



# SQL server 2005

Becas  
Control+f

**Microsoft®**



## Índice:

<b>Módulo 1 - Introducción a Bases de Datos</b>	<b>- 6 -</b>
<b>1- Conceptos de Bases de Datos</b>	<b>- 7 -</b>
Definición	- 7 -
Arquitectura Cliente/Servidor	- 7 -
Modelos	- 7 -
Sistema de gestión de base de datos (SGBD)	- 8 -
Bases de Datos OLTP (On Line Transactional Processing)	- 9 -
Bases de Datos OLAP (On Line Analytical Processing)	- 9 -
Objetos de la base de datos	- 9 -
Transacciones	- 10 -
<b>2- Diseño de Base de Datos</b>	<b>- 11 -</b>
El proceso de diseño	- 11 -
Normalización	- 11 -
Desnormalización	- 12 -
<b>Módulo 2 - Introducción al Lenguaje SQL</b>	<b>- 13 -</b>
<b>1- Introducción</b>	<b>- 14 -</b>
Transact-SQL	- 14 -
<b>2- Lenguaje</b>	<b>- 15 -</b>
Variables	- 15 -
Operadores	- 15 -
Comentarios	- 15 -
Control de Flujo	- 15 -
<b>3- Instrucción SELECT</b>	<b>- 17 -</b>
Definición	- 17 -
Descripción de las cláusulas	- 17 -
Cláusula JOIN	- 19 -
<b>4- Instrucción INSERT</b>	<b>- 22 -</b>
Definición	- 22 -
Descripción de las cláusulas	- 22 -
Consideraciones	- 22 -
<b>5- Instrucción UPDATE</b>	<b>- 24 -</b>
Definición	- 24 -
Descripción de las cláusulas	- 24 -
<b>6- Instrucción DELETE</b>	<b>- 26 -</b>
Definición	- 26 -
Descripción de las cláusulas	- 26 -
<b>Módulo 3 - Introducción a SQL Server 2005</b>	<b>- 27 -</b>
<b>1- Componentes</b>	<b>- 28 -</b>
Motor de la base de Datos	- 28 -
Analysis Service	- 28 -
SQL Server Integration Service (SSIS)	- 28 -
Notification Services	- 28 -
Full-Text Search	- 28 -
Relational Database Engine .NET CLR	- 29 -

Reporting Service	- 29 -
Replicación	- 29 -
Native HTTP Support	- 29 -
Service Broker	- 29 -
<b>2- Algunas Novedades</b>	<b>- 30 -</b>
Memoria Dinámica AWE	- 30 -
Nuevos y mejorados tipos de datos	- 30 -
Mayor tamaño de los registros	- 30 -
Tablas e Índices con particiones	- 30 -
Operación de indexado on line	- 30 -
Snapshot Isolation Level	- 30 -
Mirroring	- 30 -
SQLiMail	- 31 -
Cifrado nativo	- 31 -
Desencadenadores DDL	- 31 -
SQL Server Management Studio	- 31 -
<b>Módulo 4 - Creación de Bases de Datos</b>	<b>- 33 -</b>
<b>1- Bases de Datos</b>	<b>- 34 -</b>
Introducción	- 34 -
Diseño	- 34 -
Bases de Datos de Sistema	- 34 -
Creación	- 35 -
Argumentos:	- 36 -
Modificación y Borrado	- 36 -
Opciones de Base de Datos	- 36 -
Arquitectura física del registro de transacciones (Transaction Log)	- 38 -
Puntos de comprobación (CheckPoint)	- 39 -
Información sobre Base de datos	- 39 -
SQL Server Management Studio	- 39 -
Vista del Catálogo	- 39 -
Funciones del Sistema	- 40 -
Procedimientos Almacenados de Sistema	- 40 -
<b>2- Grupos de Archivos (FileGroups)</b>	<b>- 41 -</b>
Archivos de base de datos	- 41 -
Grupos de Archivos	- 41 -
Mejoras en el rendimiento	- 42 -
Consideraciones para la creación y organización de grupos de archivos	- 43 -
<b>3- Esquemas (Schemas)</b>	<b>- 44 -</b>
Definición	- 44 -
Ventajas	- 44 -
Resolución de nombres	- 45 -
<b>4- Instantáneas de Base de Datos (Database Snapshot)</b>	<b>- 46 -</b>
Definición	- 46 -
Funcionamiento de las instantáneas de la base de datos	- 46 -
Creación	- 47 -
Lectura sobre una instantánea de la base de datos	- 47 -
Tamaño de la instantánea de la base de datos	- 48 -
<b>Módulo 5 - Creación de Tipos de Datos y Tablas</b>	<b>- 49 -</b>

<b>1- Tipos de Datos</b>	<b>- 50 -</b>
Definición	- 50 -
Tipo de Datos SQL Server	- 50 -
Consideraciones en la elección de tipos de datos	- 51 -
Tipos de Datos de Alias	- 51 -
<b>2- Tablas</b>	<b>- 54 -</b>
Definición	- 54 -
Creación	- 54 -
Atributos de las columnas	- 55 -
Modificación y Borrado	- 55 -
Como se organizan los datos	- 55 -
<b>3- Tablas con Particiones</b>	<b>- 57 -</b>
Definición	- 57 -
Ventajas	- 57 -
Funciones de Partición	- 57 -
Argumentos	- 57 -
Esquemas de particiones	- 58 -
Crear Tabla con Particiones	- 58 -
Que operaciones pueden realizarse sobre una tabla con particiones	- 59 -
<b>Módulo 6 - Usando XML</b>	<b>- 60 -</b>
<b>1- Uso de FOR XML</b>	<b>- 61 -</b>
Cláusula FOR XML	- 61 -
Argumentos	- 61 -
Consultas en Modo RAW	- 62 -
Consultas en Modo AUTO	- 63 -
Consultas en Modo EXPLICIT	- 65 -
Consultas en Modo PATH	- 67 -
XML Anidado	- 68 -
<b>2- Usando OPENXML</b>	<b>- 69 -</b>
Definición	- 69 -
sp_xml_preparedocument	- 69 -
Sintaxis OPENXML	- 69 -
Argumentos	- 69 -
Trabajando con Espacios de Nombre XML (Namespaces)	- 71 -
<b>3- Usando el tipo de dato xml - XQuery</b>	<b>- 73 -</b>
Tipo de dato	- 73 -
XQuery	- 73 -
Sentencias FLWOR	- 73 -
Método Query	- 74 -
Método Value	- 74 -
Método Exist	- 75 -
Métodos Modify	- 75 -
Enlazar datos relacionales dentro de datos XML	- 76 -
Método Nodes	- 76 -



**Microsoft®**



# Módulo 1

## Introducción a Bases de Datos



## 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.

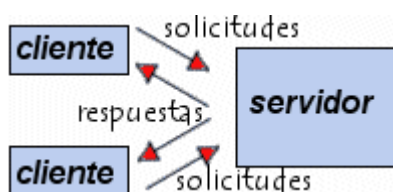
Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulan ese conjunto de datos.

### Arquitectura Cliente/Servidor

Los sistemas cliente/servidor están contruidos 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 terminaron 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 interfases 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: Decida 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: Examine cada tabla y decida 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: Analice el diseño para detectar errores. Cree las tablas y agregue algunos registros con datos de ejemplo. Compruebe si puede obtener los resultados previstos de las tablas. Realice los ajustes necesarios en el diseño.
- Aplicar las reglas de normalización: Aplique reglas de normalización de los datos para comprobar si las tablas están estructuradas correctamente. Realice 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 artículos 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.



# **Módulo 2**

## **Introducción al Lenguaje SQL**



## 1- Introducción

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 ...

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 sublenguajes:

- **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 ordenes 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: /\* \*/

### 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.
- **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	Size	SizeUnit
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00	NULL	NULL
2	829	All-Purpose Bike Stand	ST-1401	0	1	NULL	4	3	59,466	159,00	NULL	NULL
3		AW/C Logo Cap	CA-1098	0	1	Multi	4	3	6,9223	8,99	NULL	NULL
4	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00	0,00	NULL	NULL
5	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00	0,00	NULL	NULL
6	877	Bike Wash - Dissolver	CL-9009	0	1	NULL	4	3	2,9733	7,95	NULL	NULL
7	316	Blade	BL-2036	1	0	NULL	800	600	0,00	0,00	NULL	NULL
8	843	Cable Lock	LO-C100	0	1	NULL	4	3	10,3125	25,00	NULL	NULL
9	952	Chain	CH-0234	0	1	Silver	500	375	8,9866	20,24	NULL	NULL
10	324	Chain Stays	CS-2812	1	0	NULL	1000	750	0,00	0,00	NULL	NULL
11	322	Chaining	CR-7833	0	0	Black	1000	750	0,00	0,00	NULL	NULL
12	320	Chaining Bolts	CB-2903	0	0	Silver	1000	750	0,00	0,00	NULL	NULL
13	321	Chaining Nut	CN-6137	0	0	Silver	1000	750	0,00	0,00	NULL	NULL
14	866	Classic Vest, L	VE-C304-L	0	1	Blue	4	3	23,749	63,50	L	NULL
15	865	Classic Vest, M	VE-C304-M	0	1	Blue	4	3	23,749	63,50	M	NULL
16	864	Classic Vest, S	VE-C304-S	0	1	Blue	4	3	23,749	63,50	S	NULL
17	505	Cone-Shaped Race	RA-7490	0	0	NULL	1000	750	0,00	0,00	NULL	NULL
18	323	Crown Race	CR-9981	0	0	NULL	1000	750	0,00	0,00	NULL	NULL
19	504	Cup-Shaped Race	RA-7245	0	0	NULL	1000	750	0,00	0,00	NULL	NULL

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	AW/C 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	Chaining	CR-7833	0,00
12	Chaining Bolts	CB-2903	0,00
13	Chaining Nut	CN-6137	0,00
14	Classic Vest, L	VE-C304-L	63,50
15	Classic Vest, M	VE-C304-M	63,50
16	Classic Vest, S	VE-C304-S	63,50

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
```



Results Messages			
	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
13	HL Road Frame - Red, 62	FR-R92R-62	1431,50
14	HL Road Front Wheel	FW-R820	330,06
15	HL Road Handlebars	HB-R956	120,27

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
13	46660	117274.345342
14	46981	147390.932828
15	47018	107356.790400

## 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

	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
13	Women's Tights, S	359,952	0,00
14	Women's Tights, S	224,97	0,00
15	Women's Tights, S	89,988	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.EmployeeId, e.ContactId, c.FirstName, c.LastName, e.BirthDate, e.Gender,
a.AddressLine1, a.City, a.PostalCode
FROM HumanResources.Employee e
INNER JOIN HumanResources.EmployeeAddress ea
ON e.EmployeeId=ea.EmployeeId
INNER JOIN Person.Address a ON ea.AddressId=a.AddressId
INNER JOIN Person.Contact c ON e.ContactId=c.ContactId
WHERE e.Gender='M'
ORDER BY c.LastName, c.FirstName
```



Results Messages									
	EmployeeId	ContactId	FirstName	LastName	BirthDate	Gender	AddressLine1	City	PostalCode
1	288	1012	Syed	Abbas	1965-02-11 00:00:00.000	M	7484 Roundtree Drive	Bothell	98011
2	200	1268	Hazem	Abolrous	1967-11-27 00:00:00.000	M	5050 Mt. Wilson Way	Kenmore	98028
3	85	1281	Pilar	Ackerman	1962-10-11 00:00:00.000	M	5407 Cougar Way	Seattle	98104
4	208	1220	Jay	Adams	1966-03-14 00:00:00.000	M	896 Southdale	Monroe	98272
5	117	1255	François	Ajenstat	1965-06-17 00:00:00.000	M	1144 Paradise Ct.	Issaquah	98027
6	23	1173	Greg	Alderson	1960-11-18 00:00:00.000	M	8684 Military East	Bellevue	98004
7	77	1270	Sean	Alexander	1966-04-07 00:00:00.000	M	7985 Center Street	Renton	98055
8	218	1274	Gary	Altman	1961-02-21 00:00:00.000	M	2598 Brook Court	Renton	98055



## 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 'SELECT', e.EmployeeID, c.LastName, sp.SalesYTD  
FROM HumanResources.Employee AS e  
INNER JOIN Sales.SalesPerson AS sp  
ON e.EmployeeID = sp.SalesPersonID  
INNER JOIN Person.Contact AS c  
ON e.ContactID = c.ContactID  
WHERE e.EmployeeID LIKE '2%'  
ORDER BY e.EmployeeID, c.LastName
```



## 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 subselección 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 como 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.SalesPersonID = so.SalesPersonID
```





```
AND so.OrderDate = (SELECT MAX(OrderDate)
FROM Sales.SalesOrderHeader
WHERE SalesPersonID = sp.SalesPersonID);
```





## 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 subconsulta 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.SalesPersonID = sp.SalesPersonID
WHERE sp.SalesYTD > 2500000.00
```



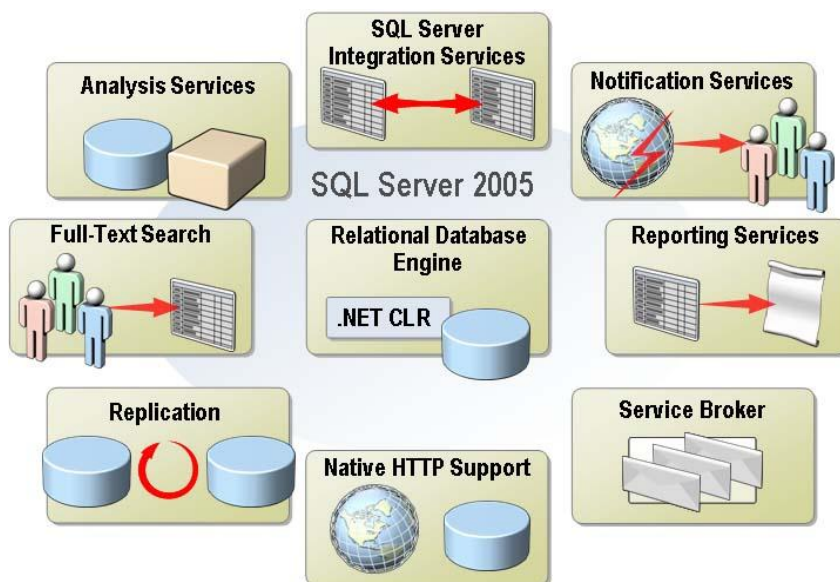


# **Módulo 3**

## **Introducción a SQL Server 2005**



## 1- Componentes



### 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.

### Analysis Service

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 suscripciones 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.



### **Relational Database Engine .NET CLR**

El Lenguaje Común de Tiempos de Ejecución de .NET (CLR) provee un ambiente de administración para código escrito en lenguaje .NET como Visual C# o Visual Basic .NET. El motor de la base de datos SQL Server 2005 tiene alojado el .NET CLR, haciendo posible desarrollar objetos de base de datos usando códigos escritos en un lenguaje de programación .NET. La habilidad de desarrollar objetos de base de datos en código .NET provee varias ventajas, incluyendo un modelo de administración de seguridad (funcionalidad que no esta disponible o es difícil de implementar en Transact-SQL) y es una mejor elección para realizar desarrollos. Puede desarrollar Procedimientos almacenados, tipos de datos, funciones y desencadenadores.

### **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.

### **Native HTTP Support**

Cuando usamos Microsoft Windows Server™ 2003, SQL Server 2005 responde de manera innata a los requerimientos HTTP sin la ayuda de software Web Server como IIS. Esto hace más fácil implementar Servicios Web (Web Services) basados en una base de datos SQL Server.

### **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 2005 una mejor plataforma para soluciones basada en Servicios (service-based).





## 2- Algunas Novedades

### Memoria Dinámica AWE

Soporte AWE permite que aplicaciones 32-bit puedan direccionar memoria física mas allá de los límites de su memoria virtual configurada.

### Nuevos y mejorados tipos de datos

- ***XML***: el tipo de datos xml es usado para almacenar datos XML directamente en la base de datos. Los valores para los datos xml pueden ser opcionalmente asociados con un XML schema
- ***VARCHAR, NVARCHAR, VARBINARY(MAX)***: El máximo de caracteres que puede ser usado para extender el largo de alguno de estos tipos de datos es ahora hasta  $2^{31}$  bytes (aproximadamente 2 GB).

### Mayor tamaño de los registros

En las ediciones previas de SQL Server, el tamaño máximo de un registro de una tabla era 8 KB. Mientras que este es aún el máximo físico de una pagina, el SQL Server 2005 puede mover el reflujo de datos varchar, nvarchar, varbinary, o sql\_variant a otra pagina manteniendo punteros de 24-byte en la pagina original, haciendo posible almacenar una fila que contenga mas de 8 KB de datos sin caer en el uso de los tipos de datos text, ntext o image.

### Tablas e Índices con particiones

Esto permite a una tabla ser esparcida a través de un numero de grupos de archivos físicos (filegroups), mejorando la performance y haciendo mas fácil administrar los procesos archivados basados en rangos de datos. Especial para tablas grandes.

### Operación de indexado on line

Acepta operaciones de INSERT, UPDATE, etc mientras se regeneran los índices, por lo cual se puede estar manteniendo o cambiando los índices on line. También soporta Restore on line en donde solo la parte de la base de datos que se esta levantando no está disponible pero si el resto.

### Snapshot Isolation Level

El SQL Server 2005 incluye una nueva transacción de isolation level llamada snapshot. Snapshot isolation permite a los usuarios leer previamente valores de datos usando una vista consistente transaccionalmente de la base de datos. En este isolation level, los lectores no bloquean otros lectores y escritores que están accediendo al mismo dato. Similarmente, los escritores no bloquean a los lectores. La operación de escritura en el snapshot isolation level esta basada en un mecanismo de bloqueo con detector de conflicto automático. El mecanismo de detección de conflictos evita que un usuario haga cambios basados en datos que otro usuario haya cambiado.

### Mirroring

Permite que el registro de transacciones pase información a un segundo servidor manteniendo ambos sincronizados. Si el primario falla la aplicación puede ser redireccionada al segundo servidor.





### **SQLiMail**

SQLiMail es un nuevo sistema en SQL Server 2005. Usa el Standard Simple Mail Transfer Protocol (SMTP) para enviar un e-mail desde una instancia de SQL Server 2005. Usando SQLiMail, las aplicaciones de base de datos pueden enviar e-mails que contengan resultados de consultas y archivos. SQLiMail esta diseñado para ser un sistema confiable, escalable y seguro para SQL Server.

### **Cifrado nativo**

SQL Server 2005 admite las funciones de cifrado dentro de la propia base de datos, totalmente integradas en una infraestructura de administración clave. De forma predeterminada, se cifran las comunicaciones cliente/servidor. Para centralizar la garantía de seguridad, se puede definir la directiva del servidor para que rechace las comunicaciones no cifradas.

### **Desencadenadores DDL**

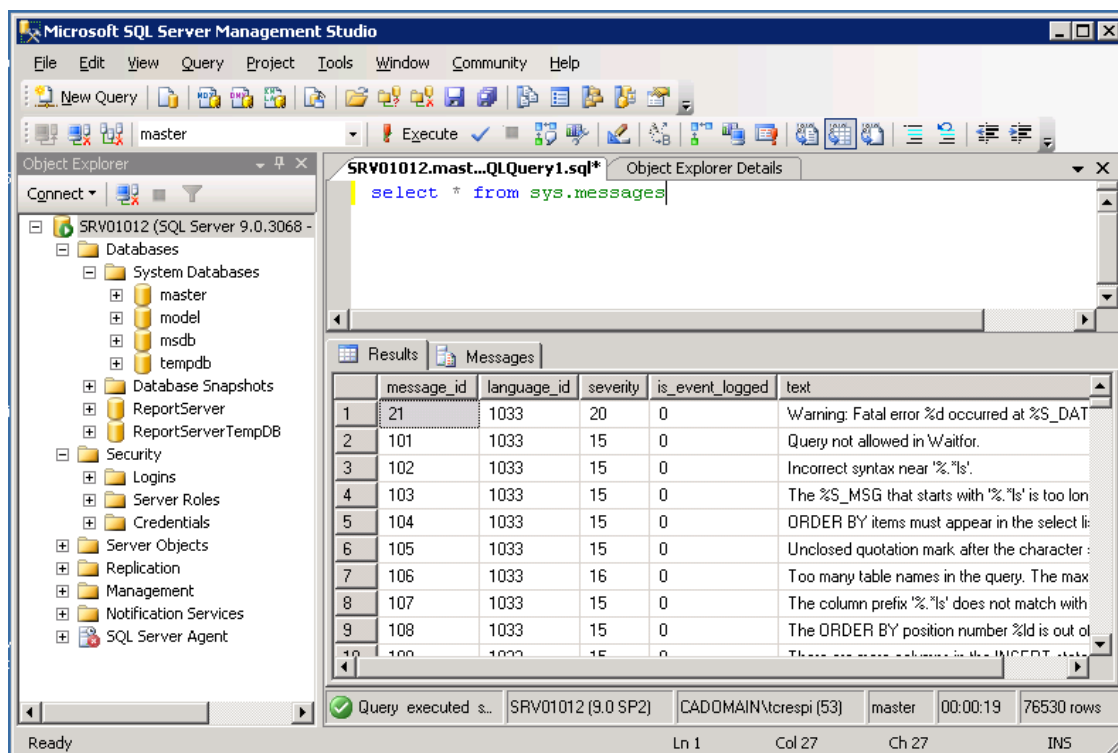
Los desencadenadores DDL, al igual que los estándar, ejecutan procedimientos almacenados como respuesta a un evento. Pero a diferencia de los desencadenadores estándar, no se ejecutan como respuesta a instrucciones UPDATE, INSERT o DELETE en una tabla o vista. En cambio, se ejecutan principalmente como respuesta a instrucciones de lenguaje de definición de datos (o DDL). Entre ellas se incluyen instrucciones CREATE, ALTER, DROP, GRANT, DENY, REVOKE y UPDATE STATISTICS. Algunos procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL también pueden activar desencadenadores DDL.

### **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



## 1- Bases de Datos

### Introducción

Las bases de datos de SQL Server 2005 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 preprocesados,.

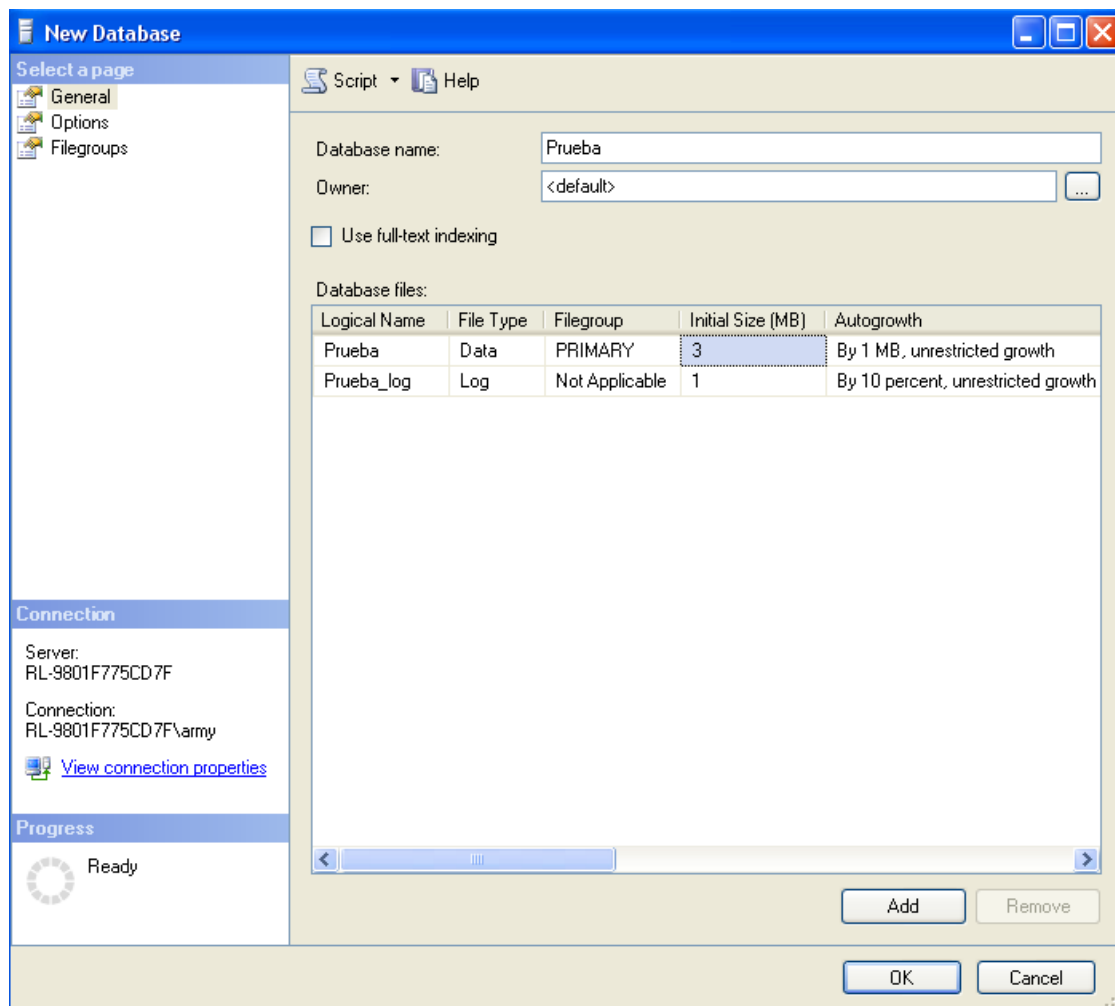
### Bases de Datos de Sistema

- **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. Si no se especifica es de 1MB. 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.

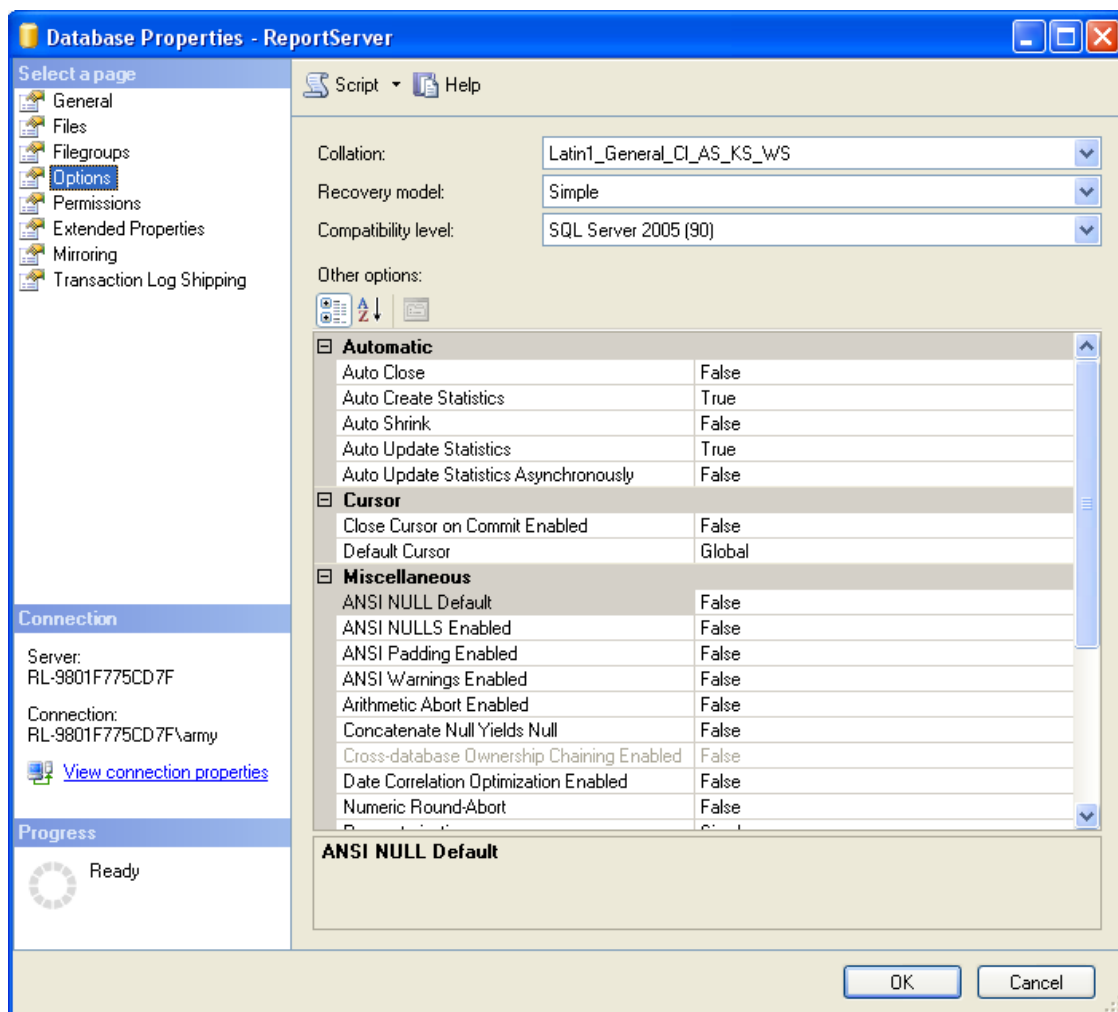
#### Opciones de Base de Datos

Una vez creada la base de datos, se pueden establecer distintos valores para sus opciones usando SQL Server Management Studio o la sentencia Transact-SQL 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.
- **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

### 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.

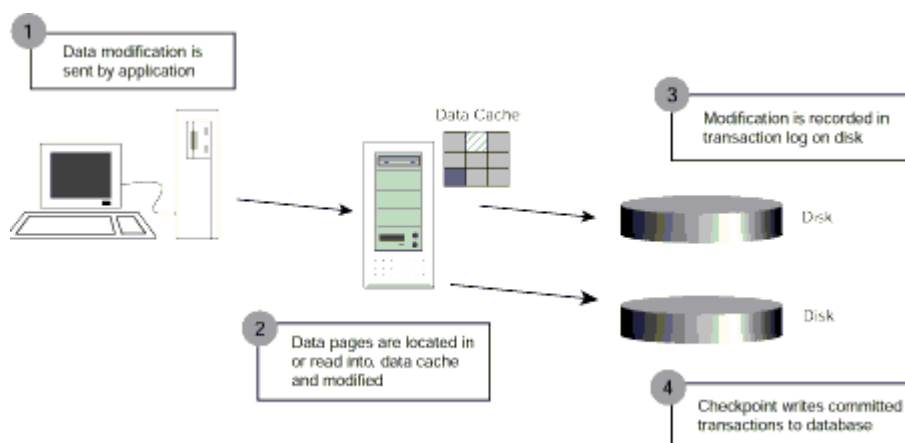
El registro de transacciones de una base de datos está asignado a uno o varios archivos físicos. Conceptualmente, el archivo de registro es una cadena de entradas de registro. El registro de transacciones es un archivo de registro circular. Las nuevas entradas del registro se agregan al final del registro 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.

### 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 ellas 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\_members** 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('Product.Production')



## 2- Grupos de Archivos (FileGroups)

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:

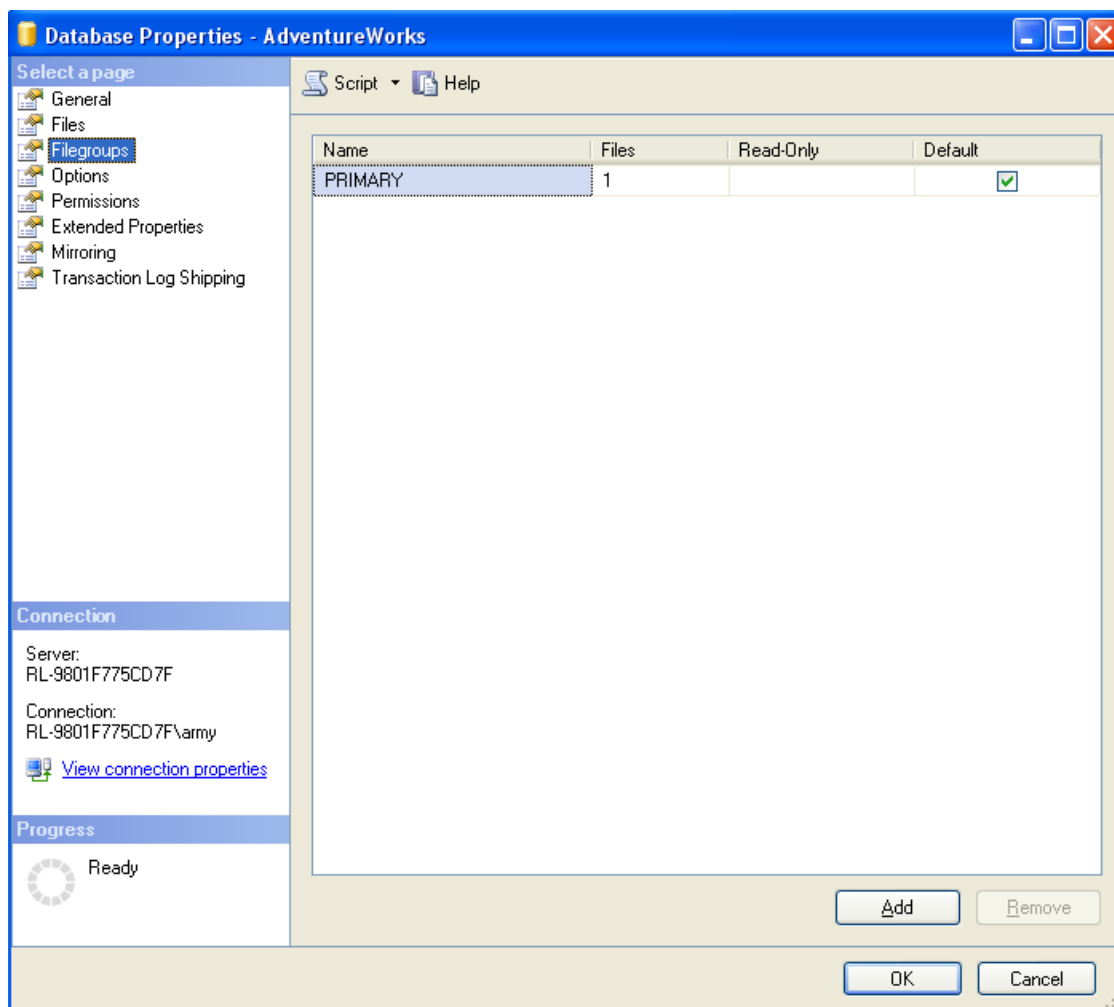
- **Archivos de datos principales:** El archivo de datos principal es el punto de partida de la base de datos y apunta a los otros archivos de la base de datos. Cada base de datos tiene un archivo de datos principal. La extensión recomendada para los nombres de archivos de datos principales es **.mdf**.
- **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**. Los archivos de registro nunca forman parte de un grupo de archivos. Ningún archivo puede pertenecer a más de un grupo de archivos. Las tablas, los índices y los datos de objetos grandes se pueden asociar a un grupo de archivos específico.



Ejemplo:

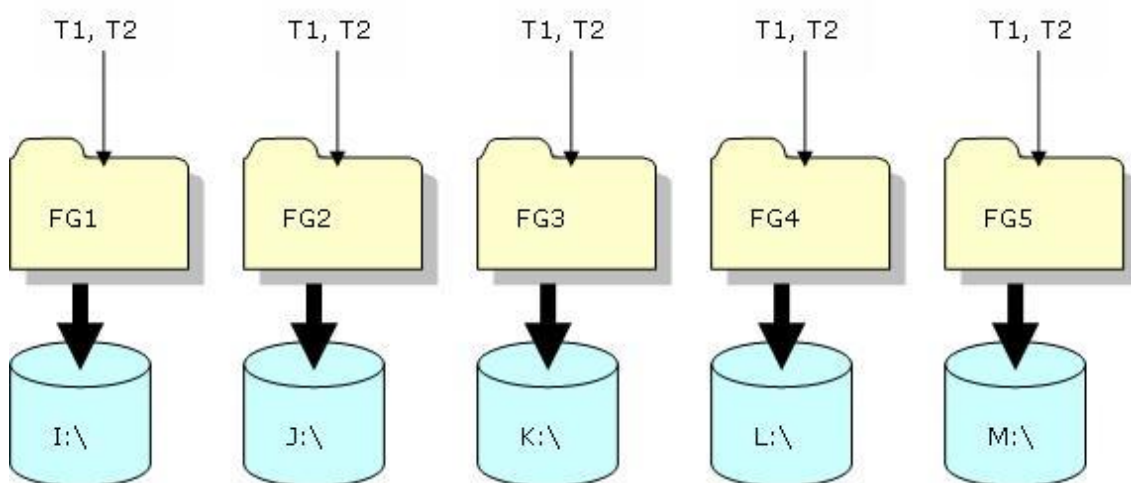
```
CREATE DATABASE [AdventureWorks] ON PRIMARY
( NAME = 'AdventureWorks_Data', FILENAME = 'C:\AdventureWorks_Data.mdf' ),
FILEGROUP [OrderHistoryGroup]
( NAME = N'OrdHist1', FILENAME = N'D:\OrdHist1.ndf' ),
( NAME = N'OrdHist2', FILENAME = N'D:\OrdHist2.ndf' )
LOG ON
( NAME = 'AdventureWorks_log', FILENAME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

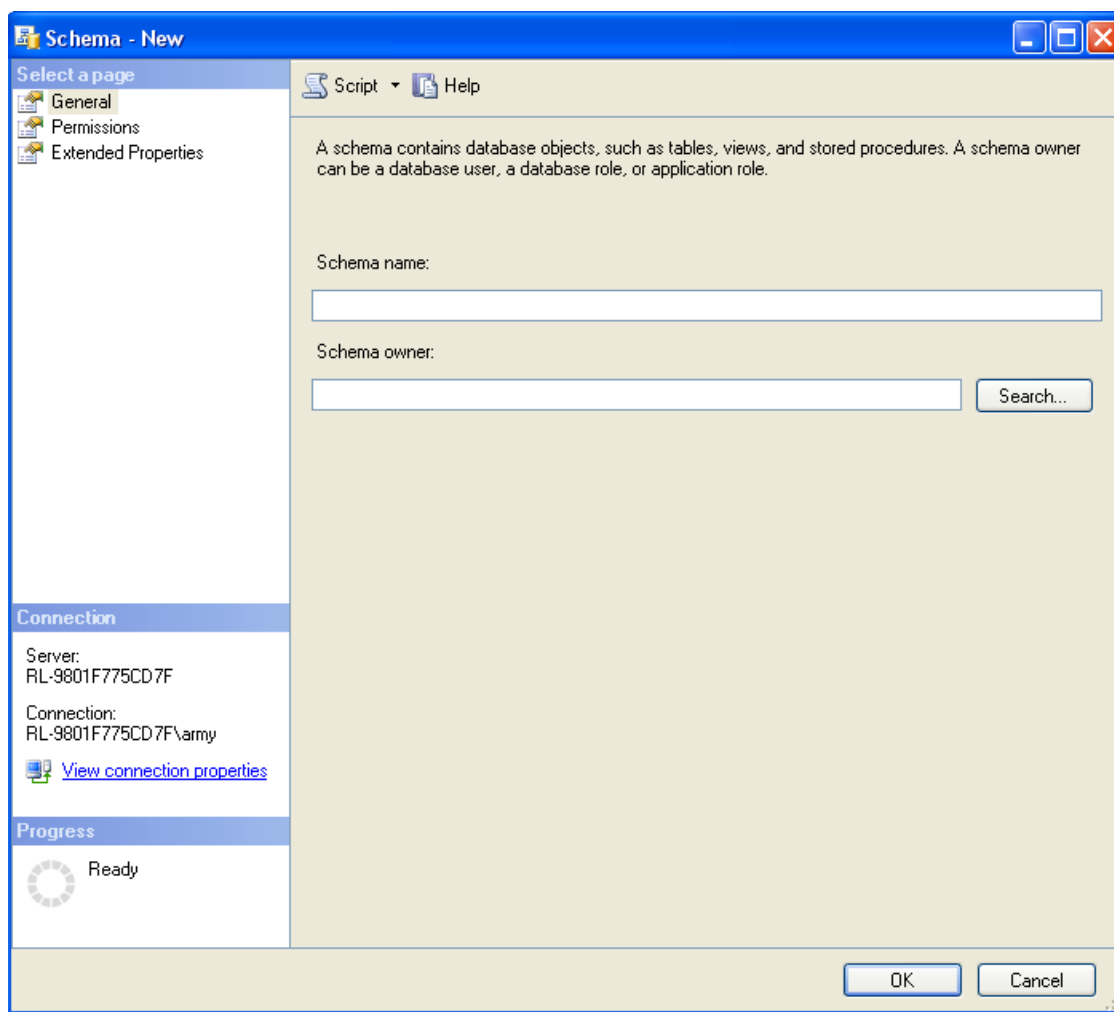
### 3- Esquemas (Schemas)

#### Definición

El comportamiento de los esquemas cambió en 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 nombres de espacio, 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 2005 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 sólo están disponibles en 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

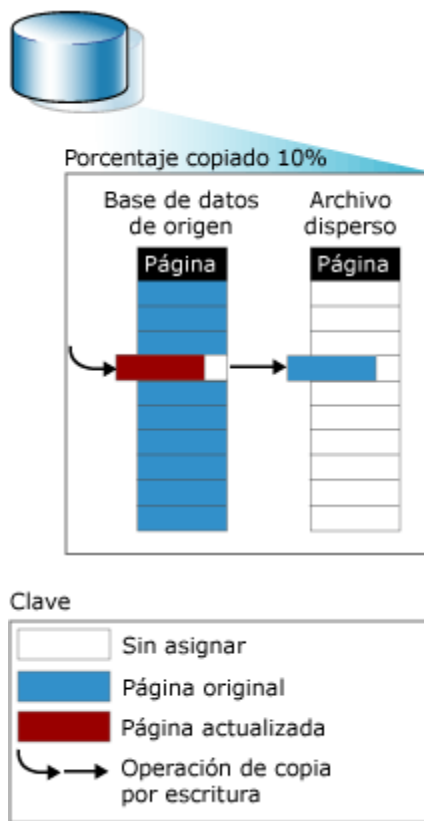
Antes de modificar por primera vez una página de la base de datos de origen, la página original se copia de la base de datos de origen a la instantánea. Este proceso se denomina operación "copiar al escribir". La instantánea almacena la página original y conserva los registros de datos en el estado en 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



## 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	
	Datetime	8	
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	Nuevo. 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.
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.
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.
GUID	uniqueidentifier	16	

XML	Xml	0 – 2GB	Nuevo
Especiales	Bit	1	Tipo de datos entero que puede aceptar los valores 1, 0 o nulo.
	Cursor	0 - 8	
	Timestamp	8	

### 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.

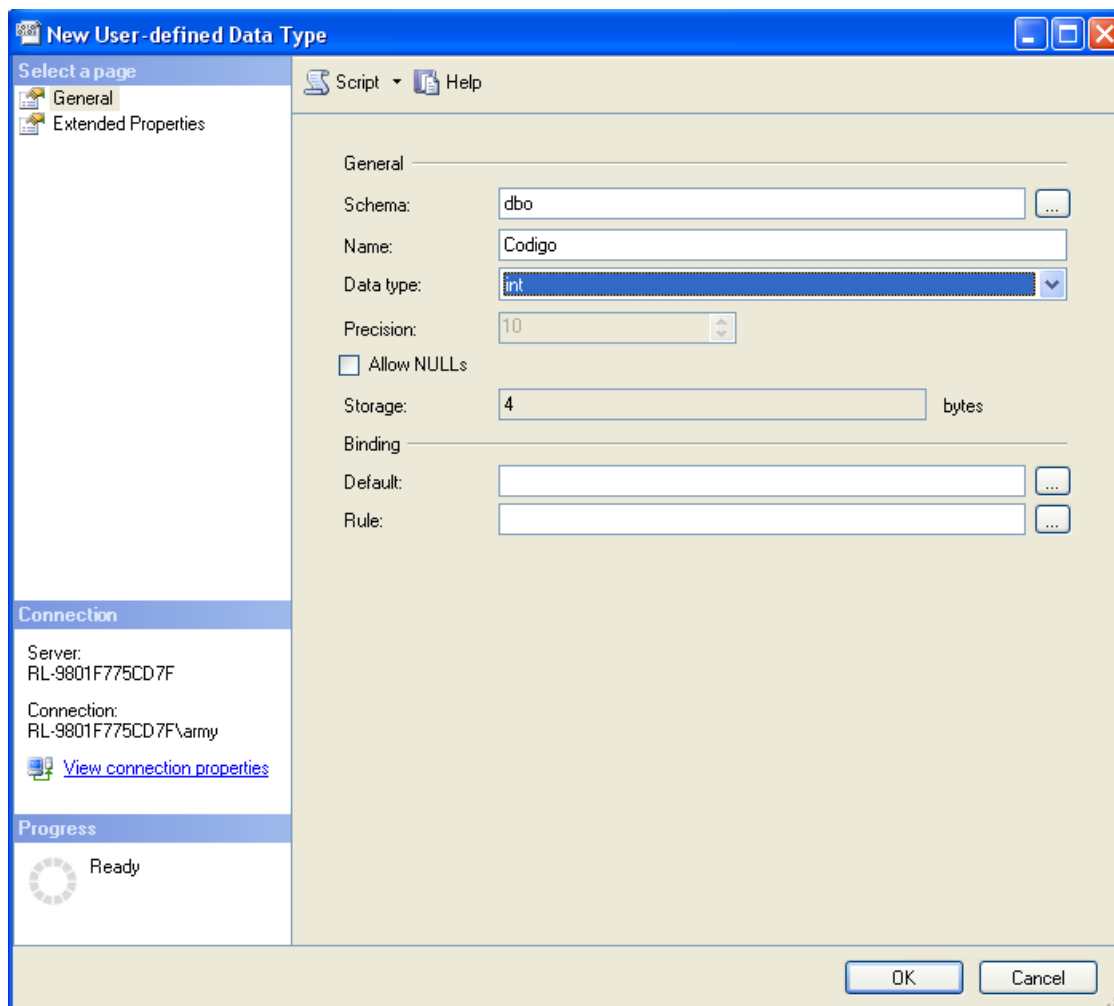
### Tipos de Datos de Alias

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 no se define explícitamente la nulabilidad, se asignará en función de la configuración de los valores ANSI\_NULL\_DEFAULT predeterminada de la base de datos o conexión. 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.



## 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.

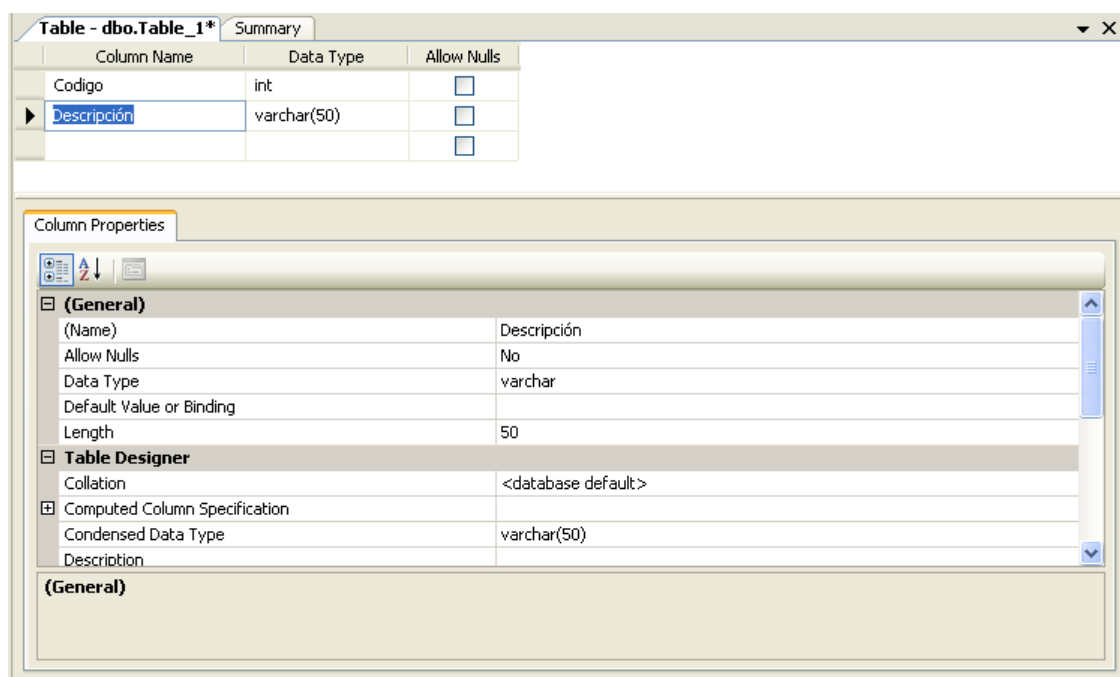
Aunque los nombres de las tablas tienen que ser únicos para cada esquema de una base de datos, puede crear varias tablas con el mismo nombre si especifica distintos esquemas para cada una de ellas.

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.

Creación de tablas en el SQL Server Management Studio:





Ejemplo:

```
CREATE TABLE Ventas.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.
- **Tipo de Dato:** Tipo de dato de la columna
- **Nulabilidad:** Si soporta nulos o no. Si no se especifica se toma el valor predetermina en NULL ANSI.
- **Longitud:** Algunos tipos de datos tienen una longitud predeterminada, en otros hay que ingresarla
- **Collation:** Juego de caracteres aceptados en esta columna. Por defecto toma la asociada a la base de datos. Cada Collation 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.

#### 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 almacenan 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 una 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.

Encabezado	Datos Fijos	NB	VB	Datos Variables
------------	-------------	----	----	-----------------

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 de 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 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.

Este código muestra una función de partición llamada pf\_FechaOrden que arma cuatro particiones. La primera para fechas anteriores a Enero de 2003, la segunda para las fechas comprendidas entre Enero del 2003 y fines de Diciembre del 2003, la tercera para fechas comprendidas entre Enero del 2004 y fines de Diciembre del 2004 y la última para las fechas posteriores a Enero del 2005.

```
CREATE PARTITION FUNCTION pf_FechaOrden (datetime)
AS RANGE RIGHT
FOR VALUES ('01/01/2003', '01/01/2004', '01/01/2005')
```

### 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.

Este 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)
```

Este otro ejemplo asigna todas las particiones al mismo grupo de archivos

```
CREATE PARTITION SCHEME ps_FechaOrden
AS PARTITION pf_FechaOrden
ALL TO ([PRIMARY])
```

### 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
CONSTRAINT OrderDetailsRangeYearCK
CHECK ([OrderDate] >= '20021001'
AND [OrderDate] < '20041001'),
[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 ('01/01/2003')
```

- **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 ('01/01/2006')
```



# Módulo 6

## Usando XML

## 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 2005 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 los 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 2005 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, CustomerType, 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" CustomerType="S" SalesOrderID="43860"/>
<row CustID="1" CustomerType="S" SalesOrderID="44501"/>
<row CustID="1" CustomerType="S" SalesOrderID="45283"/>
.....
```

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

```
SELECT Cust.CustomerID CustID, CustomerType, 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><CustomerType>S</CustomerType><SalesOrderID>43860</Sales
OrderID></row>
<row><CustID>1</CustID><CustomerType>S</CustomerType><SalesOrderID>44501</Sales
OrderID></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, CustomerType, 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" CustomerType="S" SalesOrderID="43860"/>
  <Orden CustID="1" CustomerType="S" SalesOrderID="44501"/>
  <Orden CustID="1" CustomerType="S" 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:

- Cada 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, Cust.CustomerType
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" CustomerType="S">
  <OrderHeader CustomerID="1" SalesOrderID="43860" Status="5"/>
  <OrderHeader CustomerID="1" SalesOrderID="44501" Status="5"/>
  ....
</Cust>
<Cust CustomerID="5" CustomerType="S">
  <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, Cust.CustomerType  
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>  
  <CustomerType>S</CustomerType>  
    <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, Cust.CustomerType  
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>  
    <CustomerType>S</CustomerType>  
      <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.

El modo EXPLICIT transforma en un documento XML el conjunto de filas resultante de la ejecución de la consulta. Para que el modo EXPLICIT pueda generar el documento XML, el conjunto de filas debe ajustarse a un determinado formato. Por ello, es necesario escribir la consulta SELECT para generar el conjunto de filas, la tabla universal, con un formato específico que permita a la lógica del procesamiento generar el XML deseado.

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,
```



```
EmployeeID as [Employee!1!EmpID],
NULL      as [Name!2!FName],
NULL      as [Name!2!LName]
FROM HumanResources.Employee E INNER JOIN Person.Contact C
ON E.ContactID = C.ContactID
UNION ALL
SELECT 2 as Tag,
      1 as Parent,
      EmployeeID,
      FirstName,
      LastName
FROM HumanResources.Employee E INNER JOIN Person.Contact C
ON E.ContactID = C.ContactID
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 registros. 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***

### **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, or 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
```

### 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\_Eschema:** 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)
```

El resultado es:

ProductID	Quantity
761	2
770	1

También puede armarse una consulta que especifique el patrón del XPath:

```
SELECT *
FROM OPENXML (@idoc, '/Customer/Order/OrderDetail', 1)
WITH (CustomerID int ' ../@CustomerID',
OrderID int ' ../@SalesOrderID',
OrderDate datetime ' ../@OrderDate',
ProdID int '@ProductID',
Quantity int)
```

El resultado en este caso sería:

CustomerID	OrderID	OrderDate	ProdID	Quantity
1	43860	2001-08-01 00:00:00.000	761	2
1	43860	2001-08-01 00:00:00.000	770	1

### 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 elementos 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

El tipo de datos xml es un tipo de dato nativo del 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 semiestructurada están guardadas en la misma ubicación haciendo más fácil su administración.
- Aprovechar la 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 pueden ser tratados 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 2005 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 esta 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 <Maintenance>3 year parts and labor extended maintenance is
available</Maintenance> INTO (/Root/ProductDescription/Features)[1])
-- Agrega otro
SET @Doc.modify('
INSERT <Warranty>1 year parts and labor</Warranty>
AS FIRST INTO (/Root/ProductDescription/Features)[1] ')
SELECT @Doc
```

El resultado es final es:

```
<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>
```

### Enlazar datos relacionales dentro de datos XML

Puede especificar métodos de tipo de datos xml con una variable de tipo de datos o columna xml. Por ejemplo, query() ejecuta la instrucción XQuery especificada con una instancia XML. Cuando se genera XML de esta manera, se puede utilizar un valor de un tipo de columna que no es XML o una variable Transact-SQL. Este proceso se conoce como enlazar datos relacionales dentro de XML. Permite parametrizar consultas

Ejemplo:

```
DECLARE @isbn varchar(20)
SET @isbn = '0-7356-1588-2'
SELECT xCol FROM Tabla
WHERE xCol.exist ('/book/@ISBN[. = sql:variable("@isbn")]') = 1
```

### 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">
    <LinItem ProductID="1" Price="2.99" Quantity="3" />
    <LinItem ProductID="2" Price="3.99" Quantity="1" />
  </Order>'
```

```
SELECT nCol.value('@ProductID', 'integer') ProductID,
nCol.value('@Quantity', 'integer') Quantity
FROM @xmlOrder.nodes('/Order/LinItem') AS nTable(nCol)
```

El resultado es:



ProductID	Quantity
-----------	----------

1	3
2	1

