. Esta tabla contiene una copia de los registros que fueron afectados en esta última operación. El desencadenador puede referenciar esta tabla en su código para verificarlos o cualquier otra tarea. Tenga en cuenta que los registros incluidos en la tabla **Deleted** ya no existen en la tabla original. Este desencadenador no se ejecuta con la instrucción **TRUNCATE TABLE** ya que la misma no usa el registro de transacciones.

Ejemplo:

```
CREATE TRIGGER [delCustomer] ON [Sales].[Customer]
AFTER DELETE AS
BEGIN
SET NOCOUNT ON;
EXEC master..xp_sendmail
@recipients=N'SalesManagers@Adventure-Works.com',
@message = N'Customers have been deleted!!';
END
```

Como trabaja el desencadenador de UPDATE
Un desencadenador de UPDATE se ejecuta después o en reemplazo de la instrucción UPDATE de una tabla o vista en donde el mismo fue configurado.

Al ejecutarse la instrucción UPDATE, si un desencadenador de UPDATE existe, dos nuevas tablas temporarias residentes en memoria son creada:

- **Deleted**: Contiene los registros afectados con los datos antes de la modificación
- **Inserted**: Contiene los registros afectados con los datos después de la modificación

El desencadenador puede referenciar ambas tablas en su código para comparar cambios, verificarlos o cualquier otra tarea.
Puede usar también la sentencia **IF UPDATE(columna)** que permite examinar si la columna especificada ha sido modificada de alguna manera. Esto permite que algunas acciones estén asociadas con las modificaciones a determinadas columnas.

Ejemplo:

```
CREATE TRIGGER [uWorkOrder] ON [Production].[WorkOrder]
AFTER UPDATE AS
BEGIN
      SET NOCOUNT ON;
      IF UPDATE([ProductID]) OR UPDATE([OrderQty])
      BEGIN
      INSERT INTO [Production].[TransactionHistory]([ProductID],
      [ReferenceOrderID], [TransactionType],[TransactionDate],
      [Quantity])
      SELECT inserted.[ProductID], inserted.[WorkOrderID],
      'W', GETDATE(), inserted.[OrderQty]
      FROM inserted
      UPDATE [Production].[WorkOrder]
      SET [Production].[WorkOrder].[ModifiedDate] = GETDATE()
      FROM inserted
      WHERE inserted.[WorkOrderID] =
      [Production].[WorkOrder].[WorkOrderID]
      END
END
```

Desencadenadores INSTEAD OF
Los desencadenadores INSTEAD OF pasan por alto las acciones estándar de la instrucción de desencadenamiento: INSERT, UPDATE o DELETE. Se puede definir un desencadenador INSTEAD OF para realizar comprobación de errores o valores en una o más columnas y, a continuación, realizar acciones adicionales antes de insertar el registro. Por ejemplo, cuando el valor que se actualiza en una columna de tarifa de una hora de trabajo de una tabla de nómina excede un valor específico, se puede definir un desencadenador para producir un error y

revertir la transacción, o insertar un nuevo registro en un registro de auditoría antes de insertar el registro en la tabla de nómina. Sin embargo los desencadenadores INSTEAD OF son de mayor utilidad para ampliar los tipos de actualizaciones que una vista puede admitir. Por ejemplo, pueden proporcionar la lógica para modificar varias tablas base mediante una vista.
Se puede definir desencadenadores INSTEAD OF en tablas o vistas; siempre que la vista no tenga la opción WITH CHECK OPTION.

Ejemplo:

```
CREATE TRIGGER [delEmployee] ON [HumanResources].[Employee]
INSTEAD OF DELETE NOT FOR REPLICATION AS
BEGIN
SET NOCOUNT ON;
DECLARE @DeleteCount int;
SELECT @DeleteCount = COUNT(*) FROM deleted;
IF @DeleteCount > 0
      BEGIN
      RAISERROR(N'Los empleados no pueden borrarse. Solo pueden ser
                  marcados como deshabilitados', -- Message, 10, --
      Severity.1); -- State.
      IF @@TRANCOUNT > 0
         ROLLBACK TRANSACTION;
      END;
END;
```

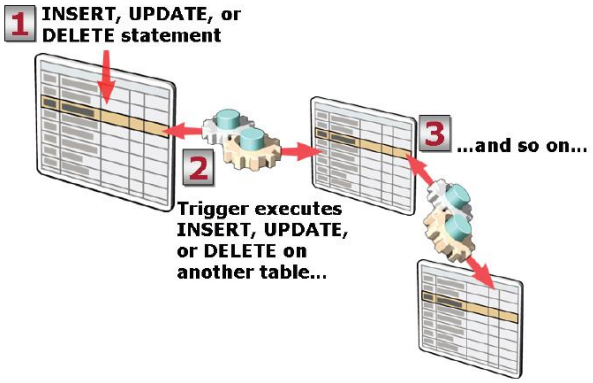
Desencadenadores Anidados
Los desencadenadores están anidados cuando un desencadenador realiza una acción que inicia otro desencadenador. Estas acciones pueden iniciar otros desencadenadores y así sucesivamente. Los desencadenadores se pueden anidar hasta un máximo de 32 niveles. Puede controlar si los desencadenadores AFTER se pueden anidar en la opción de configuración del servidor **nested triggers**. Los desencadenadores INSTEAD OF se pueden anidar independientemente de esta configuración.

Consideraciones:

- Los desencadenadores anidados están habilitados como valor predeterminado
- Un mismo desencadenador no se dispara dos veces dentro de la misma transacción. No se llama a sí mismo como respuesta a una segunda modificación de la tabla. Sin embargo si el desencadenador modifica otra tabla y dispara un segundo desencadenador y en este segundo desencadenador modifica la tabla original, el desencadenador original se disparará o no dependiendo si está habilitada o no esta funcionalidad.
- Dado que los desencadenadores se ejecutan dentro de una transacción, un error en cualquier nivel de un conjunto de desencadenadores anidados anula toda la transacción y provoca que se reviertan todas las modificaciones de datos.
- Se puede usar @@**NestLevel** para verificar el nivel de anidado en donde se encuentra

Para deshabilitar esta opción se puede usar

```
sp_configure 'nested triggers', 0
```



Desencadenadores Recursivos
SQL Server permite también la invocación recursiva de desencadenadores cuando el valor **RECURSIVE_TRIGGERS** está habilitado.

Los desencadenadores recursivos permiten dos tipos de recursividad:

- **Indirecta**: Una aplicación actualiza la tabla T1. Así se activa el desencadenador TR1 para actualizar la tabla T2. En esta situación, el desencadenador T2 activa y actualiza la tabla T1.
- **Directa**: Con la repetición directa, una aplicación actualiza la tabla T1. Así se activa el desencadenador TR1 para actualizar la tabla T1. Debido a que la tabla T1 se ha actualizado, el desencadenador TR1 se activa de nuevo, y así sucesivamente.

Las tablas **Inserted** y **Deleted** solo contienen información asociada a la última operación INSERT, DELETE o UPDATE.
Deshabilitar **RECURSIVE_TRIGGERS** sólo evita las repeticiones directas. Para deshabilitar la repetición indirecta, establezca la opción **nested triggers** del servidor. Si los desencadenadores anidados están desactivados, los desencadenadores recursivos también se deshabilitan, independientemente del valor de **RECURSIVE_TRIGGERS**
Si alguno de los desencadenadores ejecuta una instrucción ROLLBACK TRANSACTION, no se ejecuta ningún desencadenador posterior, independientemente del nivel de anidamiento.

Consideraciones:

- Los desencadenadores recursivos son complejos y deben estar muy bien diseñados y probados. Deben tener lógica de control del ciclo sino pueden terminar en un ciclo infinito.
- Pueden tener problemas si los datos deben ser modificados en un cierto orden
- Se puede crear funcionalidades similares sin usar desencadenadores recursivos diseñando el desencadenador de forma totalmente diferente, conteniendo en manejo del ciclo dentro del mismo.

Para habilitar o deshabilitar esta opción puede usar:

```
ALTER DATABASE AdventureWorks2008 SET RECURSIVE_TRIGGERS ON
EXEC sp_dboption AdventureWorks2008, 'recursive triggers', True
```

3- Esquemas XML (XML Schemas)

Definición
Los esquemas XML son documentos que se utilizan para definir y validar el contenido y la estructura de datos XML, del mismo modo que un esquema de base de datos define y valida las tablas, columnas y tipos de datos que componen una base de datos. Los elementos del esquema XML (elementos, atributos, tipos y grupos) se utilizan para definir la estructura válida, un contenido de datos válido y las relaciones de determinados tipos de datos XML.
Su sintaxis está definida por La World Wide Web Consortium (W3C).

Colecciones de Esquemas XML
Los esquemas XML son registrados dentro de colecciones de esquemas XML como objetos de la base de datos. Cada colección puede tener uno o varios esquemas, cada uno de ellos identificado con un espacio de nombres que se especifica en el atributo **targetNamespace** del esquema.

Creación de Colecciones de Esquemas

CREATE XML SCHEMA COLLECTION [<esquema_relacional>.]identificador_sql
AS Expresión

Para modificar o borrar colecciones puede usar ALTER/DROP XML SCHEMA COLLECTION.

Argumentos:

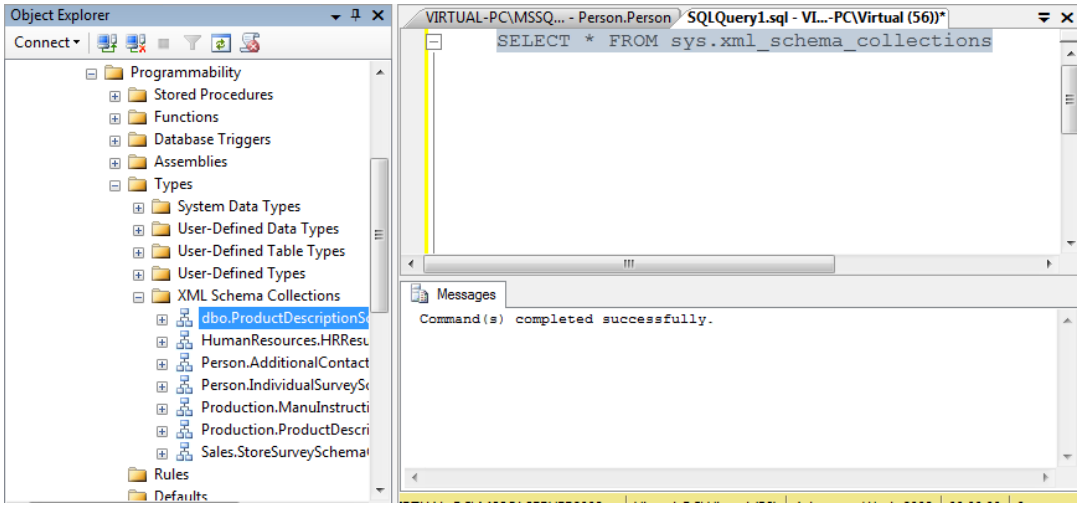
- **esquema_relacional**: Identifica el nombre del esquema relacional. Si no se especifica, se asume el esquema relacional predeterminado.
- **identificador_sql**: Es el identificador de SQL para la colección del esquema XML. Debe cumplir las reglas de los identificadores.
- **Expresión**: Es una constante de cadena o una variable escalar. Es de tipo varchar, varbinary, nvarchar o xml.

Ejemplo:

```
CREATE XML SCHEMA COLLECTION ProductDescriptionSchemaCollection
AS '<xsd:schema
targetNamespace="http://schemas.microsoft.com/sqlserver/2004/07/advent
ure-  works/ProductModelWarrAndMain"
xmlns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/ProductModelWarrAndMain"  elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
<xsd:element name="Warranty"  >
     <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="WarrantyPeriod" type="xsd:string" />
            <xsd:element name="Description" type="xsd:string"  />
        </xsd:sequence>
     </xsd:complexType>
</xsd:element>
</xsd:schema>'
```

Para obtener información acerca de estas colecciones puede usar el SQL Server Management Studio o la vista de catálogo **sys.xml_schema_collections**

```
SELECT * FROM sys.xml_schema_collections
```



XML con Tipo

Se pueden crear variables, parámetros y columnas del tipo de datos xml. Opcionalmente, se puede asociar una colección de esquemas XML a una variable, a un parámetro o a una columna de tipo xml. En este caso, se dice que la instancia del tipo de datos xml es una instancia con tipo. En los demás casos, se dice que la instancia XML es una instancia sin tipo.
Al asignarle un valor a una columna, parámetro o variable con tipo de dato xml con tipo el valor será verificado con el esquema asociado.
Columnas, parámetros y variables XML con tipo pueden almacenar documentos o contenido XML. Esto se especifica en la declaración junto a la colección de esquemas XML. Especifique DOCUMENT si cada instancia XML tiene exactamente un elemento de nivel superior (Root). En caso contrario, use CONTENT (permite XML fragmentados). De manera predeterminada, las instancias se almacenan en la columna xml con tipo como contenido XML y no como documentos XML.

Ejemplos: En este ejemplo se puede ver cómo crear una columna de una tabla con tipo de dato xml con tipo (como contenido ya que no se aclara explícitamente):

```
CREATE TABLE HumanResources.EmployeeResume
(EmployeeID int,  Resume xml (ResumeSchemaCollection))
```

En este ejemplo se crea como Documento.

```
CREATE TABLE Prueba
(Col1 xml (DOCUMENT Production.ProductDescriptionSchemaCollection))
```

También podría asociarse a un parámetro de un procedimiento almacenado:

```
CREATE PROCEDURE SampleProc @ProdDescription xml
(Production.ProductDescriptionSchemaCollection)
AS ....
```

Módulo 9

Vistas

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Introducción

Definición
Una vista (view) es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla real, una vista consta de un conjunto de columnas y filas de datos con un nombre. Sin embargo, a menos que esté indexada, una vista no existe como conjunto de valores de datos almacenados en una base de datos. Las filas y las columnas de datos proceden de tablas a las que se hace referencia en la consulta que define la vista y se producen de forma dinámica cuando se hace referencia a la vista.
Una vista actúa como filtro de las tablas subyacentes a las que se hace referencia en ella. La consulta que define la vista puede provenir de una o de varias tablas, o bien de otras vistas de la base de datos actual u otras bases de datos. Asimismo, es posible utilizar las consultas distribuidas para definir vistas que utilicen datos de orígenes heterogéneos. Esto puede resultar de utilidad, por ejemplo, si desea combinar datos de estructura similar que proceden de distintos servidores, cada uno de los cuales almacena los datos para una región distinta de la organización.

Su usan comúnmente para:

- Centrar, simplificar y personalizar la percepción de la base de datos para cada usuario. Centrarse en los datos más importantes de una tabla, en cómo se presentan o armar vistas con JOIN y UNION de uso frecuente.
- Como mecanismos de seguridad, que permiten a los usuarios obtener acceso a los datos por medio de la vista, pero no les conceden el permiso de obtener acceso directo a las tablas bases subyacentes.
- Para proporcionar compatibilidad con versiones anteriores con el fin de emular una tabla que existía pero cuyo esquema ha cambiado.
- Para exportar e importar datos.
- Para combinar datos de particiones.

Al crear una vista tener en cuenta:

- Se debe ser miembro de alguno de los siguientes roles: **sysadmin**, **db_owner** o **db_ddladmin** o tener permisos para CREATE VIEW y ALTER SCHEMA para el esquema en donde se incluirá la vista dentro de la base de datos. También se deben tener permisos de SELECT sobre las tablas y vistas referenciadas.
- Solo se pueden crear vista en la base de datos en curso
- Se puede construir vistas sobre vistas. El anidado soporta hasta 32 niveles.
- Puede contener un máximo de 1024 columnas
- No se pueden crear vistas temporarias.
- No se pueden creas vistas sobre tablas temporarias.
- No soportan desencadenadores AFTER.

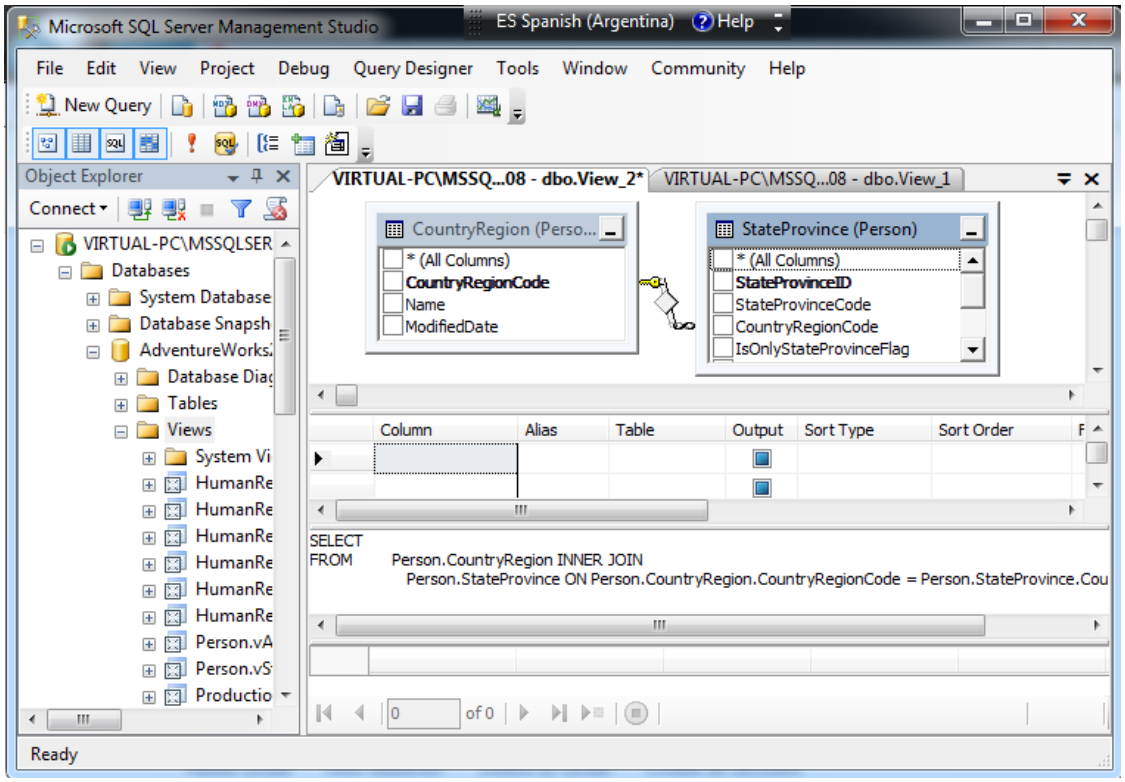
Tipos de Vistas

- **Vistas estándar**: La combinación de datos de una o más tablas mediante una vista estándar permite satisfacer la mayor parte de las ventajas de utilizar vistas.
- **Vistas indexadas**: Una vista indexada es una vista que se ha materializado. Esto significa que se ha calculado y almacenado. Se puede indexar una vista creando un índice clúster único en ella. Las vistas indexadas mejoran de forma considerable el rendimiento de algunos tipos de consultas.
- **Vistas con particiones**: Una vista con particiones junta datos horizontales con particiones de un conjunto de tablas miembro en uno o más servidores.

2- Creación

Creación
Se pueden crear usando la herramienta SQL Server Management Studio o usando la instrucción CREATE VIEW.

Creación de vista con SQL Server Management Studio




```
CREATE VIEW [nombre_esquema] Nombre_Vista [ (columnas [ ,...n ] ) ]
[ WITH [ ENCRYPTION ] [, SCHEMABINDING ] [, VIEW_METADATA ] ]
AS sentencia_Select [ ; ]
[ WITH CHECK OPTION ]
```

Argumentos

- **nombre_esquema**: Es el nombre del esquema al que pertenece la vista.
- **Nombre_Vista**: Es el nombre de la vista. Los nombres de las vistas deben cumplir las reglas de los identificadores. La especificación del nombre del propietario de la vista es opcional.
- **Columnas**: Es el nombre que se va a utilizar para una columna en una vista. Sólo se necesita un nombre de columna cuando una columna proviene de una expresión aritmética, una función o una constante; cuando dos o más columnas puedan tener el mismo nombre, normalmente debido a una combinación; o cuando una columna de una vista recibe un nombre distinto al de la columna de la que proviene. Los nombres de columna se pueden asignar también en la instrucción SELECT. Si no se especifica el parámetro columnas, las columnas de la vista adquieren los mismos nombres que las columnas de la instrucción SELECT.

- **Sentencia_Select**: Es la instrucción SELECT que define la vista. Dicha instrucción puede utilizar más de una tabla y otras vistas. Se necesitan permisos adecuados para seleccionar los objetos a los que se hace referencia en la cláusula SELECT de la vista que se ha creado. Una vista no tiene por qué ser un simple subconjunto de filas y de columnas de una tabla determinada. Es posible crear una vista que utilice más de una tabla u otras vistas mediante una cláusula SELECT de cualquier complejidad. En una definición de vista indexada, la instrucción SELECT debe ser una instrucción de una única tabla o una instrucción JOIN de varias tablas con agregación opcional.
- **WITH CHECK OPTION**: Exige que todas las instrucciones de modificación de datos ejecutadas en la vista sigan los criterios establecidos en la sentencia SELECT. Cuando una fila se modifica mediante una vista, WITH CHECK OPTION garantiza que los datos permanezcan visibles en toda la vista después de confirmar la modificación.
- **WITH ENCRYPTION*:*** Cifra las entradas de **sys.syscomments** que contienen el texto de la instrucción CREATE VIEW. El uso de WITH ENCRYPTION evita que la vista se publique como parte de la replicación de SQL Server.
- **WITH SCHEMABINDING**: Enlaza la vista al esquema de las tablas subyacentes. Cuando se especifica SCHEMABINDING, las tablas base no se pueden modificar de una forma que afecte a la definición de la vista. En primer lugar, se debe modificar o quitar la propia definición de la vista para quitar las dependencias en la tabla que se va a modificar. Cuando se utiliza SCHEMABINDING, la sentencia SELECT debe incluir los nombres de dos partes (esquema.objeto) de las tablas, vistas o funciones definidas por el usuario a las que se hace referencia. Todos los objetos a los que se hace referencia se deben encontrar en la misma base de datos. Las vistas o las tablas que participan en una vista creada con la cláusula SCHEMABINDING no se pueden eliminar a menos que se elimine o cambie esa vista de forma que deje de tener un enlace de esquema.
- **WITH VIEW_METADATA**: Especifica que la instancia de SQL Server devolverá a las API de DB-Library, ODBC y OLE DB la información de metadatos sobre la vista en vez de las tablas base cuando se soliciten los metadatos del modo de exploración para una consulta que hace referencia a la vista.

Ejemplo:

```
CREATE VIEW [HumanResources].[vEmployee]
AS
SELECT
e.[EmployeeID],c.[Title],c.[FirstName],c.[MiddleName],c.[LastName]
,c.[Suffix],e.[Title] AS [JobTitle],c.[Phone],c.[EmailAddress]
,c.[EmailPromotion],a.[AddressLine1],a.[AddressLine2],a.[City]
,sp.[Name] AS [StateProvinceName],a.[PostalCode]
,cr.[Name] AS [CountryRegionName],c.[AdditionalContactInfo]
FROM [HumanResources].[Employee] e
INNER JOIN [Person].[Contact] c ON c.[ContactID] = e.[ContactID]
INNER JOIN [HumanResources].[EmployeeAddress] ea
ON e.[EmployeeID] = ea.[EmployeeID]
INNER JOIN [Person].[Address] a ON ea.[AddressID] = a.[AddressID]
INNER JOIN [Person].[StateProvince] sp
ON sp.[StateProvinceID] = a.[StateProvinceID]
INNER JOIN [Person].[CountryRegion] cr
ON cr.[CountryRegionCode] = sp.[CountryRegionCode]
```

Restricciones de la cláusula SELECT

Las cláusulas SELECT de una definición de vista no pueden incluir lo siguiente:

- Cláusulas COMPUTE o COMPUTE BY

- Una cláusula ORDER BY, a menos que también haya una cláusula TOP en la lista de selección de la instrucción SELECT
- La palabra clave INTO
- Una referencia a una tabla temporal o a una variable de tabla

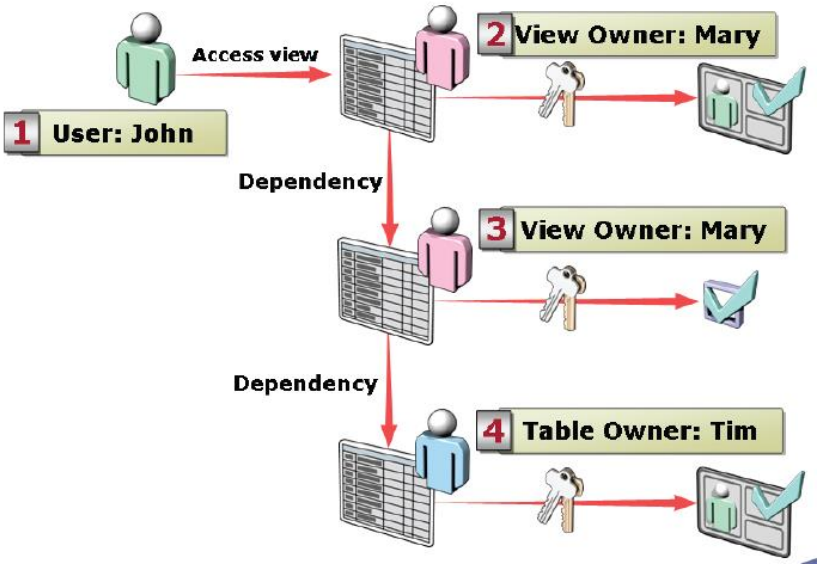
Modificación y borrado
Las vistas se pueden modificar o borrar usando la herramienta SQL Server Management Studio o las instrucciones **ALTER VIEW** o **DROP VIEW**.
Al modificar una vista se puede cambiar su sentencia SELECT o cualquiera de sus opciones. ALTER VIEW no afecta a desencadenadores ni procedimientos almacenados dependientes y no cambia los permisos asignados a la vista.
Al borrar una vista se borra la misma y todos sus permisos. No borra otros objetos que referencien a la misma. Cuando se ejecuta en una vista indexada, automáticamente se eliminan todos los índices de una vista.

Problemas de Cadenas de Propiedad
Cuando varios objetos de una base de datos obtienen acceso unos a otros de forma secuencial, la secuencia se denomina cadena. Aunque estas cadenas no existen de manera independiente, cuando SQL Server recorre los eslabones de una cadena evalúa los permisos de los objetos que la componen de manera distinta a si se estuviese obteniendo acceso a los objetos por separado. Estas diferencias tienen implicaciones importantes en lo que se refiere a administrar la seguridad.
Las cadenas de propiedad permiten administrar el acceso a varios objetos, por ejemplo, a varias tablas, estableciendo permisos en un objeto, como una vista. El encadenamiento de propiedad también ofrece una pequeña mejora en el rendimiento en los escenarios que permiten la omisión de comprobaciones de permisos.

Cuando se obtiene acceso a un objeto mediante una cadena, SQL Server primero compara al propietario del objeto con el propietario del objeto de llamada.
- Si ambos objetos tienen el mismo propietario, los permisos del objeto de referencia no se evalúan.
- Si ambos objetos tienen distinto propietario, los permisos se evalúan en cada nivel de la llamada, por lo cual se puede tener permisos por ejemplo sobre una vista pero no sobre la tabla que referencia dicha vista, por lo tanto no podrá ejecutar la vista a pesar de tener el permiso correspondiente sobre ella.

Para evitar romper la cadena de propietarios asegúrese de que todas las vistas y sus tablas subyacentes pertenezcan al mismo propietario.

Ejemplo:



Obtener información sobre vistas

- SQL Server Management Studio
- Vistas de catálogo:
  - **Sys_views** devuelve la lista de vistas de la base de datos
  - **sys.sql_modules** devuelve la lista de vistas, procedimientos, funciones y desencadenadores de la base de datos
  - **Sys_Sql_Dependencies** permite verificar las dependencias del objeto.
- Procedimiento almacenado:
  - **sp_HelpText** permite ver la definición de una vista
  - **sp_Help**
  - **sp_depends** permite verificar las dependencias del objeto.

Cifrado

La definición de una vista se guarda en la tabla de sistema **sys.syscomments**. Para proteger la lógica de esta definición es posible cifrarla usando la cláusula **WITH ENCRYPTION**. De esta manera es imposible leer la definición de la misma. Se recomienda guardar una copia de la definición antes de cifrarla, ya que si no lo hace no se podrá acceder a la definición si esto fuera necesario. Si se crea una vista cifrada al modificarla hay que incluir nuevamente la cláusula **WITH ENCRIPTYON** para mantener el cifrado.

Actualización de datos a través de una vista

Es posible modificar los datos de una tabla base subyacente mediante una vista, siempre que se cumplan las siguientes condiciones:

- Cualquier modificación, incluidas las instrucciones UPDATE, INSERT y DELETE, debe hacer referencia a las columnas de una única tabla base.
- Las columnas que se vayan a modificar en la vista deben hacer referencia directa a los datos subyacentes de las columnas de la tabla. No pueden ser derivadas de las mismas como por ejemplo funciones de agregado o algún cálculo.
- Las columnas que se van a modificar no se ven afectadas por las cláusulas GROUP BY, HAVING o DISTINCT.

- Si la vista tiene la cláusula **WITH CHECK OPTION** los datos modificados tienen que cumplir con los criterios establecidos en la sentencia SELECT, o sea no se pueden generar modificaciones que causen que el registro desaparezca de la vista.

Ejemplo:

Crea una vista sin la cláusula WITH CHECK OPTION

```
CREATE VIEW ProvinciasCanada
AS
SELECT StateProvinceID, StateProvinceCode, CountryRegionCode,
IsOnlyStateProvinceFlag, [Name], TerritoryID
FROM   Person.StateProvince
WHERE CountryRegionCode = 'CA'
```

Inserta un nuevo registro perteneciente a otro país. El mismo se agrega sin problemas

```
INSERT INTO ProvinciasCanada (StateProvinceCode, CountryRegionCode,
IsOnlyStateProvinceFlag, [Name], TerritoryID)
VALUES ('PR','DE',0,'Prueba',6)
```

Modifica la vista para incluir la cláusula **WITH CHECK OPTION**

```
ALTER VIEW ProvinciasCanada
AS
SELECT StateProvinceID, StateProvinceCode, CountryRegionCode,
IsOnlyStateProvinceFlag, [Name], TerritoryID
FROM   Person.StateProvince
WHERE CountryRegionCode = 'CA'
WITH CHECK OPTION
```

Intenta ingresar un registro perteneciente a otro país. La inserción falla por la validación de la cláusula WITH CHECK OPTION

```
INSERT INTO ProvinciasCanada (StateProvinceCode, CountryRegionCode,
      IsOnlyStateProvinceFlag, [Name], TerritoryID)
VALUES ('PP','DE',0,'Prueba 2',6)
```

Si cambia el país a ‘CA’ podrá ingresar el registro sin problemas

3- Optimización de Performance usando Vistas

Consideraciones
Cuando se usan vistas estándar, siempre existe una sobrecarga de performance ya que SQL Server debe ejecutar la sentencia SELECT contra una o más tablas para devolver los datos cada vez que se accede a la vista. Esta sobrecarga es mayor todavía si se usan vistas sobre vistas. Se debe ser muy cuidadoso al anidar vistas ya que pueden tener un impacto muy grande sobre la performance.
Para mejorar la performance se pueden usar las vistas indexadas y las vista con particiones.

Vistas Indexadas
Una vista indexada es un vista que tiene creado un índice clúster único sobre ella. Al crear el índice el conjunto de resultados de la vista se almacena en la base de datos como si se tratara de una tabla con un índice clúster.
Una ventaja de la creación de un índice en una vista es que el optimizador de consultas se inicia utilizando el índice de la vista en las consultas que no nombran directamente la vista en la cláusula FROM. Las consultas existentes pueden beneficiarse de la eficiencia mejorada de la recuperación de datos desde la vista indexada sin tener que volver a escribirse.
Según se realizan las modificaciones en los datos de las tablas base, éstas se reflejan en los datos almacenados en la vista indexada. El requisito de que el índice clúster de la vista sea único mejora la eficacia con la que SQL Server puede encontrar en el índice las filas cuyos datos se hayan modificado. Pero esto también implica una desventaja ya que genera una sobrecarga por el mantenimiento del índice.

Antes de crear una vista indexada considere:

- Que los datos involucrados no tengan actualizaciones frecuentes
- Que se mejore la performance de varias consultas de uso frecuente
- Que la mejora de performance sea considerable teniendo en cuenta la sobrecarga del mantenimiento

Los requerimientos para crear una vista indexada son:

- El primer índice debe ser un índice clúster único
- Esta vista se debe crear con la opción **SCHEMABINDING**. El enlace de esquemas asocia la vista al esquema de las tablas bases subyacentes.
- La vista no debe hacer referencia a ninguna otra vista, sólo a tablas.
- Todas las tablas base a las que hace referencia la vista deben estar en la misma base de datos que ésta y tener su mismo propietario.
- Es preciso utilizar nombres compuestos de dos partes en la vista para hacer referencia a las tablas y funciones definidas por el usuario. No se permiten nombres de una, tres o cuatro partes.
- Todas las funciones a las que se hace referencia en las expresiones de la vista deben ser deterministas.
- Las funciones definidas por el usuario a las que se hace referencia en la vista se deben crear con la opción SCHEMABINDING.
- Las opciones ANSI_NULLS y QUOTED_IDENTIFIER deben establecerse en ON cuando se ejecute la instrucción CREATE VIEW.
- La opción ANSI_NULLS debe establecerse en ON para poder ejecutar todas las instrucciones CREATE TABLE, que crean tablas a las que se hace referencia en la vista.

Ejemplo:

```
CREATE UNIQUE CLUSTERED INDEX [IX_vStateProvinceCountryRegion]
```

```
ON [Person].[vStateProvinceCountryRegion]
([StateProvinceID] ASC, [CountryRegionCode] ASC)
```

Vista con Particiones

Una vista con particiones combina los datos con particiones horizontales procedentes de un conjunto de tablas miembro residentes en uno o varios servidores de forma que parezca que la totalidad de los datos proceda de una sola tabla. Microsoft SQL Server distingue entre vistas con particiones locales y distribuidas. En una vista con particiones locales, todas las tablas que participan y la vista residen en la misma instancia de SQL Server. En una vista con particiones distribuida, al menos una de las tablas participantes reside en un servidor diferente (remoto). Además, SQL Server diferencia entre vistas con particiones que son actualizables y vistas que son copias de solo lectura de las tablas subyacentes.

Las vistas con particiones locales se incluyen en SQL Server 2008 por compatibilidad de versiones, la mejor forma de crear particiones locales en esta versión es través de las tablas con particiones.

Las vistas con particiones distribuidas se utilizan para implementar una federación de servidores de bases de datos. Una federación es un grupo de servidores que se administran independientemente, pero que colaboran para compartir la carga de procesamiento de un sistema. Permite procesamiento paralelo lo que mejora significativamente la performance.

No se pueden crear índices sobre una vista con particiones ya que estas soportan solo nombre de dos partes cuando una vista con particiones usa en general nombres de tres y cuatro partes.

Módulo 10

Procedimientos Almacenados

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Introducción

Definición
Un procedimiento almacenado (stored procedure) es una colección de sentencias Transact-SQL que se guardan dentro de la base de datos. Son métodos de encapsulación con tareas repetitivas que soportan el uso de variables, ejecuciones condicionales y otras características de programación.

Pueden:

- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida al lote o al procedimiento que realiza la llamada.
- Contener instrucciones de programación que realicen operaciones en la base de datos, incluidas las llamadas a otros procedimientos.
- Devolver un valor de estado a un lote o a un procedimiento que realiza una llamada para indicar si la operación se ha realizado correctamente o se han producido errores (y el motivo de éstos).

Los procedimientos almacenados difieren de las funciones en que no devuelven valores en lugar de sus nombres ni pueden utilizarse directamente en una expresión.

Antes de crearlos, tenga en cuenta lo siguiente:

- Se debe ser miembro de alguno de los siguientes roles: **sysadmin**, **db_owner** o **db_ddladmin** o tener permisos para CREATE PROCEDURE y ALTER SCHEMA para el esquema en donde se incluirá la vista dentro de la base de datos.
- Los procedimientos almacenados son objetos de ámbito de esquema y sus nombres deben ajustarse a las reglas para los identificadores.
- Sólo puede crear un procedimiento almacenado en la base de datos actual.
- De manera predeterminada, los parámetros admiten valores nulos. Si se pasa un valor de parámetro nulo y ese parámetro se utiliza en una instrucción CREATE TABLE o ALTER TABLE en la que la columna a la que se hace referencia no admite valores nulos, el Motor de base de datos genera un error.

Procedimientos Almacenados Temporales
Se pueden crear procedimientos almacenados temporales. De forma similar a las tablas temporales, los procedimientos almacenados temporales (tanto privados como globales) se pueden crear agregando los prefijos # y ## delante del nombre del procedimiento. # denota un procedimiento almacenado temporal local; ## denota un procedimiento almacenado temporal global. Estos procedimientos dejan de existir cuando se cierra SQL Server.

Guía para su Creación

- Especificar el esquema de los objetos que se usan en el procedimiento almacenado. Esto asegura que los objetos de otros esquemas sean accesibles desde el procedimiento. Si no se especifica el esquema el procedimiento los buscará en el esquema predeterminado.
- Diseñar cada procedimiento para cumplir con una sola tarea
- Crear y probar el procedimiento en el servidor antes de usarlo desde el cliente
- Se recomienda no crear procedimientos almacenados con el prefijo **sp_**. SQL Server utiliza el prefijo sp_ para indicar procedimientos almacenados del sistema. El nombre que elija puede entrar en conflicto con algún procedimiento futuro del sistema. Cuando se ejecuta un procedimiento cuyo nombre comienza con este prefijo SQL Server busca primero en la base de datos **master** y luego en la base de datos local.

- SQL Server guarda los valores establecidos para SET QUOTED_IDENTIFIER y SET ANSI_NULLS al momento de la creación o modificación. Usar los mismos valores para todos los procedimientos. Estos valores son usados al momento de la ejecución.
- Minimizar el uso de procedimientos almacenados temporales.

Ventajas de los Procedimientos Almacenados

- Se registran en el servidor.
- Pueden incluir atributos de seguridad (como permisos) y cadenas de propiedad; además se les pueden asociar certificados.
- Los usuarios pueden disponer de permiso para ejecutar un procedimiento almacenado sin necesidad de contar con permisos directos en los objetos a los que se hace referencia en el procedimiento.
- Mejoran la seguridad de la aplicación.
- Encapsulan lógica de aplicación, que puede ser compartida entre múltiples aplicaciones.
- Los procedimientos almacenados con parámetros pueden ayudar a proteger la aplicación ante ataques por inyección de código SQL.
- Permiten una programación modular.
- Puede crear el procedimiento una vez y llamarlo desde el programa tantas veces como desee. Así, puede mejorar el mantenimiento de la aplicación y permitir que las aplicaciones tengan acceso a la base de datos de manera uniforme.
- Constituyen código con nombre que permite el enlace diferido. Esto proporciona un nivel de direccionamiento indirecto que facilita la evolución del código.
- Pueden reducir el tráfico de red.
- Una operación que necesite centenares de líneas de código Transact-SQL puede realizarse mediante una sola instrucción que ejecute el código en un procedimiento, en vez de enviar cientos de líneas de código por la red.

Inyección de código SQL

La inyección de código SQL es un ataque en el cual se inserta código malicioso en las cadenas que posteriormente se pasan a una instancia de SQL Server para su análisis y ejecución. Todos los procedimientos que generan instrucciones SQL deben revisarse en busca de vulnerabilidades de inyección de código, ya que SQL Server ejecutará todas las consultas recibidas que sean válidas desde el punto de vista sintáctico.

La forma principal de inyección de código SQL consiste en la inserción directa de código en variables especificadas por el usuario que se concatenan con comandos SQL y se ejecutan. Existe un ataque menos directo que inyecta código dañino en cadenas que están destinadas a almacenarse en una tabla o como metadatos. Cuando las cadenas almacenadas se concatenan posteriormente en un comando SQL dinámico, se ejecuta el código dañino.

El proceso de inyección consiste en finalizar prematuramente una cadena de texto y anexar un nuevo comando.

La utilización de procedimientos almacenados con parámetros que contengan validación de los datos a procesar permite minimizar la inyección de código. No cree nunca instrucciones Transact-SQL directamente a partir de datos indicados por el usuario. No concatene nunca datos especificados por el usuario que no se hayan validado. La concatenación de cadenas es el punto de entrada principal de una inyección de scripts.

Resolución diferida de nombres y Compilación

Cuando se crea un procedimiento almacenado, se analizan sus instrucciones para asegurar la precisión sintáctica. Si se detecta un error sintáctico en la definición, se devuelve un error y no se crea el procedimiento almacenado.

Cuando se ejecuta un procedimiento almacenado por primera vez, el procesador de consultas lee el texto correspondiente en la vista de catálogo **sys.sql_modules** y comprueba si están presentes los nombres de los objetos que utiliza el procedimiento. Este proceso se denomina resolución diferida de nombres porque los objetos a los que hace referencia el procedimiento almacenado no tienen por qué existir cuando se crea éste, sino sólo cuando se ejecuta.
En la etapa de resolución, Microsoft SQL Server también realiza otras actividades de validación (por ejemplo, comprueba la compatibilidad del tipo de datos de una columna con variables).Si los objetos a los que hace referencia el procedimiento almacenado no se encuentran durante la ejecución, el procedimiento almacenado se interrumpe cuando llega a la instrucción que hace referencia al objeto que falta. En este caso, o si se detectan otros errores en la etapa de resolución, se devuelve un error.

Si la ejecución del procedimiento supera sin problemas la etapa de resolución, el optimizador de consultas de Microsoft SQL Server analiza las instrucciones Transact-SQL del procedimiento almacenado y crea un plan de ejecución. El plan de ejecución describe el método más rápido para ejecutar el procedimiento almacenado, a partir de información de este tipo:

- La cantidad de datos que hay en las tablas.
- La naturaleza y la presencia de índices en las tablas, así como la distribución de los datos en las columnas indexadas.
- Los operadores y los valores de comparación utilizados en las condiciones de las cláusulas WHERE.
- La presencia de combinaciones y de las palabras clave UNION, GROUP BY y ORDER BY.

Después de haber analizado estos factores en el procedimiento almacenado, el optimizador de consultas coloca el plan de ejecución en la memoria. El proceso consistente en analizar el procedimiento almacenado y crear un plan de consultas se denomina compilación. El plan de ejecución en memoria optimizado se utiliza para ejecutar la consulta. Este plan permanece en la memoria hasta que se reinicie SQL Server o hasta que se necesite espacio para el almacenamiento de otro objeto.

Cuando se ejecute posteriormente el procedimiento almacenado, SQL Server vuelve a utilizar el plan de ejecución existente si sigue en la memoria. Si el plan de ejecución ya no está en la memoria, se crea uno nuevo.

Volver a compilar Procedimientos Almacenados
Cuando se cambia una base de datos mediante la adición de índices o la modificación de datos de columnas indexadas, los planes de consultas originales utilizados para el acceso a las tablas deberán optimizarse de nuevo; para ello, será preciso volver a compilarlos. Esta optimización se produce automáticamente la primera vez que se ejecuta un procedimiento almacenado tras reiniciar Microsoft SQL Server. También se produce si cambia una tabla subyacente utilizada por el procedimiento almacenado. Pero si se agrega un nuevo índice del que se puede beneficiar el procedimiento almacenado, la optimización no se produce automáticamente hasta que se vuelva a ejecutar el procedimiento almacenado tras reiniciar Microsoft SQL Server. En esta situación, puede ser útil forzar que se vuelva a compilar el procedimiento almacenado la próxima vez que se ejecute.
Otro motivo para forzar que se vuelva a compilar un procedimiento almacenado es contrarrestar, cuando sea necesario, el comportamiento de "examen de parámetros" de la compilación de procedimientos almacenados. Cuando SQL Server ejecuta procedimientos almacenados, los valores de parámetros utilizados por el procedimiento cuando se compila se incluyen como parte de la generación del plan de consultas. Si esos valores representan los valores típicos con los que el procedimiento se invoca posteriormente, el procedimiento

almacenado se beneficia del plan de consultas cada vez que se compila y se ejecuta. En caso contrario, el rendimiento puede verse afectado.

SQL Server proporciona tres formas de forzar que un procedimiento almacenado se vuelva a compilar:

- El procedimiento almacenado del sistema **sp_recompile** fuerza que se vuelva a compilar un procedimiento almacenado la próxima vez que se ejecute.
- La creación de un procedimiento almacenado que especifique la opción **WITH RECOMPILE** en su definición indica a SQL Server que no guarde en memoria caché ningún plan para ese procedimiento, que vuelve a compilarse cada vez que se ejecuta. Utilice la opción WITH RECOMPILE cuando los procedimientos almacenados adopten parámetros cuyos valores varíen considerablemente entre distintas ejecuciones del procedimiento almacenado, de modo que se creen distintos planes de ejecución cada vez. Esta opción no se utiliza con mucha frecuencia y provoca que el procedimiento almacenado se ejecute más lentamente, ya que debe compilarse de nuevo cada vez que se ejecuta.
- Puede forzar que se vuelva a compilar el procedimiento almacenado si especifica la opción **WITH RECOMPILE** al ejecutarlo. Utilice esta opción únicamente si el parámetro que está suministrando es atípico o si los datos han cambiado considerablemente desde que se creó el procedimiento almacenado.

Anidar Procedimientos Almacenados

Los procedimientos almacenados se anidan cuando un procedimiento almacenado llama a otro. Puede anidar hasta 32 niveles de procedimientos almacenados. El nivel de anidamiento aumenta en uno cuando el procedimiento almacenado a los que se ha llamado empieza a ejecutarse, y disminuye en uno cuando finaliza su ejecución. Si se intenta superar el límite máximo de 32 niveles de anidamiento, se producirá un error general en la cadena de llamada. El nivel actual de anidamiento de los procedimientos almacenados en ejecución se almacena en la función **@@NESTLEVEL**.

2- Creación y Ejecución

Creación
Puede crear procedimientos almacenados usando la instrucción **CREATE PROCEDURE** de Transact-SQL.

Cuando cree un procedimiento almacenado, deberá especificar lo siguiente:

- Todos los parámetros de entrada y de salida del lote o del procedimiento que realiza la llamada.
- Las instrucciones de programación que realicen operaciones en la base de datos, incluidas las llamadas a otros procedimientos.
- El valor de estado devuelto al lote o al procedimiento que realiza la llamada, a fin de indicar que la operación se ha realizado correctamente o que se ha producido un error (y el motivo del mismo).
- Las instrucciones de control de errores necesarias para detectar y administrar posibles errores.

```
CREATE { PROC | PROCEDURE } [Nombre_esquema.] Nombre_Procedimiento
[ { @parametro [ nombre_esquema. ] tipo_dato }
[ VARYING ] [ = default ] [ [ OUT [ PUT ] ]
[ ,...n ]
[ WITH <opciones> [ ,...n ]
AS sentencias_sql [;][ ...n ]
<opciones> ::=
[ ENCRYPTION ]
[ RECOMPILE ]
[ EXECUTE_AS_Clause ]
```

Argumentos

- **nombre_esquema**: Es el nombre del esquema al que pertenece el procedimiento.
- **Nombre_Procedimiento**: Es el nombre del procedimiento. Los nombres de los procedimientos almacenados deben cumplir las reglas de los identificadores. La especificación del nombre del propietario opcional.
- **@parametros**: Especifique un nombre de parámetro con un signo (@) como primer carácter. El nombre del parámetro se debe ajustar a las reglas de los identificadores. Los parámetros son locales para el procedimiento; los mismos nombres de parámetro se pueden utilizar en otros procedimientos. De manera predeterminada, los parámetros sólo pueden ocupar el lugar de expresiones constantes; no se pueden utilizar en lugar de nombres de tabla, nombres de columna o nombres de otros objetos de base de datos.
- **[type_schema_name.]data_type*:*** Es el tipo de datos del parámetro y el esquema al que pertenece. Se pueden utilizar todos los tipos de datos como un parámetro de un procedimiento almacenado Transact-SQL.
- **VARYING*:*** Especifica el conjunto de resultados admitido como un parámetro de salida. Este parámetro es creado de forma dinámica por el procedimiento almacenado y su contenido puede variar. Sólo se aplica a los parámetros de tipo cursor.
- **OUTPUT**: Indica que se trata de un parámetro de salida. Utilice los parámetros OUTPUT para devolver valores al autor de la llamada del procedimiento. Los parámetros text, ntext e image no se pueden utilizar como parámetros OUTPUT.
- **Sentencias_sql*:*** Una o más instrucciones Transact-SQL que se van a incluir en el procedimiento.

- **ENCRYPTION**: Cifra las entradas de **sys.syscomments** que contienen el texto de la instrucción CREATE PROCEDURE.
- **RECOMPILE**: Indica que el Motor de base de datos no almacena en caché un plan para este procedimiento y que éste se compila en tiempo de ejecución. Esta opción no se puede utilizar cuando se especifica FOR REPLICATION.
- **EXECUTE AS**: Especifica el contexto de seguridad en el que se ejecuta el procedimiento almacenado.

Ejemplo:

```
CREATE PROC Production.LongLeadProducts
AS
SELECT Name, ProductNumber
FROM Production.Product
WHERE DaysToManufacture >= 1
```

Ejecución
Para ejecutar un procedimiento almacenado, utilice la instrucción **EXECUTE** o **EXEC** de Transact-SQL. También puede ejecutar un procedimiento almacenado sin necesidad de utilizar la palabra clave EXECUTE si el procedimiento almacenado es la primera instrucción del lote.

Ejemplo:

```
EXEC Production.LongLeadProducts
```

Modificación y Borrado
Para cambiar las instrucciones o los parámetros de un procedimiento almacenado, puede eliminar y volver a crear el procedimiento, o bien modificar el procedimiento almacenado en un solo paso. Cuando elimina un procedimiento almacenado y lo vuelve a crear, se pierden todos los permisos que están asociados con él. Cuando modifica el procedimiento almacenado, se cambia la definición de los parámetros o del procedimiento pero se conservan los permisos definidos para el procedimiento almacenado, y los procedimientos almacenados o desencadenadores dependientes no se ven afectados.
También es posible modificar un procedimiento almacenado para cifrar la definición o provocar que se vuelva a compilar el procedimiento cada vez que se ejecute.
Para modificar o borrar un procedimiento almacenado use las instrucciones **ALTER / DROP PROCEDURE**. Antes de modificar o borrar un procedimiento verifique sus dependencias de manera de controlar como afectará esto al resto de los objetos.

Obtener información sobre procedimientos almacenados
- SQL Server Management Studio
- Vistas de catálogo:
  - **Sys_procedures** devuelve la lista de procedimientos de la base de datos
  - **Sys_parameters** devuelve la definición de los parámetros de los procedimientos
  - **sys.sql_modules** devuelve la lista de vistas, procedimientos, funciones y desencadenadores de la base de datos
  - **Sys_Sql_Dependencies** permite verificar las dependencias del objeto.
- Procedimiento almacenado:
  - **sp_HelpText** permite ver la definición de una vista
  - **sp_Help**
  - **sp_depends** permite verificar las dependencias del objeto.

3- Procedimientos Almacenados con Parámetros

Un procedimiento almacenado se comunica con el programa que lo llama mediante sus parámetros. Cuando un programa ejecuta un procedimiento almacenado, es posible pasarle valores mediante los parámetros del procedimiento. Estos valores se pueden utilizar como variables estándar en el lenguaje de programación Transact-SQL. El procedimiento almacenado también puede devolver valores al programa que lo llama mediante parámetros OUTPUT. Un procedimiento almacenado puede tener hasta 2.100 parámetros, cada uno de ellos con un nombre, un tipo de datos, una dirección y un valor predeterminado.

Manejo de Parámetros
Cada parámetro de un procedimiento almacenado debe definirse con un nombre único. Los nombres de los procedimientos almacenados deben empezar por un solo carácter @, como una variable estándar de Transact-SQL, y deben seguir las reglas definidas para los identificadores de objetos. El nombre del parámetro se puede utilizar en el procedimiento almacenado para obtener y cambiar el valor del parámetro.
Es posible pasar valores a los procedimientos almacenados de dos maneras: asignando un nombre explícitamente a los parámetros y asignando el valor apropiado, o bien suministrando los valores del parámetro especificados en la instrucción CREATE PROCEDURE sin asignarles un nombre.

Por ejemplo, si el procedimiento almacenado miProcedimiento espera tres parámetros llamados @Primero, @Segundo y @Tercero, los valores pasados al procedimiento almacenado pueden asignarse a los nombres de los parámetros; por ejemplo:

```
EXECUTE miProcedimiento @Segundo = 2, @Primero = 1, @Tercero = 3
```

O bien por posición, sin asignarles ningún nombre:

```
EXECUTE miProcedimiento 1, 2, 3
```

Si se asigna un nombre a los parámetros cuando se ejecuta el procedimiento almacenado, es posible especificarlos en cualquier orden. Si no se asigna ningún nombre a los parámetros, deben especificarse en el mismo orden (de izquierda a derecha) en el que están definidos en el procedimiento almacenado. Además, todos los parámetros que precedan a un parámetro determinado deberán especificarse aunque sean opcionales y tengan valores predeterminados. Por ejemplo, si todos los parámetros de miProcedimiento son opcionales, se podría ejecutar miProcedimiento mediante la especificación de los valores correspondientes a los dos primeros parámetros, pero no a los del segundo y tercer parámetros solamente. Esto es necesario, ya que en caso contrario, Microsoft SQL Server no puede identificar los parámetros que se especifican.

Dirección de los parámetros
La dirección de un parámetro puede ser de entrada, que indica que un valor se pasa al parámetro de entrada de un procedimiento almacenado, o de salida, que indica que el procedimiento almacenado devuelve un valor al programa que lo llama mediante un parámetro de salida. El valor predeterminado es un parámetro de entrada. Un parámetro de salida se especifica con la cláusula **OUTPUT**.

Valores Predeterminados
Puede crear un procedimiento almacenado con parámetros opcionales especificando un valor predeterminado para los mismos. Al ejecutar el procedimiento almacenado, se utilizará el valor

predeterminado si no se ha especificado ningún otro. Es necesario especificar valores predeterminados, ya que el sistema devuelve un error si en el procedimiento almacenado no se especifica un valor predeterminado para un parámetro y el programa que realiza la llamada no proporciona ningún otro valor al ejecutar el procedimiento.
Si no se puede especificar ningún valor predeterminado para el parámetro, siempre se puede especificar NULL y dejar que el procedimiento almacenado devuelva un mensaje personalizado en caso de ejecutarse sin que el parámetro tenga un valor.

Parámetros de Entrada

Ejemplo:

```
ALTER PROCEDURE Production.LongLeadProducts @MinimumLength int = 1
AS
IF (@MinimumLength < 0)
      BEGIN
      RAISERROR('Dato inválido', 14, 1)
      RETURN
      END
SELECT Name, ProductNumber, DaysToManufacture
FROM Production.Product
WHERE DaysToManufacture >= @MinimumLength
ORDER BY DaysToManufacture DESC, Name
```

Ejecución:

Por nombre:

```
EXEC Production.LongLeadProducts @MinimumLength=4
```

Por posición:

```
EXEC Production.LongLeadProducts 4
```

Como el parámetro tiene un valor predeterminado asociado se puede llamar al procedimiento sin argumentos, el mismo asumirá el valor 1

```
EXEC Production.LongLeadProducts
```

Parámetros de Salida
Para especificar un parámetro de salida, debe indicar la palabra clave **OUTPUT** en la definición del parámetro del procedimiento almacenado. El procedimiento almacenado devuelve el valor actual del parámetro de salida al programa que lo llama cuando se abandona el procedimiento almacenado. El programa que realiza la llamada también debe utilizar la palabra clave **OUTPUT** al ejecutar el procedimiento almacenado, a fin de guardar el valor del parámetro en una variable que se pueda utilizar en el programa que llama.

Ejemplo:

```
CREATE PROC HumanResources.AddDepartment
            @Name nvarchar(50), @GroupName nvarchar(50),
            @DeptID smallint OUTPUT
AS
```

```
INSERT INTO HumanResources.Department (Name, GroupName)
VALUES (@Name, @GroupName)
SET @DeptID = SCOPE_IDENTITY()
```

Ejecución:

```
DECLARE @dept int
EXEC AddDepartment 'Refunds', '', @dept OUTPUT
SELECT @dept
```

Parámetros de Retorno

Un procedimiento almacenado puede devolver un valor **entero**, denominado código de retorno, para indicar el estado de ejecución de un procedimiento. Se especifica el código de retorno para un procedimiento almacenado mediante la instrucción **RETURN**. Al igual que con los parámetros **OUTPUT**, debe guardar el código de retorno en una variable cuando se ejecute el procedimiento almacenado a fin de utilizar su valor en el programa que realiza la llamada.

Los códigos de retorno suelen utilizarse en los bloques de control de flujo dentro de los procedimientos almacenados con el fin de establecer el valor del código de retorno para cada posible situación de error.

Los procedimientos almacenados devuelven un cero si no se especifica la cláusula **RETURN**.

Ejemplo:

```
ALTER PROC HumanResources.AddDepartment
            @Name nvarchar(50), @GroupName nvarchar(50),
            @DeptID smallint OUTPUT
AS
IF ((@Name = '') OR (@GroupName = ''))
      RETURN -1
INSERT INTO HumanResources.Department (Name, GroupName)
VALUES (@Name, @GroupName)
SET @DeptID = SCOPE_IDENTITY()
RETURN 0
```

Ejecución:

```
DECLARE @dept int, @result int
EXEC @result = AddDepartment 'Refunds', '', @dept OUTPUT
IF (@result = 0)
      SELECT @dept
ELSE
      SELECT 'Error'
```

4- Manejo de Errores

Manejo Estructurado de Excepciones
El manejo estructurado de excepciones de SQL Server 2008 es un requerimiento importante para varias sentencias Transact-SQL particularmente las que involucran transacciones. Reduce el trabajo requerido para controlar los errores y genera un código más confiable.

TRY...CATCH en Transact-SQL
Los errores en el código de Transact-SQL se pueden procesar mediante una construcción TRY…CATCH similar a las características de control de excepciones de los lenguajes Microsoft Visual C++ y Microsoft Visual C#. Las construcciones TRY…CATCH constan de dos partes: un bloque TRY y uno CATCH. Cuando se detecta una condición de error en una instrucción de Transact-SQL que se encuentra dentro de un bloque TRY, se pasa el control a un bloque CATCH donde se puede procesar el error.
Cuando el bloque CATCH controla la excepción, el control se transfiere a la primera instrucción de Transact-SQL siguiente a la instrucción END CATCH. Si la instrucción CATCH es la última instrucción de un procedimiento almacenado o un desencadenador, el control vuelve al código que los invocó. No se ejecutarán las instrucciones de Transact-SQL del bloque TRY que siguen a la instrucción que genera un error.
Si no existen errores en el bloque TRY, el control pasa a la instrucción inmediatamente después de la instrucción END CATCH asociada. Si la instrucción END CATCH es la última instrucción de un procedimiento almacenado o un desencadenador, el control pasa a la instrucción que invocó al procedimiento almacenado o el desencadenador.
Un bloque TRY se inicia con la instrucción BEGIN TRY y finaliza con la instrucción END TRY. Se pueden especificar una o varias instrucciones de Transact-SQL entre las instrucciones BEGIN TRY y END TRY. Un bloque TRY debe ir seguido inmediatamente por un bloque CATCH. Un bloque CATCH se inicia con la instrucción BEGIN CATCH y finaliza con la instrucción END CATCH. En Transact-SQL, cada bloque TRY se asocia a un sólo bloque CATCH.

Sintaxis:

```
BEGIN TRY
        {sentencias SQL | bloque sentencias}
END TRY
BEGIN CATCH
        {sentencias SQL | bloque sentencias}
END CATCH
```

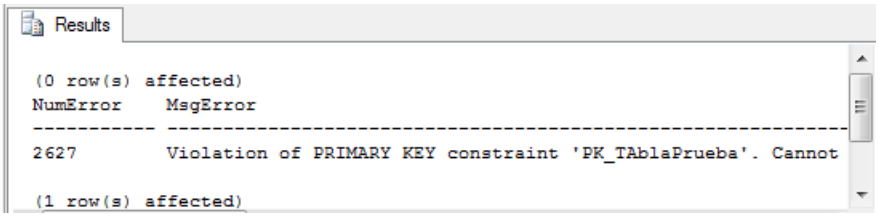
Ejemplo:

```
CREATE PROCEDURE dbo.AgregarDatos @a int, @b varchar(50)
AS
BEGIN TRY
       INSERT INTO TablaPrueba VALUES (@a, @b)
END TRY
BEGIN CATCH
      SELECT ERROR_NUMBER() as NumError, ERROR_MESSAGE()  as MsgError
END CATCH
```

Ejecución:

```
EXEC dbo.AgregarDatos 1, 'Primero'
EXEC dbo.AgregarDatos 2, 'Segundo'
EXEC dbo.AgregarDatos 1, 'Tercero'
```

Este último trata de insertar nuevamente la clave 1 la cual ya fue ingresada. La respuesta a la ejecución es:



Guía para el Control de Errores

- Cada construcción TRY…CATCH debe encontrarse en un solo lote, procedimiento almacenado o desencadenador.
- Un bloque CATCH debe seguir inmediatamente a un bloque TRY.
- Las construcciones TRY…CATCH pueden estar anidadas. Esto significa que las construcciones TRY…CATCH se pueden colocar dentro de otros bloques TRY y CATCH. Cuando se produce un error dentro de un bloque TRY anidado, el control del programa se transfiere al bloque CATCH que está asociado al bloque TRY anidado.
- Para controlar un error que se produce en un bloque CATCH determinado, escriba un bloque TRY…...CATCH en el bloque CATCH especificado.
- El bloque TRY…CATCH no controlará los errores con una gravedad de 20 o superior que hacen que el Motor de base de datos cierre la conexión. No obstante, TRY…CATCH controlará los errores con una gravedad de 20 o superior siempre que la conexión no se cierre.
- Los errores con un gravedad de 10 o inferior se consideran advertencias o mensajes informativos, y los bloques TRY…CATCH no los controlan.

Funciones de error

TRY…CATCH utiliza las siguientes funciones de error para capturar información de errores:

- **ERROR_NUMBER**() devuelve el número de error.
- **ERROR_MESSAGE**() devuelve el texto completo del mensaje de error. El texto incluye los valores suministrados para los parámetros sustituibles, como longitudes, nombres de objeto u horas.
- **ERROR_SEVERITY**() devuelve la gravedad del error.
- **ERROR_STATE**() devuelve el número de estado del error.
- **ERROR_LINE**() devuelve el número de línea dentro de la rutina en que se produjo el error.
- **ERROR_PROCEDURE**() devuelve el nombre del procedimiento almacenado o desencadenador en que se produjo el error.

La información de errores se recupera mediante estas funciones desde cualquier sitio en el ámbito del bloque CATCH de una construcción TRY…CATCH. Las funciones de error devolverán un valor NULL si se llaman desde fuera del ámbito de un bloque CATCH. Se puede hacer referencia a funciones de error contenidas en un procedimiento almacenado y pueden utilizarse para recuperar información de errores cuando el procedimiento almacenado se ejecuta en el bloque CATCH. Al hacerlo, no será necesario repetir el código de control de errores en cada bloque CATCH.

Módulo 11

Funciones de Usuario

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Introducción

Definición
Al igual que las funciones en los lenguajes de programación, las funciones definidas por el usuario de Microsoft SQL Server son rutinas que aceptan parámetros, realizan una acción, como un cálculo complejo, y devuelven el resultado de esa acción como un valor. El valor devuelto puede ser un valor escalar único o un conjunto de resultados.

Ventajas de las Funciones

- Permiten una programación modular: Puede crear la función una vez, almacenarla en la base de datos y llamarla desde el programa tantas veces como desee. Las funciones definidas por el usuario se pueden modificar, independientemente del código de origen del programa.
- Permiten una ejecución más rápida: Al igual que los procedimientos almacenados, las funciones definidas por el usuario Transact-SQL reducen el costo de compilación del código Transact-SQL almacenando los planes en la caché y reutilizándolos para ejecuciones repetidas. Esto significa que no es necesario volver a analizar y optimizar la función definida por el usuario con cada uso, lo que permite obtener tiempos de ejecución mucho más rápidos.
- Pueden reducir el tráfico de red: Una operación que filtra datos basándose en restricciones complejas que no se puede expresar en una sola expresión escalar se puede expresar como una función. La función se puede invocar en la cláusula WHERE para reducir el número de filas que se envían al cliente.

Tipos de Funciones

- **Escalares (scalar functions)**: Devuelven un único valor de datos del tipo definido en la cláusula RETURNS.
- **Con Valores de Tabla en Línea (inline table-valued functions)**: Devuelven un tipo de datos **Table**. No tienen cuerpo; la tabla es el conjunto de resultados de una sola instrucción SELECT.
- **Con Valores de Tabla con múltiples instrucciones (multi-statement tabled-value functions)**: Devuelven un tipo de datos **Table**. El cuerpo de la función, definido en un bloque BEGIN...END, contiene una serie de instrucciones Transact-SQL que generan e insertan filas en la tabla que se va a devolver.

Instrucciones válidas en una Función

- Las instrucciones DECLARE pueden utilizarse para definir variables y cursores de datos locales de la función.
- La asignación de valores a objetos locales de la función, como la utilización de SET para asignar valores a variables locales escalares y de tabla.
- Las operaciones de cursores que hacen referencia a cursores locales que están declarados, abiertos, cerrados y no asignados en la función. No se admiten las instrucciones FETCH que devuelven datos al cliente. Sólo se permiten las instrucciones FETCH que asignan valores a variables locales mediante la cláusula INTO.
- Instrucciones de control de flujo excepto instrucciones TRY...CATCH.
- Instrucciones SELECT que contienen listas de selección con expresiones que asignan valores a las variables locales para la función.
- Instrucciones UPDATE, INSERT y DELETE que modifican las variables de tabla locales de la función.
- Instrucciones EXECUTE que llaman a un procedimiento almacenado extendido.

Parámetros
Una función definida por el usuario tiene de cero a varios parámetros de entrada y devuelve un valor escalar o una tabla. Una función puede tener un máximo de 1024 parámetros de entrada. Cuando un parámetro de la función tiene un valor predeterminado, debe especificarse la palabra clave DEFAULT al llamar a la función para poder obtener el valor predeterminado. Este comportamiento es diferente del de los parámetros con valores predeterminados de procedimientos almacenados definidos por el usuario, para los cuales omitir el parámetro implica especificar el valor predeterminado. Las funciones definidas por el usuario no admiten parámetros de salida.

Volver a escribir procedimientos almacenados como funciones
Si desea invocar un procedimiento almacenado directamente desde una consulta, reorganice el código como si fuera una función definida por el usuario. En general, si el procedimiento almacenado devuelve un solo conjunto de resultados, defina una función con valores de tabla. Si el procedimiento almacenado calcula un valor escalar, defina una función escalar.

2- Creación

Creación
Las funciones se crean con la instrucción **CREATE FUNCTION**, se modifican con **ALTER FUNCTION** y se borran con **DROP FUNCTION**. La instrucción CREATE FUNCTION es diferente para cada tipo de función.
CREATE FUNCTION admite una cláusula SCHEMABINDING que enlaza la función con el esquema de cualquier objeto al que haga referencia, como tablas, vistas y otras funciones definidas por el usuario.

Funciones Escalar (Scalar Function)
Las funciones escalares definidas por el usuario devuelven un único valor de datos del tipo definido en la cláusula **RETURNS**. Las funciones escalares insertadas no tienen cuerpo; el valor escalar es el resultado de una sola instrucción. Para una función escalar de múltiples instrucciones, el cuerpo de la función, definido en un bloque BEGIN...END, contiene una serie de instrucciones Transact-SQL que devuelven el valor único. El tipo devuelto puede ser de cualquier tipo de datos excepto text, ntext, image, cursor y timestamp.

Ejemplo:

```
CREATE FUNCTION Sales.SumSold(@ProductID int)
RETURNS int
AS
BEGIN
      DECLARE @ret int
      SELECT @ret = SUM(OrderQty)
      FROM Sales.SalesOrderDetail WHERE ProductID = @ProductID
      IF (@ret IS NULL)
            SET @ret = 0
      RETURN @ret
END
```

Las funciones con valores escalares deben invocarse como mínimo con el nombre de dos partes de la función. En estas ubicaciones se permiten funciones definidas por el usuario que devuelven valores escalares:

- Consultas
  - Como expresión de la lista de una instrucción SELECT:
  - Como expresión de cláusula WHERE o HAVING
  - Como expresión de las cláusulas GROUP BY u ORDER BY
  - Como expresión de la cláusula SET en una instrucción UPDATE
  - Como expresión en el VALUES de una instrucción INSERT
- Definición de Tablas
  - En una restricción CHECK
  - En una restricción DEFAULT
  - En columnas calculadas
- Sentencias SQL
  - En una condición control de flujo
  - En una expresiones CASE
  - En una sentencia PRINT
- Funciones y procedimientos almacenados

Ejecución:

```
SELECT ProductID, Name, Sales.SumSold(ProductID) AS SumSold
FROM Production.Product
```

Funciones Deterministas y no Deterministas
Las funciones deterministas devuelven siempre el mismo resultado cada vez que se invocan con un conjunto específico de valores de entrada y cuando el estado de la base de datos es el mismo. Las funciones no deterministas pueden devolver resultados diferentes cada vez que se llaman con un conjunto específico de valores de entrada aunque el estado de la base de datos a la que tienen acceso permanezca sin cambios.
Funciones Deterministas: Day, Month, Year, DateDiff, IsNull, etc
Funciones No Deterministas: getdate(), Rand, Convert, etc.

Funciones con Valores de Tabla en Línea (inline table-valued functions)
Las funciones con valores de tabla definidas por el usuario devuelven un tipo de datos **table**. Las funciones con valores de tabla en línea no tienen cuerpo; la tabla es el conjunto de resultados de una sola instrucción **SELECT**. Pueden utilizarse para obtener la funcionalidad de vistas con parámetros, ya que las vista no manejan parámetros.

Las funciones en línea definidas por el usuario siguen las reglas siguientes:

- La cláusula RETURNS sólo contiene la palabra clave **table**. No es necesario definir el formato de una variable de retorno, ya que se define mediante el formato del conjunto de resultados de la instrucción SELECT en la cláusula RETURN.
- El cuerpo de la función no está delimitada por BEGIN ni END.
- La cláusula RETURN contiene una sola instrucción SELECT entre paréntesis. El conjunto de resultados de la instrucción SELECT forma la tabla devuelta por la función. La instrucción SELECT utilizada en una función en línea está sujeta a las mismas restricciones que las instrucciones SELECT utilizadas en las vistas.
- La función con valores de tabla sólo acepta constantes o argumentos @local_variable.

Ejemplo:

```
CREATE FUNCTION Production.ProductsByColor (@Color varchar(15))
RETURNS TABLE
AS
RETURN (SELECT Productid, Name, ProductNumber, Color
FROM Production.Product
WHERE Color = @Color )
```

Ejecución:

```
SELECT * FROM Production.ProductsByColor('blue')
```

Funciones con Valores de Tabla con Múltiples Instrucciones (multi-statement tabled-value functions)
Las funciones con valores de tabla definidas por el usuario devuelven un tipo de datos **table**. Pueden ser unas eficaces alternativas a las vistas. Una función definida por el usuario con valores de tabla se puede usar donde se permiten las expresiones de vista o de tabla en las consultas Transact-SQL. Mientras que las vistas se limitan a una única instrucción SELECT, las funciones definidas por el usuario pueden contener instrucciones adicionales que permiten una lógica más eficaz que en las vistas.

También puede reemplazar **procedimientos almacenados** que devuelven un solo conjunto de resultados. En la cláusula FROM de una instrucción Transact-SQL es posible hacer referencia a la tabla que devuelve una función definida por el usuario, pero esto no es posible con los procedimientos almacenados que devuelven conjuntos de resultados.
Ninguna instrucción Transact-SQL de una función con valores de tabla puede devolver un conjunto de resultados directamente a un usuario. La única información que la función puede devolver al usuario es el tipo de datos **table** devuelto por la función.

En una función definida por el usuario con valores de tabla:

- La cláusula **RETURNS** define el nombre de una variable de retorno local para la tabla devuelta por la función. La cláusula RETURNS también define el formato de la tabla. El nombre de una variable de retorno local tiene un ámbito local dentro de la función.
- Las instrucciones Transact-SQL del cuerpo de la función generan e insertan filas en la variable de retorno definida por la cláusula RETURNS.
- Al ejecutar una instrucción **RETURN**, las filas insertadas en la variable se devuelven desde la función en formato tabular. La instrucción RETURN no puede tener un argumento.

Ejemplo:

```
CREATE FUNCTION HumanResources.EmployeeNames
(@format nvarchar(9))
RETURNS @tbl_Employees TABLE
(EmployeeID int PRIMARY KEY, [Employee Name] nvarchar(100))
AS
BEGIN
      IF (@format = 'SHORTNAME')
            INSERT @tbl_Employees
            SELECT BusinessEntityID, LastName
            FROM HumanResources.vEmployee
      ELSE IF (@format = 'LONGNAME')
            INSERT @tbl_Employees
            SELECT BusinessEntityID, (FirstName + ' ' + LastName)
            FROM HumanResources.vEmployee
      RETURN
END
```

Ejecución:

```
SELECT * FROM HumanResources.EmployeeNames('LONGNAME')
SELECT * FROM HumanResources.EmployeeNames('SHORTNAME')
```

Obtener información sobre funciones

- SQL Server Management Studio
- Vistas de catálogo:
  - **Sys_sql_modules** devuelve la lista de procedimientos, funciones, vistas y desencadenadores de la base de datos
  - **Sys_parameters** devuelve la definición de los parámetros de las funciones
  - **Sys_Sql_Dependencies** permite verificar las dependencias del objeto.
- Procedimiento almacenado:
  - **sp_HelpText** permite ver la definición de una función
  - **sp_Help**
  - **sp_depends** permite verificar las dependencias del objeto.

3- Contexto de Ejecución

Definición
El contexto de ejecución está determinado por el usuario o el inicio de sesión que está conectado a la sesión o que está ejecutando o llamando a un módulo. Representa la identidad para la que se comprueban los permisos para ejecutar instrucciones o realizar acciones, identidad que puede causar problemas si rompe la cadena de propiedad.
En SQL Server 2008, el contexto de ejecución de una sesión puede cambiarse explícitamente mediante la especificación de un nombre de usuario o inicio de sesión en una instrucción **EXECUTE AS**. El contexto de ejecución de un módulo como, por ejemplo, un procedimiento almacenado, un desencadenador o una función definida por el usuario, puede cambiarse implícitamente mediante la especificación de un nombre de usuario o inicio de sesión en una cláusula **EXECUTE AS** en la definición del módulo. Cuando el contexto de ejecución cambia a otro usuario o inicio de sesión, SQL Server comprueba los permisos para ese otro usuario o inicio de sesión. Básicamente, esa cuenta se suplanta durante el transcurso de la sesión o ejecución del módulo.

EXECUTE AS
Establece el contexto de ejecución de una sesión.
De forma predeterminada, una sesión empieza cuando un usuario inicia la sesión y termina cuando el usuario la cierra. Todas las operaciones durante una sesión están sujetas a comprobaciones de permisos de ese usuario. Cuando se ejecuta una instrucción EXECUTE AS, el contexto de ejecución de la sesión se cambia al nombre de usuario o inicio de sesión especificado. Después de cambiar el contexto, los permisos se comprueban con los testigos de seguridad del usuario y el inicio de sesión de esa cuenta en vez de los de la persona que llama a la instrucción EXECUTE AS. Básicamente, la cuenta de inicio de sesión o usuario se suplanta durante la ejecución del módulo o la sesión, o el cambio de contexto se revierte de forma explícita.

```
{ EXEC | EXECUTE ] AS { LOGIN | USER } = 'nombre'
[ WITH { NO REVERT | COOKIE INTO @variable_varbinary } ] | CALLER}
```

Argumentos:

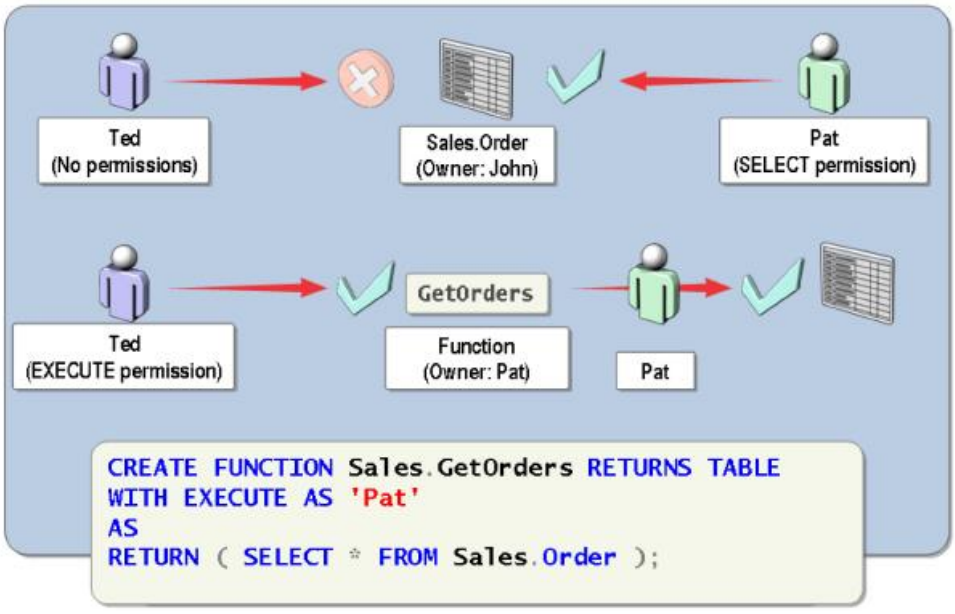
- **LOGIN**: Especifica que el contexto de ejecución que se va a suplantar es un inicio de sesión. El ámbito de la suplantación se produce en el nivel de servidor.
- **USER**: Especifica que el contexto de ejecución que se va a suplantar es un usuario de la base de datos actual. El ámbito de la suplantación se restringe a la base de datos actual. Un cambio de contexto a un usuario de base de datos no hereda los permisos en el nivel de servidor de ese usuario.
- **' nombre'**: Es un inicio de sesión o usuario válido. Debe ser miembro de la función fija de servidor **sysadmin** o existir como entidad de seguridad. Se puede especificar como una variable local.
- **NO REVERT**: Especifica que el cambio de contexto no se puede volver al contexto anterior.
- **COOKIE INTO** @variable_varbinary: Especifica que el contexto de ejecución sólo se puede volver al contexto anterior si la llamada a la instrucción REVERT WITH COOKIE contiene el valor de @varbinary_variable correcto.
- **CALLER**: Cuando se utiliza dentro de un módulo, especifica que las instrucciones dentro del módulo se ejecutan en el contexto del llamador del módulo. Cuando se utiliza fuera de un módulo, la instrucción no tiene ninguna acción.

EXECUTE AS (Cláusula)
En SQL Server 2008 puede definir el contexto de ejecución de los siguientes módulos definidos por el usuario: funciones (excepto funciones con valores de tabla en línea), procedimientos y desencadenadores.
Al especificar el contexto en el que se ejecuta el módulo, puede controlar qué cuenta de usuario usa el Motor de base de datos de SQL Server 2008 para validar permisos en objetos a los que el módulo hace referencia. De esta forma se dispone de mayor control y flexibilidad para administrar los permisos a lo largo de la cadena de objetos que existe entre los módulos definidos por el usuario y los objetos a los que esos módulos hacen referencia. Los permisos deben concederse a usuarios únicamente del propio módulo, sin tener que concederles permisos explícitos en los objetos referenciados. Sólo el usuario que ejecuta el módulo debe tener permisos en los objetos a los que tiene acceso el módulo.

EXECUTE AS { CALLER | SELF | OWNER | nombre_usuario }

Argumentos

- **CALLER**: Especifica que las instrucciones dentro del módulo se ejecutan en el contexto del llamador del módulo. El usuario que ejecuta el módulo debe tener los permisos adecuados no sólo en el propio módulo, sino también en los objetos de la base de datos a los que el módulo hace referencia. Es el valor predeterminado.
- **SELF**: Es equivalente a EXECUTE AS usuario, donde el usuario especificado es la persona que crea o modifica el módulo.
- **OWNER**: Especifica que las instrucciones dentro del módulo se ejecutan en el contexto del propietario actual del módulo. Si el módulo no tiene un propietario especificado, se usa el propietario del esquema del módulo. OWNER no se puede especificar para desencadenadores.
- **Nombre_usuario**: Especifica que las instrucciones dentro del módulo se ejecutan en el contexto del usuario especificado en nombre_usuario. Los permisos para los objetos dentro del módulo se comprueban para ese usuario.



Módulo 12

Transacciones y Bloqueos

1- Transacciones

Definición

Las transacciones son grupos de operaciones que se combinan en unidades lógicas de trabajo. Se usan para controlar y mantener la coherencia y la integridad de cada acción de una transacción, a pesar de los errores que puedan producirse en el sistema.

Las transacciones enlazan varias tareas entre sí. Por ejemplo, si una aplicación realiza dos tareas, crea en primer lugar una tabla nueva en una base de datos y, después, llama a un objeto especializado para recopilar, dar formato e insertar datos en la tabla nueva. Estas dos tareas están relacionadas entre sí e incluso son interdependientes, de modo que se ha de evitar la creación de una tabla nueva salvo que se pueda rellenar con datos. La ejecución de ambas tareas dentro del ámbito de una transacción individual fuerza la conexión entre ellas. Si la segunda tarea presenta errores, la primera se deshace hasta un punto anterior a la creación de la tabla nueva.

En términos de procesamiento, las transacciones se confirman o se anulan. Para que una transacción se confirme, todos los participantes deben garantizar la permanencia de los cambios efectuados en los datos. Los cambios deben conservarse aunque el sistema se bloquee o tengan lugar otros eventos imprevistos. Basta con que un solo participante no pueda garantizar este punto para que la transacción no funcione en su totalidad. Todos los cambios efectuados en datos dentro del ámbito de la transacción se deshacen hasta un punto específico establecido.

Las transacciones garantizan que se siguen las propiedades **ACID** (atomicidad, coherencia, aislamiento y duración) de forma que los datos se confirmen correctamente en la base de datos.

Los programadores de SQL son los responsables de iniciar y finalizar las transacciones en puntos que exijan la coherencia lógica de los datos. El programador debe definir la secuencia de modificaciones de datos que los dejan en un estado coherente en relación con las reglas de negocios de la organización. El programador incluye estas instrucciones de modificación en una sola transacción de forma que el Motor de base de datos de SQL Server puede hacer cumplir la integridad física de la misma.

Es responsabilidad de un sistema de base de datos corporativo, como una instancia del Motor de base de datos, proporcionar los mecanismos que aseguren la integridad física de cada transacción. SQL Server proporciona:

- Servicios de bloqueo que preservan el aislamiento de la transacción.
- Servicios de registro que aseguran la durabilidad de la transacción. Aunque se produzca un error en el hardware del servidor, el sistema operativo o la instancia del Motor de base de datos, la instancia utiliza registros de transacciones, al reiniciar, para revertir automáticamente las transacciones incompletas al punto en que se produjo el error del sistema.
- Características de administración de transacciones que exigen la atomicidad y coherencia de la transacción. Una vez iniciada una transacción, debe concluirse correctamente; en caso contrario, la instancia del Motor de base de datos deshará todas las modificaciones de datos realizadas desde que se inició la transacción.

Las transacciones pueden ser:

- **Locales**: Se considera que una transacción es local cuando es de una sola fase y la base de datos la trata directamente. O sea cuando todos los datos modificados dentro de la transacción pertenecen a la misma base de datos
- **Distribuida**: Una transacción distribuida es una transacción que actualiza datos en dos o más bases de datos conectadas a una red.

Tipos de Transacciones

- **Implícitas**: Cada instrucción SQL se confirma o se deshace cuando finaliza. No tiene que realizar ninguna acción para delinear el inicio de una transacción.
- **Explícitas**: Una transacción explícita es aquella en la que se definen explícitamente el inicio y el final de la transacción. Se pueden especificar mediante instrucciones SQL o funciones API de la base de datos.

Instrucciones para Transacciones Explícitas

Con SQL Server Management Studio se pueden utilizar las siguientes instrucciones SQL para definir transacciones explícitas:

- **BEGIN TRANSACTION**: Marca el punto de inicio de una transacción explícita para una conexión.
- **COMMIT TRANSACTIO*N***: Finaliza correctamente una transacción si no se han encontrado errores. Todos los datos modificados por la transacción se convierten en parte permanente de la base de datos. Se liberan los recursos ocupados por la transacción.
- **ROLLBACK TRANSACTION**: Borra una transacción en la que se han encontrado errores. Todos los datos modificados por la transacción vuelven al estado en el que estaban al inicio de la transacción. Se liberan los recursos ocupados por la transacción.

También puede utilizar transacciones explícitas en ADO .NET y OLE DB. En ADO .NET, use el método **BeginTransaction** en un objeto **SqlConnection** para iniciar una transacción explícita. Para finalizar la transacción, llame al método **Commit** o **Rollback** del objeto **SqlTransaction**.

BEGIN TRANSACTION

Marca el punto de inicio de una transacción local explícita. La instrucción BEGIN TRANSACTION incrementa @@TRANCOUNT en 1.

```
BEGIN { TRAN | TRANSACTION }
   [ { nombre_transacción | @variable_tran }
     [ WITH MARK [ 'descripcion' ] ]  ]
```

Argumentos:

- **Nombre_transacción**: Es el nombre asignado a la transacción. Este parámetro debe cumplir las reglas de los identificadores, pero no se admiten identificadores de más de 32 caracteres. Utilice nombres de transacciones solamente en la pareja más externa de instrucciones BEGIN…COMMIT o BEGIN…ROLLBACK anidadas.
- **@tran_name_variable*:*** Se trata del nombre de una variable definida por el usuario que contiene un nombre de transacción válido. La variable debe declararse con un tipo de datos char, varchar, nchar o nvarchar. Si se pasan más de 32 caracteres a la variable, sólo se utilizarán los primeros 32; el resto de caracteres se truncará.
- **WITH MARK ['descripcion']:** Especifica que la transacción está marcada en el registro. El parámetro 'descripcion' es una cadena que describe la marca. Si utiliza WITH MARK, debe especificar un nombre de transacción. WITH MARK permite restaurar un registro de transacciones hasta una marca con nombre.

BEGIN TRANSACTION representa un punto en el que los datos a los que hace referencia una conexión son lógica y físicamente coherentes. Si se producen errores, se pueden revertir todas las modificaciones realizadas en los datos después de BEGIN TRANSACTION para devolver los datos al estado conocido de coherencia. Cada transacción dura hasta que se completa sin

errores y se emite **COMMIT TRANSACTION** para hacer que las modificaciones sean una parte permanente de la base de datos, o hasta que se produzcan errores y se borren todas las modificaciones con la instrucción **ROLLBACK TRANSACTION**.
BEGIN TRANSACTION inicia una transacción local para la conexión que emite la instrucción. Según la configuración del nivel de aislamiento de la transacción actual, la transacción bloquea muchos recursos adquiridos para aceptar las instrucciones Transact-SQL emitidas por la conexión hasta que la misma finaliza con una instrucción COMMIT TRANSACTION o ROLLBACK TRANSACTION. Las transacciones que quedan pendientes durante mucho tiempo pueden impedir que otros usuarios tengan acceso a estos recursos bloqueados y pueden impedir también el truncamiento del registro.
Aunque BEGIN TRANSACTION inicia una transacción local, ésta no se guardará en el registro de transacciones hasta que la aplicación realice posteriormente una acción que se deba almacenar en el registro, como la ejecución de una instrucción INSERT, UPDATE o DELETE. Una aplicación puede realizar acciones tales como adquirir bloqueos para proteger el nivel de aislamiento de transacciones de instrucciones SELECT, pero no se guarda ningún dato en el registro hasta que la aplicación realiza una acción de modificación.

Ejemplo:

```
DECLARE @TranName VARCHAR(20);
SELECT @TranName = 'MyTransaction';

BEGIN TRANSACTION @TranName;
DELETE FROM HumanResources.JobCandidate
    WHERE JobCandidateID = 13;
COMMIT TRANSACTION @TranName;
```

BEGIN DISTRIBUTED TRANSACTION
Especifica el inicio de una transacción distribuida de Transact-SQL administrada mediante el **Coordinador de transacciones distribuidas de Microsoft (MS DTC).**
La instancia del Motor de base de datos de SQL Server que ejecuta la instrucción **BEGIN DISTRIBUTED TRANSACTION** es el originador de la transacción y controla su realización. Posteriormente, cuando en la sesión se ejecuta una instrucción COMMIT TRANSACTION o ROLLBACK TRANSACTION, la instancia que controla la transacción solicita a MS DTC que administre la realización de la transacción distribuida entre todas las instancias participantes.
La principal manera en que las instancias remotas del Motor de base de datos se dan de alta en una transacción distribuida es cuando una sesión ya dada de alta en la transacción distribuida ejecuta una consulta distribuida que hace referencia a un servidor vinculado.

Una transacción local iniciada por la instrucción BEGIN TRANSACTION aumenta al nivel de transacción distribuida si se realizan las siguientes acciones antes de confirmarla o revertirla:

- Se ejecuta una instrucción INSERT, DELETE o UPDATE que hace referencia a una tabla remota de un servidor vinculado.
- Se realiza una llamada a un procedimiento almacenado remoto cuando la opción REMOTE_PROC_TRANSACTIONS es ON.

La copia local de SQL Server se convierte en el controlador de la transacción y utiliza el Coordinador de transacciones distribuidas de Microsoft (MS DTC) para administrar la transacción distribuida.

Ejemplo:

```
BEGIN DISTRIBUTED TRANSACTION;
-- Borra un registro sobre la base de datos local
DELETE AdventureWorks2008.HumanResources.JobCandidate
```

```
WHERE JobCandidateID = 13;
-- Borra un registro sobre la base de datos remota
DELETE RemoteServer.AdventureWorks2008.HumanResources.JobCandidate
WHERE JobCandidateID = 13;
COMMIT TRANSACTION;
```

COMMIT TRANSACTION
Marca el final de una transacción correcta, implícita o explícita. Si @@**TRANCOUNT** es 1, COMMIT TRANSACTION hace que todas las modificaciones efectuadas sobre los datos desde el inicio de la transacción sean parte permanente de la base de datos, libera los recursos mantenidos por la transacción y reduce @@**TRANCOUNT** a 0. Si @@**TRANCOUNT** es mayor que 1, COMMIT TRANSACTION sólo reduce @@TRANCOUNT en 1 y la transacción sigue activa.

COMMIT { TRAN | TRANSACTION } [nombre_transaccion | @variable_tran]]

Es responsabilidad del programador de Transact-SQL utilizar COMMIT TRANSACTION sólo en el punto donde todos los datos a los que hace referencia la transacción sean lógicamente correctos.
Si la transacción que se ha confirmado era una transacción Transact-SQL distribuida, COMMIT TRANSACTION hace que MS DTC utilice el protocolo de confirmación en dos fases para confirmar los servidores involucrados en la transacción. Si una transacción local afecta a dos o más bases de datos de la misma instancia del Motor de base de datos, la instancia utiliza una confirmación interna en dos fases para confirmar todas las bases de datos involucradas en la transacción.
Cuando se utiliza en transacciones anidadas, las confirmaciones de las transacciones anidadas no liberan recursos ni hacen permanentes sus modificaciones. Las modificaciones sobre los datos sólo quedan permanentes y se liberan los recursos cuando se confirma la transacción más externa. Cada COMMIT TRANSACTION que se ejecute cuando @@TRANCOUNT sea mayor que 1 sólo reduce @@TRANCOUNT en 1. Cuando @@TRANCOUNT llega a 0, se confirma la transacción externa entera.

Nota: No se puede revertir una transacción después de ejecutar una instrucción COMMIT TRANSACTION, porque las modificaciones sobre los datos ya son parte permanente de la base de datos.

ROLLBACK TRANSACTION
Revierte una transacción explícita o implícita hasta el inicio de la transacción o hasta un punto de retorno dentro de la transacción. Una transacción no se puede revertir después de ejecutar una instrucción COMMIT TRANSACTION.

ROLLBACK { TRAN | TRANSACTION }
[nombre_transaction | @variable_tran
| nombre_savepoint | @variable_savepoint]

ROLLBACK TRANSACTION borra todas las modificaciones de datos realizadas desde el inicio de la transacción o hasta un punto de retorno. También libera los recursos que retiene la transacción.
ROLLBACK TRANSACTION sin nombre_savepoint o nombre_transaction revierte todas las instrucciones hasta el principio de la transacción. Cuando se trata de transacciones anidadas, esta misma instrucción revierte todas las transacciones internas hasta la instrucción BEGIN TRANSACTION más externa. En ambos casos, ROLLBACK TRANSACTION disminuye la

función del sistema @@TRANCOUNT a 0. ROLLBACK TRANSACTION con nombre_savepoint no disminuye @@TRANCOUNT. Si se especifica un nombre_savepoint libera todos los bloqueos adquiridos más allá del punto de retorno, a excepción de las extensiones y las conversiones. Estos bloqueos no se liberan y no vuelven a convertirse a su modo de bloqueo previo.

ROLLBACK TRANSACTION no puede hacer referencia a un argumento nombre_savepoint en transacciones distribuidas que se inician de forma explícita con BEGIN DISTRIBUTED TRANSACTION o que se extienden desde una transacción local.

En procedimientos almacenados, las instrucciones ROLLBACK TRANSACTION sin un argumento nombre_savepoint o nombre_transaction revierten todas las instrucciones hasta la instrucción BEGIN TRANSACTION más externa. Una instrucción ROLLBACK TRANSACTION de un procedimiento almacenado que provoca que @@TRANCOUNT muestre un valor diferente cuando finaliza el procedimiento almacenado del valor de @@TRANCOUNT en el momento de la llamada al procedimiento almacenado, genera un mensaje informativo. Este mensaje no afecta a los siguientes procesos.

Si se emite la instrucción ROLLBACK TRANSACTION en un desencadenador:

- Se revierten todas las modificaciones de datos realizadas hasta ese punto de la transacción actual, incluidas las que realizó el desencadenador.
- El desencadenador continúa la ejecución del resto de las instrucciones después de la instrucción ROLLBACK. Si alguna de estas instrucciones modifica datos, no se revierten las modificaciones. La ejecución de las instrucciones restantes no activa ningún desencadenador anidado.
- Tampoco se ejecutan las instrucciones del lote después de la instrucción que activó el desencadenador.

Una instrucción ROLLBACK TRANSACTION no produce ningún mensaje para el usuario. Si necesita indicar advertencias en procedimientos almacenados o en desencadenadores, utilice las instrucciones RAISERROR o PRINT. RAISERROR es la instrucción más adecuada para indicar errores.

SAVE TRANSACTION
Establece un punto de retorno dentro de una transacción.

SAVE {TRAN | TRANSACTION } { nombre_savepoint | @ variable_savepoint }

Un usuario puede establecer un punto de retorno, o marcador, dentro de una transacción. El punto de retorno define una ubicación a la que la transacción puede volver si se cancela parte de la transacción de forma condicional. Si se revierte una transacción hasta un punto de retorno, se debe continuar hasta su finalización con más instrucciones Transact-SQL si es necesario y una instrucción COMMIT TRANSACTION o se debe cancelar completamente al revertir la transacción hasta su inicio. Para cancelar una transacción completa, utilice el formato ROLLBACK TRANSACTION nombre_transaction. Se deshacen todas las instrucciones o procedimientos de la transacción.
En una transacción se permiten nombres de puntos de retorno duplicados, pero una instrucción ROLLBACK TRANSACTION que especifique el nombre de un punto de retorno sólo revertirá la transacción hasta la instrucción SAVE TRANSACTION más reciente que también utilice ese nombre.
No se admite SAVE TRANSACTION en transacciones distribuidas iniciadas explícitamente con BEGIN DISTRIBUTED TRANSACTION u originadas a partir de una transacción local.

Cuando una transacción comienza, los recursos utilizados durante la transacción se mantienen hasta la finalización de la misma (es decir, los bloqueos). Cuando se revierte parte de una transacción hasta un punto de retorno, se mantienen los recursos hasta la finalización de la transacción o hasta revertir la transacción completa.

Ejemplo:

```
CREATE PROCEDURE SaveTranExample @InputCandidateID INT
AS
DECLARE @TranCounter INT;
SET @TranCounter = @@TRANCOUNT;
IF @TranCounter > 0
    -- la transaccion es llamada dentro de una transaccion activa.
Crea un SavePoint
    -- para poder revertir solo este proceso si el procedimiento falla
    SAVE TRANSACTION ProcedureSave;
ELSE
    -- Comienza una transacción independiente
    BEGIN TRANSACTION;

-- Modifica la base de datos
BEGIN TRY
      DELETE HumanResources.JobCandidate
      WHERE JobCandidateID = @InputCandidateID;
      IF @TranCounter = 0
            -- la transacción comenzo dentro del procedimiento
            COMMIT TRANSACTION;
END TRY
BEGIN CATCH
      IF @TranCounter = 0
            -- la transacción comenzo dentro del procedimiento
            ROLLBACK TRANSACTION;
      ELSE
            BEGIN
            -- La transacción comenzó antes de la llamada al
Procedimiento
            IF XACT_STATE() <> -1
                -- La transacción es valida hace rollback hasta el
savepoint
                 ROLLBACK TRANSACTION ProcedureSave;
            END

      DECLARE @ErrorMessage NVARCHAR(4000);
      DECLARE @ErrorSeverity INT;
      DECLARE @ErrorState INT;
      SELECT @ErrorMessage = ERROR_MESSAGE();
      SELECT @ErrorSeverity = ERROR_SEVERITY();
      SELECT @ErrorState = ERROR_STATE();
      RAISERROR (@ErrorMessage, -- Message text.
                 @ErrorSeverity, -- Severity.
                 @ErrorState) -- State.);
END CATCH
```

XACT_STATE
Función escalar que notifica el estado de la transacción de usuario de una solicitud que se está ejecutando actualmente. Indica si la solicitud tiene una transacción de usuario activa y si se puede confirmar la transacción.
Devuelve los siguientes valores:

- 0: No hay ninguna transacción de usuario activa para la solicitud actual.
- 1: La solicitud actual tiene una transacción de usuario activa. La solicitud puede realizar cualquier acción, incluida la escritura de datos y la confirmación de la transacción.
- -1: La solicitud actual tiene una transacción de usuario activa, pero se ha producido un error por el cual la transacción se clasificó como no confirmable. La solicitud no puede confirmar la transacción o revertirla a un punto de retorno; sólo puede solicitar que la transacción se revierta completamente. Tampoco puede realizar operaciones de escritura hasta que se revierta la transacción. Mientras tanto, sólo puede realizar operaciones de lectura. Una vez que la transacción se ha revertido, la solicitud puede realizar operaciones de lectura y escritura, y puede iniciar transacciones nuevas.

SET IMPLICIT_TRANSACTIONS
Establece el modo de transacción implícita para la conexión.

```
SET IMPLICIT_TRANSACTIONS { ON | OFF }
```

Cuando es ON, SET IMPLICIT_TRANSACTIONS establece la conexión al modo de transacción implícita. Cuando es OFF, restablece la conexión al modo de transacción con confirmación automática.
Cuando una conexión está en modo de transacción implícita y actualmente no está realizando una transacción, la ejecución de cualquiera de las instrucciones siguientes inicia una transacción:

- CREATE
- DELETE
- DROP
- FETCH
- GRANT
- REVOKE
- INSERT
- DELETE
  UPDATE
- SELECT
- TRUNCATE TABLE

Si la conexión tiene ya una transacción abierta, estas instrucciones no inician una nueva transacción.
Al término de las transacciones que se abran automáticamente por ser esta opción ON, el usuario debe confirmarlas o revertirlas explícitamente. En caso contrario, las transacciones y todos los cambios que se realicen en los datos se revertirán cuando el usuario se desconecte. Una vez confirmada una transacción, la ejecución de alguna de las instrucciones anteriores iniciará una transacción nueva.
El modo de transacción implícita permanecerá activo hasta que la conexión ejecute una instrucción SET IMPLICIT_TRANSACTIONS OFF, que restablece el modo de confirmación automática en la conexión. En el modo de confirmación automática, todas las instrucciones individuales se confirman cuando finalizan correctamente.

Transacciones Anidadas

Las transacciones explícitas se pueden anidar. El objetivo principal de esto es aceptar transacciones en procedimientos almacenados a los que se puede llamar desde un proceso que ya esté en una transacción o desde procesos que no tengan transacciones activas.

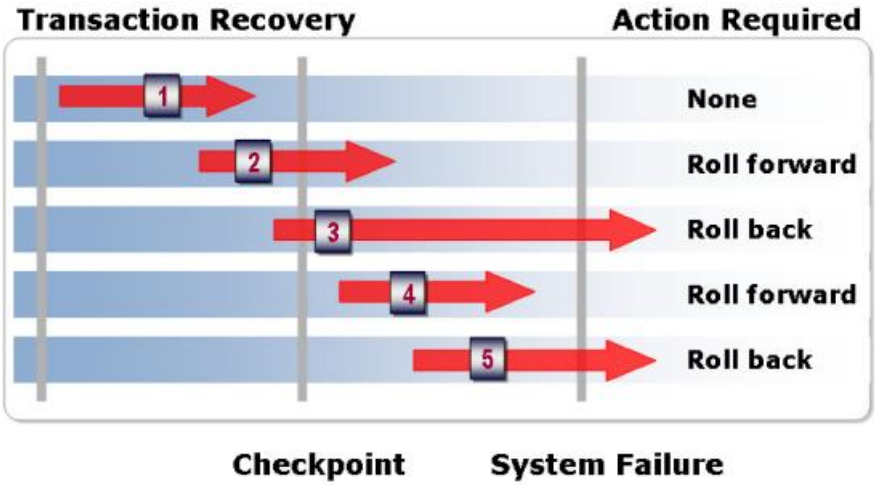
El Motor de base de datos de SQL Server omite la confirmación de las transacciones internas. La transacción se confirma o se revierte basándose en la acción realizada al final de la transacción más externa. Si se confirma la transacción externa, también se confirmarán las transacciones anidadas internas. Si se revierte la transacción externa, también se revertirán todas las transacciones internas, independientemente de si se confirmaron individualmente o no.

Cada llamada a COMMIT TRANSACTION se aplica a la última instrucción BEGIN TRANSACTION ejecutada. Si las instrucciones BEGIN TRANSACTION están anidadas, la instrucción COMMIT sólo se aplica a la última transacción anidada, que es la más interna. Aunque una instrucción COMMIT TRANSACTION nombre_transaccion de una transacción anidada haga referencia al nombre de la transacción externa, la confirmación sólo se aplicará a la transacción más interna.

No es válido que el parámetro nombre_transaction de una instrucción ROLLBACK TRANSACTION haga referencia a las transacciones internas de un conjunto de transacciones anidadas con nombre, sólo puede hacer referencia al nombre de la transacción más externa.

La función **@@TRANCOUNT** registra el nivel de anidamiento de la transacción actual. Cada instrucción BEGIN TRANSACTION incrementa @@TRANCOUNT en uno. Cada instrucción COMMIT TRANSACTION o COMMIT WORK reduce @@TRANCOUNT en uno. Una instrucción ROLLBACK TRANSACTION que no tenga nombre de transacción revertirá todas las transacciones anidadas y establecerá @@TRANCOUNT en 0. Una instrucción ROLLBACK TRANSACTION que utilice el nombre de la transacción más externa de un conjunto de transacciones anidadas revertirá todas las transacciones anidadas y establecerá @@TRANCOUNT en 0. Cuando no esté seguro de si ya está en una transacción, use SELECT @@TRANCOUNT para determinar si es 1 o más. Si @@TRANCOUNT es 0, no está en una transacción.

Recuperación de Transacciones



SQL Server automáticamente garantiza que todas las transacciones confirmadas se reflejen en la base de datos en el caso de un fallo. Se utiliza el registro de transacciones y los checkpoints.

2- Bloqueos

Modos de Bloqueos

El Motor de base de datos de SQL Server de Microsoft bloquea los recursos con diferentes modos de bloqueo que determinan el modo en que las transacciones simultáneas pueden tener acceso a los recursos.

En la siguiente tabla se indican los modos de bloqueo de recursos que se emplean:

Modo de Bloqueo	Descripción
Compartido (S)	Se utiliza para operaciones de lectura que no cambian ni actualizan datos, como la instrucción SELECT. Ninguna otra transacción podrá modificar los datos mientras el bloqueo compartido (S) exista en el recurso. Los bloqueos compartidos (S) en un recurso se liberan tan pronto como finaliza la operación de lectura, a menos que se haya establecido el nivel de aislamiento de la transacción como REPEATABLE READ o más alto, o bien se utilice una sugerencia de bloqueo para mantener los bloqueos compartidos (S) durante la transacción.
Actualizar (U)	Se utiliza en recursos que se pueden actualizar. Evita una forma común de interbloqueo que se produce cuando varias sesiones leen, bloquean y actualizan recursos. En una transacción de lectura repetible o serializable, la transacción lee los datos, adquiere un bloqueo compartido (S) en el recurso (página o fila) y, a continuación, modifica los datos, lo que requiere una conversión del bloqueo en un bloqueo exclusivo (X). Si dos transacciones adquieren bloqueos compartidos en un recurso y, a continuación, intentan actualizar los datos simultáneamente, una de ellas intenta convertir el bloqueo en un bloqueo exclusivo (X). La conversión de bloqueo compartido en exclusivo debe esperar, ya que el bloqueo exclusivo de una transacción no es compatible con el bloqueo compartido de la otra. Por tanto, se produce una espera de bloqueos. La segunda transacción intenta adquirir un bloqueo exclusivo (X) para realizar su actualización. Debido a que ambas transacciones intentan convertir los bloqueos en exclusivos (X) y cada una espera a que la otra libere su bloqueo de modo compartido, se produce un interbloqueo. Para evitar este posible problema de interbloqueo, se utilizan los bloqueos de actualización (U). Dos transacciones no pueden obtener simultáneamente un bloqueo de actualización (U) para un recurso. Si una transacción modifica un recurso, el bloqueo de actualización (U) se convierte en un bloqueo exclusivo (X).
Exclusivo (X)	Se utiliza para operaciones de modificación de datos, como INSERT, UPDATE o DELETE. Garantiza que no puedan realizarse varias actualizaciones simultáneamente en el mismo recurso. Los bloqueos exclusivos (X) evitan que transacciones simultáneas tengan acceso a un recurso. Al utilizar un bloqueo exclusivo (X), el resto de las transacciones no pueden modificar los datos; las operaciones de lectura sólo se pueden realizar si se utiliza la sugerencia NOLOCK o el nivel de aislamiento de lectura no confirmada.
Intensión	Se utiliza para establecer una jerarquía de bloqueos. Los tipos de bloqueos con intención son: Intención compartida (IS), Intención exclusiva (IX) e Intención compartida exclusiva (SIX). El Motor de base de datos utiliza bloqueos con intención para proteger la aplicación de un bloqueo compartido (S) o exclusivo (X) en un recurso inferior en la jerarquía de bloqueos. Los bloqueos con intención se denominan así porque se adquieren antes que los bloqueos de los niveles inferiores y, por lo tanto, señalan la intención de aplicar bloqueos en un nivel inferior. Los bloqueos con intención se utilizan con dos fines: Evitar que otras transacciones modifiquen el recurso de nivel superior de forma que invaliden el bloqueo del nivel inferior y para mejorar la eficacia del Motor de base de datos para detectar conflictos de bloqueo en el nivel superior de granularidad.

Esquema	Se utiliza cuando se ejecuta una operación que depende del esquema de una tabla. Los tipos de bloqueos de esquema son: Modificación del esquema (Sch-M) y Estabilidad del esquema (Sch-S). El Motor de base de datos utiliza bloqueos de modificación del esquema (Sch-M) cuando se realiza una operación de lenguaje de definición de datos (DDL) en tablas como, por ejemplo, agregar una columna o quitar una tabla. Mientras se conserva, el bloqueo Sch-M evita el acceso simultáneo a la tabla. Esto significa que el bloqueo Sch-M bloquea todas las operaciones externas hasta que el bloqueo se libera. Algunas operaciones del lenguaje de manipulación de datos (DML), como el truncamiento de tablas, utilizan los bloqueos Sch-M para impedir el acceso a las tablas afectadas por operaciones simultáneas. El Motor de base de datos usa bloqueos de estabilidad del esquema (Sch-S) al compilar y ejecutar consultas. Los bloqueos Sch-S no impiden los bloqueos de transacciones, incluidos los bloqueos exclusivos (X). Por tanto, otras transacciones, incluidas las que tienen bloqueos X de una tabla, pueden seguir ejecutándose mientras se compila una consulta. No obstante, en la tabla no se pueden realizar operaciones DDL simultáneas ni operaciones DML simultáneas que adquieren bloqueos Sch-M.
Actualización Masiva (BU)	Se utiliza cuando se copian datos de forma masiva en una tabla y se especifica la sugerencia TABLOCK. Los bloqueos de actualización masiva (BU) permiten que varios subprocesos copien datos de forma masiva y simultánea en la misma tabla, pero impiden que otros procesos que no están copiando datos de forma masiva tengan acceso a la tabla.
Intervalo de Clases	Protege el intervalo de filas que lee una consulta cuando se utiliza el nivel de aislamiento de transacciones serializables. Garantiza que otras transacciones no puedan insertar filas que podrían incluirse como respuesta de las consultas de la transacción serializable si las consultas se volvieran a ejecutar.

Compatibilidad de Bloqueos

La compatibilidad de bloqueos controla si varias transacciones pueden adquirir bloqueos sobre el mismo recurso a la vez. Si un recurso ya está bloqueado por otra transacción, sólo se puede conceder una nueva solicitud de bloqueo si el bloqueo solicitado es compatible con el modo del bloqueo existente. Si el modo del bloqueo solicitado no es compatible con el bloqueo existente, la transacción que solicita el nuevo bloqueo espera a que se libere el bloqueo existente o a que caduque el intervalo de tiempo de espera del bloqueo. Por ejemplo, ningún modo de bloqueo es compatible con bloqueos exclusivos. Mientras se mantiene un bloqueo exclusivo (X), ninguna otra transacción puede adquirir un bloqueo de ninguna clase (compartido, de actualización o exclusivo) en dicho recurso hasta que se libere el bloqueo exclusivo. Como alternativa, si se ha aplicado un bloqueo compartido (S) a un recurso, otras transacciones también pueden adquirir un bloqueo compartido o de actualización (U) en el elemento, aunque la primera transacción no haya terminado. Sin embargo, otras transacciones no pueden adquirir un bloqueo exclusivo si no se anula el bloqueo compartido.

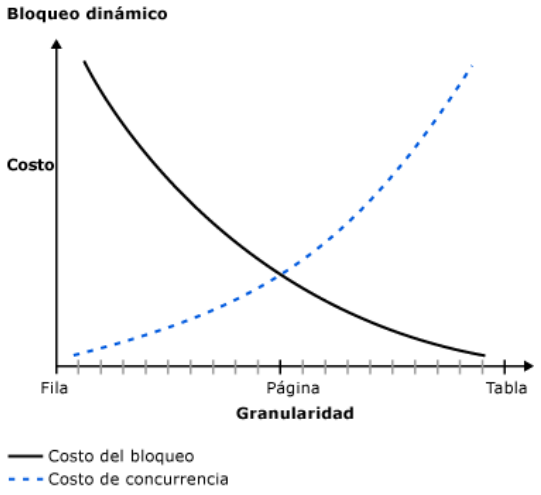
En la tabla siguiente se muestra la compatibilidad de los modos de bloqueo más frecuentes:

	Modo Otorgado					
Modo Solicitado	**IS**	**S**	**U**	**IX**	**SIX**	**X**
Intención Compartida (IS)	Sí	Sí	Sí	Sí	Sí	No
Compartido (S)	Sí	Sí	Sí	No	No	No
Actualizar (U)	Sí	Sí	No	No	No	No
Con Intención Exclusivo (IX)	Sí	No	No	Sí	No	No
Compartido con Intención Exclusivo (SIX)	Sí	No	No	No	No	No
Exclusivo (X)	No	No	No	No	No	No

Bloqueo dinámico
La utilización de bloqueos de bajo nivel, como los de fila, aumenta la simultaneidad reduciendo la probabilidad de que dos transacciones soliciten bloqueos de los mismos datos al mismo tiempo. También aumenta el número de bloqueos y los recursos necesarios para administrarlos. Los bloqueos de alto nivel de tabla o página producen una sobrecarga menor, pero a costa de reducir la simultaneidad.
El Motor de base de datos de SQL Server utiliza una estrategia de bloqueo dinámico para determinar los bloqueos que son más eficaces. Determina automáticamente los bloqueos más apropiados cuando se ejecuta la consulta, basándose en las características del esquema y de la consulta. Por ejemplo, para reducir la sobrecarga de bloqueos, el optimizador puede decidir la realización de bloqueos de página en un índice al realizar un recorrido del índice.

El bloqueo dinámico presenta las ventajas siguientes:

- Administración simplificada de la base de datos. Los administradores de bases de datos no tienen que preocuparse de ajustar los umbrales de extensión de bloqueo.
- Mayor rendimiento. El Motor de base de datos minimiza la sobrecarga del sistema al utilizar los bloqueos apropiados para la tarea.
- Los programadores de aplicaciones se pueden concentrar en la programación.



Personalizar el Tiempo de Espera de Bloqueos
Cuando una instancia del Motor de base de datos de SQL Server de Microsoft no puede conceder un bloqueo a una transacción porque otra transacción ya posee un bloqueo en conflicto en el recurso, la primera transacción queda bloqueada a la espera de que se libere el bloqueo existente. De forma predeterminada no hay un tiempo de espera obligatorio, ni tampoco existe ningún modo de comprobar si un recurso está bloqueado antes de intentar bloquearlo, excepto intentar tener acceso a los datos (con el riesgo de quedar bloqueado indefinidamente).
El parámetro **LOCK_TIMEOUT** permite a una aplicación establecer el tiempo máximo que una instrucción esperará a un recurso bloqueado. Cuando una instrucción ha esperado más tiempo del indicado en LOCK_TIMEOUT, la instrucción bloqueada se cancela automáticamente y se devuelve el mensaje de error a la aplicación. Sin embargo, SQL Server no cancela ni revierte ninguna transacción que contenga la instrucción. Si una aplicación no intercepta el error, puede continuar sin ser consciente de que se ha cancelado una instrucción individual de la transacción y de que esto puede producir errores, ya que las instrucciones posteriores de la transacción podrían depender de la instrucción que nunca se ejecutó.

```
SET LOCK_TIMEOUT período
```

Argumentos:

- **Período**: Es el número de milisegundos que transcurrirán antes de que Microsoft SQL Server devuelva un error de bloqueo. El valor -1 (predeterminado) indica que no hay límite de espera (es decir que se espera indefinidamente). El valor 0 significa no esperar y devolver un mensaje en cuanto se encuentre un bloqueo.

Para determinar el valor actual de LOCK_TIMEOUT, ejecute la función @@**LOCK_TIMEOUT**:

Ejemplo:

```
SET LOCK_TIMEOUT 1800
SELECT @@lock_timeout
```

Granularidad y Jerarquías de Bloqueos

El Motor de base de datos de SQL Server admite bloqueo multigranular. Esta función permite que una transacción bloquee diferentes tipos de recursos. Para minimizar el costo del bloqueo, el Motor de base de datos bloquea automáticamente los recursos en el nivel apropiado para la tarea. Los bloqueos de menor granularidad, como es el caso de las filas, aumentan la simultaneidad. Sin embargo, se produce una sobrecarga mayor porque cuantas más filas se bloquean, más bloqueos se deben mantener. Los bloqueos realizados en una granularidad alta, por ejemplo en tablas, reducen la simultaneidad porque el bloqueo de toda una tabla restringe el acceso de otras transacciones a cualquier parte de la tabla. Sin embargo, produce una sobrecarga menor debido a que se mantienen menos bloqueos.

El Motor de base de datos a menudo se ve en la obligación de adquirir bloqueos en distintos niveles de granularidad para brindar una protección completa a un recurso. Este grupo de bloqueos en distintos niveles de granularidad se denomina jerarquía de bloqueos. Por ejemplo, para brindar protección completa a la lectura de un índice, probablemente sea necesario que una instancia del Motor de base de datos adquiera bloqueos compartidos en filas y bloqueos con intención compartida en las páginas y la tabla.

En la siguiente tabla se muestran los recursos que el Motor de base de datos puede bloquear:

Recurso	Descripción
RID	Identificador de fila que se utiliza para bloquear una sola fila de un montón.
KEY	Bloqueo de fila dentro de un índice que se utiliza para proteger intervalos de claves en transacciones serializables.
PAGE	Página de 8 kilobytes (KB) de una base de datos, como páginas de datos o de índices.
EXTENT	Grupo contiguo de ocho páginas, como páginas de datos o de índices.
HOBT	Montón o árbol B. Bloqueo que protege un índice o el montón de páginas de datos en una tabla que no contiene un índice clúster.
TABLE	Tabla completa, con todos los datos e índices.
FILE	Archivos de la base de datos.
APPLICATION	Recurso especificado por la aplicación.

METADATA	Bloqueos de metadatos.
ALLOCATION_UNIT	Unidad de asignación
DATABASE	Base de datos completa.

Efectos de la simultaneidad

Los usuarios que modifican datos pueden afectar a otros usuarios que leen o modifican los mismos datos a la vez. Se dice que estos usuarios tienen acceso a los datos de forma simultánea. Si un sistema de almacenamiento de datos no dispone de control de simultaneidad, los usuarios se pueden encontrar con los siguientes efectos secundarios:

- **Actualizaciones perdidas**: Este problema surge cuando dos o más transacciones seleccionan la misma fila y, a continuación, la actualizan de acuerdo con el valor seleccionado originalmente. Ninguna transacción es consciente de las otras transacciones. La última actualización sobrescribe las actualizaciones realizadas por las otras transacciones y, en consecuencia, se pierden datos.
- **Dependencia no confirmada (lectura no actualizada)**: Este problema se produce cuando una transacción selecciona una fila que está siendo actualizada por otra transacción. La segunda transacción lee datos que no han sido confirmados aún y pueden ser modificados por la transacción que está actualizando la fila.
- **Análisis incoherente (lectura repetitivas)**: Este problema se produce cuando una transacción obtiene acceso a la misma fila varias veces y en cada ocasión lee datos diferentes. El análisis incoherente es similar a la dependencia no confirmada en tanto que una transacción está modificando los datos que está leyendo una segunda transacción. Sin embargo, en el caso del análisis incoherente, los datos que lee la segunda transacción están confirmados por la transacción que realizó el cambio. Además, el análisis incoherente comprende varias lecturas (dos o más) de la misma fila y las transacciones modifican la información cada vez; de ahí el término de lectura irrepetible.
- **Lecturas ficticias**: Este problema se produce cuando se realiza una acción de insertar o eliminar en una fila y ésta pertenece a un intervalo de filas que está leyendo una transacción. La primera lectura que hizo la transacción en el intervalo de filas muestra una fila que ya no existe en la segunda lectura o en lecturas sucesivas, porque otra transacción la ha eliminado. De forma similar, la segunda lectura o las lecturas sucesivas de la transacción muestran una fila que no existía en la primera lectura, como consecuencia de una inserción realizada por otra transacción.

Nivel de Aislamiento de una Transacción

Las transacciones especifican un nivel de aislamiento que define el grado en que se debe aislar una transacción de las modificaciones de recursos o datos realizadas por otras transacciones. Los niveles de aislamiento se describen en cuanto a los efectos secundarios de la simultaneidad que se permiten, como las lecturas desfasadas o ficticias.

Control de los niveles de aislamiento de transacción:

- Controla si se realizan bloqueos cuando se leen los datos y qué tipos de bloqueos se solicitan.
- Duración de los bloqueos de lectura.
- Si una operación de lectura que hace referencia a filas modificadas por otra transacción:
  - Se bloquea hasta que se libera el bloqueo exclusivo de la fila.
  - Recupera la versión confirmada de la fila que existía en el momento en el que empezó la instrucción o la transacción.

- Lee la modificación de los datos no confirmados.

La selección de un nivel de aislamiento de transacción no afecta a los bloqueos adquiridos para proteger la modificación de datos. Siempre se obtiene un bloqueo exclusivo en los datos modificados de una transacción, bloqueo que se mantiene hasta que se completa la transacción, independientemente del nivel de aislamiento seleccionado para la misma. En el caso de las operaciones de lectura, los niveles de aislamiento de transacción definen básicamente el nivel de protección contra los efectos de las modificaciones que realizan otras transacciones.
Un nivel de aislamiento menor significa que los usuarios tienen un mayor acceso a los datos simultáneamente, con lo que aumentan los efectos de simultaneidad que pueden experimentar, como las lecturas desfasadas o la pérdida de actualizaciones. Por el contrario, un nivel de aislamiento mayor reduce los tipos de efectos de simultaneidad, pero requiere más recursos del sistema y aumenta las posibilidades de que una transacción bloquee otra. El nivel de aislamiento apropiado depende del equilibrio entre los requisitos de integridad de los datos de la aplicación y la sobrecarga de cada nivel de aislamiento.

SET TRANSACTION ISOLATION LEVEL
El Motor de base de datos de SQL Server admite los niveles de aislamiento de transacción definidos en SQL-92. La configuración de dichos niveles permite a los programadores compensar el riesgo superior de que se produzcan ciertos problemas de integridad al permitirse un mayor acceso simultáneo a los datos.

Los niveles de aislamiento de transacción son los siguientes y se establecen usando SET TRANSACTION ISOLATION LEVEL.

- **READ UNCOMMITTED**: Especifica que las instrucciones pueden leer filas que han sido modificadas por otras transacciones pero todavía no se han confirmado. Las transacciones que se ejecutan en el nivel READ UNCOMMITTED no emiten bloqueos compartidos para impedir que otras transacciones modifiquen los datos leídos por la transacción actual. Las transacciones READ UNCOMMITTED tampoco se bloquean mediante bloqueos exclusivos que impedirían que la transacción actual leyese las filas modificadas pero no confirmadas por otras transacciones. Cuando se establece esta opción, es posible leer las modificaciones no confirmadas, denominadas lecturas de datos sucios. Los valores de los datos se pueden cambiar, y las filas pueden aparecer o desaparecer en el conjunto de datos antes de que finalice la transacción.
- **READ COMMITTED**: Especifica que las instrucciones no pueden leer datos que hayan sido modificados, pero no confirmados, por otras transacciones. Esto evita las lecturas de datos sucios. Otras transacciones pueden cambiar datos entre cada una de las instrucciones de la transacción actual, dando como resultado lecturas no repetibles o datos ficticios. Esta opción es la predeterminada para SQL Server.
  El comportamiento de READ COMMITTED depende del valor de la opción de base de datos **READ_COMMITTED_SNAPSHOT**:
  - Si se establece en **OFF** (valor predeterminado), el Motor de base de datos utiliza bloqueos compartidos para impedir que otras transacciones modifiquen las filas mientras la transacción actual esté ejecutando una operación de lectura. Los bloqueos compartidos impiden también que la instrucción lea las filas modificadas por otras transacciones hasta que la otra transacción haya finalizado. El tipo de bloqueo compartido determina cuándo se liberará. Los bloqueos de fila se liberan antes de que se procese la fila siguiente. Los bloqueos de página se liberan cuando se lee la página siguiente, y los bloqueos de tabla se liberan cuando la instrucción finaliza.
  - Si se establece en **ON**, el Motor de base de datos utiliza versiones de fila para presentar a cada instrucción una instantánea coherente, desde el punto de vista

transaccional, de los datos tal como se encontraban al comenzar la instrucción. No se utilizan bloqueos para impedir que otras transacciones actualicen los datos.

- **REPEATABLE READ**: Especifica que las instrucciones no pueden leer datos que han sido modificados pero aún no confirmados por otras transacciones y que ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que ésta finalice. Se aplican bloqueos compartidos a todos los datos leídos por cada instrucción de la transacción, y se mantienen hasta que la transacción finaliza. De esta forma, se evita que otras transacciones modifiquen las filas que han sido leídas por la transacción actual. Otras transacciones pueden insertar filas nuevas que coincidan con las condiciones de búsqueda de las instrucciones emitidas por la transacción actual. Si la transacción actual vuelve a ejecutar la instrucción, recuperará las filas nuevas, dando como resultado lecturas ficticias. Debido a que los bloqueos compartidos se mantienen hasta el final de la transacción en lugar de liberarse al final de cada instrucción, la simultaneidad es inferior que en el nivel de aislamiento predeterminado READ COMMITTED. Utilice esta opción solamente cuando sea necesario.
- **SNAPSHOT**: Especifica que los datos leídos por cualquier instrucción de una transacción sean la versión coherente, desde el punto de vista transaccional, de los datos existentes al comienzo de la transacción. La transacción únicamente puede reconocer las modificaciones de datos confirmadas antes del comienzo de la misma. Las instrucciones que se ejecuten en la transacción actual no verán las modificaciones de datos efectuadas por otras transacciones después del inicio de la transacción actual. El efecto es el mismo que se obtendría si las instrucciones de una transacción obtuviesen una instantánea de los datos confirmados tal como se encontraban al comienzo de la transacción. Las transacciones SNAPSHOT no solicitan bloqueos al leer los datos, excepto cuando se recupera una base de datos. Las transacciones SNAPSHOT que leen datos no bloquean la escritura de datos de otras transacciones. Las transacciones que escriben datos no bloquean la lectura de datos de las transacciones SNAPSHOT. Durante la fase de reversión de la recuperación de una base de datos, las transacciones SNAPSHOT solicitan un bloqueo si se intenta leer datos bloqueados por otra transacción que está en proceso de reversión. La transacción SNAPSHOT se bloquea hasta que finalice la reversión de esa transacción. El bloqueo se libera justo después de haberse concedido.

  La opción de base de datos **ALLOW_SNAPSHOT_ISOLATION** debe establecerse en ON para poder iniciar una transacción que utilice el nivel de aislamiento SNAPSHOT. Si una transacción que utiliza el nivel de aislamiento SNAPSHOT obtiene acceso a datos de varias bases de datos, será necesario establecer ALLOW_SNAPSHOT_ISOLATION en ON en cada una de ellas.

  Una transacción que se ejecuta en el nivel de aislamiento SNAPSHOT puede ver los cambios realizados por esa transacción. Por ejemplo, si la transacción realiza una operación UPDATE en una tabla y después emite una instrucción SELECT para la misma tabla, los datos modificados se incluirán en el conjunto de resultados.
- **SERIALIZABLE**: Especifica lo siguiente:
  - Las instrucciones no pueden leer datos que hayan sido modificados, pero aún no confirmados, por otras transacciones.
  - Ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que la transacción actual finalice.
  - Otras transacciones no pueden insertar filas nuevas con valores de clave que pudieran estar incluidos en el intervalo de claves leído por las instrucciones de la transacción actual hasta que ésta finalice.

  Se colocan bloqueos de intervalo en el intervalo de valores de clave que coincidan con las condiciones de búsqueda de cada instrucción ejecutada en una transacción. De esta manera, se impide que otras transacciones actualicen o inserten filas que satisfagan los requisitos de alguna de las instrucciones ejecutadas por la transacción actual. Esto significa que, si alguna de las instrucciones de una transacción se ejecuta

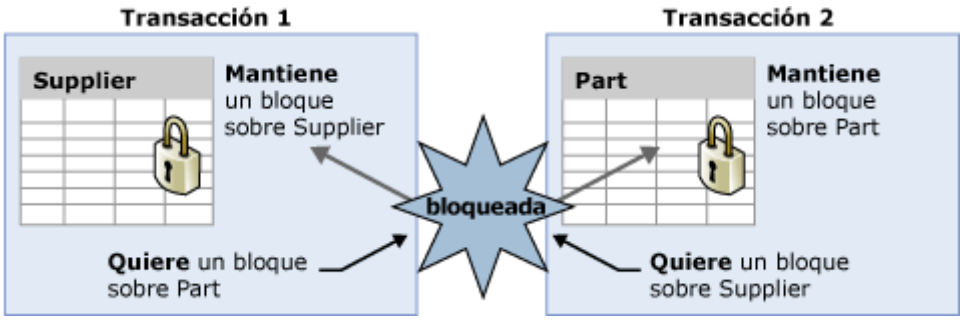
por segunda vez, leerá el mismo conjunto de filas. Los bloqueos de intervalo se mantienen hasta que la transacción finaliza. Éste es el nivel de aislamiento más restrictivo, porque bloquea intervalos de claves completos y mantiene esos bloqueos hasta que la transacción finaliza. Al ser menor la simultaneidad, sólo se debe utilizar esta opción cuando sea necesario.

Cada nivel de aislamiento ofrece más aislamiento que el anterior porque contiene más bloqueos restrictivos durante periodos más largos, pero existen dos excepciones: SNAPSHOT y READ COMMITTED cuando el valor de READ_COMMITTED_SNAPSHOT es ON. Estos niveles de aislamiento no adquieren bloqueos compartidos en filas de datos durante las operaciones de lectura. Los únicos bloqueos que se mantienen en las tablas son los bloqueos SCH-S.

Interbloqueos (DeadLocks)

Un interbloqueo se produce cuando dos o más tareas se bloquean entre sí permanentemente teniendo cada tarea un bloqueo en un recurso que las otras tareas intentan bloquear. Por ejemplo:

- La transacción A tiene un bloqueo compartido de la fila 1.
- La transacción B tiene un bloqueo compartido de la fila 2.
- La transacción A ahora solicita un bloqueo exclusivo de la fila 2 y se bloquea hasta que la transacción B finalice y libere el bloqueo compartido que tiene de la fila 2.
- La transacción B ahora solicita un bloqueo exclusivo de la fila 1 y se bloquea hasta que la transacción A finalice y libere el bloqueo compartido que tiene de la fila 1.
- La transacción A no puede completarse hasta que se complete la transacción B, pero la transacción B está bloqueada por la transacción A. Esta condición también se llama dependencia cíclica: la transacción A tiene una dependencia de la transacción B y la transacción B cierra el círculo teniendo una dependencia de la transacción A.

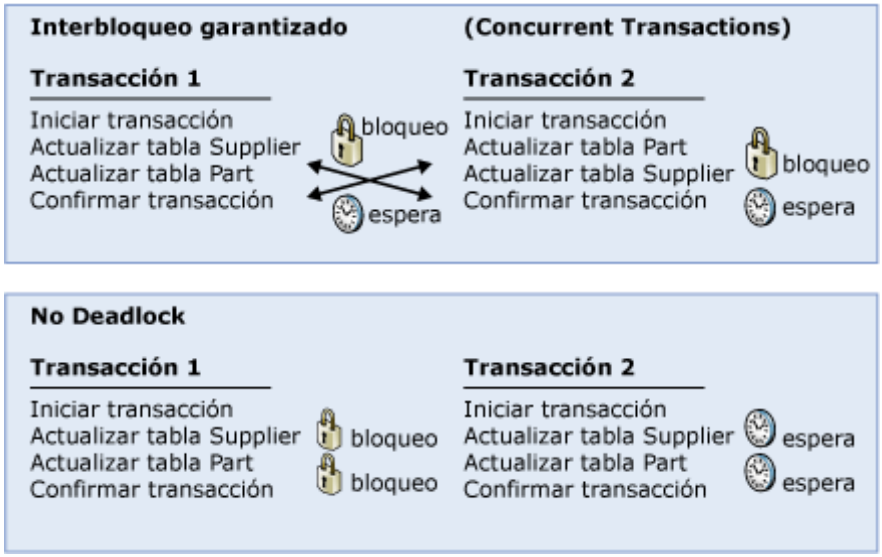


Ambas transacciones con un interbloqueo esperarán para siempre, a no ser que un proceso externo rompa el interbloqueo. La supervisión de interbloqueos del Motor de base de datos de SQL Server de Microsoft comprueba periódicamente si hay tareas con un interbloqueo. Si el monitor detecta una dependencia cíclica, selecciona una de las tareas como el sujeto y finaliza su transacción con un error. Esto permite a la otra tarea completar su transacción. La aplicación con la transacción que terminó con un error puede reintentar la transacción, que suele completarse después de que la otra transacción interbloqueada haya finalizado.

El uso de ciertas convenciones de código en las aplicaciones reduce la probabilidad de que las aplicaciones causen interbloqueos.

- Obtenga acceso a los objetos en el mismo orden.

- Evite la interacción con los usuarios en las transacciones.
- Mantenga transacciones cortas y en un proceso por lotes.
- Utilice un nivel de aislamiento inferior.
- Utilice un nivel de aislamiento basado en las versiones de las filas.
  - Establezca la opción de base de datos READ_COMMITTED_SNAPSHOT en ON para que las transacciones de lectura confirmada utilicen las versiones de filas.
  - Utilice el aislamiento de instantánea.



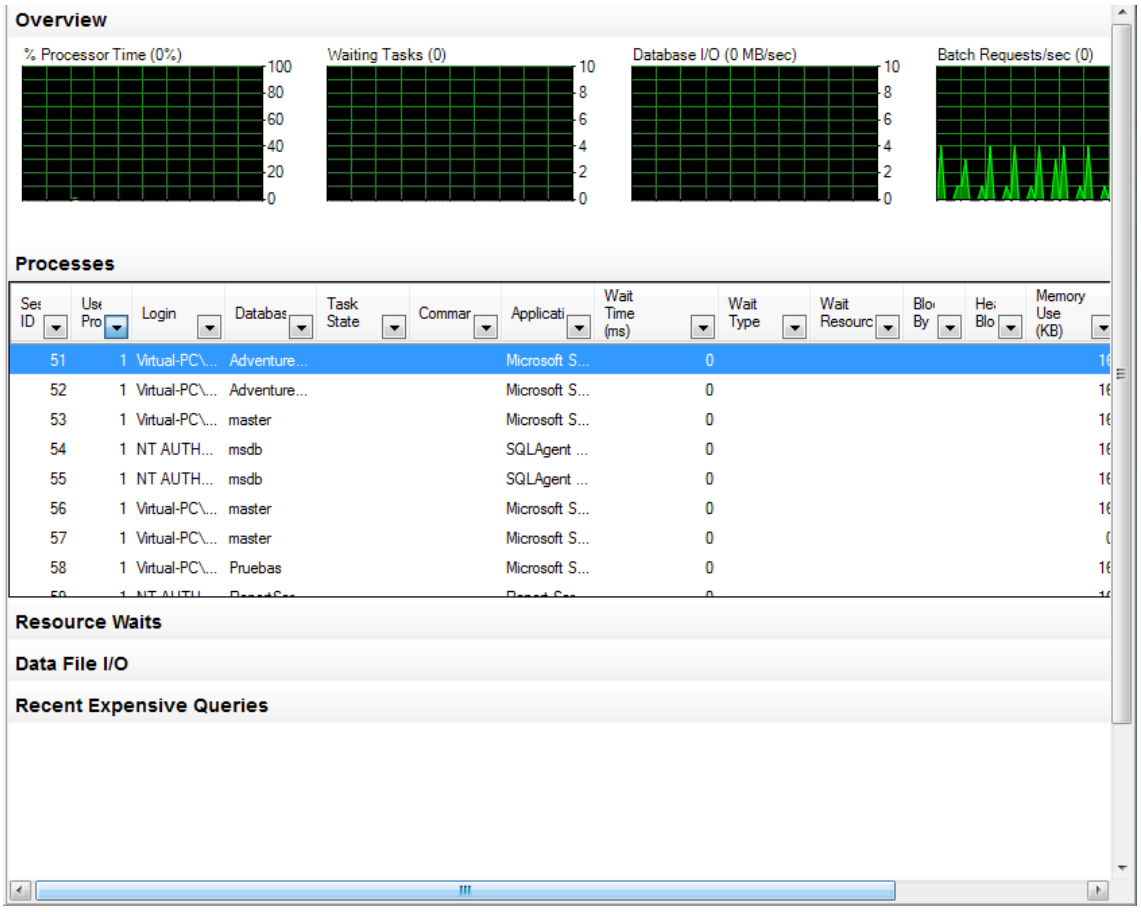
A menudo se confunden los interbloqueos con los bloqueos normales. Cuando una transacción solicita un bloqueo en un recurso bloqueado por otra transacción, la transacción solicitante espera hasta que se libere el bloqueo. De forma predeterminada, las transacciones de SQL Server no tienen tiempo de espera, a menos que se establezca **LOCK_TIMEOUT**. La transacción solicitante está bloqueada, no interbloqueada, porque la transacción solicitante no ha hecho nada para bloquear la transacción a la que pertenece el bloqueo. Finalmente, la transacción a la que pertenece el bloqueo se completará y liberará el bloqueo, y a la transacción solicitante se le concederá el bloqueo y continuará.

Información sobre bloqueos

- Vistas de catálogo:
  - **sys.dm_tran_locks**: Devuelve información acerca de los recursos del administrador de bloqueos activos actualmente. Cada fila representa una solicitud activa al administrador de bloqueos sobre un bloqueo que se ha concedido o está esperando a ser concedido.
  - **sys.dm_tran_database_transactions**: Devuelve información sobre transacciones en el nivel de base de datos.
  - **sys.dm_tran_active_transactions**: Devuelve información sobre transacciones de la instancia de SQL Server.
- Procedimiento almacenado:
  - **sp_who*:*** Proporciona información acerca de los usuarios, sesiones y procesos actuales
  - **sp_lock*:*** Genera información acerca de los bloqueos.

Monitor de Actividades

Utilice esta página para ver las propiedades de los procesos actuales. Utilice el Monitor de actividad al solucionar problemas de bloqueo y para terminar un proceso en interbloqueo o que no responde.



Módulo 13

Código Administrado en SQL Server

BUFFA SISTEMAS Av de Mayo 676 Cap Fed Tel // (5411) 5273-4400

1- Introducción

Introducción a la integración del CLR y SQL Server
Common Language Runtime (CLR) es el núcleo de Microsoft .NET Framework y proporciona el entorno de ejecución de todo el código de .NET Framework. El código que se ejecuta en CLR se conoce como **código administrado**. El CLR proporciona diversas funciones y servicios necesarios para la ejecución de los programas, como compilación just-in-time (JIT), asignación y administración de memoria, aplicación de la seguridad de tipos, control de excepciones, administración de subprocesos y seguridad.
Con el CLR hospedado en Microsoft SQL Server (lo que se denomina integración con CLR), puede crear procedimientos almacenados, desencadenadores, funciones definidas por el usuario, tipos definidos por el usuario y agregados definidos por el usuario en código administrado. Como el código administrado se compila a código nativo antes de su ejecución, en algunas situaciones puede conseguir aumentos significativos del rendimiento.
El código administrado utiliza seguridad de acceso del código (CAS), vínculos a código y dominios de aplicación para impedir que los ensamblados realicen determinadas operaciones.

Características del CLR

- **Soporte de Clases de .Net Framework**: Integra la ejecución con la biblioteca de clases de .Net Framework (BCL)
- **SubProcesos (Threads)**: Provee clases e interfaces que habilitan la programación para múltiples subprocesos de memoria.
- **COM**: Interoperabilidad con objetos COM (Component Object Model)
- **Control de Tipo de Datos**: Todos los tipos de datos son seguros. No permite conversiones de datos inseguras ni variables no inicializadas.
- **Control de Excepciones**: Provee un manejo estructurado de errores.
- **Motor de Seguridad**: Provee seguridad basada en evidencia, en la identidad del usuario y en el origen del código
- **Motor de Depuración (Debug)**: Permite depurar las aplicaciones
- **MSIL a Código Nativo**: Convierte código MSIL a código nativo durante la ejecución en un proceso llamado JIT Compilation (Just in Time Compilation)
- **Control de Código**: Controla el código durante la ejecución
- **Garbage Collector**: Controla la vida de los objetos en memoria
- **Cargador de clases**: Carga las clases en memoria, maneja la metadata

Cuando un componente se está ejecutando, el CLR es el responsable de asignar y controlar la memoria, empezar y terminar los subprocesos de ejecución, reforzar las políticas de seguridad y resolver las dependencias que un componente tiene con otros componentes
Durante el desarrollo, el CLR asegura que el código sea correcto y que los tipos de datos sean seguros. También reduce la cantidad de código que un desarrollador tiene que escribir para transformar lógica de negocios en componentes.

Ventajas del Código Administrado

- **Lenguaje de Programación**: Se puede crear código administrado en varios lenguajes .Net entre ellos Visual Basic y Microsoft Visual C#®. Esta flexibilidad en la elección del lenguaje habilita la posibilidad de elegir el lenguaje más apropiado teniendo en cuenta los requerimientos del negocio y/o las habilidades y conocimientos del programador.
- **Seguridad de Tipos de Datos**: La biblioteca de clases de .Net Framework maneja tipos de datos seguros en todos sus lenguajes. Esto asegura que el acceso de los datos en memoria es seguro permitiendo leer solo datos que han sido previamente escritos.

- **Seguridad de Aplicaciones**: Provee seguridad para aplicaciones con código administrado. Se puede usar seguridad de acceso a código (limita las acciones que la aplicación puede ejecutar) o seguridad basada en roles (seguridad aplicada a grupos de usuarios similar a los roles de SQL Server).
- **Desarrollo rápido**: Provee una biblioteca de clases muy extensa que reducen el esfuerzo de los programadores para codificar una aplicación.
- **Interoperabilidad**: Puede interoperar con componentes escritos en código no administrado como COM o APIs (Windows application programming interfaces)

Integración del CLR con SQL Server 2008
La integración del CLR con SQL Server 2008 permite ejecutar código administrado directamente dentro de la base de datos. En su forma más básica, el concepto es similar al llamado de procedimientos almacenados extendidos de las versiones anteriores de SQL Server. Sin embargo, el CLR provee un entorno mucho más poderoso y flexible que los procedimientos almacenados extendidos tradicionales.

Un **ensamblado (assembly)** es una unidad de distribución de código administrado. Existen dos tipos de ensamblados dentro de .Net Framework: .**dll** (trabajan dentro del proceso de memoria de la aplicación que las invoca) y .**exe** (trabajan en un proceso de memoria independiente). SQL Server 2008 puede trabajar solo con ensamblados de tipo dll.

Esta integración permite crear procedimientos almacenados, tipos de datos de usuario, funciones y desencadenadores usando los lenguajes de .Net. Estos objetos de código administrado se usan como si fueran cualquiera de los otros objetos de la base de datos.
Esta integración provee muchos beneficios cuando se está desarrollando una aplicación de base de datos: Estos son:

- Los lenguajes .Net son más ricos y ofrecen más funcionalidades de desarrollo que el lenguaje Transact-SQL.
- El código administrado ejecuta dentro del entorno del CLR. Esto provee más seguridad que los procedimientos almacenados extendidos
- Se usa la misma herramienta para desarrollar los objetos de la base de datos que para el resto de la aplicación
- El código administrado tiene mejor performance que el código Transact-SQL

Código Administrado vs. Transact-SQL
Ahora que es posible crear objetos usando Transact-SQL o código administrado será necesario considerar cada escenario antes de decidir cuál de las dos opciones elegir.

Elija preferentemente código administrado:

- Cuando se requiere una programación compleja para completar una tarea. Los lenguajes .Net tienen características poderosas como orientación a objetos, manejo estructurado de excepciones, construcciones avanzadas de condiciones, arreglos, colecciones, etc.
- Se necesita usar la biblioteca de clases de .Net Framework. Estas contienen clases que permiten acceder a Servicios Web, trabajar con el sistema de archivos y otras tareas que son difíciles o imposibles en Transact-SQL
- Se requiere funcionalidad que necesita de un uso intenso del procesador.

Elija preferentemente Transact-SQL cuando:

- Cuando básicamente el código necesite acceso a los datos con nada o muy poca lógica procedural

2- Importar y configurar ensamblados

Ensamblado (Assembly)
Una ensamblado es un bloque construido de una aplicación .Net Framework, forma una unidad de distribución que maneja control de versión, reusabilidad, manejo de permisos y ámbito de activación. Es una colección de tipos y recursos que están construidos para trabajar juntos y forman una unidad lógica de funcionalidad. Un ensamblado provee al CLR con información que necesita para conocer la estructura e implementación de los tipos incluidos. Pueden ser de dos tipos:

- Archivo .exe (trabaja fuera del proceso)
- Archivo .dll (trabaja dentro del proceso)

SQL Server 2008 solo soporta los ensamblados de tipo dll a través de SQLCLR. Esto provee muy buena performance ya que la ejecución del código administrado se realiza dentro del ejecutable de SQL Server.

Para trabajar con ensamblados .Net en SQL Server se debe:

- Crear el código dentro de un proyecto .Net
- Compilar y generar el ensamblado
- Importar el ensamblado dentro de SQL Server

Habilitar la integración con CLR
La característica de integración de Common Language Runtime (CLR) está desactivada de forma predeterminada en Microsoft SQL Server y se debe habilitar para utilizar los objetos que se implementan con la integración CLR. Para habilitar la integración con CLR mediante Transact-SQL, utilice la opción **clr enabled** del procedimiento almacenado **sp_configure**

```
sp_configure 'clr enabled', 1
GO
RECONFIGURE
GO
```

Como importar un ensamblado
La instrucción **CREATE ASSEMBLY** permite crear un módulo de aplicación administrada (assembly o ensamblado) que contiene metadatos de clase y código administrado como un objeto en una instancia de SQL Server. Mediante este módulo, puede crear en la base de datos funciones CLR, procedimientos almacenados, desencadenadores, funciones de agregado definidas por el usuario y tipos definidos por el usuario.
Si desea modificarlo usar **ALTER ASSEMBLY**. ALTER ASSEMBLY lo actualiza a la última copia de los módulos de Microsoft .NET Framework que conservan su implementación y agrega o quita los archivos asociados con él.
Para quitar el ensamblado usar la sentencia **DROP ASSEMBLY**. DROP ASSEMBLY devuelve un error si otro ensamblado que existe en la base de datos hace referencia al ensamblado o si se utiliza en procedimientos, desencadenadores, tipos definidos por el usuario, agregados o funciones de CLR en la base de datos actual. Se deberán quitar todos los objetos de la base de datos referenciados a este módulo antes de quitar el ensamblado.

```
CREATE ASSEMBLY [nombre_assembly]
[ AUTHORIZATION nombre_dueño]
FROM { ruta_archivo_assembly | assembly_bits [ ,...n ] }
[ WITH PERMISSION_SET = { SAFE | EXTERNAL_ACCESS | UNSAFE } ]
```

Argumentos

- **Nombre_assembly**: Es el nombre del ensamblado. El nombre debe ser único en la base de datos y un identificador válido.
- **AUTHORIZATION**: Especifica el nombre de un usuario o función como propietario del ensamblado. El valor de nombre_dueño debe ser el nombre de una función de la que el usuario actual es miembro o el usuario actual debe tener el permiso IMPERSONATE sobre nombre_dueño. Si no se especifica, la propiedad se otorga al usuario actual.
- **FROM < ruta_archivo_assembly>:** Especifica la ruta de acceso local o ubicación de red en la que se encuentra el ensamblado que se carga y, además, el nombre de archivo de manifiesto que se corresponde con el ensamblado. No admite la carga de ensamblados de varios módulos. SQL Server también busca cualquier ensamblado dependiente de este ensamblado en la misma ubicación y lo carga con el mismo propietario que el ensamblado de nivel raíz. Si estos ensamblados dependientes no se encuentran o aún no están cargados en la base de datos actual, CREATE ASSEMBLY produce un error. Si los ensamblados dependientes ya están cargados en la base de datos actual, el propietario de los mismos deberá ser el mismo que el del ensamblado nuevo que se crea.
- **FROM <assembly_bits>**: Es la lista de valores binarios que forman el ensamblado y sus ensamblados dependientes. El primer valor de esta lista se considera el ensamblado raíz. Los valores correspondientes a los ensamblados dependientes pueden suministrarse en cualquier orden. Se ignora cualquier valor que no se corresponda con los ensamblados dependientes del ensamblado raíz.
- **PERMISSION_SET {SAFE | EXTERNAL_ACCESS | UNSAFE}:** Especifica un conjunto de permisos de acceso a código que se conceden al ensamblado cuando SQL Server tiene acceso al mismo. **SAFE** es el valor predeterminado y el más recomendable. Es el conjunto de permisos más restrictivo. El código que ejecuta un ensamblado con permisos **SAFE** no puede tener acceso a recursos externos del sistema, como archivos, la red, variables de entorno o el registro. **EXTERNAL_ACCESS** permite que los ensamblados tengan acceso a determinados recursos externos del sistema, como archivos, redes, variables de entorno y el registro. **UNSAFE** permite que los ensamblados tengan acceso ilimitado a los recursos, tanto dentro como fuera de una instancia de SQL Server. El código que se ejecuta desde dentro de un ensamblado **UNSAFE** puede llamarse código no administrado.

Ejemplo:

```
CREATE ASSEMBLY Contacts
FROM 'C:\ContactsApp\Contacts.dll'
```

Nota de Seguridad: **SAFE** es el ajuste de permiso recomendado para ensamblados que realizan tareas de administración de datos y cálculos sin tener acceso a los recursos fuera de una instancia de SQL Server.
Es recomendable el uso de **EXTERNAL_ACCESS** para ensamblados que tengan acceso a recursos fuera de una instancia de SQL Server. Los ensamblados EXTERNAL_ACCESS incluyen la protección de escalabilidad y confiabilidad ofrecida por los ensamblados SAFE; sin embargo, desde el punto de vista de la seguridad, son similares a los ensamblados UNSAFE. Esto se debe a que el código en los ensamblados EXTERNAL_ACCESS se ejecuta de forma predeterminada con la cuenta del servicio SQL Server y tiene acceso a los recursos externos con esa misma cuenta, a no ser que el código suplante al autor de la llamada de forma explícita.
Si especifica **UNSAFE**, otorga libertad absoluta al código en el ensamblado para realizar operaciones en el espacio de procesos de SQL Server que pueden comprometer la solidez de

SQL Server. Los ensamblados UNSAFE también pueden trastornar potencialmente el sistema de seguridad de SQL Server o de Common Language Runtime.
Si se especifica EXTERNAL_ACCESS o UNSAFE, el inicio de sesión de SQL Server debe tener el permiso **EXTERNAL ACCESS ASSEMBLY** en el servidor. Si se especifica UNSAFE, es necesaria la pertenencia a la función fija de servidor **sysadmin**.

Ejemplo:

```
Use master
GO
CREATE ASYMMETRIC KEY AdventureWorks_Login
FROM EXECUTABLE FILE = 'C:\ContactsApp\Contacts.dll'

CREATE LOGIN AWorksCLR
FROM ASYMMETRIC KEY AdventureWorks_Login
GRANT  EXTERNAL ACCESS ASSEMBLY TO AWorksCLR

USE AdventureWorks2008
GO
CREATE ASSEMBLY Contacts
FROM 'C:\ContactsApp\Contacts.dll'
WITH PERMISSION_SET = EXTERNAL_ACCESS
```

3- Creación de objetos administrados en la base de datos

Conceptos de los ensamblados

- **Espacio de nombres (NameSpaces)**: Es un grupo lógico de clase, estructuras, enumeraciones y otros ítems del código. Permite al programador a organizar ítems relacionados. Permite también duplicar los nombres de los ítems dentro de diferentes espacios de nombres.
- **Clase**: Contiene una serie de variables, propiedades (atributos), métodos (operaciones que puede ejecutar) y eventos (situaciones que puede reconocer). Es una plantilla para objetos.
- **Objeto**: En ejecución un objeto es creado o instanciado basado en una clase especifica. Es el objeto el que existe en memoria y ejecuta las acciones.
- **Método**: Operaciones que puede ejecutar.

SQL Server 2008 permite relacionar objetos de la base de datos a código administrado de la siguiente manera:

- **Clase**: Funciones de agregado y tipos de datos de usuario
- **Método**: Procedimientos almacenados, funciones y desencadenadores

Procedimientos almacenados, Funciones y Desencadenadores

Para usar código administrado en la base de datos se necesita crear objetos que llamen a esos códigos. Una vez registrado el ensamblado usando CREATE ASSEMBLY, se pueden crear códigos para procedimientos almacenados, funciones y desencadenadores y relacionarlos con métodos del ensamblado.

Para ello se usan las instrucciones **CREATE PROCEDURE**, **CREATE TRIGGER** o **CREATE FUNCTION** según el objeto a crear. La sintaxis de dichas sentencias es la misma que para los objetos estándar excepto que la cláusula **AS** cambia. Generalmente en AS es donde se especifican las sentencias Transact-SQL. Es estos casos se deberá usar la cláusula **EXTERNAL NAME** para especificar el nombre del ensamblado, el nombre del espacio de nombres (optativo), el nombre de la clase y el nombre del método. La sintaxis de esta cláusula es:

EXTERNAL NAME NombreEnsamblado.[[Espacio de nombre].Clase].Método

Ejemplo:

```
CREATE PROCEDURE Person.UpdatePhoneList AS
EXTERNAL NAME Contacts.[Contacts.PhoneList].SaveList
```

En este ejemplo Contacts es el nombre del ensamblado que fue previamente registrado usando CREATE ASSEMBLY. Este ensamblado contiene una clase llamada PhoneList dentro del espacio de nombres Contacts. Esta clase contiene un método llamado SaveList. Cada vez que se invoca al procedimiento almacenado UpdatePhoneList se genera una llamada a este método.

Si el método tuviera parámetros estos pueden ser pasados por el procedimiento almacenado o función.

Funciones de Agregado y Tipos de Datos de Usuario

Al igual que los procedimientos almacenados, funciones y desencadenadores este tipo de objetos se crean igual que los objetos estándar cambiando la cláusula AS. En este caso se deben usar las instrucciones **CREATE AGREGATE** y **CREATE TYPE** y estás se relacionaran con clases en vez de generar llamadas a métodos.

EXTERNAL NAME NAME NombreEnsamblado.[[Espacio de nombre].Clase]

Ejemplo:

```
CREATE AGGREGATE Concatenate(@input nvarchar(4000))
RETURNS nvarchar(4000)
EXTERNAL NAME Utilities.[Utilities.Concatenate]
```

Ejecución:

```
SELECT AccountNumber, dbo.Concatenate(SalesOrderNumber) Orders
FROM Sales.SalesOrderHeader
GROUP BY AccountNumber
```