beginning

SQL SERVER 2008

for developer

CESAR PEREDA TORRES

Consultor Y Especialista
Sistemas, Redes y Servidores

Dentro de mi experiencia profesional y laboral con mas de 19 años, he brindado mis servicios a Instituciones Privadas y GuberNombresntales, Universidades ,Centros de Estudios Tecnológicos y Pedagógicos, dentro de estas Entidades son el Ministerio de la Presidencia PRONAP, Escuela Nacional de estadística ENEI Ucayali. Instituto Superior Tecnológico Suiza, Instituto superior Tecnológico Horacio Zeballos Games, Instituto Superior Tecnológico Tokio, Universidad Nacional de Ucayali, Ministerio de Salud, Par salud, Ministerio de Justicia – Instituto Nacional Penitenciario, Policía Nacional del Perú Cooperativa, Asociación de Civiles de la Fuerza Aérea del Perú, y Empresas particulares, en referencia al desarrollo y funcionamiento de las Áreas y direcciones de la tecnología informática.

Dentro de las cuales 0cho años de experiencia profesional en docencia en Centros de Estudios de Educación Superior.

Microsoft® SQL Server® 2008

DEDICATORIA

Un día vi pasar tan rápido las cosas alrededor de mi, tanto fue que no pude darme cuenta lo hermoso de los días tan valiosos que estuve perdiendo y no pude estar mas cerca de EL, es por esto dedico este libro a La Gran Misericordia de DIOS por darme una linda familia Y una Hija Bendita, unos Padres Maravillosos y una Hermanita quienes estuvieron siempre a mi lado en los momentos difíciles y por todas las Bendiciones que día a día ha puesto en mis caminos.

Día a día ha sido para mi incontables poder editar este libro pero aprendi que DIOS tiene un propósito con cada uno de nosotros, aun cuando para uno no fuese lo que uno quisiera, pero DIOS sabe en que momento se hacen las cosas cuando se las entregamos a EL.

"El Plan que yo tengo para tu futuro esta lleno de esperanza"
Jeremias 29:11

"Si me buscas con todo tu corazón me encontraras"
Deuteronomio 4:29

CESAR PEREDA TORRES

Consultor Especialista en Informática y Sistemas

Dentro de mi experiencia profesional y pedagogica durante los siete años he dictado cursos de Ofimatica a Nivel avanzado Macros, así como entorno XML en correlación a Base de datos y registros en los lenguajes de programación.
Asimismo he vendio desarrollando diferentes tipos de libros y manuales de ayuda para el usuario y el Especialista a fin de que el operador y usuario pueda contar con memorias de ayuda, tales como : Manual de Bolsillo de SQL SERVER, VFox y SQL Conexiones, Manual de Ayuda de SQL ANYWHERE, Manual del Bolsillo de EXCEL Avanzado, Implementación e Instalación de Servidores WINDOWS SERVER.

Siaba

Agradecimientos:

Juana Maria Torres Espinoza
Jesús Wilmer Pereda López
Mariluisa Pascal Ampudia Esposa
Mariluisa Harumi Pereda Pascal
Cintia Julia Pereda Torres
Moises Olvos Pereda
Gerrardo Edgardo Lopez

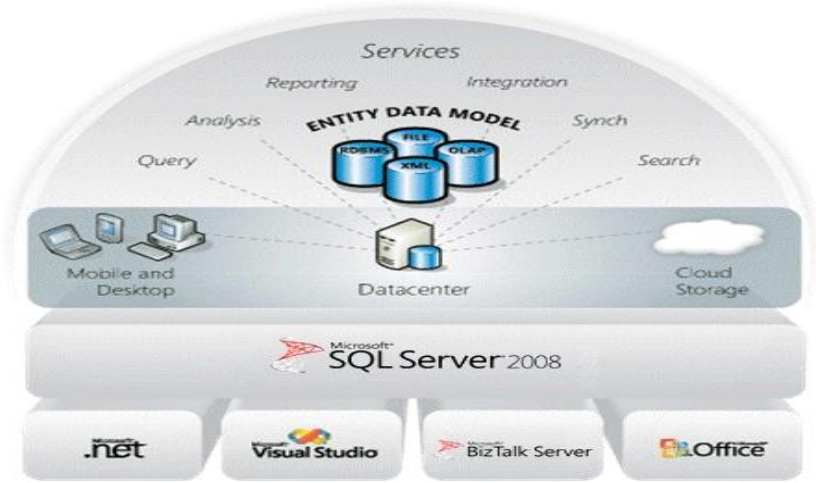
INTRODUCCION

INTRODUCCION

Microsoft SQL Server 2008 incluye un completo conjunto de herramientas gráficas y utilidades de la línea de comandos que permiten a OPERADORs, programadores y administradores aumentar su productividad.

El lenguaje de consulta estructurado (**SQL**) es un lenguaje de base de datos normalizado, utilizado por los diferentes **motores de bases de datos** para realizar determinadas operaciones sobre los **datos o sobre la estructura** de los mismos. Pero como sucede con cualquier sistema de normalización hay excepciones para casi todo; de hecho, cada motor de bases de datos tiene sus peculiaridades y lo hace diferente de otro motor, por lo tanto, el lenguaje SQL normalizado (ANSI) no nos servirá para resolver todos los problemas, aunque si se puede asegurar que cualquier sentencia escrita en ANSI será interpretable por cualquier motor de datos.

SQL Server 2008 Database Engine (Motor de base de datos de SQL Server 2008) de Microsoft es el servicio principal para almacenar, procesar y proteger datos. El Database Engine (Motor de base de datos) proporciona acceso controlado y procesamiento de transacciones rápido para cumplir con los requisitos de las aplicaciones consumidoras de datos más exigentes de su empresa. El Database Engine (Motor de base de datos) también proporciona compatibilidad completa para mantener una alta disponibilidad.



Microsoft SQL Server consolida la administración de servidores y la creación de objetos comerciales en dos entornos integrados: SQL Server Management Studio y Business Intelligence Development Studio. Ambos entornos utilizan soluciones y proyectos para fines de administración y organización. Además, ambos ofrecen una funcionalidad de control de código fuente totalmente integrada (si hay un proveedor de control de código fuente como Microsoft Visual SourceSafe instalado).

Aunque ambos entornos de estudio usan los contenedores y los elementos visuales establecidos en Microsoft Visual Studio 2005, (por ejemplo, proyectos, soluciones, Explorador de soluciones y Cuadro de herramientas) estos entornos no forman parte, por sí mismos, de Visual Studio 2005. En su lugar, los entornos de estudio incluidos con SQL Server son entornos independientes que están diseñados para programadores de aplicaciones empresariales que funcionan con SQL Server, SQL Server Compact 3.5 SP1, Analysis Services, Integration Servicesy Reporting Services. No es posible utilizar estas herramientas para crear aplicaciones personalizadas o acometer grandes proyectos de desarrollo.

SQL Server Management Studio

SQL Server Management Studio es un entorno integrado para obtener acceso a todos los componentes de SQL Server, así como para configurarlos y administrarlos. SQL Server Management Studio combina un amplio grupo de herramientas gráficas con un editor de texto enriquecido para ofrecer acceso a SQL Server a los programadores y administradores, sin importar su nivel de especialización.

SQL Server Management Studio combina las funciones del Administrador corporativo y el Analizador de consultas, herramientas incluidas en versiones anteriores de SQL Server, en un único entorno. Además, SQL Server Management Studio proporciona un entorno para administrar Analysis Services, Integration Services, Reporting Services y XQuery. Este entorno ofrece a los programadores una experiencia familiar y proporciona a los administradores de bases de datos una herramienta única para realizar sus tareas con la facilidad de las herramientas gráficas y una experiencia de Scripts enriquecida.

Business Intelligence Development Studio

Business Intelligence Development Studio es un entorno integrado para desarrollar construcciones de inteligencia empresarial, como cubos, orígenes de datos, informes y paquetes de Integration Services. Business Intelligence Development Studio incluye plantillas de proyecto que proporcionan un contexto para desarrollar construcciones específicas. Por ejemplo, se puede optar por un proyecto de Analysis Services si el objetivo es crear una base de datos de Analysis Services que contenga cubos, dimensiones o modelos de minería de datos.

En Business Intelligence Development Studio, es posible desarrollar proyectos que formen parte de una solución independiente de un servidor concreto. Por ejemplo, puede incluir un proyecto de Analysis Services, de Integration Services y de Reporting Services en la misma solución. Puede implementar los objetos en un servidor de prueba para probarlos durante el desarrollo y, posteriormente, implementar el resultado de los proyectos en uno o más servidores de ensayo o de producción.

Soluciones, proyectos y elementos

Tanto SQL Server Management Studio como Business Intelligence Development Studio proporcionan proyectos que se organizan en soluciones. Los proyectos de SQL Server se guardan como Scripts de SQL Server, de Analysis Server y de SQL Server Compact 3.5 SP1. Los proyectos de Business Intelligence Development Studio se guardan como proyectos de Analysis Services, de Integration Services y de informes. Los proyectos deben abrirse en la misma herramienta en la que han sido creados.

Elegir entre SQL Server Management Studio y Business Intelligence Development Studio

SQL Server Management Studio está diseñado para desarrollar y administrar objetos de base de datos y para administrar y configurar objetos existentes de Analysis Services. Business Intelligence Development Studio está diseñado para desarrollar aplicaciones de Business Intelligence. Si está implementando una solución que utiliza servicios de bases de datos de SQL Server o si está administrando una solución existente que utiliza SQL Server, Analysis Services, Integration Services o Reporting Services, debe utilizar SQL Server Management Studio. Si está desarrollando una solución que utiliza Analysis Services, Integration Services o Reporting Services, debe utilizar Business Intelligence Development Studio.

Parte del estudio recopilado del portal, con la finalidad que el alumno pueda contar con una fuente de ayuda web.

http://technet.microsoft.com/es-es/library/ms174170.aspx

CARACTERISTICAS DE SQL SERVER 2008

La intención de este artículo es la de comentar una de las nuevas capacidades de SQL Server 2008 que es la posibilidad de crear índices filtrados, pero me parece que es una buena oportunidad para mencionar que son los índices, cual es su objetivo, que tipos de índices existen y dejar para final del artículo este asunto de los índices filtrados.

Comencemos por la idea más básica que es la de preguntarse qué es un índice y para qué sirve, un índice es un mecanismo que permite acceder a un conjunto de datos en forma más eficiente que si no se utilizase dicho mecanismo, considerando a la velocidad de acceso a los datos como el factor de eficiencia que los índices optimizan. En el primer tipo de índice que vamos a comentar la estrategia de optimización consiste en ordenar físicamente los datos de forma que puedan encontrarse más rápidamente, esto significa que en este tipo de índice existirá una o varias columnas que definirán de que manera estará la tabla físicamente ordenada.

Esta idea no es nueva, para quienes hayan programado alguna vez en cualquier lenguaje sabrán que si debemos buscar un valor en un vector ordenado, podremos utilizar algunas técnicas como por ejemplo la búsqueda binaria que permitirán encontrar los datos buscados en orden logarítmico a diferencia del inmejorable orden lineal cuando los valores dentro del vector están desordenados. Para quien nunca haya programado podrá recordar un diccionario, en un diccionario un usuario busca una definición (datos) a partir de una clave (palabra a buscar) y el hecho de que los datos estén ordenados por la clave (o sea las definiciones por las palabras) permitirá que el usuario no tenga que recorrer todas las palabras del diccionario hasta encontrar la palabra deseada. De forma similar dentro de la estructura de tablas del SQL Server el hecho que los datos se encuentren ordenados físicamente por la clave permitirá un acceso más rápido a los mismos. No estará quien se pregunte qué sucederá cuando se inserte un nuevo registro con la performance, y no hay dudas que será menos eficiente que si los datos estuviesen desordenados, pero no hay que olvidar que lo que se desea es eficiencia en las operaciones de búsquedas, que son las que se realizan con mayor frecuencia.

La forma más sencilla de ver la diferencia que puede provocar un índice de este tipo es crear una tabla simple en nuestro motor de base de datos SQL Server y ver el plan de ejecución en ambos casos (con y sin el índice), comencemos creando la tabla y agregando algunos valores:

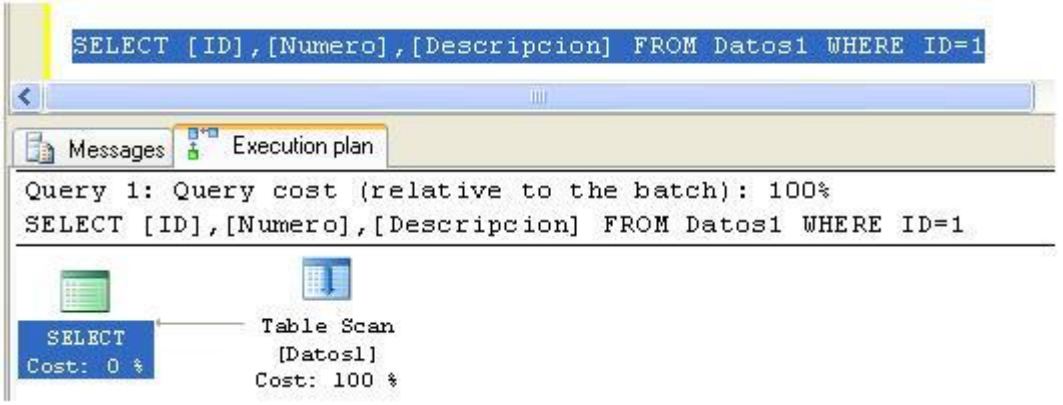
```
CREATE TABLE [dbo].[Datos1](
    [ID] [int] NOT NULL,
    [Numero] [int]NOT NULL,
    [Descripcion] [nvarchar](50) NOT NULL,
)
INSERT INTO Datos1 ([ID],[Numero],[Descripcion]) VALUES (1,1,'D1')
INSERT INTO Datos1 ([ID],[Numero],[Descripcion]) VALUES (2,2,'D2')
INSERT INTO Datos1 ([ID],[Numero],[Descripcion]) VALUES (3,3,'D3')
INSERT INTO Datos1 ([ID],[Numero],[Descripcion]) VALUES (4,4,'D4')
```

Luego iniciaremos una búsqueda y veremos el plan de ejecución. El plan de ejecución mostrará de qué manera el query optimizer intentará acceder a los datos durante una consulta, (El query optimizer es el encargado de diseñar la estrategia del acceso a los datos).
Existen varias maneras de ver el plan de ejecución, utilizaremos en estos ejemplos la forma grafica.

Luego de haber ejecutado el script previo deberemos escribir lo siguiente en un query analizer:

```
SELECT [ID], [Numero], [Descripcion] FROM Datos1 WHERE ID=1
```

Y luego presionar CTRL+L. Se obtendrá un resultado similar a lo siguiente:



Los planes de ejecución en formato gráfico deben leerse de izquierda a derecha y de arriba hacia abajo, y aunque pueden ser extremadamente largos y complejos de leer, en nuestro caso podemos ver el mismo está compuesto por solamente dos iconos y una flecha que los une a ambos. Cada icono representará una operación y la flecha simbolizará el movimiento de datos entre las dos operaciones, indicándonos que la operación “Table Scan” ha tomado los datos que la operación SELECT procesará, en realidad la operación SELECT no ha hecho nada en este caso. Este diagrama nos indica que está haciendo interNombresnte el motor de base de datos.

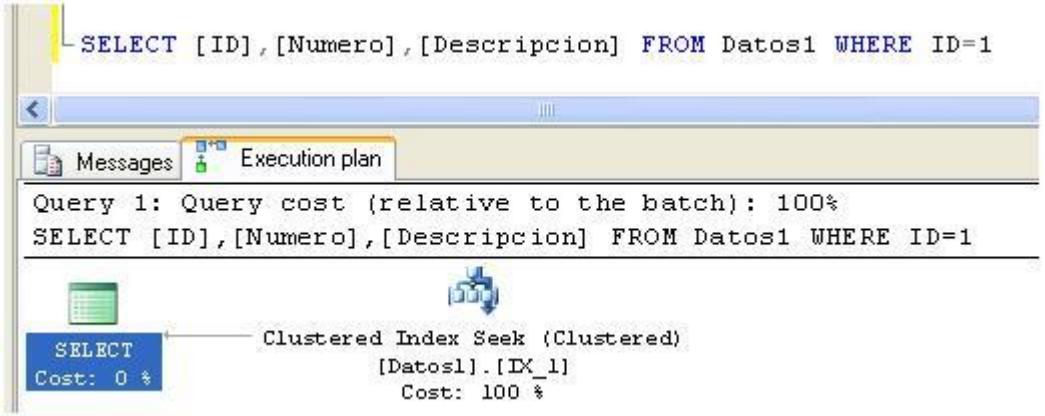
Una operación “Table Scan” nos está indicando que el motor ha necesitado recorrer secuencialmente la tabla Datos1 para poder encontrar los registros que cumplan con la condición pedida.

La operación “Table Scan” es equivalente a tener un diccionario desordenado donde es necesario recorrerlo secuencialmente hasta encontrar la palabra que deseamos buscar, pero además la palabra puede existir más de una vez, así que siempre deberemos recorrerlo hasta la última palabra para asegurarnos que hemos encontrado todas las definiciones. Cuando no hay índices creados la performance de las búsquedas quedan gravemente comprometidas.

En contraposición crearemos un índice y veremos que cambios se producen en el plan de ejecución, ejecutaremos la siguiente línea de código:

CREATE CLUSTERED INDEX IX_1 ON [dbo].[Datos1] (ID)

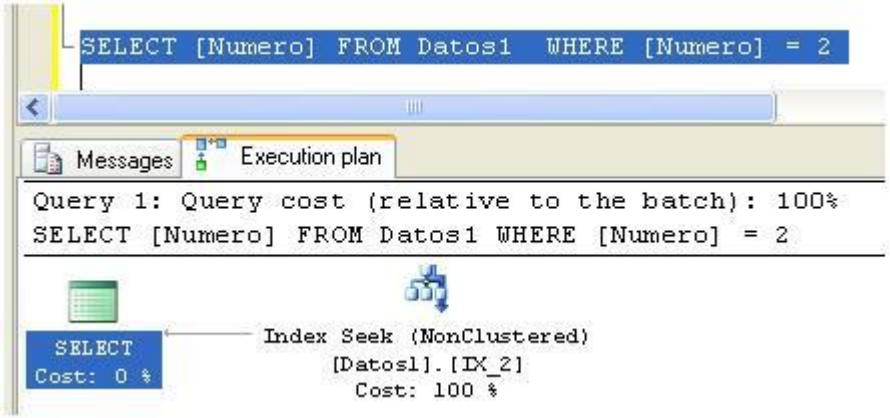
Donde hemos indicado la creación de un índice por la columna “ID”,(la palabra CLUSTERED indicará que la tabla se ordernenará físicamente por el índice solicitado, luego veremos que existe otro tipo de índices que no impone tal condición.) Si volvemos a ejecutar la consulta anterior, el plan de ejecución tomará el siguiente formato:



Indicando que en este caso la búsqueda de datos está utilizando el índice IX_1, de manera que el motor ya no debe recorrer toda la tabla para encontrar los registros pedidos.
Podemos ahora preguntarnos que pasaría si además es necesario realizar búsquedas por otro campo, supongamos por el campo “Numero”, en este caso no podremos reordenar la tabla físicamente por “Numero”, ya que al hacer esto perderíamos el orden físico que ya habíamos establecido por el campo “ID”, es claro que el orden físico puede establecerse solo para una clave (ya sea compuesta por un solo o varios campos). Para estos casos existen otro tipo de índices conocidos como índices non-clustered, ya que no modifican el orden físico de los registros en la tabla original, estos índices guardarán en otra estructura una copia de los valores involucrados en la clave y un puntero al registro original de la tabla. Para probar lo antes comentado ejecutaremos el siguiente comando:

CREATE INDEX IX_2 ON [dbo].[Datos1] (Numero)

Y luego veremos el plan de la siguiente búsqueda:



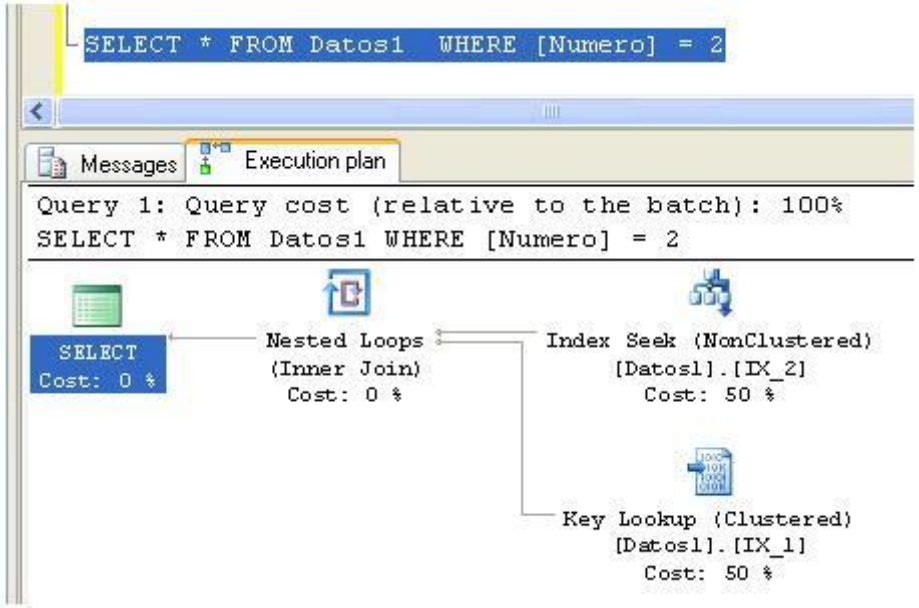
Donde puede verse que el query optimizer ha decidido utilizar el nuevo índice IX_2.

Habrá seguramente quien se haya percatado que en este último query solo estamos incluyendo a la columna “Numero” y se pregunte el por qué de esta decisión?, y más aun, habrá quien pareciéndole extraño realizará la misma búsqueda pero esta vez con todos los campos (al menos eso espero). Si es así, quien realice esta prueba descubrirá algo pertubador, y es que el query optimizer habrá decidió utilizar el índice IX_1, y no IX_2, pero por que? podrán preguntarse y la respuesta es la siguiente: Como comentamos previamente los índices non-clustered guardan una copia de las claves y un puntero al registro original, de esta manera cuando hemos buscado solamente por “Numero” el índice IX_2 es capaz de devolver la información solicitada ya que posee el valor de la columna “Numero”, pero cuando hemos pedido otros datos como “ID” y “Descripcion” que no existen en IX_2 el query optimizer ha decidido que es menos costoso recorrer la tabla por IX_1 para devolver los datos que IX_2 no posee. Cuando un índice non-clustered cubre todos los datos solicitados en la consulta se dice que es un covered-index, el caso contrario no será un covered-index y el query optimizer deberá buscar alguna estrategia para obtener los datos faltantes, obviamente los clustered index son siempre covered index, ya que poseen el registro completo.

El query optimizer puede utilizar otras estrategias para obtener los datos faltantes como veremos a continuación. Si ejecutamos el siguiente código:

```
DELETE FROM Datos1
DECLARE @C int =1
WHILE @C < 10000
BEGIN
   INSERT INTO Datos1 ([ID],[Numero],[Descripcion])
   VALUES (@C,@C + 1,'D1' + cast(@C as nvarchar(10)))
   SET @C+=1
END
```

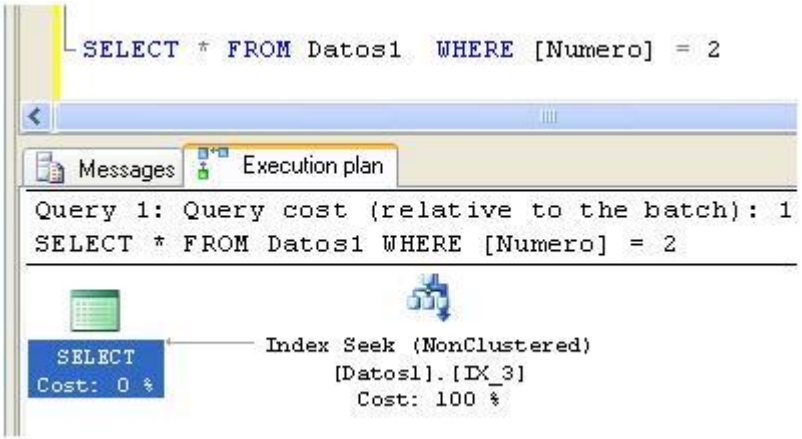
Donde solamente hemos agregado más datos y volvemos a ejecutar la consulta anterior veremos lo siguiente:



Ahora el query optimizer ha utilizado nuestro índice IX_2 pero para recuperar los datos faltantes a requerido efectuar una operación de Key Lookup extra utilizando el índice IX_1, para finalmente unir los datos en la operación Nested Loops. Si creamos un nuevo índice que cubra todos los datos pedidos de la siguiente forma:

```
1 CREATE INDEX IX_3  ON [dbo].[Datos1] (Numero,ID,Descripcion)
```

No debería sorprendernos el siguiente resultado:



Otra opción para incluir las columnas restantes es utilizar la sentencia INCLUDE de la siguiente forma:

```
1 CREATE INDEX IX_3 ON [dbo].[Datos1] (Numero)  INCLUDE (Descripcion, ID)
```

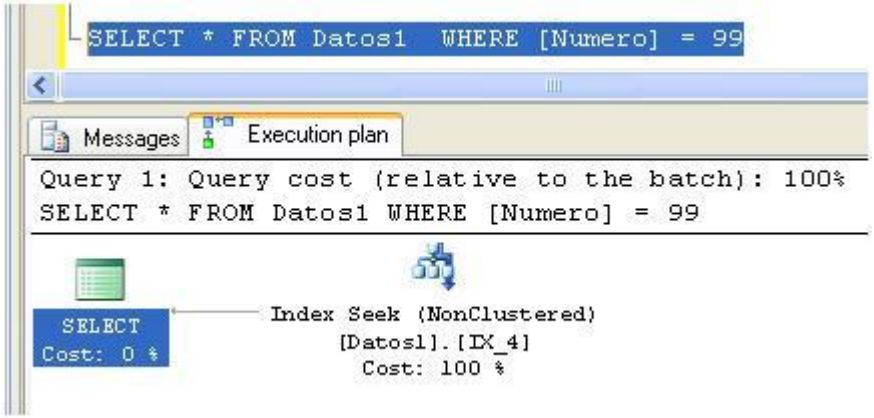
En el segundo caso, las columnas son agregadas al índice pero no forman parte del mismo.

En ambos tipos de índices, clustered o non-clustered existe la posibilidad de definirlos como únicos (unique), un índice único no admite repetición de valores, y permite una mayor optimización en las búsquedas. Las claves primarias de las tablas están compuestas por índices “unique” que pueden ser o no clustered.

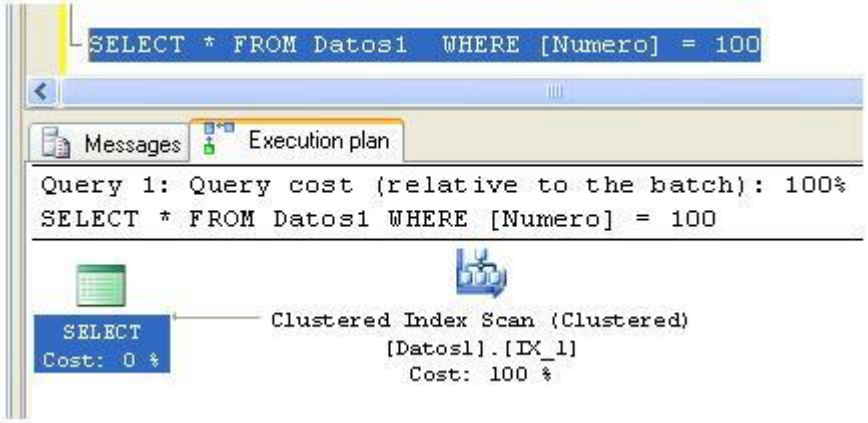
En Sql Server 2008 existe además la posibilidad de crear índices filtrados, o sea índices que se aplican solo a un grupo de datos. Para probarlo podemos eliminar los índices IX_2 e IX_3 y crear un nuevo índice IX_4 filtrado, las siguientes líneas de código efectuan estas operaciones:

```
1 DROP INDEX IX_2 ON [dbo].[Datos1]
2 DROP INDEX IX_3 ON [dbo].[Datos1]
3
4 CREATE INDEX IX_4 ON [dbo].[Datos1] (Numero,ID,Descripcion) WHERE Numero < 100
```

De esta forma el índice IX_4 será aplicable para algunas condiciones solamente, por ejemplo si ejecutamos el siguiente query:



El query optimizer ha decidido emplear IX_4 mientras que en el caso de:

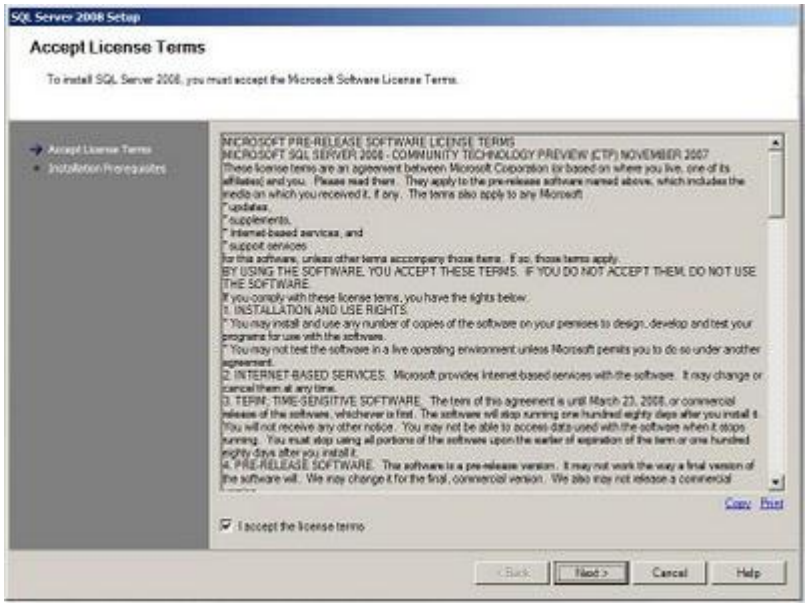


CAPITULO I

INSTALACION DE SQL SERVER 2008

Para iniciar con el proceso de instalación de SQL Server. Ubícate en la carpeta \Servers\ e inicie **setup.exe**. Si está instalando desde un recurso compartido de red, navegue a la carpeta \Servers\ en la carpeta de red e inicie **setup.exe**.

Si aparece el cuadro de diálogo de instalación Microsoft .NET Framework versión 2.0, haga clic en la casilla para aceptar el Contrato de licencia de .NET Framework 2.0 y, a continuación, haga clic en **Siguiente** para realizar la instalación. Para salir de la instalación de SQL Server 2008, haga clic en **Cancelar**. Cuando se complete la instalación de .NET Framework 2.0, haga clic en **Finalizar**.



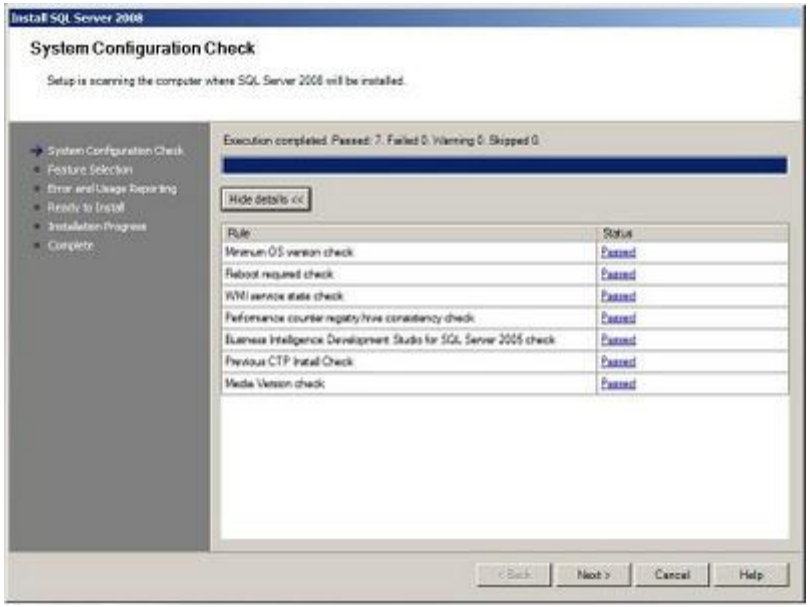
En la página **Términos de licencia**, lea el contrato de licencia y active la casilla para aceptar los términos y condiciones de la licencia. Una vez aceptado el contrato de licencia, se activará el botón **Siguiente**. Para continuar, haga clic en **Siguiente**. Para salir del programa de instalación, haga clic en **Cancelar**.

El Asistente para la instalación instalará los requisitos previos de SQL Server si aún no están en el equipo. Son los siguientes:

- .NET Framework 2.0
- SQL Server Native Client
- Archivos auxiliares de instalación de SQL Server

Para instalar los requisitos previos, haga clic en **Instalar**.

En la página **SQL Server 2008 Installation Center**, haga clic en el vínculo **Nueva instalación**.

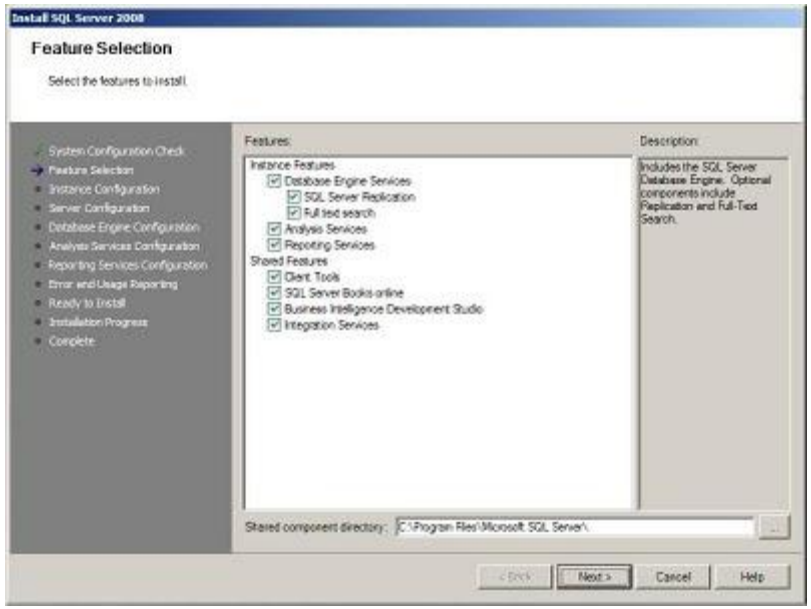


Al hacer clic en el vínculo de instalación, el Comprobador de configuración del sistema comprobará el equipo donde se está ejecutando la instalación. Las comprobaciones de esta versión incluyen:

- Comprobación de la versión del sistema operativo: comprueba que el sistema operativo se admite en esta versión. Para obtener información de los requisitos,
- Comprobación del servicio WMI: comprueba que el servicio Windows Installer se está ejecutando.
- Comprobación de la coherencia de los contadores de rendimiento: comprueba los valores de las claves del Registro para comprobar el incremento correcto de la instalación de los contadores de perfmon de SQL Server.

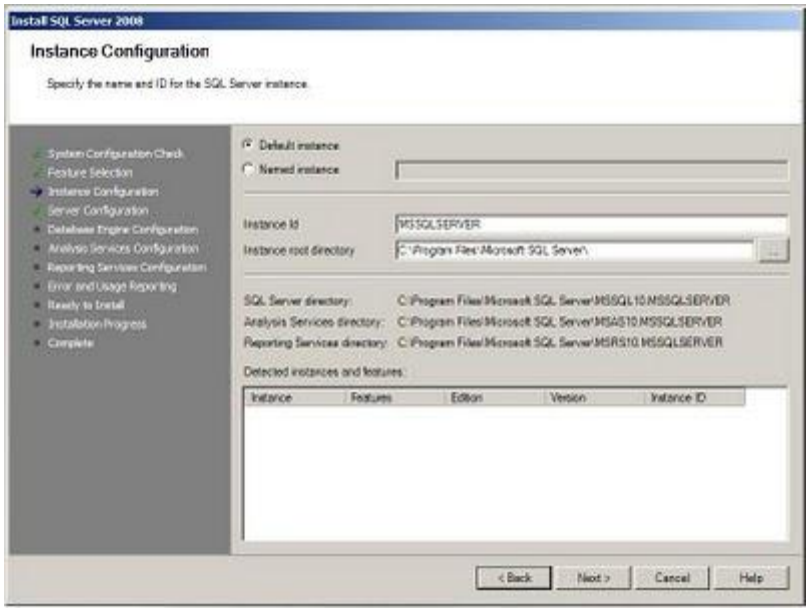
  Comprobación de Business Intelligence Development Studio: comprueba que Business Intelligence Development Studio no está instalado, ya que la actualización de este componente no se admite.

Comprobación de la instalación de SQL Server 2008 anterior: comprueba que las instalaciones de CTP anteriores de SQL Server 2008 no están presentes en el equipo donde se está ejecutando el programa de instalación.



En la página **Selección de características**, seleccione los componentes de la instalación. Después de seleccionar el nombre de la característica, la descripción de cada grupo de componentes aparece en el panel derecho. Puede activar las casillas de verificación que desee. Para obtener más información,

Para cambiar la ruta de instalación de los componentes compartidos, actualice el nombre de ruta en el campo que se proporciona en la parte inferior del cuadro de diálogo o haga clic en el botón **...** para navegar a un directorio de instalación. La ruta de acceso de instalación predeterminada es C:\Archivos de programa\Microsoft SQL Server\.



En la página **Configuración de instancia**, especifique si desea instalar una instancia predeterminada o una instancia con nombre. Para tener en cuenta consideraciones sobre la denominación de instancias,

Sufijo de id. de instancia: de forma predeterminada, el nombre de instancia se utiliza como sufijo del identificador de instancia. Se usa para identificar los directorios de instalación y las claves del Registro para la instancia de SQL Server. Es así en las instancias predeterminadas y en las instancias con nombre. Con una instancia predeterminada, el nombre y el sufijo del identificador serían MSSQLSERVER. Para

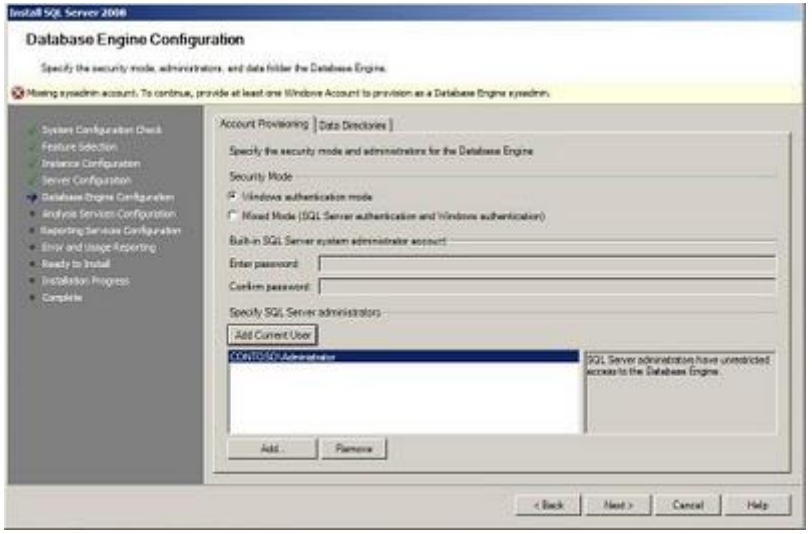
utilizar un sufijo de identificador de instancia no predeterminado, haga clic en la casilla **Sufijo de id. de instancia** y suministre un valor en el campo proporcionado.

Las instancias independientes típicas de SQL Server 2008, tanto si son predeterminadas como si son instancias con nombre, no utilice un valor no predeterminado para la casilla**Sufijo de id. de instancia**.

Directorio raíz de instancia: de forma predeterminada, el directorio raíz de la instancia es C:\Archivos de programa\Microsoft SQL Server\. Para especificar un directorio raíz no predeterminado, utilice el campo proporcionado o haga clic en el botón **Examinar** y navegue a una carpeta de instalación.

Todos los componentes de una instancia determinada de SQL Server se administran como una unidad. Todos los Service Packs y actualizaciones de SQL Server se aplicarán a cada componente de una instancia de SQL Server.

Instancias detectadas y características: la cuadrícula mostrará las instancias de SQL Server que están en el equipo en el que se ejecuta el programa de instalación. Para actualizar una de esas instancias en lugar de crear una nueva, seleccione el nombre y compruebe que aparece en el……, a continuación, haga clic en **Siguiente**.



En la página **Configuración del servidor: Cuentas de servicio**, especifique las cuentas de inicio de sesión para los servicios de SQL Server. Los servicios reales configurados en esta página dependen de las características seleccionadas para ser instaladas.

Puede asignar la misma cuenta de inicio de sesión a todos los servicios de SQL Server, o configurar cada cuenta de servicio individualmente. También puede especificar si los servicios se inician automática o manualmente, o están deshabilitados. Microsoft recomienda que configure de forma individual las cuentas de servicio para proporcionar los mínimos privilegios para cada servicio, donde a los servicios de SQL Server se les conceden los permisos mínimos que necesitan para completar sus tareas. Para obtener más información,.

Para especificar la misma cuenta de inicio de sesión para todas las cuentas de servicio en esta instancia de SQL Server, las credenciales se proporcionan en los campos de la parte inferior de la página.

Nota de seguridad No utilice una contraseña en blanco. Utilice una contraseña segura.

Cuando termine de especificar información de inicio de sesión para los servicios de SQL Server, haga clic en **Siguiente**.

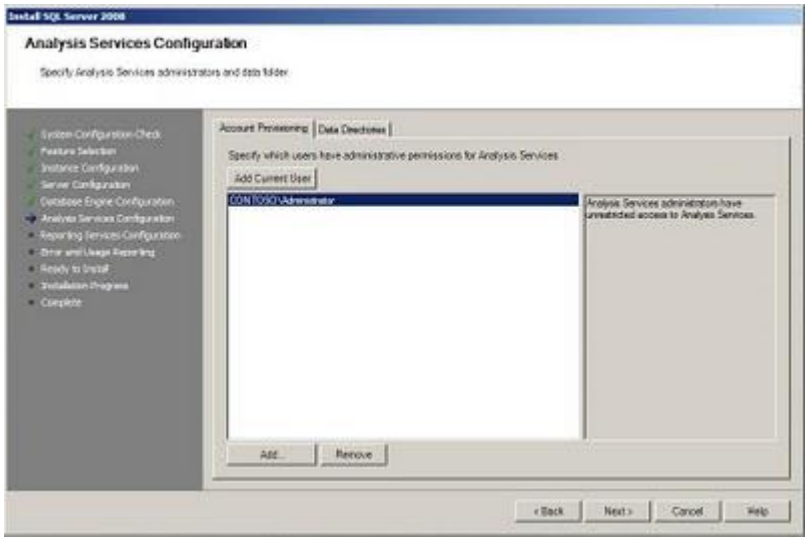
Utilice la ficha **Configuración del servidor - Intercalación** para especificar intercalaciones no predeterminadas para Database Engine (Motor de base de datos) y Analysis Services. Para obtener más información,

Use la página **Configuración del motor de base de datos - Aprovisionamiento de cuentas** para especificar lo siguiente:

Modo de Seguridad: seleccione la autenticación de Windows o la autenticación de modo mixto para su instancia de SQL Server.

Si selecciona la autenticación de modo mixto, debe proporcionar y, a continuación, confirmar una contraseña segura para la cuenta de administrador del sistema de SQL Server integrada.

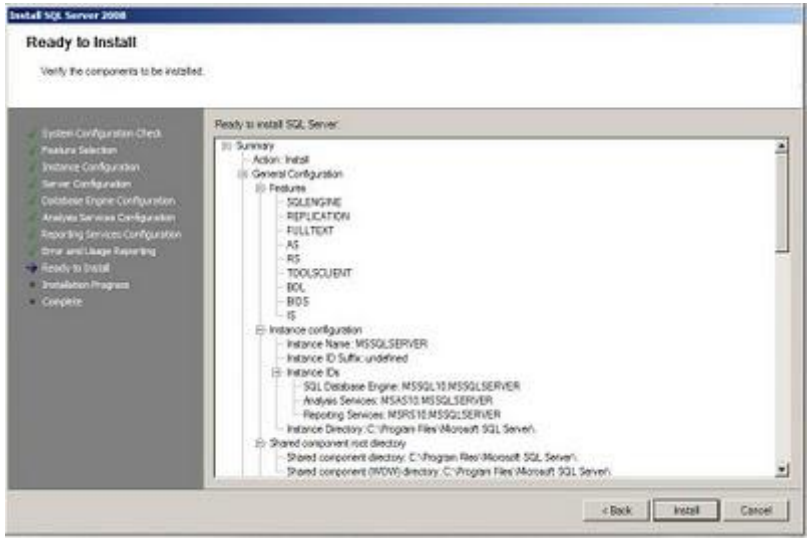
Una vez que un dispositivo establezca una conexión correcta con SQL Server, el mecanismo de seguridad es el mismo para la autenticación de Windows y para el modo mixto. Para obtener más información acerca de cómo aprovisionar las cuentas,



Administradores de SQL Server: debe especificar al menos un administrador del sistema para la instancia de SQL Server.

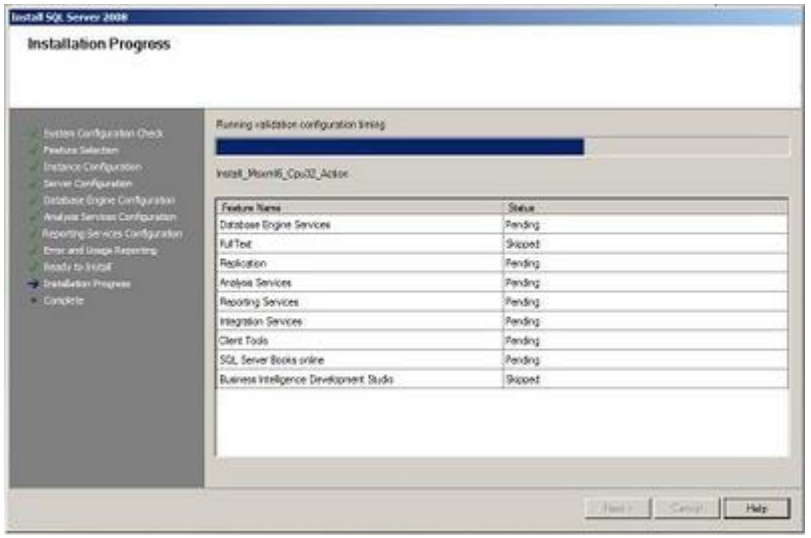
Para agregar la cuenta en la que se ejecuta el programa de instalación de SQL Server, haga clic en el botón **Agregar OPERADOR actual**. Para agregar o quitar las cuentas de la lista de administradores del sistema, haga clic en **Agregar** o en **Quitar**, y a continuación modifique la lista de OPERADORs, grupos o equipos que tendrán privilegios de administrador para la instancia de SQL Server. Para obtener más información acerca de cómo aprovisionar las cuentas,

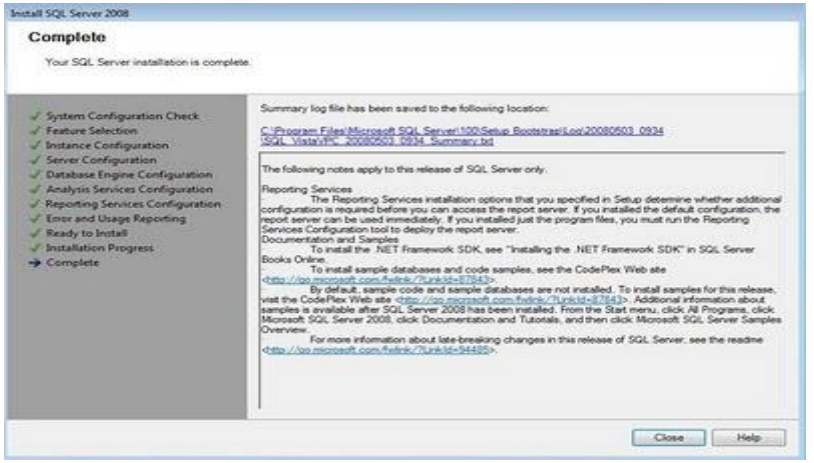
Cuando termine de modificar la lista, haga clic en **Aceptar** y, a continuación, compruebe la lista de administradores en el cuadro de diálogo de configuración. Cuando la lista esté completa, haga clic en **Siguiente** para continuar.



Utilice la página **Configuración del motor de base de datos - Directorios de datos** para especificar los directorios de instalación no predeterminados. Para instalar en los directorios predeterminados, haga clic en **Siguiente**.

Luego veremos un compendio de todo los servicios que se van a instalar con SQL SERVER 2008 y elegimos si deseamos regresar a cambiarla algún parámetro o sino proceder con la Instalación haciendo clic el botón **INSTALL**





Podemos ver que comienza el proceso de instalación del motor de Bases de Datos y todos los servicios del SQL Server 2008

Después de terminado podemos ver el siguiente mensaje indicándonos como termino el proceso de instalación.

Recopilación de memoria ayuda por franklin zhunio

Añadiendo ante ello, debemos considerar en el proceso de instalación tener los privilegios de administrador del equipo de cómputo

1. Inserte el disco de instalación de SQL Server. Desde la carpeta raíz, haga doble clic en setup.exe. Para realizar la instalación desde un recurso compartido de red, localice la carpeta raíz de dicho recurso y, a continuación, haga doble clic en setup.exe. Si aparece el cuadro de diálogo Programa de instalación de Microsoft SQL Server 2008, haga clic en **Aceptar** para instalar los requisitos previos y, a continuación, haga clic en **Cancelar** para salir de la instalación de SQL Server 2008.
2. Si aparece el cuadro de diálogo de instalación de .NET Framework 3.5 SP1, active la casilla para aceptar el contrato de licencia de .NET Framework 3.5 SP1. Haga clic en **Siguiente**. Para salir de la instalación de SQL Server 2008, haga clic en **Cancelar**. Cuando se complete la instalación de .NET Framework 3.5 SP1, haga clic en **Finalizar**.
3. También se requiere Windows Installer 4.5, que se puede instalar con el Asistente para la instalación. Si se le solicita que reinicie el equipo, hágalo y, a continuación, reinicie el archivo setup.exe de SQL Server 2008.
4. Una vez instalados los requisitos previos, el Asistente para la instalación ejecutará el Centro de instalación de SQL Server. Para crear una nueva instalación de SQL Server 2008, haga clic en **Nueva instalación independiente de SQL Server o agregar características a una instalación existente**.
5. El Comprobador de configuración del sistema ejecutará una operación de detección en su equipo. Para continuar, haga clic en **Aceptar**. Se crean los archivos de registro de la instalación. Para obtener más información,
6. En la página Clave del producto, seleccione un botón de opción para indicar si está instalando una edición gratuita de SQL Server o una versión de producción del producto que tiene una clave de PID. Para obtener más información,
7. En la página Términos de licencia, lea el contrato de licencia y active la casilla para aceptar los términos y condiciones de la licencia.
8. El Asistente para la instalación instalará los requisitos previos de SQL Server si aún no están en el equipo. Entre ellos, figuran:
    - .NET Framework 3.5 SP1
    - SQL Server Native Client
    - Archivos auxiliares del programa de instalación de SQL Server

    Para instalar los requisitos previos, haga clic en **Instalar**.

9. El Comprobador de configuración del sistema comprobará el estado del sistema de su equipo antes de seguir con la instalación.
10. En la página Selección de características, seleccione los componentes de la instalación. Después de seleccionar el nombre de la característica se muestra una descripción de cada grupo de componentes en el panel derecho. Puede activar una combinación de casillas. Para obtener más información,

    Si desea especificar un directorio personalizado para los componentes compartidos, use el campo situado en la parte inferior de la página Selección de características. Para cambiar la ruta de instalación de los componentes compartidos, actualice el nombre de ruta en el campo situado en la parte inferior del cuadro de diálogo o haga clic en **Examinar** para moverse a un directorio de instalación. La ruta de instalación predeterminada es C:\Archivos de programa \ Microsoft SQL Server\100\.

11. En la página Configuración de instancia, especifique si desea instalar una instancia predeterminada o una instancia con nombre. Para obtener más información, Para continuar, haga clic en Siguiente.

    Id. de instancia: de forma predeterminada, el nombre de instancia se utiliza como identificador de la instancia. Se usa para identificar los directorios de instalación y las claves del Registro para la instancia de SQL Server. Es así en las instancias predeterminadas y en las instancias con nombre. En el caso de una instancia predeterminada, el nombre y el identificador de la citada instancia

serían MSSQLSERVER. Para utilizar un identificador de instancia no predeterminado, active la casilla **Id. de instancia** y proporcione un valor.

Directorio raíz de instancia: de forma predeterminada, el directorio raíz de instancia es C:\Archivos de programa\Microsoft SQL Server\100\. Para especificar un directorio raíz no predeterminado, utilice el campo proporcionado o haga clic en **Examinar** para buscar una carpeta de instalación.

Todos los Service Pack y actualizaciones de SQL Server se aplicarán a cada componente de una instancia de SQL Server.

Características e instancias detectadas: la cuadrícula muestra las instancias de SQL Server que están en el equipo en el que se ejecuta el programa de instalación. Si ya hay una instancia predeterminada instalada en el equipo, debe instalar una instancia con nombre de SQL Server 2008.

12. La página Requisitos de espacio en disco calcula el espacio en disco necesario para las características que ha especificado. A continuación, compara el espacio necesario con el espacio en disco disponible. Para obtener más información,
13. El flujo de trabajo en el resto del tema depende de las características que haya especificado en la instalación. Dependiendo de las selecciones, es posible que no vea todas las páginas.
14. En la página Configuración del servidor - Cuentas de servicio, especifique las cuentas de inicio de sesión para los servicios de SQL Server. Los servicios reales que se configuran en esta página dependen de las características que se van a instalar.

    Puede asignar la misma cuenta de inicio de sesión a todos los servicios de SQL Server, o configurar cada cuenta de servicio individualmente. También puede especificar si los servicios se inician automática o manualmente, o si están deshabilitados. Microsoft recomienda que configure de forma individual las cuentas de servicio para proporcionar los privilegios mínimos para cada servicio, donde a los servicios de SQL Server se les conceden los permisos mínimos que necesitan para completar sus tareas.

    Para especificar la misma cuenta de inicio de sesión para todas las cuentas de servicio en esta instancia de SQL Server, las credenciales se proporcionan en los campos de la parte inferior de la página.

    Nota de seguridad No utilice una contraseña en blanco. Utilice una contraseña segura.

    Cuando termine de especificar la información de inicio de sesión para los servicios de SQL Server, haga clic en **Siguiente**.

15. Utilice la ficha **Configuración del servidor - Intercalación** para especificar intercalaciones no predeterminadas para Database Engine (Motor de base de datos) y Analysis Services.
16. Use la página Configuración de Database Engine (Motor de base de datos) - Aprovisionamiento de cuentas para especificar lo siguiente:
    - Modo de Seguridad: seleccione la autenticación de Windows o la autenticación de modo mixto para su instancia de SQL Server. Si selecciona la autenticación de modo mixto, debe proporcionar una contraseña segura para la cuenta de administrador del sistema de SQL Server integrada.

      Una vez que un dispositivo establezca una conexión correcta con SQL Server, el mecanismo de seguridad será el mismo para la autenticación de Windows y para el modo mixto. Para obtener más información, .

    - Administradores de SQL Server: debe especificar al menos un administrador del sistema para la instancia de SQL Server. Para agregar la cuenta en la que se ejecuta el programa de instalación de SQL Server, haga clic en **Agregar OPERADOR actual**. Para agregar o quitar cuentas de la lista de administradores del sistema, haga clic en **Agregar** o en **Quitar** y, a continuación, modifique la lista de OPERADORs, grupos o equipos que tendrán privilegios de administrador para la instancia de SQL Server. Para obtener más información .

Cuando haya terminado de modificar la lista, haga clic en **Aceptar**. Compruebe la lista de administradores en el cuadro de diálogo de configuración. Cuando la lista esté completa, haga clic en **Siguiente**.

17. Use la página Configuración de Database Engine (Motor de base de datos) - Directorios de datos para especificar los directorios de instalación no predeterminados. Para instalar en los directorios predeterminados, haga clic en **Siguiente**.
18. Para obtener más información,
19. Use la página Configuración Database Engine (Motor de base de datos) - FILESTREAM para habilitar FILESTREAM para la instancia de SQL Server. Para obtener más información,
20. Use la página Configuración de Analysis Services - Aprovisionamiento de cuentas para especificar los OPERADORs o las cuentas que tendrán permisos de administrador para Analysis Services. Debe especificar al menos un administrador del sistema para Analysis Services. Para agregar la cuenta en la que se ejecuta el programa de instalación de SQL Server, haga clic en **Agregar OPERADOR actual**. Para agregar o quitar cuentas de la lista de administradores del sistema, haga clic en **Agregar** o **Quitar** y, a continuación, modifique la lista de OPERADORs, grupos o equipos que tendrán privilegios de administrador para Analysis Services .

    Cuando haya terminado de modificar la lista, haga clic en **Aceptar**. Compruebe la lista de administradores en el cuadro de diálogo de configuración. Cuando la lista esté completa, haga clic en **Siguiente**.

21. Use la página Configuración de Analysis Services - Directorios de datos para especificar los directorios de instalación no predeterminados. Para instalar en los directorios predeterminados, haga clic en **Siguiente**.
22. Use la página Configuración de Reporting Services para especificar el tipo de instalación de Reporting Services que se creará. Entre las opciones posibles se encuentran las siguientes:
    - Configuración predeterminada del modo nativo
    - Configuración predeterminada del modo de SharePoint
    - Instalación de Reporting Services sin configurar
23. En la página **Informes de errores y de uso**, especifique la información que desee enviar a Microsoft y que ayudará a mejorar SQL Server. De forma predeterminada, las opciones para los informes de errores y el uso de características están habilitadas
24. El Comprobador de configuración del sistema ejecutará uno o varios conjuntos de reglas para validar la configuración del equipo con las características de SQL Server que ha especificado.
25. La página Listo para instalar muestra una vista de árbol de las opciones de instalación que se especificaron durante la instalación. Para continuar, haga clic en **Instalar**.
26. La página Progreso de la instalación muestra el estado para que pueda supervisar el progreso de la instalación durante la ejecución del programa de instalación.
27. Después de la instalación, la página Operación completada proporciona un vínculo al archivo de registro de resumen para la instalación y otras notas importantes. Para completar el proceso de instalación de SQL Server, haga clic en **Cerrar**.
28. Si el programa indica que se reinicie el equipo, hágalo ahora. Es importante leer el mensaje del Asistente para la instalación tras finalizar el programa de instalación.

}

MEMORIA AYUDA

Que se debe considerar en el momento de la implementación e instalación de SQL SERVER ENTERPRISE EN WINDOWS Vista o en Windows XP

Recordemos que la pregunta existe hasta la fecha desde cuando se instalaba SQL Server 7 y las nuevas actualizaciones hasta la fecha.

No puedes, necesitas la versión professional de SQL o instalar SQLExpress o cambiar tu sistema operativo a Windows 2000,2003 o 20008 Server.

La versión Enterprise de SQL server (versión 7.0, 2000, 2005 o 2008) es para servidores, tanto vista como XP son Workstations, para instalar en un Workstation necesitas la versión Professional o standard

Con la versión Enterprise podrás instalar las herramientas de cliente para conectar a un SQL server remoto pero nunca un servidor de SQL en un equipo que no sea un servidor.

Cómo actualizar a SQL Server 2008

El Asistente para la instalación de SQL Server proporciona un único árbol de características para la actualización de los componentes de SQL Server. También puede instalar SQL Server 2008 en paralelo con una versión anterior, o migrar los valores de configuración y las bases de datos existentes de una versión anterior de SQL Server y aplicarlos a una instancia de SQL Server 2008.

Debemos considerar las actualizaciones de cada versión.

Los siguientes escenarios de actualización se admiten en esta versión de SQL Server.

SQL Server 2000 (32 bits) Developer SP41,4	SQL Server 2008 Developer
SQL Server 2000 (32 bits) Enterprise SP41,4	SQL Server 2008 Enterprise
SQL Server 2000 Enterprise Evaluation (32 bits, IA64)4,5	No se admite la actualización.
SQL Server 2000 (64 bits) Developer SP41,4	SQL Server 2008 (64 bits) IA64 Developer
SQL Server 2000 (64 bits) IA64 Enterprise SP43,4,5	SQL Server 2008 (64 bits) IA64 Enterprise
SQL Server 2000 (32 bits) Personal SP4	No se admite la actualización.
SQL Server 2005 (32 bits) Express1	SQL Server 2008 Express
	SQL Server 2008 Express Tools
	SQL Server 2008 Express Advanced
	SQL Server 2008 Workgroup
SQL Server 2005 (32 bits) Express1 Advanced1	SQL Server 2008 Express Advanced
	SQL Server 2008 Workgroup
SQL Server 2005 (32 bits) Workgroup1	SQL Server 2008 Workgroup
	SQL Server 2008 Standard

	SQL Server 2008 Enterprise
SQL Server 2005 (32 bits) Standard1	SQL Server 2008 Standard
	SQL Server 2008 Enterprise
SQL Server 2005 (32 bits) Developer1	SQL Server 2008 Developer
SQL Server 2005 (32 bits) Enterprise1	SQL Server 2008 Enterprise
SQL Server 2005 Enterprise Evaluation (32 bits, IA64, X64)	No se admite la actualización.
SQL Server 2005 IA64 (64 bits) Developer	SQL Server 2008 IA64 (64 bits) Developer
SQL Server 2005 IA64 (64 bits) Standard	SQL Server 2008 IA64 (64 bits) Enterprise
SQL Server 2005 IA64 (64 bits) Enterprise	SQL Server 2008 IA64 (64 bits) Enterprise
SQL Server 2005 X64 (64 bits) Developer	SQL Server 2008 X64 (64 bits) Developer
SQL Server 2005 X64 (64 bits) Standard	SQL Server 2008 X64 (64 bits) Standard
	SQL Server 2008 X64 (64 bits) Enterprise
SQL Server 2005 X64 (64 bits) Enterprise	SQL Server 2008 X64 (64 bits) Enterprise
SQL Server 2008 Express1	SQL Server 2008 Express
	SQL Server 2008 Express Tools
	SQL Server 2008 Express Advanced
	SQL Server 2008 Workgroup
	SQL Server 2008 Standard
	SQL Server 2008 Developer

	SQL Server 2008 Enterprise
SQL Server 2008 Express Tools	SQL Server 2008 Express Tools
	SQL Server 2008 Express Advanced
	SQL Server 2008 Workgroup
	SQL Server 2008 Standard
	SQL Server 2008 Developer
	SQL Server 2008 Enterprise
SQL Server 2008 Express Advanced1	SQL Server 2008 Express Advanced
	SQL Server 2008 Workgroup
	SQL Server 2008 Standard
	SQL Server 2008 Developer
	SQL Server 2008 Enterprise
SQL Server 2008 Express x64 (64 bits)	SQL Server 2008 Express x64 (64 bits)
	SQL Server 2008 Express Tools x64 (64 bits)
	SQL Server 2008 Express Advanced x64 (64 bits)
	SQL Server 2008 Workgroup x64 (64 bits)

	SQL Server 2008 Standard x64 (64 bits)
	SQL Server 2008 Developer x64 (64 bits)
	SQL Server 2008 Enterprise x64 (64 bits)
SQL Server 2008 Express Tools x64 (64 bits)	SQL Server 2008 Express Tools x64 (64 bits)
	SQL Server 2008 Express Advanced x64 (64 bits)
	SQL Server 2008 Workgroup x64 (64 bits)
	SQL Server 2008 Standard x64 (64 bits)
	SQL Server 2008 Developer x64 (64 bits)
	SQL Server 2008 Enterprise x64 (64 bits)
SQL Server 2008 Express Advanced x64 (64 bits)	SQL Server 2008 Express Advanced x64 (64 bits)
	SQL Server 2008 Workgroup x64 (64 bits)
	SQL Server 2008 Standard x64 (64 bits)
	SQL Server 2008 Enterprise x64 (64 bits)
SQL Server 2008 Workgroup1	SQL Server 2008 Workgroup
	SQL Server 2008 Standard
	SQL Server 2008 Enterprise
SQL Server 2008 Web1	SQL Server 2008 Web

SQL Server 2008 Standard1,2	SQL Server 2008 Standard
	SQL Server 2008 Enterprise
SQL Server 2008 Developer1,2	SQL Server 2008 Workgroup
	SQL Server 2008 Standard
	SQL Server 2008 Developer
	SQL Server 2008 Enterprise
SQL Server 2008 Enterprise1,2	SQL Server 2008 Enterprise
SQL Server 2008 Enterprise Evaluation2	SQL Server 2008 Enterprise Evaluation
	SQL Server 2008 Web
	SQL Server 2008 Workgroup
	SQL Server 2008 Standard
	SQL Server 2008 Developer
	SQL Server 2008 Enterprise
SQL Server 2008 IA64 (64 bits) Enterprise Evaluation2	SQL Server 2008 IA64 (64 bits) Enterprise
	SQL Server 2008 IA64 (64 bits) Developer
	SQL Server 2008 IA64 (64 bits) Enterprise Evaluation
SQL Server 2008 x64 (64 bits) Enterprise Evaluation2	SQL Server 2008 Enterprise Evaluation

	SQL Server 2008 Web
	SQL Server 2008 Workgroup
	SQL Server 2008 x64 (64 bits) Standard
	SQL Server 2008 x64 (64 bits) Developer
	SQL Server 2008 x64 (64 bits) Enterprise
SQL Server 2008 IA64 (64 bits) Developer2	SQL Server 2008 IA64 (64 bits) Developer
	SQL Server 2008 IA64 (64 bits) Enterprise
SQL Server 2008 Developer x64 (64 bits)2	SQL Server 2008 Workgroup x64 (64 bits)
	SQL Server 2008 Standard x64 (64 bits)
	SQL Server 2008 Developer x64 (64 bits)
	SQL Server 2008 Enterprise x64 (64 bits)
SQL Server 2008 x64 (64 bits) Standard2	SQL Server 2008 x64 (64 bits) Standard
	SQL Server 2008 x64 (64 bits) Enterprise
SQL Server 2008 IA64 (64 bits) Enterprise2	SQL Server 2008 IA64 (64 bits) Enterprise
SQL Server 2008 x64 (64 bits) Enterprise2	SQL Server 2008 x64 (64 bits) Enterprise

No puede agregar componentes a una instalación existente de SQL Server durante la actualización a SQL Server 2008. Cuando haya actualizado una instancia de SQL Server a SQL Server 2008, podrá agregar características con el Asistente para la instalación de SQL Server 2008: Setup.exe.

Compatibilidad entre idiomas

- La versión en inglés de SQL Server es compatible con todas las versiones traducidas de los sistemas operativos admitidos.
- Las versiones traducidas de SQL Server son compatibles con sistemas operativos traducidos que estén en el mismo idioma que la versión traducida de SQL Server.
- Las versiones localizadas de SQL Server se pueden actualizar a versiones localizadas de SQL Server 2008 del mismo idioma.
- Las versiones localizadas de SQL Server no se pueden actualizar a la versión en inglés de SQL Server 2008.
- Las versiones localizadas de SQL Server no se pueden actualizar a versiones localizadas de SQL Server 2008 de un idioma distinto.
- Las versiones traducidas de SQL Server también son compatibles con las versiones en inglés de los sistemas operativos admitidos mediante la configuración del Paquete de interfaz de OPERADOR multilingüe (MUI) de Windows. No obstante, deberá comprobar algunas configuraciones del sistema operativo antes de instalar una versión traducida de SQL Server en un servidor que ejecute un sistema operativo en inglés con una configuración de MUI que no sea en inglés. Compruebe que las siguientes configuraciones del sistema operativo coinciden con el idioma de SQL Server que desea instalar:
    - Configuración de la interfaz de OPERADOR del sistema operativo
    - Configuración regional del OPERADOR del sistema operativo
    - Configuración regional del sistema

  Si estas configuraciones del sistema operativo no coinciden con el idioma de la versión traducida de SQL Server, deberá establecerlas correctamente antes de instalar SQL Server 2008.

Tal y como habrás visto en la presentación previa del curso, veremos también el lenguaje T-SQL, lenguaje basado en SQL pero específico de Microsoft que nos permitirá diseñar código con mayores posibilidades de lo que ofrece SQL. De igual modo, tampoco es necesario conocimientos de programación ya que iremos viendo todo desde un principio.

Para aquellos alumnos que ya tengan nociones de bases de datos o lenguaje SQL, afiancen y amplíen esos conocimientos y puedan realizar las principales tareas de administración de uno de los servidores preferidos por muchas codigocli, SQL Server 2008. Por otro lado, este curso esta orientado también a aquellas CLEINTES que se dedican al desarrollo de aplicaciones informáticas, tanto páginas web, intranets y programas de escritorio, con el aprendizaje de SQL y la administración de servidores de datos, comprenderán mejor el enlace de sus aplicaciones con las bases de datos y serán capaces de separar el desarrollo de sus aplicaciones de la capa de negocio que supone la parte de la base de datos, mediante el lenguaje T-SQL podrán incluir objetos que realicen tareas que solucionan cantidad de problemas que se plantean durante el desarrollo de aplicaciones, mejorando enormemente la eficacia y la seguridad de las aplicaciones.

Con la aparición de la informática, las codigocli son capaces de gestionar los mismos datos en unas horas que lo que antes gestionaban durante meses. Según se han ido modernizando las características de hardware y software, cualquier empresa puede cubrir la necesidad del control de información de gran valor para su desarrollo y crecimiento de un modo sencillo y rentable para el resultado que obtienen.

Actualmente podemos encontrarnos con varios servidores de base de datos (RDBMS):

Oracle, DB2, MySQL, SQL Server, Y otros

Además la reciente aparición de SQL Server 2008, conlleva que este preparado para la expansión por la red de redes (Internet) ya que por ejemplo es capaz de generar automáticamente Libroos XML, se trata del formato estándar de datos que facilita la transmisión de datos en Internet.

Como veremos en el siguiente punto, tenemos diferentes versiones de SQL Server , cada una orientada a cubrir unas determinadas necesidades de diferentes tipos de codigocli o clientes, pero podemos enumerar una serie de propiedades comunes para todas ellas, que demuestran que SQL Server es bastante más que un servidor de base de datos:

- Servidor de base de datos, de gran rendimiento.
- RDBMS que pueden ser instalados tanto en sistemas de OPERADORs como Windows XP, máquinas de multiprocesador de 64 bits, redes de ordenadores.
- La administración se facilita mediante interfaz gráfica de OPERADOR.
- Capaz de tener varias instancias del servido en una única máquina.
- Acceso directo a datos desde página Web, gracias a la generación automática de Libroos XML, consiguiendo una completa integración con Internet.
- Posibilidades de data warehousing y data mining, para almacenar y analizar datos, funcionando como Online Transaction Processing (OLTP) y con servicios Online Analytical Processing (OLAP).
- Comunicación perfecta con otras aplicaciones Microsoft, pudiendo presentar información en hojas de Excel, por citar un ejemplo.
- Integración perfecta con herramientas de desarrollo de software como Visual Studio 2005.
- Lenguaje T-SQL para ampliar las posibilidades de las tareas a realizar.

Capacidad para interpretar funciones realizadas con CLR (Common Language Runtime) de plataformas .NET, esto nos permite realizar funciones en lenguajes muy conocidos como Visual Basic o C#.

En cualquier caso, si vamos a realizar una instalación sobre otra versión es más que recomendable realizar una copia de seguridad de toda la información, y sobre todo de nuestras bases de datos, para evitar problemas y sorpresas.

Actualizar SQL Server con el Asistente para copiar bases de datos

En este tema se describe cómo utilizar el Asistente para copiar bases de datos para actualizar una base de datos de SQL Server a una versión posterior.
Cuando utilice el Asistente para copiar bases de datos para actualizar una base de datos, tenga en cuenta los siguientes requisitos:

- Antes de proceder a la actualización, asegúrese de que no haya ninguna aplicación o servicio tratando de tener acceso a la base de datos. No utilice el modo de sólo lectura ya que ocasionará un error.
- No se puede cambiar el nombre de la base de datos durante esta operación.

Para actualizar una base de datos de SQL Server 2000 a una versión posterior

1. Conéctese a cualquier instancia de SQL Server Database Engine (Motor de base de datos de SQL Server) utilizando el Explorador de objetos en SQL Server Management Studio.
2. Expanda **Bases de datos**, haga clic con el botón secundario, seleccione **Tareas**y, a continuación, haga clic en **Copiar base de datos**.
3. Complete los pasos del asistente.

Para asegurarse del rendimiento óptimo de una base de datos actualizada,

ejecute sp_updatestats (actualizar estadísticas) en la base de datos actualizada.
Después de utilizar el Asistente para copiar bases de datos con el fin de actualizar una base de datos de SQL Server 2005 o SQL Server 2000 a SQL Server 2008, la base de datos está disponible inmediatamente y se actualiza de forma automática a continuación. Si la base de datos tiene índices de texto completo, el proceso de actualización los importa, los restablece o los vuelve a generar, dependiendo del valor de la propiedad del servidor **Opción de actualización de texto completo**. Si la opción de actualización se establece en **Importar** o en**Volver a generar**, los índices de texto completo no estarán disponibles durante la actualización. Dependiendo de la cantidad de datos que se indicen, la importación puede requerir varias horas y volver a generar puede requerir hasta diez veces más. Observe también que cuando la opción de actualización se establece en **Importar**, si no se dispone de un catálogo de texto completo, se vuelven a generar los índices de texto asociados. Para obtener información sobre cómo ver o cambiar la configuración de la propiedad **Opción de actualización de texto completo**,

Copiar bases de datos con Copia de seguridad y restauración

En SQL Server 2008, se puede crear una base de datos nueva restaurando una copia de seguridad de una base de datos que se creó con SQL Server 2000, SQL Server 2005 o SQL Server 2008. Sin embargo, las copias de seguridad las bases de datos **maestra**, de **modelo** y **msdb** creadas mediante SQL Server 2000 o SQL Server 2005 no pueden restaurarse con SQL Server 2008. Asimismo, las copias de seguridad de SQL Server 2008 no se pueden restaurar con versiones anteriores de SQL Server.

El formato de las copias de seguridad de bases de datos creadas mediante SQL Server 7.0 o versiones anteriores no es compatible y, por lo tanto, estas bases de datos no pueden restaurarse en SQL Server 2008. Para obtener información acerca de cómo migrar una base de datos creada mediante SQL Server 6.5 o versiones anteriores en SQL Server 2005,

SQL Server 2008 utiliza una ruta de acceso predeterminada distinta a la de las versiones anteriores. Por lo tanto, para restaurar una base de datos creada en la ubicación predeterminada de SQL Server 2000 o SQL Server 2005 a partir de las copias de seguridad, es preciso utilizar la opción MOVE. Para obtener información acerca de la nueva ruta de acceso predeterminada,

Pasos generales para utilizar las funciones de copia de seguridad o restauración para copiar una base de datos

Cuando se utiliza la copia de seguridad o la restauración para copiar una base de datos a otra versión de SQL Server, los equipos de origen y de destino pueden ser de cualquier plataforma en la que se ejecute SQL Server.

Los pasos generales son:

1. Cree una copia de seguridad de la base de datos de origen que puede alojarse en una instancia de SQL Server 2000, SQL Server 2005 o SQL Server 2008. El equipo en el que se ejecute esta versión de SQL Server será el equipo de origen.
2. En el equipo al que desee copiar la base de datos (el equipo de destino), conéctese a una sesión de SQL Server en la que tenga previsto restaurar la base de datos. Si es necesario, cree en la instancia de servidor de destino los mismos dispositivos de copia de seguridad utilizados para la copia de seguridad de las bases de datos de origen.
3. Restaure la copia de seguridad de la base de datos de origen en el equipo de destino. Al restaurar la base de datos se crean automáticamente todos los archivos de la base de datos.

En los siguientes temas se abordan aspectos adicionales que pueden afectar al proceso.

ANTES DE RESTAURAR LOS ARCHIVOS DE BASE DE DATOS

La restauración de una base de datos crea automáticamente los archivos necesarios para la base de datos que se restaura. De forma predeterminada, los archivos que crea SQL Server durante el proceso de restauración utilizan el mismo nombre y las mismas rutas de acceso que los archivos de la base de datos original en el equipo de origen. Para evitar errores y consecuencias no deseadas, determine los archivos que se crean de forma automática al realizar la restauración antes de ejecutarla porque:

- Es posible que los nombres de archivos ya existan en el equipo, lo que provocará un error.
- Es posible que no haya espacio suficiente en la ubicación de destino.
- Es posible que la estructura de directorios o asignación de unidades no exista en el equipo.
  Por ejemplo, la copia de seguridad contiene un archivo que es necesario restaurar en la unidad E:, pero el equipo de destino no contiene una unidad E:.
- Si se pueden reemplazar los archivos de la base de datos, se sobrescriben las bases de datos y archivos existentes que tengan los mismos nombres en la copia de seguridad, a menos que dichos archivos pertenezcan a una base de datos diferente.

Tenga en cuenta que si reutiliza un nombre de base de datos y un destino existentes cuyos archivos se puedan sobrescribir, se sobrescribirán todos los archivos existentes cuyo nombre sea idéntico al de los de la copia de seguridad.

Si es preciso, se puede especificar la asignación de dispositivos, los nombres de archivo o la ruta de acceso para restaurar una base de datos.

MOVER LOS ARCHIVOS DE BASE DE DATOS

Si no se puede restaurar los archivos de la copia de seguridad de la base de datos en el equipo de destino debido a las raLugars mencionadas anteriormente, es necesario mover los archivos a una nueva ubicación a medida que se restauran. Por ejemplo:

- Suponga que desea restaurar una base de datos a partir de las copias de seguridad creadas en la ubicación predeterminada de SQL Server 2000 o SQL Server 2005.
- Puede ser necesario restaurar algunos archivos de la base de datos de la copia de seguridad en una unidad diferente debido a consideraciones de capacidad. Probablemente se trate de un hecho frecuente, porque la mayor parte de los equipos de una organización no tienen el mismo número y tamaño de unidades de disco o idénticas configuraciones de software.
- Puede ser necesario crear una copia de una base de datos existente en el mismo equipo para realizar pruebas. En este caso, los archivos de la base de datos original ya existen, por lo que se necesita especificar diferentes nombres de archivo al crear la copia de la base de datos durante la operación de restauración.

Cambiar el nombre de la base de datos

Se puede cambiar el nombre de la base de datos cuando se restaura en el equipo de destino, sin necesidad de restaurar primero la base de datos y después cambiar manualmente el nombre. Por ejemplo, es posible que sea necesario cambiar el nombre de la base de datos de **Contenidos** a **ContenidosCopy** para indicar que se trata de una copia de la base de datos.
El nombre de base de datos que se proporciona explícitamente al restaurar una base de datos se utiliza de forma automática como el nuevo nombre de la base de datos. Debido a que el nombre de la base de datos no existe, se crea uno nuevo con los archivos de la copia de seguridad.

Actualizar una base de datos utilizando la restauración

Al restaurar copias de seguridad de SQL Server 2000 o SQL Server 2005, es útil conocer de antemano si la ruta de acceso (unidad y directorio) de cada uno de los catálogos de texto completo de una copia de seguridad existe en el equipo de destino. Para obtener una lista de los nombres lógicos y físicos, la ruta y el nombre de archivo de todos los archivos de una copia de seguridad, incluidos los archivos de catálogo, utilice una instrucción RESTORE FILELISTONLY FROM <backup_device>.
Si no existe la misma ruta de acceso en el equipo de destino, son dos las alternativas válidas:

- Cree la asignación de unidades/directorios equivalente en el equipo de destino.
- Mueva los archivos de catálogo a una ubicación nueva durante la operación de restauración con la cláusula WITH MOVE de la instrucción RESTORE DATABASE.

Propiedad de la base de datos

Cuando se restaura una base de datos en otro equipo, el inicio de sesión de SQL Server o el OPERADOR de Microsoft Windows que inicia la operación de restauración se convierte automáticamente en el propietario de la nueva base de datos. Una vez restaurada la base de datos, el administrador del sistema o el nuevo propietario de la base de datos pueden cambiar la propiedad de la base de datos. Para evitar restauraciones no autorizadas de una base de datos, utilice contraseñas en los medios o en el conjunto de copia de seguridad. .

Administrar metadatos al restaurar una base de datos en otra instancia de servidor

Al restaurar una base de datos en otra instancia de servidor, para proporcionar una experiencia coherente a los OPERADORs y las aplicaciones, puede que tenga que volver a crear algunos o todos los metadatos de la base de datos, por ejemplo los inicios de sesión y los trabajos, en la otra instancia de servidor.

Copiar bases de datos de SQL Server 7.0 o anterior

Al instalar SQL Server 2008, se actualizarán automáticamente las bases de datos

existentes. Para copiar una base de datos actualizada, puede usar cualquiera de los métodos de copia compatibles con las bases de datos de SQL Server 2008.
Para obtener información sobre cómo usar una base de datos de SQL Server 7.0, SQL

Bases de datos de SQL Server 7.0

Puede convertir una base de datos de SQL Server versión 7.0 en SQL Server 2008 usando uno de los métodos siguientes:

- Para actualizar una base de datos de SQL Server 7.0 a SQL Server 2000 o SQL Server 2005, adjunte la base de datos a una instancia que ejecute cualquiera de dichas versiones. A continuación, puede actualizar la base de datos a SQL Server 2008. Generalmente éste es el método preferido.
  Para obtener información sobre cómo usar la operación de adjuntar para actualizar una base de datos de SQL Server 2000 o SQL Server 2005,.
- Utilice el Asistente para importación y exportación de SQL Server para copiar los datos entre varias instancias de SQL Server. Este asistente trabaja con cualquier origen y destino para los que exista un proveedor, aunque pueden producirse problemas en la conversión de datos dependiendo del origen de éstos.
- Para migrar los datos de una base de datos creada en SQL Server 7.0, realice las operaciones siguientes:

    1. Utilice la versión 7.0 de **bcp** para exportar los datos a un archivo de datos mediante un comando **bcpout**.

    2. Utilice la versión de SQL Server 2008 (versión 10.0) de **bcp**, para importar los datos del archivo de datos mediante un comando **bcpin**. Si dicho archivo contiene formatos de datos nativos, especifique las opciones **-V70** y **–n**, que indican al comando **bcp in** que debe usar los tipos de datos nativos de SQL Server 7.0.

Bases de datos de SQL Server 6.0 o SQL Server 6.5

Para migrar los datos de una base de datos de SQL Server versión 6.0 o SQL Server versión 6.5, use el programa **bcp** de dicha versión de SQL Server para exportar los datos a un archivo en modo de carácter (**bcpout**). A continuación, podrá importar los datos de caracteres en una base de datos de SQL Server 2008. Sin embargo, SQL Server 2008 no admite los formatos de datos nativos de SQL Server 6.0 y SQL Server 6.5. Esto significa que la versión de bcp.exe de SQL Server 2008 no admite la opción de línea de comandos **-6**, ni las opciones **60** y **65** de la opción de línea de comandos **–V**.
El formato de las copias de seguridad de bases de datos creadas mediante SQL Server 6.5 o versiones anteriores no es compatible y, por lo tanto, estas bases de datos no pueden restaurarse en SQL Server 2005 ni en versiones posteriores

Nivel de compatibilidad de la base de datos después de actualizar

Los niveles de compatibilidad de las bases de datos **tempdb**, **model**, **msdb** y **Resource** quedan establecidos en 100 después de la actualización.
La base de datos **maestra** del sistema conserva el nivel de compatibilidad que tenía antes de la actualización, a menos que dicho nivel sea inferior a 80. Si el nivel de compatibilidad de la base de datos **maestra** era inferior a 80 antes de la actualización, se establece en 80 después de la misma.

Si el nivel de compatibilidad de una base de datos de OPERADOR era 80 o 90 antes de la actualización, permanece igual después de la misma. Si el nivel de compatibilidad era igual o inferior a 70 antes de la actualización, en la base de datos actualizada, el nivel de compatibilidad se establece en 80, que es el nivel de compatibilidad mínimo admitido en SQL Server 2008.

Las nuevas bases de datos de OPERADOR heredarán el nivel de compatibilidad de la base de datos **model**.

CAPITULO II

BASE DE DATOS

El Database Engine (Motor de base de datos) es el servicio principal para almacenar, procesar y proteger datos. El Database Engine (Motor de base de datos) proporciona acceso controlado y procesamiento de transacciones rápido para cumplir con los requisitos de las aplicaciones consumidoras de datos más exigentes de su empresa.
Use Database Engine (Motor de base de datos) para crear bases de datos relacionales para el procesamiento de transacciones en línea o datos de procesamiento analítico en línea. Esto incluye la creación de tablas para almacenar datos y objetos de base de datos (p.ej., índices, vistas y procedimientos almacenados) para ver, administrar y proteger datos. Puede usar SQL Server Management Studio para administrar los objetos de bases de datos y SQL Server Profiler para capturar eventos de servidor.

El concepto más general de una base de datos es el lugar donde se guardan los datos.

- Campo: Contiene un dato en particular, como puede ser el primer punto que hace referencia al precio de un libro.
- Registro: Almacena todos los datos de un determinado objeto de información, vemos que el segundo punto de nuestras necesidades reclama los aspectos más importantes de un libro. En este caso, el libro es el objeto de información, y sus aspectos (Título, Autor, ISBN, Páginas,...) de ese objeto de información serían un grupo de campos, al igual que sucede con el precio.
- Tabla: Almacena información de varios objetos de información que comparten aspectos similares. Estamos mencionando el tercer punto de nuestra librería, donde queremos almacenar la información de todos los libro, podemos pensar, pero cada libro es diferente al resto, y es cierto, pero todos los libros tienen en común que cada uno de ellos tiene un determinado Título, Autor, ISBN, Páginas, Género, etc... Por lo tanto, si hemos entendido bien, los conceptos de los dos anteriores niveles, podemos asegurar que una tabla almacena una serie de registros (libros).

- Base de datos: Cuarto y último nivel, de nuestro primer vistazo a la idea de base de datos, relacionada con el cuarto punto de nuestra librería el cual nos indica que queremos almacenar los aspectos de la empresa al completo, por lo tanto, este nivel guarda información de varios aspectos, no sólo de libros, sino de MOVIMIENTOS, compras, clientes etc...Por lo tanto la base de datos, dicho de un modo muy simple y muy genérico, almacena las tablas.

Acabamos de mencionar los cuatro conceptos básicos de toda base de datos, si es la primera vez que te introduces en este mundillo deben quedarte muy claros estos cuatro pilares de información.

Tal y como hemos avisado, esta definición de base de datos es demasiado simple, decir que la base de datos se encarga de almacenar la información estructurada en esos cuatro niveles es decir demasiado poco. Muchos fabricantes ofrecen en sus servidores la posibilidad de almacenar muchas mas funcionalidades que estas cuatro.

Microsoft SQL Server 2008ofrece una cantidad enorme de objetos, que al igual que los datos se almacenan en la base de datos, pero cuya función no es guardar información, sino trabajar con ella. Así a primera vista, puede parecer complicado, ¿Una base de datos almacena algo más que datos? Veremos que así es, y que son de una importancia grandísima, ya que tienen tareas tan importantes como asegurar que esos datos se almacenan correctamente, de la seguridad, del rendimiento que obtenemos de esos datos, etc...Pero como te digo, los iremos viendo a lo largo del curso.

ESTRUCTURACIÓN DE UNA BASE DE DATOS

Estructura física

Una base de datos se almacena en varios ficheros o archivos en disco. Como mínimo tendremos dos ficheros que explicaremos más adelante.
Tenemos la posibilidad de almacenar estos ficheros en discos que no estén ni tan siquiera formateados o que no tengan una partición hecha, pero este método no es el más aconsejable. Es más razonable almacenar estos archivos en un disco ya formateado, con formato NTFS.
En codigocli cuyo volumen de datos es altísimo y el trabajo que se realiza sobre la base de datos soporta una actividad elevada, se almacenan los archivos en grupos de discos denominados RAID por hardware. Este método mejora considerablemente el rendimiento, y nos asegura que en caso de fallos inesperados no perdamos esa valiosa información.

Como es lógico, nosotros para realizar nuestros ejemplos, no vamos a basarnos en esta tipo de estructuras de hardware, lo almacenaremos en nuestro disco duro, aunque veremos como asegurar nuestros datos mediante planes de mantenimiento con copias de seguridad automáticas.

Como hemos mencionado, como mínimo tendremos dos archivos donde almacenar la base de datos:

- Archivo de datos.
- Archivo de registro de transacciones.

Pero debes saber que tenemos otras posibilidades y podemos utilizar archivos extras para mejorar el rendimiento de nuestra base de datos, podemos usar varios archivos, si

pensamos que nuestra base de datos va a alcanzar un tamaño grande. O si deseamos que nuestros datos se almacenen en diferentes dispositivos de almacenamiento u ordenadores, y de este modo permitir un trabajo más rápido al poder acceder a la información en paralelo.

Centrándonos en lo principal:

- El archivo de datos, o aquellos que añadimos como extras, son los archivos que tendrán almacenada la información, los datos. Pero recuerda que hemos dicho que SQL Server 2008 nos permite también crear en nuestras bases de datos, no sólo información, sino también una serie de objetos que trabajan con la información. Pues bien, esta serie de objetos también se almacena en el archivo de datos.
- Por otro lado, tenemos el archivo de registro de transacciones. Este fichero es tan importante como el anterior. Su importante tarea es garantizar que esa base de datos permanece integra. Gracias a estos archivos de registros (puede haber más de uno), en caso de ser necesario, podremos recuperar la base de datos, ya que almacena las modificaciones que se producen debido a la actividad o la explotación de la base de datos.

Nombres de archivos.

El modo de nombrar una base de datos, parte de una base fija, de un nombre principal que generalmente entrega el administrador de la base de datos. Una vez que tenemos este nombre principal, SQL Server 2008 se encarga de añadir terminaciones y unas determinadas extensiones, a ese nombre principal. El administrador además de seleccionar el nombre principal, puede elegir el destino donde se almacenarán los ficheros que forman la base de datos.

Vamos a suponer que estamos en una empresa como administradores, y estamos creando su base de datos. Nosotros como administradores le damos el nombre principal " miEmpresa ". Ese será el nombre de la base de datos, pero los ficheros donde se almacenará su información y el registro de transacciones, serán:

- Archivo de datos: miEmpresa_Data.MDF
- Archivo de registro de tranasacciones: miEmpresa_Log.LDF

En caso de tener archivos extras, nosotros como administradores también podremos darles su nombre principal, y la extensión que suele utilizarse es .NDF

Siguiendo con nuestra tarea de administrador, ahora sería el momento de seleccionar el lugar de almacenamiento, como ya sabes podemos seleccionar una determinada carpeta o directorio, incluso diferentes unidades físicas. Lo más aconsejable es guardar en diferentes unidades, por un lado el archivo de datos, y por otro el archivo de registro de transacciones. De modo que en caso de fallo, por lo menos tengamos uno de ellos.

A continuación puedes ver una figura que representa la estructura física de la base de datos, tomando como ejemplo el nombre principal "MiEmpresa".

No debes quedarte con la idea de que una base de datos, se compone sencillamente de dos archivos, es algo mucho más completo que todo eso lo que representa una base de datos como entidad.

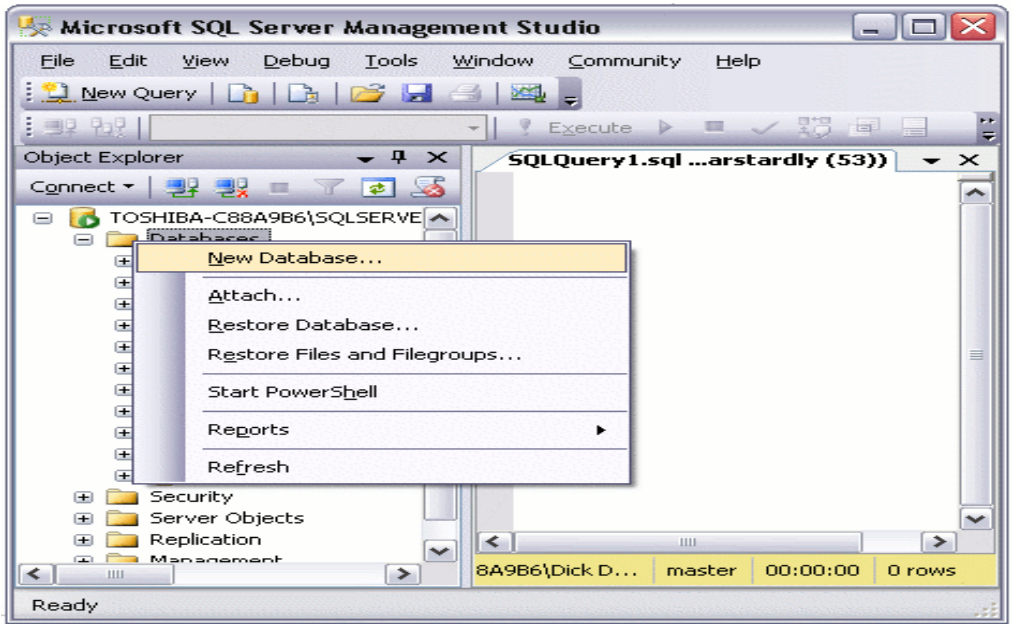
Tamaño de la base de datos.

En el momento de crear la base de datos, es casi imposible conocer la cantidad de memoria que necesitará para almacenar toda la información. Es cierto que hay ciertas técnicas que nos permiten calcular el tamaño que podrá alcanzar la base de datos, pero estas estimaciones pueden venirse a bajo, por modificaciones imprevistas, como puede ser el crecimiento de la empresa y que se intensifique la actividad realizada sobre la información, por citar un ejemplo.
Tampoco es nada aconsejable pecar de precavidos y reservar una cantidad de memoria exagerada, y pensar que con esta cantidad casi infinita no tendremos problemas de espacio para nuestros datos. De acuerdo, puede que no haya problemas de espacio (o quizá si), pero lo que es seguro es que tendremos muchísimos problemas de rendimiento, de fragmentación etc...
SQL Server 2008 nos permite olvidarnos hasta cierto punto de este problema. Los archivos de datos y de registro, crecen automáticamente. No crecen con cada dato que se añade. Nosotros como administradores, le daremos un tamaño inicial sencillo de estimar (una cantidad muy pequeña, unos Megabytes), en ese momento SQL Server 2008 crea la estructura correcta para la base de datos, y una vez que nuestra base de datos está en explotación cuando alcanza el tamaño limite, lo incrementa una cantidad dada por un factor predeterminado.

Estructura lógica

Para entender que es la estructura lógica de una base de datos vamos a poner un sencillo ejemplo.
Cuando nosotros nos compramos un equipo de música, poco nos importa como funcionan los circuitos integrados, los elementos electrónicos que componen nuestro equipo. En este caso, esos circuitos, esos dispositivos electrónicos, sería la estructura física del equipo de música, al igual que hemos visto la estructura física de nuestra base de datos.
A lo que nosotros como OPERADORs vamos a dar importancia es al manejo del equipo de música: como subir el volumen, encenderlo, cambiar de emisoras, introducir un CD. De igual modo, como OPERADORs de la base datos, debemos conocer la estructura lógica de la base de datos para poder gestionar o trabajar con los datos.
Una estructura lógica mínima puede ser el ejemplo de la librería que hemos visto a modo de introducción en esta lección.

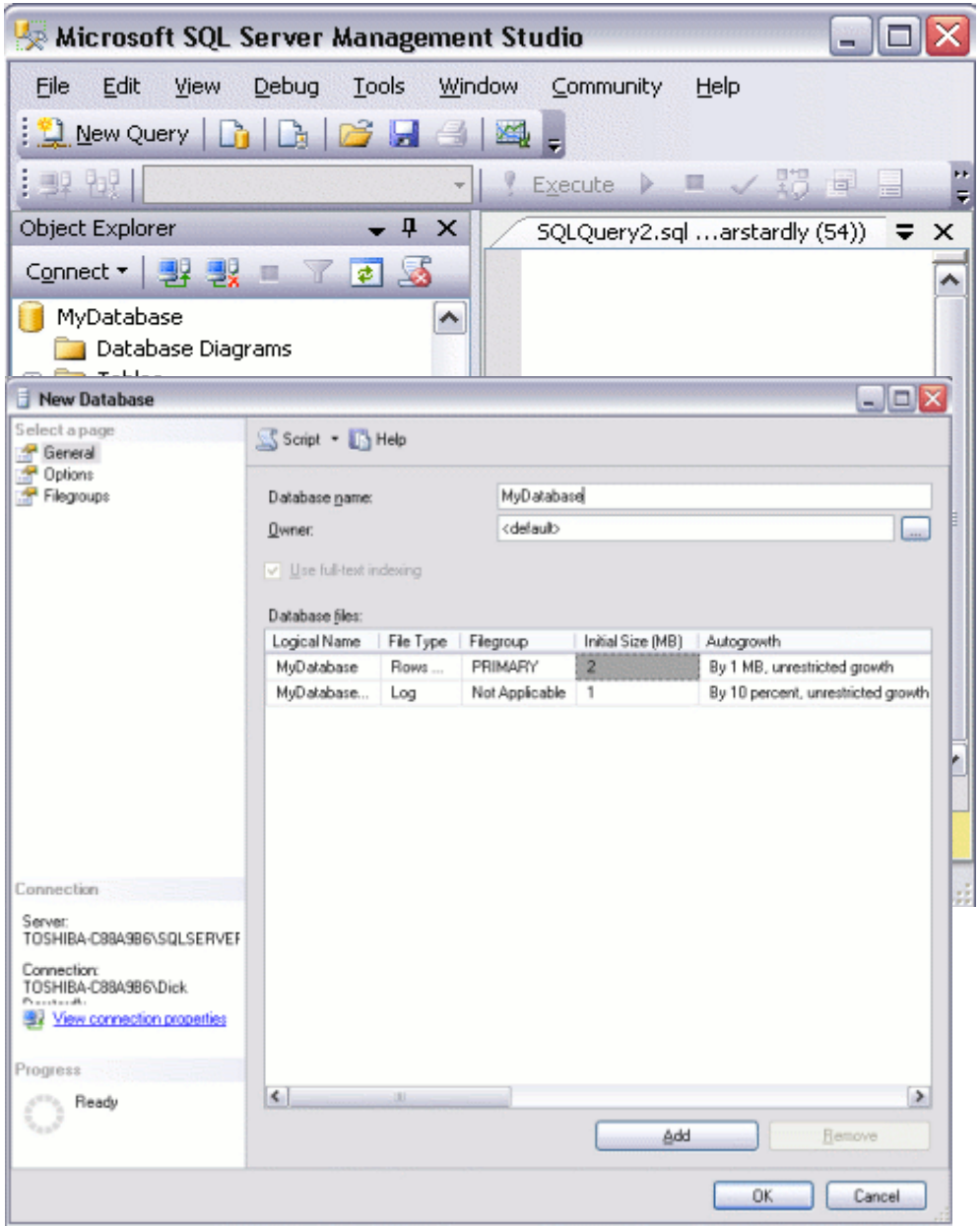
Lo que vamos a exponer a continuación a modo de introducción son los elementos principales que componen la estructura lógica de una base de datos, de modo que sepas de que estamos hablando en caso de que se mencionen en las diferentes lecciones. Sin embargo, los iremos viendo con más detenimiento más adelante, de momento es suficiente con que te suenen y las vayas conociendo.



Los pasos siguientes muestran como creamos una base de datos usando SQL Server Management Studio.

1. Dar click derecho en la opción "Databases" y seleccionar "New Database..."
2. Luego dar click sobre el nombre de la Base de datos

Ahora se dará cuenta de su nueva base de datos aparece en la "Base de datos" de SQL Server Management Studio.

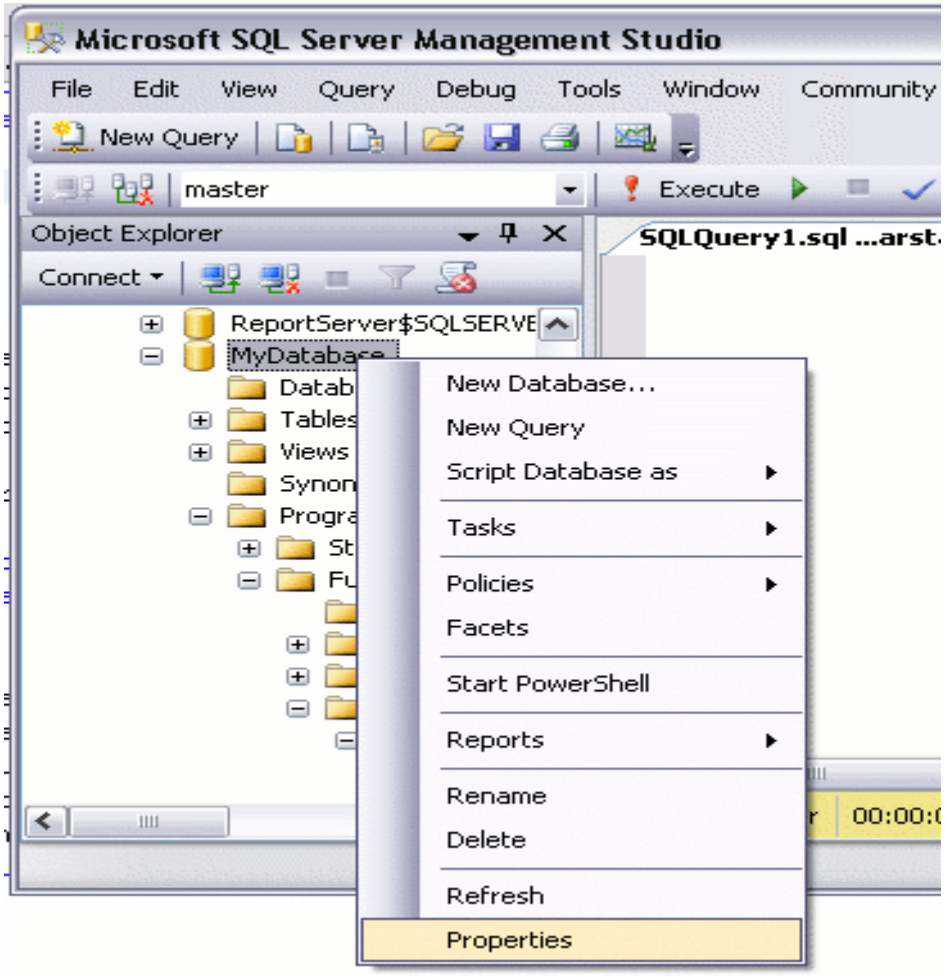


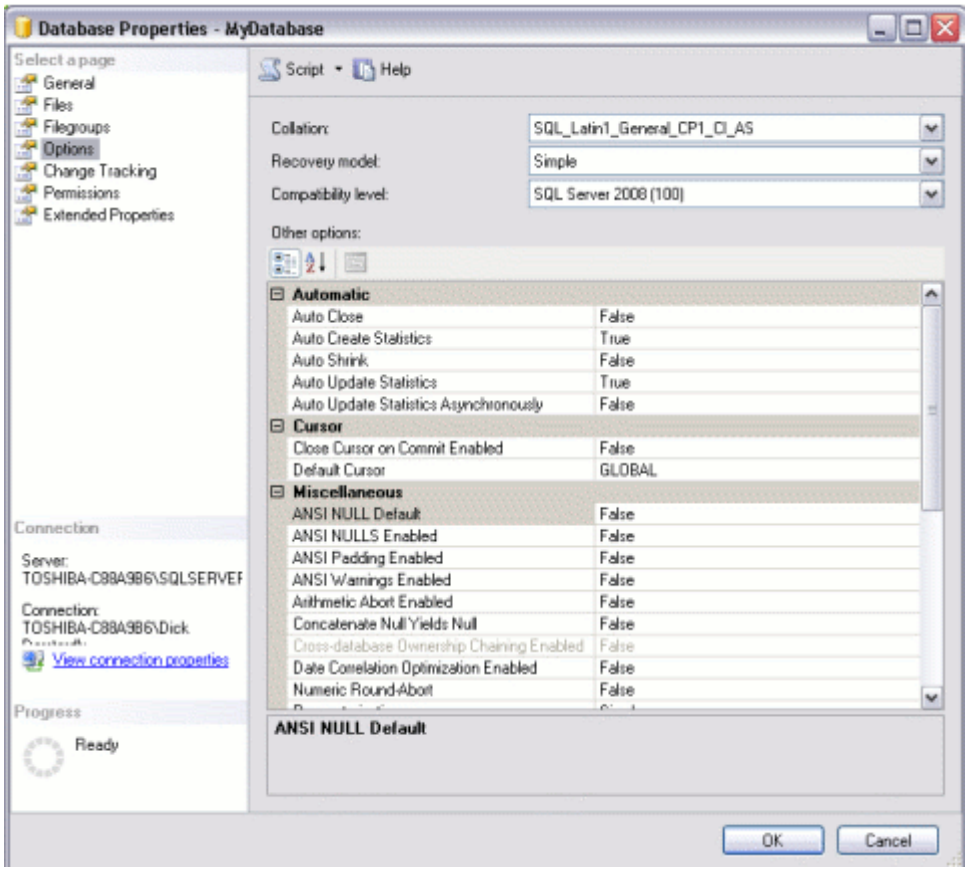
Su nueva base de datos se basa en el modelo de "base de datos. La base de datos de modelo es una base de datos del sistema que se utiliza como una plantilla cada vez que una nueva base de datos se crea. Si utiliza el panel de la izquierda para navegar hasta su base de datos y expanda el árbol, te darás cuenta de que su base de datos ya contiene una serie de objetos.Por ejemplo, ya contiene las funciones del sistema, las vistas del sistema, procedimientos almacenados del sistema, y (oculta) las tablas del sistema. Estos son los objetos del sistema que proporcionan información sobre la base de datos.

Acabamos de crear una base de datos utilizando las opciones predeterminadas. Cuando creamos la base de datos, un "Archivo de datos" y un "registro de transacciones" fueron creadas. Fueron creados en la ubicación por defecto para nuestro servidor.

Si hubiéramos querido, podríamos haber especificado una ubicación diferente para estos archivos. También podría haber cambiado las especificaciones para permitir o no el archivo de crecer de forma automática (como almacenes de datos más y más), y en caso afirmativo, ¿cómo que el crecimiento debe ser administrado. Podríamos haber hecho eso en el paso 2.Pero no todo está perdido. Todavía podemos hacerlo ahora que hemos creado la base de datos.Podemos hacerlo a través del cuadro de diálogo Propiedades.

Para ver o cambiar las propiedades de base de datos, simplemente haga clic derecho sobre la base de datos y seleccionar "Propiedades:





El cuadro de diálogo Propiedades contiene un gran número de opciones para cambiar la configuración de su base de datos. Por ahora, podemos dejar todo en su configuración por defecto.

SINTAXIS DE CREAR UNA BASE DE DATOS

```
CREATE DATABASE database_Nombres
    [ ON
        [ PRIMARY ] [ <filespec> [ ,...n ]
        [ , <filegroup> [ ,...n ] ]
    [ LOG ON { <filespec> [ ,...n ] } ]
    ]
    [ COLLATE collation_Nombres ]
    [ WITH <external_access_option> ]
]
[;]

To attach a database
CREATE DATABASE database_Nombres
    ON <filespec> [ ,...n ]
    FOR { ATTACH [ WITH <service_broker_option> ]
        | ATTACH_REBUILD_LOG }
[;]

<filespec> ::=
{
(
    NOMBRES = logical_file_Nombres ,
        UBICACION = { 'os_file_Nombres' | 'filestream_path' }
        [ , SIZE = size [ KB | MB | GB | TB ] ]
        [ , MAXSIZE = { max_size [ KB | MB | GB | TB ] | UNLIMITED } ]
        [ , FILEGROWTH = growth_increment [ KB | MB | GB | TB | % ] ]
) [ ,...n ]
}

<filegroup> ::=
{
FILEGROUP filegroup_Nombres [ CONTAINS FILESTREAM ] [ DEFAULT ]
    <filespec> [ ,...n ]
}

<external_access_option> ::=
{
  [ DB_CHAINING { ON | OFF } ]
  [ , TRUSTWORTHY { ON | OFF } ]
}
<service_broker_option> ::=
{
    ENABLE_BROKER
  | NEW_BROKER
  | ERROR_BROKER_CONVERSATIONS
}

Create a database snapshot
CREATE DATABASE database_snapshot_Nombres
    ON
        (
        NOMBRES = logical_file_Nombres,
        UBICACION = 'os_file_Nombres'
        ) [ ,...n ]
    AS SNAPSHOT OF source_database_Nombres
[;]
```

database_Nombres

Es el nombre de la nueva base de datos. nombres de base de datos debe ser único dentro de una instancia de SQL Server y cumplir con las reglas de los identificadores.

database_Nombres puede ser un máximo de 128 caracteres, a menos que un nombre lógico, no se especifica para el archivo de registro. Si un nombre de archivo de registro lógico no se especifica, SQL Server genera el logical_file_Nombres y el os_file_Nombres para el registro añadiendo un sufijo a database_Nombres. Esto limita a 123 caracteres database_Nombres para que el nombre de archivo generado lógica no es más que 128 caracteres.

Si los datos de nombre de archivo no se especifica, SQL Server utiliza database_Nombres ya que tanto el logical_file_Nombres y como el os_file_Nombres.

ON

Especifica que los archivos de disco utilizado para almacenar las secciones de datos de la base de datos, archivos de datos, se definen explícitamente. ON se requiere cuando es seguida por una lista separada por comas de <filespec> artículos que definen los archivos de datos para el grupo de archivos primario. La lista de archivos del grupo de archivos principal puede ser seguido por una lista opcional, separada por comas de <filegroup> artículos que definen los grupos de archivos de OPERADOR y sus archivos.

PRIMARIA

Especifica que la lista de asociados <filespec> define el archivo principal. El primer archivo especificado en la entrada <filespec> del grupo de archivos principal se convierte en el archivo principal. Una base de datos sólo puede tener un archivo principal. Para obtener más información,

Si no se especifica PRIMARIA, el primer archivo enumerados en la instrucción CREATE DATABASE se convierte en el archivo principal.

LOG ON

Especifica que los archivos de disco utilizado para almacenar el registro de base de datos, archivos de registro, se definen explícitamente. LOG ON es seguido por una lista separada por comas de <filespec> artículos que definen los archivos de registro. Si LOG ON no se especifica un archivo de registro se crea automáticamente que tiene un tamaño que es un 25 por ciento de la suma de los tamaños de todos los archivos de datos para la base de datos o 512 KB, lo que es más grande. LOG ON no se puede especificar en una instantánea de base de datos.

COLLATE

Especifica la colación por defecto para la base de datos. nombre de intercalación puede ser un nombre de intercalación de Windows o un nombre de intercalación de SQL. Si no se especifica, la base de datos se le asigna la intercalación predeterminada de la instancia de SQL Server. A nombre de la colación no se puede especificar en una instantánea de base de datos.

A nombre de la colación no se puede especificar con el PARA COLOCAR DE cláusulas o ATTACH_REBUILD_LOG. Para obtener información acerca de cómo cambiar la intercalación de una base de datos adjunta, visite este sitio Web de Microsoft.

Para obtener más información acerca de Windows y los nombres de intercalación SQL, vea COLLATE (Transact-SQL).

PARA COLOCAR

Especifica que la base de datos se crea adjuntando un conjunto existente de archivos del sistema operativo. Debe haber una entrada <filespec> que especifica el archivo principal. Las entradas sólo otros <filespec> requeridos son los de los archivos que tienen un camino diferente de cuando la base de datos fue creada el pasado o se adjunta. Una entrada <filespec> se debe especificar para estos archivos.

PARA COLOCAR requiere lo siguiente:

Todos los archivos de datos (MDF y NDF) deben estar disponibles. Si existen varios archivos de registro, todos ellos deben estar disponibles. Si una lectura / escritura de base de datos tiene un único archivo de registro que no está ahora disponible, y si la base de datos se cerró sin OPERADORs o transacciones abiertas antes de la operación de adjuntar, PARA COLOCAR reconstruye automáticamente el archivo de registro y actualiza el archivo principal. En cambio, para una base de datos de sólo lectura, el registro no puede ser reconstruido debido a que el archivo principal no se puede actualizar. Por lo tanto, al adjuntar una base de datos de sólo lectura cuyo registro no está disponible, usted debe proporcionar los archivos de registro o en la cláusula FOR ATTACH.

Nota:

Una base de datos creada por una versión más reciente de SQL Server no puede adjuntarse en versiones anteriores. La fuente de datos debe ser al menos la versión 80 (SQL Server 2000) para conectar a SQL Server 2008. SQL Server 2000 o SQL Server 2005 las bases de datos que tienen un nivel de compatibilidad inferior a 80 se establecerá en compatibilidad 80 cuando se adjuntan.

En SQL Server, los archivos de texto que forman parte de la base de datos que se adjunta se adjuntará con la base de datos. Para especificar una nueva ruta de acceso del catálogo de texto, especifique la nueva ubicación sin el nombre del archivo de texto del sistema operativo.

Nota de seguridad:

Le recomendamos que no adjuntar bases de datos de fuentes desconocidas o no confiables.Estas bases de datos podría contener código malicioso que podría ejecutar código Transact-SQL no deseado o provocar errores al modificar el esquema o la estructura de base de datos física. Antes de utilizar una base de datos desde un origen desconocido o no es de confianza, ejecute DBCC CHECKDB en la base de datos en un servidor de no producción, así como examinar el código, como procedimientos almacenados u otro código definido por el OPERADOR, en la base de datos.
Para obtener más información acerca de cómo adjuntar y separar bases de datos, vea Separar y adjuntar bases de datos.

Nota:

Si la base de datos utiliza Service Broker, vea también <service_broker_option>.

Para obtener información sobre los permisos de archivos que se establecen cada vez que una base de datos se separa y adjunto, vea Proteger los datos y archivos de registro.

Al adjuntar una base de datos replicada que fue copiada en lugar de desprenderse de ser, considere lo siguiente:

Si adjunta la base de datos a la misma instancia de servidor y la versión como la base de datos original, no se requieren pasos adicionales.

Si adjunta la base de datos a la misma instancia de servidor, pero con una versión actualizada, debe ejecutar sp_vupgrade_replication para mejorar la replicación después de la operación de colocar se ha completado.

Si adjunta la base de datos a una instancia de servidor diferente, independientemente de la versión, debe ejecutar sp_removedbreplication para quitar la replicación después de la operación de colocar se ha completado.

Nota:

Adjuntar trabaja con el formato de almacenamiento vardecimal, pero el SQL Server Database Engine se debe actualizar por lo menos a SQL Server 2005 Service Pack 2. No puede adjuntar una base de datos utilizando el formato de almacenamiento vardecimal a una versión anterior de SQL Server. Para obtener más información sobre el formato de almacenamiento vardecimal, vea Almacenar datos decimales como longitud variable.

Para obtener información acerca de cómo actualizar una base de datos mediante el uso de adjuntar, vea Cómo actualizar una base de datos mediante Separar y Adjuntar (Transact-SQL).

PARA ATTACH_REBUILD_LOG

Especifica que la base de datos se crea adjuntando un conjunto existente de archivos del sistema operativo. Esta opción se limita a leer y escribir bases de datos. Si uno o más archivos de registro de transacciones se ha omitido, el archivo de registro se vuelve a generar. Debe haber una entrada <filespec> especificando el archivo principal.

Nota:

Si los archivos de registro están disponibles, el motor de base de datos va a utilizar esos archivos en lugar de reconstruir los archivos de registro. PARA ATTACH_REBUILD_LOG requiere lo siguiente:

Un cierre correcto de la base de datos.

Todos los archivos de datos (MDF y NDF) deben estar disponibles.

Importante:

Esta operación rompe la cadena de copia de seguridad de registro. Se recomienda que una copia de seguridad completa se realizó después de la operación se ha completado. Para obtener más información, vea BACKUP (Transact-SQL). Típicamente, para ATTACH_REBUILD_LOG se usa cuando se copia una lectura / escritura de base de datos con un registro de gran tamaño a otro servidor donde se va a los textos que utilizan en su mayoría, o sólo las operaciones, para leer, y por lo tanto requieren menos espacio de registro de la base de datos original. PARA ATTACH_REBUILD_LOG no se puede especificar en una instantánea de base de datos.

<filespec>

Controla las propiedades del archivo.

NOMBRE logical_file_Nombres

Especifica el nombre lógico para el archivo. NOMBRES es necesario cuando se especifica FICHERO, excepto cuando se especifica una de las cláusulas FOR ATTACH. Un grupo de archivos FILESTREAM no se puede nombrar PRIMARIA.

logical_file_Nombres
Es el nombre lógico utilizado en SQL Server cuando se hace referencia al archivo.Logical_file_Nombres debe ser único en la base de datos y cumplir con las reglas de los identificadores. El nombre puede ser un carácter Unicode o constante, o un identificador regular o delimitado.

NOMBREARCHIVO ('os_file_Nombres' filestream_path | '')
Especifica el sistema operativo (física) de nombre de archivo.

"Os_file_Nombres '
Es la ruta y el nombre utilizado por el sistema operativo cuando se crea el archivo. El archivo debe residir en uno de los siguientes dispositivos: el servidor local en el que está instalado SQL Server, una Storage Area Network [SAN], o de una red basada en iSCSI. La ruta especificada debe existir antes de ejecutar la instrucción CREATE DATABASE. Para obtener más información, consulte "Base de datos de archivos y grupos de archivos" en la sección Notas.

SIZE, MAXSIZE y FILEGROWTH
Parámetros no se puede establecer cuando una ruta de acceso UNC se especifica para el archivo.
Si el archivo está en una partición primas, os_file_Nombres debe especificar sólo la letra de unidad de una partición primas existentes. Sólo un archivo de datos se pueden crear en cada partición primas.
Los archivos de datos no pueden ser ejecutados en sistemas de archivos comprimidos a menos que los archivos son archivos de sólo lectura secundaria, o la base de datos es de sólo lectura. Los archivos de registro no debe ser puesto en sistemas de archivos comprimidos.Para obtener más información, consulte grupos de archivos de sólo lectura y compresión.
"Filestream_path "
Para un grupo de archivos FILESTREAM, UBICACION hace referencia a una ruta en la que los datos de FILESTREAM se almacenarán. El camino hasta la última carpeta debe existir, y la última carpeta no debe existir. Por ejemplo, si especifica la ruta C: \ MyFiles \ MyFilestreamData, C: \ MyFiles debe existir antes de ejecutar ALTER DATABASE, pero la carpeta MyFilestreamData no debe existir.
El grupo de archivos y el archivo (<filespec>) se debe crear en la misma declaración. Sólo puede haber un archivo, <filespec>, por un grupo de archivos FILESTREAM.

El tamaño, MAXSIZE y FILEGROWTH propiedades no se aplican a un grupo de archivos FILESTREAM.
TAMAÑO
Especifica el tamaño del archivo.
SIZE no se puede especificar cuando el os_file_Nombres se especifica como una ruta de acceso UNC. SIZE no se aplica a un grupo de archivos FILESTREAM.

tamaño

¿Es el tamaño inicial del archivo.

Cuando el tamaño no se suministra para el archivo principal, el motor de base de datos utiliza el tamaño del archivo principal en la base de datos modelo. Cuando un archivo de datos secundario o archivo de registro se especifica pero el tamaño no se especifica para el archivo, el motor de base de datos hace que el archivo de 1 MB. El tamaño especificado para el archivo principal debe ser al menos tan grande como el archivo principal de la base de datos modelo.

El kilobytes (KB), megabyte (MB), gigabyte (GB), o terabyte (TB) se puede utilizar sufijos. El valor predeterminado es MB. Especifique un número entero, no incluyen un decimal. El tamaño es un valor entero. Para valores mayores que 2147483647, utilice unidades más grandes.

MAXSIZE max_size

Especifica el tamaño máximo que el archivo puede crecer. MAXSIZE no se puede especificar cuando el os_file_Nombres se especifica como una ruta de acceso UNC. MAXSIZE no se aplica a un grupo de archivos FILESTREAM.

max_size
¿Es el tamaño máximo de archivo. El KB, MB, GB, TB y sufijos pueden ser utilizados. El valor predeterminado es MB. Especifique un número entero, no incluyen un decimal. Si no se especifica max_size, el archivo crece hasta que el disco está lleno. Max_size es un valor entero. Para valores mayores que 2147483647, utilice unidades más grandes.

ILIMITADO
Especifica que el archivo aumenta hasta que el disco está lleno. En SQL Server, un archivo de registro especificado con un crecimiento ilimitado tiene un tamaño máximo de 2 TB, y un archivo de datos tiene un tamaño máximo de 16 TB.

FILEGROWTH growth_increment

Especifica el incremento de crecimiento automático del archivo. El FILEGROWTH escenario de un archivo no puede superar el valor MAXSIZE. FILEGROWTH no se puede especificar cuando el os_file_Nombres se especifica como una ruta de acceso UNC. FILEGROWTH no se aplica a un grupo de archivos FILESTREAM.

growth_increment
Es la cantidad de espacio adicional en el archivo cada vez que es necesario un nuevo espacio.

El valor se puede especificar en MB, KB, GB, TB, o porcentaje (%). Si se especifica un número sin un sufijo MB, KB o%, el valor predeterminado es MB. Cuando se especifica%, el tamaño de incremento de crecimiento es el porcentaje especificado del tamaño del fichero en el momento el incremento se produce. El tamaño especificado se redondea con una precisión de 64 KB.

Un valor de 0 indica que el crecimiento automático está apagado y no hace falta espacio permitido.

Si FILEGROWTH no se especifica, el valor predeterminado es 1 MB para archivos de datos y 10% para los archivos de registro, y el valor mínimo es de 64 KB.

Nota:
En SQL Server, el incremento de crecimiento por defecto para los archivos de datos ha cambiado de 10% a 1 MB. El valor por defecto del archivo de registro del 10% permanece sin cambios.

<filegroup>
Controla el grupo de archivos de propiedades. Grupo de archivos no se pueden especificar en una instantánea de base de datos.

FILEGROUP filegroup_Nombres

Es el nombre lógico del grupo de archivos.

filegroup_Nombres
filegroup_Nombres debe ser único en la base de datos y no pueden ser los nombres proporcionados sistema-primaria y PRIMARY_LOG. El nombre puede ser un carácter Unicode o constante, o un identificador regular o delimitado. El nombre debe cumplir con las reglas de los identificadores.

CONTIENE FILESTREAM

Especifica que el grupo de archivos FILESTREAM almacena objetos binarios grandes (BLOB) en el sistema de archivos.

DEFAULT
Especifica el grupo de archivos es el llamado grupo de archivos predeterminado en la base de datos.

<external_access_option>
Controles externos de acceso hacia y desde la base de datos.

DB_CHAINING (ON | OFF)

Cuando se especifica ON, la base de datos puede ser el origen o el destino de una cadena de propiedad entre bases de datos.

Cuando es OFF, la base de datos no puede participar en la propiedad entre bases de datos de encadenamiento. El valor predeterminado es OFF.

Importante:
La instancia de SQL Server reconoce esta configuración cuando la propiedad cruzada db encadenamiento servidor opción es 0 (OFF). Cuando el encadenamiento de propiedad cross db es 1 (ON), todas las bases de datos de OPERADORs pueden participar en cadenas de propiedad entre bases de datos, independientemente del valor de esta opción. Esta opción se establece mediante sp_configure.

Para establecer esta opción, es necesario ser miembro de la función de servidor sysadmin fija.La opción DB_CHAINING no se puede establecer en estas bases de datos del sistema: master, model, tempdb.

Para obtener más información, vea Cadenas de propiedad.

CONFIABLE (ON | OFF)

Cuando se especifica ON, los módulos de base de datos (por ejemplo, vistas, funciones definidas por el OPERADOR o procedimientos almacenados) que utilizan un contexto de suplantación pueden tener acceso a recursos fuera de la base de datos.

Cuando es OFF, los módulos de base de datos en un contexto de suplantación no pueden tener acceso a recursos fuera de la base de datos. El valor predeterminado es OFF.

Digno de confianza es en la posición OFF cuando la base de datos se adjunta.

Por defecto, todas las bases de datos del sistema, excepto la base de datos msdb CONFIABLE han ajustado en OFF. El valor no se puede cambiar para el modelo y las bases de datos tempdb. Le recomendamos que nunca se establece la opción de confianza para ON para la base de datos master.

Para establecer esta opción, es necesario ser miembro de la función de servidor sysadmin fija.

<service_broker_option>
Controla las opciones de Service Broker en la base de datos.

Opciones servicio Broker sólo se puede especificar cuando la cláusula FOR ATTACH se utiliza.

ENABLE_BROKER
Especifica que Service Broker está habilitado para la base de datos especificada. Es decir, is_broker_enabled se establece en true en la vista de catálogo sys.databases y la entrega de mensajes se ha iniciado.

NEW_BROKER
Crea un nuevo valor en ambos service_broker_guid sys.databases y la base de datos restaurada y termina todos los extremos de conversación con la limpieza. El corredor está habilitado, pero ningún mensaje se envía al extremos de conversación a distancia.

ERROR_BROKER_CONVERSATIONS
Finaliza todas las conversaciones con un error que indica que la base de datos se adjunta o restaurada. El corredor está desactivado hasta que esta operación se ha completado y ha permitido a continuación.

database_snapshot_Nombres
Es el nombre de la instantánea de base de datos nueva. nombres de base de datos de instantáneas deben ser únicos en una instancia de SQL Server y cumplir con las reglas de los identificadores. database_snapshot_Nombres puede ser un máximo de 128 caracteres.

ON (NOMBRES = logical_file_Nombres, UBICACION = 'os_file_Nombres ») [, ... N]

Para crear una instantánea de base de datos, especifica una lista de archivos en la base de datos de origen. Por la instantánea funcione, todos los archivos de datos se debe especificar individualmente. Sin embargo, los archivos de registro no se permite para las instantáneas de bases de datos.

Para obtener descripciones de NOMBRE y el nombre y sus valores ver las descripciones de los valores <filespec> equivalente.

Nota:
Cuando se crea una instantánea de base de datos, las opciones y los otros <filespec> PRIMARIA palabra clave no se permiten.

AS instantánea de source_database_Nombres

Específica que la base de datos se está creando es una instantánea de base de datos de base de datos de origen especificada por source_database_Nombres. La instantánea y la fuente de base de datos debe estar en la misma instancia.

Comentarios
La base de datos principal debe ser respaldada cada vez que una base de datos de OPERADOR se crea, modifica o se ha caído.

La instrucción CREATE DATABASE se debe ejecutar en modo de confirmación automática (modo por defecto de la operación de gestión) y no se permite en una transacción explícita o implícita. Para obtener más información, vea Transacciones de confirmación automática.

Puede utilizar una instrucción CREATE DATABASE para crear una base de datos y los archivos que almacenan la base de datos. SQL Server implementa la instrucción CREATE DATABASE utilizando los pasos siguientes:

El SQL Server utiliza una copia de la base de datos model para inicializar la base de datos y sus metadatos.

Un corredor de servicio GUID se asigna a la base de datos.

El motor de base de datos a continuación, llena el resto de la base de datos con páginas vacías, a excepción de las páginas que tienen los datos internos que registra cómo el espacio se utiliza en la base de datos. Para obtener más información, consulte la base de datos de archivo de inicialización.

Un máximo de 32.767 bases de datos se puede especificar en una instancia de SQL Server.

Cada base de datos tiene un propietario que puede llevar a cabo actividades especiales en la base de datos. El propietario es el OPERADOR que crea la base de datos. El propietario de la base se puede cambiar mediante el uso de sp_changedbowner.

Base de datos de archivos y grupos de archivos

Cada base de datos tiene por lo menos 2 archivos, un archivo principal y un archivo de registro de transacciones, y al menos un grupo de archivos. Un máximo de 32.767 archivos y grupos de archivos 32.767 se puede especificar para cada base de datos. Para obtener más información, vea Arquitectura de archivos y grupos de archivos.

Cuando se crea una base de datos, haga los archivos de datos lo más grande posible sobre la base de la cantidad máxima de datos que esperar en la base de datos.

Le recomendamos que utilice un Storage Area Network (SAN), la red basada en iSCSI, o conectada localmente en disco para el almacenamiento de sus archivos de base de datos SQL Server, ya que esta configuración optimiza el rendimiento de SQL Server y fiabilidad. De forma predeterminada, los archivos de red utilizando bases de datos (almacenado en un servidor de red o almacenamiento conectado a red) no está habilitado para SQL Server. Sin embargo, puede crear una base de datos con archivos de base de datos basados en la red utilizando el indicador de traza 1807. Para obtener información sobre este indicador de traza y el rendimiento y consideraciones importantes de mantenimiento, consulte el sitio Web de Microsoft.

Base de datos de instantáneas

Usted puede utilizar la instrucción CREATE DATABASE para crear una de sólo lectura, visión estática, una instantánea de base de datos, de una base de datos existente, la base de datos de origen. Una instantánea de base de datos transaccional es coherente con la fuente de base de datos tal como existía en el momento en que se creó la instantánea. Una fuente de base de datos puede tener varias instantáneas.

Nota:

Cuando se crea una instantánea de base de datos, la instrucción CREATE DATABASE no puede archivos de registro de referencia, los archivos sin conexión, la restauración de archivos y archivos de desaparecida.

Si va a crear una base de datos falla instantánea, se convierte en sospechoso y de instantáneas debe ser borrada.

Cada instantánea persiste hasta que se suprime mediante DROP DATABASE.

Opciones de base de datos

Varias opciones de base de datos se ajusta automáticamente cada vez que cree una base de datos. Para obtener una lista de estas opciones y sus valores predeterminados, vea Configurar las opciones de base de datos. Estas opciones se pueden modificar mediante la instrucción ALTER DATABASE.

El modelo de base de datos y Crear Nuevas bases de datos

Todos los objetos definidos por el OPERADOR en la base de datos model se copian en todas las bases de datos recién creada. Puede agregar objetos, como tablas, vistas, procedimientos almacenados, tipos de datos, y así sucesivamente, a la base de datos modelo que se incluye en todas las bases de datos recién creada. Cuando una instrucción CREATE BASE DE DATOS database_Nombres se especifica sin parámetros volumen adicional, el archivo de datos principal que se haga del mismo tamaño que el archivo principal en la base de datos modelo.

A menos que se especifique PARA COLOCAR, cada base de datos nueva hereda los valores de las opciones de base de datos de la base de datos model. Por ejemplo, la opción de base de auto retráctil se establece en true en el modelo y, en cualquier base de datos nueva que cree. Si cambia las opciones de la base de datos model, esta configuración nueva opción se utilizan en las nuevas bases de datos que cree. Cambio de operaciones en la base de datos del modelo no afecta a las bases de datos existentes. Si PARA COLOCAR se especifica en la instrucción CREATE DATABASE, la base de datos nueva hereda los valores de las opciones de base de datos de la base de datos original.

Visualización de la información de base de datos

Puede utilizar vistas de catálogo, funciones del sistema y procedimientos almacenados del sistema para devolver información sobre las bases de datos, archivos y grupos de archivos.Para obtener más información, consulte Visualización de los metadatos de base de datos.

Permisos

Requiere CREATE DATABASE, CREATE ANY DATABASE, o el permiso ALTER ANY DATABASE.

Para mantener el control sobre el uso del disco en una instancia de SQL Server, el permiso para crear bases de datos se limita normalmente a una entrada pocas cuentas. Los permisos de los archivos de registro de datos y En SQL Server, ciertos permisos se establecen en los datos y archivos de registro de cada base de datos. Los permisos siguientes se establecen las siguientes operaciones cada vez que se aplican a una base de datos:

Creado Modificado para agregar un nuevo archivo Adjunto Copia de seguridad Separado Restauradas

Los permisos de evitar que los archivos de forma accidental alterado si residen en un directorio que tiene permisos abiertos. Para obtener más información, consulte Protección de datos de registros y ficheros.

EJEMPLO 001

Crear una base de datos llamada BDEJEMPLO03, con un tamaño de 8 MB y un Maximo de 10 MB. Dentro de la carpeta DATOS, conteniendo una clave primaria

```
USE MASTER
GO
CREATE DATABASE BDEJEMPLO03
ON
PRIMARY (NOMBRES = BDEJEMPLO03_data,
UBICACION = “C:\DATOS\BDEJEMPLO03.MDF”,
SIZE = 8 MB,
MAXSIZE = 10 MB,
FILEGROWTH = 1 MB)
LOG ON
(NOMBRES = BDEJEMPLO03_LOG,
UBICACION = “C:\DATOS\BDEJEMPLO03.IDF”,
SIZE = 5 MB,
MAXSIZE = 10 MB,
FILEGROWTH = 10 %)
GO
```

EJEMPLO 002

Crear una base de datos llamada EJEMPLO, con un tamaño de 5 MB y un Maximo de 10 MB. Dentro de la carpeta DATOS

```
USE MASTER
GO
CREATE DATABASEBD_ EJEMPLO
ON
(NOMBRES = BD_EJEMPLO_DATA,
UBICACION = “C:\DATOS\BD_EJEMPLO_DATA.MDF”,
SIZE = 5 MB,
MAXSIZE = 10 MB,
FILEGROWTH = 2 MB)
LOG ON
(NOMBRES = BD_EJEMPLO_LOG,
UBICACION = “C:\DATOS\BD_EJEMPLO.LDF”,
SIZE = 5 MB,
MAXSIZE = 8 MB,
FILEGROWTH =2 MB)
GO
```

EJEMPLO 003

Cambiar el estado de una opción de la base de datos REGISTROS

```
USE MASTER
GO
EXEC SP_DBOPTION REGISTROS
GO
```

EJEMPLO 004

Ahora, vamos a establecer la base de datos REGISTROS, como de solo lectura

```
EXEC SP_DBOPTION REGISTROS, “READ ONLY”, TRUE
EXEC SP_DBOPTION REGISTROS
GO
```

EJEMPLO 005

Ahora, empleando la función de ALTER DATABASE, vamos a cambiar la configuración de la base de datos en reemplazo de SP_DBOPTION

```
USE MASTER
GO
ALTER DATABASE REGISTROS
SET READ_WRITE
GO
```

EJEMPLO 006

Verificar la base de datos de los cambios efectuados.

```
SELECT DATABASE PROPERTY (“REGISTROS”, “IS Read Only”)
GO
```

---Retorna el valor de la opción de configuración de la base de datos especificada, ojo si el resultado es cero lo que indica fue Read Only es falso.

EJEMPLO 007

Crear la base de datos BDEJEMPLO03, mediante la especificación de multiples archivos de registro de datos y de transacciones, que contenga 03 archivos de datos de 5MB y 02 archivos de transacciones de 8 MB

```
USE MASTER
GO
CREATE DATABASE BDEJEMPLO03
ON PRIMARY
 (NOMBRES = BDEJEMPLO03_data,
UBICACION = “C:\DATOS\BDEJEMPLO03.MDF”,
SIZE = 5 MB,
MAXSIZE =80 MB,
FILEGROWTH = 10 MB)
 (NOMBRES =ARCHIV2_DATA,
UBICACION =”C:\DATOS\ARCHIV2.NDF”,
SIZE = 5 MB,
MAXSIZE = 10 MB,
FILEGROWTH = 10),
(NOMBRES =ARCHIV3_DATA,
UBICACION =”C:\DATOS\ARCHIV3.NDF”,
SIZE = 5 MB,
MAXSIZE = 10 MB,
FILEGROWTH = 10)
LOG ON
(NOMBRES =BDEJEMPLO03_LOG,
UBICACION =”C:\DATOS\BDEJEMPLO03.LDF”,
SIZE = 5 MB,
MAXSIZE = 10 MB,
FILEGROWTH = 10),

(NOMBRES =ARCHIVLOG2,
UBICACION =”C:\DATOS\ARCHIV.LDF”,
SIZE = 5 MB,
MAXSIZE = 10 MB,
FILEGROWTH = 10)

GO
```

EJEMPLO 008

Crear la base de datos BDEJEMPLO04,siendo un único archivo.

```
USE MASTER
GO
CREATE DATABASE BDEJEMPLO04
ON
 (NOMBRES = BDEJEMPLO04_data,
UBICACION = “C:\DATOS\BDEJEMPLO04.MDF”,
SIZE = 10,
Maxsize = 15,
Filegrowth = 2)
```

EJEMPLO 009

Crear la base de datos BDEJEMPLO06,siendo un único archivo (por defecto crear un archivo de transacciones de 1 MB), con las siguientes características:
Nombre de la Base de datos BDEJEMPLO_06
Nombre del archivo lógico BDEJEMPL_06_DAT
Nombre del archivo fisico EJEMPLO_06_DATA.MDF
Tamaño inicial 5 MB
Tamaño Maximo 20 MB
Porcentaje de incremento archivo 30%

```
USE MASTER
GO
CREATE DATABASE BDEJEMPLO06
ON
 (NOMBRES = BDEJEMPLO06_dat,
```

UBICACION = “C:\DATOS\BDEJEMPLO_06_dat,mdf”,
SIZE = 5,
Maxsize = 20,
FILEGROTW = 30%)
LOG ON
(NOMBRES =BD_EJEMPLO_06_LDF,
SIZE = 8 MB,
MAXSIZE = 2 MB,
FILEGROTW =2)
GO

EJEMPLO 010

Crear la base de datos BDEJEMPLO_07, que especifique los archivos de registro de datos y de transacciones con los siguientes características:
Nombre del archivo lógico BDEJEMPL_07_DAT
Nombre del archivo fisico EJEMPLO_07_DATA.MDF
Tamaño inicial 10 MB
Tamaño Maximo 30 MB
Porcentaje de incremento archivo 25%

```
USE MASTER
GO
CREATE DATABASE BDEJEMPLO_07,
ON
 (NOMBRES = BDEJEMPLO_07_data,
UBICACION = “C:\DATOS\BDEJEMPLO_07..MDF”,
SIZE =105 MB,
MAXSIZE =30 MB,
FILEGROWTH = 25 %)
LOG ON
 (NOMBRES =BDEJEMPLO_07.LOG,
UBICACION =”C:\DATOS\BDEJEMPLO_07_LOG.NDF”,
SIZE = 10 MB,
MAXSIZE = 30 MB,
FILEGROWTH =25%),
GO
```

EJEMPLO 011

Cambiar o incrementar el tamaño de la base de datos
ALTER DATABASE

```
USE MASTER
GO
SP_HELPDB PERUANO
```

EJEMPLO 012

Modificar el tamaño de la base de datos PERUANO

```
ALTER DATABASE PERUANO
MODIFY FILE (
NOMBRES = PERUANO_DATA,
SIZE= 15 MB)
GO
```

EJEMPLO 013

Modificar la base de datos BDEJEMPLO_04, para agregar el archivo de datos 5MB

```
USE MASTER
ALTER DATABASE BDEJEMPLO_04
ADD FILE
(
NOMBRES = BDEJEMPLO_04_DAT,
UBICACION =”C:\DATOS\BDEJEMPLEO_04.NDF”,
SIZE= 5 MB
MAXSIZE = 10 MB,
FILEGROWTH = 5MB)
GO
```

EJEMPLO 014

Ampliar la base de datos en 5 MB de disco de capacidad, asimismo agregar un grupo de archivos con nombre BD_EJEMPLO04FGI ala base de datos BD_EJEMPLO04, posteriormente debemos agregar dos archivos de 05 MB, al grupo de archivos y finalmente agregue el grupo BD_EJEMPLO04 FGI sea el grupo predeterminado.

```
USE MASTER
GO
ALTER DATABASE BD_EJEMPLO04
ADD FILEGROUP BD_EJEMPLO04FGI
GO
ALTER DATABASE BD_EJEMPLO04
```

```
ADD FILE
(NOMBRES=BDEJEMPLO04_DAT,
UBICACION = “C:\DATOS\EJEMPLO_05.NDF”,
SIZE = 5MB,
MAXSIZE = 10 MB,
FILEGROWTH = 5MB)
TO FILEGROUP BD_EJEMPLO04FGI
ALTER DATABASE BDEJEMPLO04GI
MODIFY FILEGROUP BDEJEMPLO04GI DEFAULT
```

EJEMPLO 015

Añadir un archivo secundario a la base de datos PERUANOS

```
USE MASTER
GO
ALTER DATABASE PERUANOS
ADD FILE
 (NOMBRES = PERUANOS_DAT,
UBICACION = “C:\DATOS\EJEMPLO_05.NDF”,
SIZE = 5MB,
MAXSIZE = 10 MB,
FILEGROWTH = 1 MB)
```

EJEMPLO 016

Diga usted, como eliminamos un archivo de la base de datos

```
USE MASTER
GO
ALTER DATABASE PERUANOS
REMOVE FILE PERUANOS_DAT
GO
```

EJEMPLO 017

Reducción del tamaño de una base de datos mediante la instrucción DBCC SHRINKFILE, debiendo reducir el tamaño del archivo primario de la base de datos peruanos hasta 10 MB.

```
USE PERUANOS
GO
DBCC SHRINKFILE (PERUANOS_DATA,10)
GO
```

---NOTA : SI hubiésemos colocado 50 en vez de 10, así como muestra DBCC SHRINKFILE (PERUANOS_DATA,50), esto indicad que se reducirá un 50 %

EJEMPLO 018

Vaciar el archivo test1_data de la base de datos BDEJEMPLO_04 y usa la opción REMOVE FILE para eliminar el archivo de la base de datos

```
USE BDEJEMPLO_04
GO
ALTER DATABASE BDEJEMPLO_04
REMOVE FILE TEST1_DATA
GO
```

--Con esta opción estamos eliminando un archivo

EJEMPLO 019

```
USE BDEJEMPLO_04
GO
DBCC SHRINKFILE (TEST1_DATA, EMPTYFILE)
GO
ALTER DATABASE BDEJEMPLO_04 REMOVE FILE TEST1_DATA
```

--Con esta opción estamos vaciando un archivo

--Recordemos que la opción EMPTYFILE migra todos los datos del archivo especificado al mismo grupo de archivos.

EJEMPLO 020

Diga usted como renombrar una base de datos llamada BDEJEMPLO_04 por BDEJEMPLO_04C

```
USE MASTER
GO
EXEC SP_DBOPTION BD EJEMPLO_04, “single User” TRUE
EXEC SP_RENOMBRESDB “BDEJEMPLO_04”, “BDEJEMPLO_04C”
```

EXEC SP_DBOPTION BDEJEMPLO_04C, “Single User”, False

EJEMPLO 021

Diga usted como eliminar una base

```
USE MASTER
GO
DROP DATABASE BD_EJEMPLO04
GO
```

EJEMPLO 022

Diga usted como eliminar dos a mas base de datos grabadas (B.D; PERSONAL Y TRABAJO).

```
USE MASTER
GO
DROP DATABASE PERSONAL, TRABAJO
GO
```

Ejercicios Propuestos

EJEMPLO 023

Modificar la base de datos BDEJEMPLO07 para agregarle archivos de datos de la siguiente manera:

- **Para el Archivo personal**
- **Nombre de archivo de datos** **Adicional_dat**
- **Nombre de archivo fisico** **adicional_dat.mdf**
- **Tamaño inicial** **03 MB**
- **Tamaño Maximo** **06 MB**
- **Porcentaje incremento** **05%**

```
USE MASTER
GO
ALTER DATABASE BDEJEMPLO07
MODIFY FILE
(NOMBRES=Adicional_dat,
UBICACION =”C:\COPIA01\Adicional_dat.mdf”
SIZE = 3 MB
MAXSIZE = 6 MB,
FILEGROWTH = 5%)
GO
```

NOTA

Para mirar la información de la base de datos procesada lo ejecutaremos empleando la función SP_HELPDB, para el ejemplo anterior lo ejecutaremos de esta manera : SP_HELPDB BDEJEMPLO07

EJEMPLO 024

Modificar la base de datos BDEJEMPLO07 para agregarle un grupo de archivos de datos con las siguientes características:

- **Nombre del grupo de archivos BDEJEMPLOX**

```
USE MASTER
GO
ALTER DATABASE BDEJEMPLO07
DD FILEGROUP BDEJEMPLOX
GO
```

EJEMPLO 025

Modificar la base de datos BDEJEMPLO07 para agregarle dos archivos de datos y hacer que estos pertenezcan l grupo ya creado en el punto anterior, considerando que las características son las siguientes:

- **Para el Archivo de datos1**
- **Nombre de archivo de datos Adicional2_dat**
- **Nombre de archivo fisico adicional2_dat.mdf**
- **Tamaño inicial 05 MB**
- **Tamaño Maximo 10 MB**
- **Porcentaje incremento 2 MB**
- **Para el Archivo de datos2**
- **Nombre de archivo de datos Adicional3_dat**
- **Nombre de archivo fisico adicional3_dat.mdf**
- **Tamaño inicial 05 MB**
- **Tamaño Maximo 10 MB**
- **Porcentaje incremento 2 MB**

```
USE BDEJEMPLO07
GO
ALTER DATABASE BDEJEMPLO07
ADD FILE
(NOMBRES = Adicional2_dat,
UBICACION = ”C:\COPIA01\Adicional2_dat.mdf ”,
SIZE =5 MB,
MAXSIZE =10MB,
FILEGROWTH = 2MB),
(NOMBRES =Adicional3_dat,
UBICACION = “C:\COPIA01\Adicional3_dat.mdf”,
SIZE = 5 MB,
MAXSIZE =10 MB,
FILEGROWTH =2MB)
GO
```

EJEMPLO 026

Cambiar el tamaño de la base de datos BDEJEMPLO07 aumentándole el tamaño al archivo de datos de la siguiente manera:

- **Nombre del Archivo Adicional_dat**
- **Aumentar tamaño 20 MB**

```
USE BDEJEMPLO07
GO
ALTER DATABASE BDEJEMPLO07
MODIFY FILE
(NOMBRES = BDEJEMPLO07_dat,
SIE = 20 MB)
GO
```

NOTA

Recuerde que FILEGROTW no puede exceder del valor MAXSIZE

EJEMPLO 027

Reducir el tamaño de la base de datos BDEJEMPLO07, debiendo vaciar el archivo llamado adicional_data luego reducir el tamaño de la base de datos adicional_dat.

```
USE BDEJEMPLO07
GO
ALTER DATABASE BDEJEMPLO07
REMOVE FILE Adicional_dat
GO
ALTER DATABASE BDEJEMPLO07
DBCC SHRINKFILE (BDEJEMPLO07, EMPTYFILE)
GO
ALTER DATABASE BDEJEMPLO07 REMOVE FILE adicional_dat
```

EJEMPLO 028-029

Cambiar el nombre de la base de datos BDEJEMPLO_07, por el de BDEJEMPLO_CAMBIO, luego proceder a eliminar la base de datos BDEJEMPLO_CAMBIO

```
USE BDEJEMPLO07
GO
EXEC SP_DBOPTION BDEJEMPLO07, ´Single User´ TRUE
EXEC SP_RENOMBRESDB ´BDEJEMPLO07´,´BDEJEMPLO_CAMBIO´
EXEC SP_DBOPTION BDEJEMPLO_CAMBIO, ´Single User´, FALSE
GO
SP_DBOPTION
EXEC SP_DBOPTION
```

EJEMPLO 030

Ejecutar la Revisión y Cambios de la Configuración de la Base de Datos

Tablas

Las tablas son las unidades que almacenan los datos. Como norma general se suele imponer que cada tabla, almacena información común sobre una entidad en particular (recuerda los libros). Esta norma se conoce como **normalización**.

Las tablas deben tener un nombre como máximo de 128 caracteres y el nombre debe empezar por un carácter alfabético, a excepción de las tablas temporales que se crean con el signo # delante del nombre y ## para las tablas temporales globales accesibles a todos los OPERADORs.

Tablas Temporales : Son tablas que crea el OPERADOR durante la ejecución de un procedimiento almacenado u otro mecanismo y se eliminan automáticamente cuando la conexión que las creo desaparece.

Estas tablas no se almacenan en la base de datos de trabajo sino que están almacenadas en la base de datos Tempdb.

Estructuras de las Tablas

Una base de datos en un sistema relacional está compuesta por un conjunto de tablas, que corresponden a las relaciones del modelo relacional. En la terminología usada en **SQL** no se alude a las relaciones, del mismo modo que no se usa el término atributo, pero sí la palabra columna, y no se habla de tupla, sino de línea.

Creación de Tablas Nuevas

CREATE TABLE tabla (
campo1 tipo (tamaño) índice1,
campo2 tipo (tamaño) índice2,... ,
índice multicampo , ...)

En donde:

tabla	Es el nombre de la tabla que se va a crear.
campo1 campo2	Es el nombre del campo o de los campos que se van a crear en la nueva tabla. La nueva tabla debe contener, al menos, un campo.
tipo	Es el tipo de datos de campo en la nueva tabla. (Ver Tipos de Datos)
tamaño	Es el tamaño del campo sólo se aplica para campos de tipo texto.
índice1 índice2	Es una cláusula CONSTRAINT que define el tipo de índice a crear. Esta cláusula en opcional.
índice multicampos	Es una cláusula CONSTRAINT que define el tipo de índice multicampos a crear. Un índice multicampo es aquel que está indexado por el contenido de varios campos. Esta cláusula en opcional.

Tipos de datos : Definen el tipo de datos que los objetos pueden detallarse o contenerse

Bit : Es un dato lógico que se usa para almacenar información booleana el cual los marcadores, los almacenan como 0 y 1.

Text, Image : Se usan cuando los valores que se van almacenar exceden al limite de columna de 8000 caracteres. Estos datos se pueden almacenar hasta 2 Gb entre binarios y textos.

Sql Variant: Es un tipo de datos especial que almacena valores de múltiples datos; en la misma columna se puede almacenar valores: nchar, valores int y valores decimales.

Smalldatetime : Almacena la hora también datetime

Tipos de moneda:

Money valor monetaria de 08 bytes

Smallmoney valor monetario de 04 bytes

Ambos almacenan 04 dígitos a la derecha del punto decimal, al ingresar datos monetarios debe antecederlos con el signo dólar 99999.9999 (4 dígitos)

Timestamp Cuando se agregue un nuevo registro a una tabla, en este campo se agregaran valores de hora de forma automática.

INTRODUCCION

El lenguaje de definición de datos (DDL, Data Definition Language) es el encargado de permitir la descripcion de los objetos que forman una base de datos.

El lenguaje de definición de datos le va a permitir llevar a cabo las siguientes acciones:

- Creación de tablas, índices y vistas.
- Modificación de las estructura de tablas, índices y vistas.
- Supresión de tablas, índices y vistas.

Pero antes de continuar vamos a comentar la nomenclatura que emplearemos, si tiene algún conocimiento de programación le resultará familiar.

Nomenclatura

La sintaxis empleada para la sentencias en las diferentes páginas esta basada en la notación EBNF. Vamos a ver el significado de algunos simbolos.

Carácter	Significado del carácter
< >	Encierran parámetros de una orden que el OPERADOR debe sustituir al escribir dicha orden por los valores que queramos dar a los parámetros.
[ ]	Indica que su contenido es opcional.
{ }	Indica que su contenido puede repetirse una o mas veces.
|	Separa expresiones. Indica que pueden emplearse una u otra expresión pero no más de una a la vez.

Además las palabras clave aparecen en mayúscula negrita y los argumentos en minúscula cursiva.

Creación de tablas

La creación de la base de datos debe comenzar por con la creación de una o más tablas. Para ello utilizaremos la sentencia **CREATE TABLE.**
La sintaxis de la sentencia es la siguiente:

```
CREATE TABLE <nombre_tabla>
(
<nombre_campo> <tipo_datos(tamaño)>
[null | not null] [default <valor_por_defecto>]
{
,<nombre_campo> <tipo_datos(tamaño)>
[null | not null] [default <valor_por_defecto>]}
[
, constraint <nombre> primary key (<nombre_campo>[ ,...n ])]
[
, constraint <nombre> foreign key (<nombre_campo>[ ,...n ])
references <tabla_referenciada> ( <nombre_campo> [ ,...n ] ) ]
) ;
```

Ejemplo: Vamos a simular una base de datos para un negocio de alquiler de coches, por lo que vamos a empezar creando una tabla para almacenar los carros

Las claves primarias y externas (o foráneas) se pueden implementar directamente a través de la instrucción **CREATE TABLE**, o bien se pueden agregar a través de sentencias **ALTER TABLE**.

Cada gestor de bases de datos implementa distintas opciones para la instrucción **CREATE TABLE**, pudiendo especificarse gran cantidad de parámetros y pudiendo variar el nombre que damos a los tipos de datos, pero la sintaxis standart es la que hemos mostrado aquí. Si queremos conocer más acerca de las opciones de **CREATE TABLE** lo mejor es recurrir a la Libroación de nuestro gestor de base de datos.

Modificación de tablas

En ocasiones puede ser necesario modificar la estructura de una tabla, comúnmente para añadir un campo o restricción. Para ello disponemos de la instruccción **ALTER TABLE**.
ALTER TABLE nos va a permitir:

- Añadir campos a la estructura incial de una tabla.
- Añadir reestriciones y referencias.
- Modifica el diseño de una tabla ya existente, se pueden modificar los campos o los índices existentes. Su sintaxis es:
  ALTER TABLE tabla {ADD {COLUMN tipo de campo[(tamaño)]
  [CONSTRAINT índice]
  CONSTRAINT índice multicampo} |
  DROP {COLUMN campo I CONSTRAINT nombre del índice}}
- En donde:

tabla	Es el nombre de la tabla que se desea modificar.
campo	Es el nombre del campo que se va a añadir o eliminar.
tipo	Es el tipo de campo que se va a añadir.
tamaño	Es el tamaño del campo que se va a añadir (sólo para campos de texto).
índice	Es el nombre del índice del campo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.
índice multicampo	Es el nombre del índice del campo multicampo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.

Operación	Descripción
ADD	**Se utiliza para añadir un nuevo campo a la tabla, indicando el**

COLUMN	**nombre, el tipo de campo y opcionalmente el tamaño (para campos de tipo texto).**
ADD	**Se utiliza para agregar un índice de multicampos o de un único campo.**
DROP COLUMN	**Se utiliza para borrar un campo. Se especifica únicamente el nombre del campo.**
DROP	**Se utiliza para eliminar un índice. Se especifica únicamente el nombre del índice a continuación de la palabra reservada CONSTRAINT.**

Eliminación de tablas.

Podemos eliminar una tabla de una base de datos mediante la instruccion **DROP TABLE**.

```
DROP TABLE <nombre_tabla>;
```

La instrucción **DROP TABLE** elimina de forma permanente la tabla y los datos en ella contenida.

Si intentamos eliminar una tabla que tenga registros relacionados a través de una clave externa la instrucción **DROP TABLE** fallará por integridad referencial.

Cuando eliminamos una tabla eliminamos también sus índices.

Valores Nulos

Los valores nulos se conocen como NULL, pero aunque se conoce como valor nulo, no debes pensar que se almacena un valor, el concepto de NULL podría ser un marcador que informa que hay que pasar por alto los datos de esa celda, son datos que se ignoran. Como veremos, hay que tener mucho cuidado con el uso de esta "ausencia" de información, ya que si tenemos algún despiste puede ser el causante de que no recibamos la información que realmente estamos reclamando en una consulta. Si realizamos operaciones matemáticas con varios valores de nuestra base de datos y uno de estos es NULL, el resultado siempre será NULL.

Indices

Seguimos hablando de la unidad o entidad principal de la base de datos, las tablas. Podemos tener tablas con millones de registros, si realizamos una consulta para recuperar información de un grupo de estos registros, podemos tener un rendimiento bajo debido a la gran cantidad de información que almacena esa tabla.

Para acelerar este tipo de consultas contamos con la ayuda de los **índices**. Un índice es una característica más de las tablas, el cual es una conjunto de valores clave. Este conjunto tiene una estructura estudiada para que el servidor pueda realizar las consultas con un rendimiento mucho mayor.

Estos valores claves pueden almacenar el contenido de una o varias columnas de la tabla sobre la que operan.

Además de mejorar el rendimiento, existen índices que pueden asegurar la integridad de los datos, indicando en que orden deben almacenarse los datos en un tabla. Más adelante veremos como trabajar con los índices.

Por lo tanto podemos decir que teóricamente nuestras tablas deberían todas incluir al menos un índice que asegure un mejor rendimiento. Y en la práctica, suele ser lo más común, pero debes tener en cuenta que cada vez que realizamos una tarea sobre una tabla que está relacionada con índices, el servidor, no sólo opera sobre la tabla para realizar las modificaciones que se le demanden, sino que también debe realizar operaciones sobre los índices para asegurar que la labor de estos sigue siendo la adecuado para las modificaciones realizadas sobre la información de la tabla. Por lo tanto la regla de tres es sencilla, a mayor número de índices, mas tiempo dedicará a las tareas pedidas.

Restricciones

Las restricciones son normas que imponemos a la información que pude ser almacenada, de modo que si no se cumple una de estas condiciones no permitamos que incluya ese valor en nuestra base de datos.

Si tenemos una tabla de clientes, podríamos poner como restricción que no se pueda almacenar un cliente cuya edad no supere los 18 años. El servidor se encargará de no permitir que se incluya ningún registro que incumpla nuestra condición.

Estas restricciones además de permitir controlar que valores pueden ser almacenadas, con esta tarea aseguramos también la integridad de nuestros datos. Piensa que por descuido un OPERADOR introduce por error un cero como el número de unidades que se han vendido a un minorista. Al calcular el precio de la venta y multiplicarlo por cero unidades, tendremos como precio de facturación cero.

Pero no sólo debemos pensar en que es el OPERADOR quien comete el error, puede que el programador que ha desarrollado un software de facturación haya cometido un error al escribir el código. El programa por sí sólo, si no ha tenido en cuenta esta posibilidad, no lanzará ningún error, y todo parecerá ir correctamente y emitirá la factura al minorista. Pero gracias al servidor de base de datos, podemos tener controlado que uso hace el software que trabaja con nuestros datos.

Vistas

Las consultas que se realizan sobre algunas de las tablas de la base de datos pueden ser repetitivas, de modo que día tras día cientos de OPERADORs realizan las mismas consultas sobre la tabla. Todas esas consultas repetitivas reciben el mismo grupo de datos.

Para evitar la repetición de este tipo de consultas tenemos las vistas.

Podemos pensar que una vista es un conjunto de registros determinados de una o varias tablas. De hecho se trabaja sobre ella como una tabla, pero no es una tabla. Lo que almacena en realidad es una consulta. Pero debes tener claro que no almacena datos sino que los extrae.

Una vista puede crear los enlaces necesarios para obtener información de varias tablas como si fuese una única tabla. Esto puede facilitar mucho la tarea al desarrollador de software que no tiene que preocuparse de las tablas donde se almacena la información que quiere recoger, ya que lo tiene todo en una vista sobre la que puede operar como si fuese una tabla. Por lo tanto se olvida de construir complicadas sentencias de SQL que recoja esa información de múltiples tablas, con diferentes enlaces entre ellas.

CONSULTAS - SQL

Las consultas y las tareas de gestión que se realizan durante la explotación de una base de datos vienen escritas en lenguaje SQL, Structured Query Language, que como ya hemos mencionado significa Lenguaje de Consulta Estructurado.

El ANIS (Instituto Nacional de Normalización Estadounidense) ideo este lenguaje estándar, denominado ASNI SQL, o también SQL-92, por el último año en el que ANSI aceptó modificaciones sobre el estándar.

Como suele ocurrir en tantas ocasiones en el mundo del mercado informático. Este estándar fue recogido por los fabricantes para personalizarlos y crear sus propias extensiones para sus productos. Microsoft así lo hizo, para crear Transact-SQL, o más comúnmente conocido por su abreviatura T-SQL para sus servidores de base de datos SQL Server.

Como veremos, SQL Server y sus antecesores, pueden trabajar con SQL, pero gracias a T-SQL podemos realizar sentencias más completas que solventarán fácilmente problemas. Por lo tanto T-SQL no es un sucesor de SQL para nosotros, sino una ampliación, una herramienta extra que utilizaremos para fines más avanzados.

Procedimientos almacenados

SQL Server no sólo puede ejecutar las consultas de las tablas, o las vistas que ya hemos visto. También permite que desarrollemos procedimientos con código escrito íntegramente en SQL o con la ayuda extra de T-SQL. Tanto con el estándar como con la versión de Microsoft, podemos crear sentencias que vayan más allá de consultas, ya que como lenguajes de programación que son, pueden contener sentencias condicionales, bucles, etc... Si nunca te has introducido en ningún lenguaje de programación, no te preocupes porque veremos estos conocimientos con detenimiento, y si ya conoces otros lenguajes de programación, aprenderás la sintaxis específica de este lenguaje.

Los procedimientos almacenados, como cualquier función de otro lenguaje, pueden recibir parámetros de entrada y de salida, o no recibir ni devolver nada. Además de devolver parámetros, pueden devolver incluso tablas virtuales, vistas, etc...

Los procedimientos almacenados los almacena SQL Server 2005 del modo más optimo para sacarles el mejor rendimiento posible. De este modo las instrucciones quedan almacenadas en la propia base de datos. Esto es una gran ventaja, en cuanto a seguridad y rendimiento, ya que los programas desarrollados por los programadores no necesitan tener estas sentencias SQL en el código de su software, y por lo tanto esta información que supone el propio código SQL, no tiene que "viajar" del programa a la base de datos. Y como es lógico pensar, cuanta menos información "viaje" del programa del cliente al servidor, ganaremos en seguridad y en rendimiento.

Propiedades

Partiendo de las entidades que hemos definido en nuestro conjunto como resultado del estudio de necesidades, vamos a analizar las propiedades de cada una de estas entidades.

Las propiedades definidas para cada tabla, son de carácter temporal y no son definitivas, veremos como en el proceso de normalización, tendremos que realizar modificaciones como ya hemos comentado.

Hemos visto como las tablas formaban un conjunto de entidades. Nunca debes pensar en una tabla como un elemento aislado, cada una de las tablas forma parte de nuestro conjunto, y el conjunto vendría a ser la base de datos. Por lo tanto, cada tabla es la parte de un nivel superior y no puede ser tratada individualmente, ya que como parte de un todo, tendrá dependencias con el resto de tablas de la base de datos.

Ese es precisamente el trabajo que debemos llevar a cabo ahora, analizar las dependencias que tiene cada tabla con el resto para poder representarla la estructura lógica de la base de datos.

Estas dependencias reciben el nombre de relaciones, y estudiaremos que tipo de relaciones podemos tener, de momento, para nuestro ejemplo las veremos de un modo muy sencillo. Podemos encontrarnos tablas que dependen únicamente de una segunda tabla, mientras que habrá otras que dependerán de varias. Otro caso que nos podemos encontrar son tablas que tienen una relación única, como iremos viendo.

Normalización

La normalización es el mecanismo de toma de decisiones con el objetivo de recoger todos los datos de la información que se almacenará en una base de datos y distribuirlos en tablas.

Para tomar estas decisiones tenemos un número de **formas normales** que nos ayudará a diseñar la mejor estructura lógica con el mayor rendimiento posible.

Las formas normales, son los modelos o maneras en que se pueden representar la estructura de tablas. Gracias a estos modelos conseguiremos mayor eficacia. Pero no

entiendas por eficacia como una reducción del tamaña, nos estamos refiriendo a que obtendremos una estructura muy bien organizada, de tal modo que será escalable fácilmente, permitiendo realizar modificaciones en un futuro sin muchos problemas. Aunque habrá veces donde gracias a la normalización también se reduzca el tamaño, este no es el objetivo que buscamos.

La función de la normalización es favorecer la integridad de los datos, sin importar la actividad que se desarrolle sobre la base de datos. Trata de evitar lo máximo posible la posibilidad de introducir datos que no sean razonables. Dentro del proceso de normalización podemos distinguir cuatro tipo de integridades:

- Integridad de entidad.
- Integridad de dominio.
- Integridad referencial.
- Integridad definida por el OPERADOR.

Vamos a explicar cada una de estas integridades y al final de cada una nombraremos que herramientas nos ofrece SQL Server 2005 para cumplir con estas integridades, si desconoces estas herramientas, tranquilo porque las veremos con más detenimiento en las siguientes lecciones, tan sólo que te suene para cuando lleguemos a verlas con más detenimiento.

Integridad de entidad

Hasta ahora hemos utilizado en varias ocasiones la palabra entidad. Una entidad se define como un concepto del mundo real, de modo que nuestras bases de datos guardan información sobre entidades. Estas entidades puede ser de diferente carácter:

- Entidades físicas: un libro, una bebida, un empleado
- Entidades conceptuales: una empresa
- Entidades como eventos: una alerta de nuestra agenda que nos recuerda una tarea.

Uno de los pasos de nuestro proceso de planificación es detectar estas entidades que están relacionadas con la base de datos.

La integridad de entidad pretende que cada entidad que se guarda en la base de datos sea identificable de un modo único, es decir, que evitemos la información redundante.

¿Qué criterio debemos seguir entonces para identificar que es una entidad en nuestra base de datos?

La respuesta a esta pregunta dependerá de lo que deseemos hacer con estos datos. Lo más razonable es que se identifique como identidad aquellas cosas con las que vas a trabajar de modo unitario. Dicho de un modo más claro, la información que se almacena unida (de modo unitario) es más cómodo trabajar con ella, o recuperar esa información en una única operación.

Integridad de dominio

Ya hemos visto que la integridad de identidad permite obtener los datos almacenados en una base de datos. Con la integridad de dominio conseguimos controlar la información que guardamos en la base de datos. Como dominio, podemos entender como un conjunto de normas de negocio que gestionan la disponibilidad de datos en una determinada columna de una tabla. Por ejemplo que sólo podamos introducir nombres de fabricantes validados por un dominio de valores.

Tenemos una integridad de dominio básica, como no poder introducir letras en campos destinados para almacenar números. A mayor número de limitaciones, mejor aseguraremos el correcto funcionamiento de nuestra base de datos.

Estas normas o reglas de integridad de dominio pueden indicar que campos son necesarios tener obligatoriamente con valores (no se pueden dejar vacíos, NULL) para que la base de datos no tenga datos sin conectar en el caso de tener relaciones o dependencias entre tablas.

Las herramientas que nos ofrece SQL Server para asegurar la integridad de dominio y que iremos estudiando son:

- Tipos de datos
- Tipos de datos definidos por el OPERADOR.
- Restricciones:
- CHECK
- DEFAULT
- FOREIGN KEY
- Reglas
- NOT NULL

Integridad Referencial.

Para ver este tipo de integridad tienes que pensar en las dependencias de tablas que hemos visto en la base de datos que hemos puesto como ejemplo de la empresa de venta de bebidas.

Hemos visto por ejemplo que para cada registro de la tabla Bebidas, teníamos un registro en la tabla Almacén. Y otro tipo de dependencias o relaciones que habíamos denominado relaciones "uno a muchos".

Estas relaciones se producen entre columnas comunes de las tablas que se relacionan. Como pude ser el nombre de una bebida, el de una distribuidora etc...

Con la integridad referencial tratamos de asegurar que las filas relacionadas entre tablas, no dejen de estarlo, o varíen esta relación cuando llevemos modificaciones a los datos. Con esta integridad limitaremos la actividad que puede realiza un OPERADOR sobre la base de datos.

Vamos a ponernos en un ejemplo sencillo, nuestra tabla bebidas tiene una columna llamada "Distribuidoras", que se relaciona con nuestra tabla "Distribuidora" mediante esta misma columna. Ya hemos comentado este tipo de dependencia en este mismo tema. Llevando a cabo una integridad referencial, limitaremos las siguientes tareas a un OPERADOR:

- El OPERADOR no podrá cambiar el nombre de una distribuidora en una de las tablas, ya que si así lo hace, este valor no será el mismo en las dos tablas, y provoca que la relación quede rota. Un registro o varios (dependiendo de en que tabla realice esa modificación) se quedará sin su pareja y no podrá encontrar la relación.
- No podrá eliminar registros de la tabla distribuidora que se encuentren en la tabla Bebidas. Ya que todos aquellos registros de la tabla Bebidas que estuviesen vinculados a la Distribuidora eliminada se quedarán sin relación.
- No puede añadir registros nuevos en la tabla bebida cuyo campo Distribuidora no coincida con ninguna de las distribuidoras añadidos en la tabla Distribuidoras.

Por lo tanto, lo que debemos comprender de la integridad referencial es que existen relaciones entre tablas que deben permanecer invariables sea cual sea la actividad sobre ellas.

Para mantener esta integridad SQL Server nos ofrece:

- Restricciones FOREIGN KEY.
- Restricciones CHECK.
- Desencadenadores y procedimientos almacenados.

Integridad fijada por OPERADOR.

Las tres integridades que acabamos de ver, están todas integradas en las bases de datos. Además no son exclusivas para SQL Server 2005, sino que las encontrarás en cualquier base de datos. Si bien puede que no estén completamente integradas y funcionales, son compatibles en cualquier ámbito.

La integridad que vamos a ver en este apartado, recoge todas las reglas que no están incluidas en ninguna de las integridades anteriores.

Un ejemplo de este tipo de integridad de OPERADOR, sería obligar a que una determinada bebida siempre tenga dos tipos de envases. Este tipo de integridad no la cubre ni la de entidad, ni de dominio, ni referencial. Únicamente podemos controlarla mediante procedimientos almacenados, desencadenadores o reglas que se almacenen en la base de datos.

Esta integridad puede ser controlada también desde los programas clientes que conectan a la base de datos. Mediante el código de programación estos programas pueden comprobar antes de enviar los datos al servidor, si estos cumplen con un determinado juego de normas. De este modo el OPERADOR estará limitado al utilizar el interface del programa, recibiendo los pertinentes avisos del modo de introducir los datos.

Ahora bien, aunque es completamente válido implementar esta integridad en el programa de cliente, lo más eficaz es colocarlo en el servidor, en la propia base de datos. Ya que no sabemos ni el número ni el tipo de programas que se conectará a la base de datos, y nosotros como desarrolladores tendríamos que incluir este tipo de restricciones en cada uno de los programas desarrollados, a parte del peligro que supondría aquellos programas clientes, que nuestra empresa a adquirido y a los que no tenemos acceso para modificar e incluir estas reglas.

Formas de normalización

Las formas normales definen una serie de normas o reglas que ayudan a organizar los datos en la estructura lógica de una base de datos.

Cada una de las formas que vamos a ir explicando heredan las reglas de su antecesora, así la forma normal C, incluye las reglas de las formas A y B. Para entender desde un sentido practico los diferentes modos de normalización, vamos a tomar como ejemplo la base de datos de la empresa OPERADORa de bebidas.

Recordamos la tabla Bebidas de esta base de datos:

Vamos a suponer que tenemos esta tabla con los siguientes datos:

Debes tener claro que esta tabla contiene los datos sin ser normalizados, si bien son los datos que deseamos gestionar. A continuación, nos basaremos en esta tabla para ver como aplicar sobre ella el proceso de normalización. Por supuesto, sólo es una tabla de prueba, los campos que en un caso deberíamos controlar serían muchos más.

Podemos concluir el proceso de normalización cuando analizando nuestras tablas comprobamos que somos capaces de realizar una actualización sin tener que cambiar más de un dato para cada actualización.

Cada uno de los campos de la tabla solo puede almacenar un tipo de datos, y además cada dato sólo se almacenar por separado, es decir individualmente

Esta regla que hemos anunciado puedes encontrarla con la definición de la regla de datos atómicos o indivisibles.

Para entender mejor el significado de esta regla, vamos a explicar como podríamos quebrantarla. En la tabla que acabamos de presentar, vemos que estamos guardando los datos del fabricante y de la distribuidora en dos campos diferentes. Un modo de saltarnos la regla de la forma normal A, es guardar el nombre del fabricante y el nombre de la distribuidora en un único campo. De este modo no estamos cumpliendo la norma, ya que estamos guardando información de diferentes características en un único campo.

Otra manera de no cumplir la regla que estamos viendo es la repetición de un campo. Esta técnica es muy común en administradores que se están iniciando en el desarrollo de bases de datos.

Con el proceso de normalización hemos conseguido evitar al máximo la redundancia de datos, permitiendo realizar modificaciones de un modo cómodo.

Para ello hemos indicado que desglosamos nuestra base de datos en tantas tablas como sea necesario.

De todas las reglas que hemos visto hay una que está por encima de todas, la lógica y la experiencia del administrador. Estas reglas no son obligatorias, son aconsejables en muchos casos y son de gran ayuda.

La lógica del programador puede indicarle que siguiendo la normalización de su base de datos, ha conseguido desglosar su estructura en tantas tablas con sus consiguientes relaciones. Esto puede provocar que la búsqueda de un registro tenga que llevarse a cabo a través de varias tablas y relaciones, con un rendimiento que deja mucho que desear.

Para solucionar esto, el desarrollador lleva a cabo el proceso de desnormalización, que tendrá consecuencias de redundancia de datos, pero que posiblemente sean necesario para la mejora del rendimiento.

Para conseguir alcanzar el término medio entre el proceso de normalización y desnormalización, el mejor medio es la experiencia. Se expone la base de datos a explotación como prueba piloto y se analiza la actividad que se realiza sobre ella, estudiando si los resultados se adaptan a las necesidades y cumplen con el rendimiento esperado, sino es así, gracias a estos estudios podremos ver que debemos modificar para mejorar nuestro diseño. SQL Server tiene la herramienta SQL Server Profiler que nos ayuda a realizar este tipo de análisis.

El servidor SQL Server ofrece un grupo de herramientas que ayudan en el proceso de normalización. Gracias a estas herramientas podremos gestionar nuestras tablas de modo que los datos se añadan con la lógica deseada y que las modificaciones cumplan los requisitos deseados.

Con estas herramientas podremos indicar a nuestro servidor como debe administrar la normalización, y nos ahorraremos muchas líneas de código en aplicaciones para que se encarguen de ella. Esto supone una gran ventaja frente a otras bases de datos.

Vamos a explicar brevemente las herramientas que tendremos ocasión de ver como utilizarlas en próximos capítulos. Estas herramientas son:

- Identidad
- Restricciones
- Integridad en relaciones
- Disparadores

Definición de claves principales.

Una clave principal contiene información de un registro de tal modo que gracias a esa información podamos distinguir ese registro de todos los demás, visto de otro forma, la información de la clave principal hace a un registro único e irrepetible en una tabla.

Un clave principal puede estar compuesta por una o varias columnas. En caso de estar formada por varias columnas es requisito indispensable que ninguna de esas columnas tenga información repetida en un mismo registro.

- Una buena clave principal puede ser aquella que tiene una información a la que los OPERADORs pueden acceder con facilidad, o que de la que tienen mayor conocimiento.
- Puede tener varias columnas. La mejor opción es tratar de que el número de columnas que forman la clave principal sean las menos posibles. Si una clave principal es válida con dos columnas, añadir más columnas no incremente la exclusividad de la clave principal (si es exclusiva con dos, será imposible aumentarla), lo único que provocamos si añadimos más columnas es bajar el rendimiento de las operaciones que se realicen con ellas.

En una base de datos, nos podemos encontrar tablas de las cuales no podemos encontrar ninguna columna que sea clave candidata a formar una clave principal. Para cumplir con la forma normal A del proceso de normalización debemos incluir una clave principal como mínimo en nuestras tablas. La única solución que nos queda si se nos presenta una tabla de este tipo es incluir una columna extra en nuestra tabla, la cual no almacena información que defina la información que almacenamos, pero tiene la importante tarea de ser la clave principal que distinga un registro del resto, cumpliendo con la integridad de entidad, y nos ayude con las relaciones.

Por ejemplo podemos tener una agenda de teléfonos, con los campos nombre, teléfono y dirección. Esta persona puede cambiar de teléfono, dirección y hasta si nos ponemos drásticos, incluso de nombre. Solucionamos el problema añadiendo un campo con un número para cada uno de los contactos que tenemos en esta tabla, y el problema queda resuelto. Como vemos, este problema es muy frecuente en nuestras tablas y es una solución muy cómoda, incluso para tablas en las que dudemos que sus campos puedan cumplir con los tres conceptos que hemos definido para ayudarnos con la selección de claves principales.

Identidad

Podemos tener una columna configurada como columna identidad, esta columna identidad es la manera más sencilla de garantizar la integridad de identidad.

La columna de identidad es una columna para la cual el propio servidor de base de datos se encarga de asignarle valores automáticamente. Por defecto, el primer valor es uno, y los siguientes registros van aumentando este valor de unidad en unidad. Aunque estos valores, son por defecto, veremos como se pueden modificar.

Una columna identidad es el mejor modo de añadir claves suplentes, como explicamos en el anterior capítulo, cuando una tabla no puede dar de modo natural una clave principal, será tarea nuestra añadir columnas que formen esa clave principal, pues el modo más eficaz es añadir columnas identidad.

Con este tipo de claves suplentes mejoramos considerablemente la relación entre tablas por columnas numéricas, bastante más eficaces que las claves principales formadas por textos.

Restricciones

Mediante las restricciones ponemos limitaciones a los datos que se van a introducir en la base de datos. Determinamos que datos son válidos para insertar en la columna de una tabla.

Tenemos las restricciones UNIQUE, DEFAULT y CHECK que fuerzan la integridad de identidad, dominio y la marcada por OPERADOR. Y por otro lado contamos con las restricciones PRIMARY KEY y FOREIGN KEY para garantizar la integridad referencial en las relaciones.

UNIQUE

Esta restricción obliga a que todos los valores de una determinada columna no estén repetidos en otros registros. Si tenemos varias restricciones UNIQUE en una misma tabla, todas deben ser cumplidas a la vez para cada registro.

Con la restricción UNIQUE aseguramos la integridad de identidad de la tabla, ya que cumplimos con la norma de que cada registro es diferente al resto. Si aplicamos claves principales a una tabla, automáticamente se asigna esta restricción a esa columna.

No debes pensar que una columna identidad que se incrementa ella sólo automáticamente, es otro modo de tener una restricción UNIQUE, ya que se pueden dar casos en que tengamos valores duplicados, a no se que marquemos esa columna como clave suplente o principal. Puedes llegar a esta conclusión errónea si has trabajado con ACCESS, pero en SQL Server no es así.

DEFAULT

Como su propio nombre indica, esta restricción introduce un valor por defecto en una columna cuando no se índica ningún valor para insertar. Con esta restricción aseguramos la integridad de dominio, ya que aseguramos valores válidos para nuevos registros que se inserten.

CHECK

Esta restricción evalúa por medio de expresiones los valores que se insertan en una columna. Esta expresión, una vez que se evalúa devuelve un resultado, en función de si el dato es válido (Verdadero) o no (Falso), por lo tanto devuelve un valor booleano que indica si el dato tendrá permiso para ser ingresado o no.

Como puedes ver, nos ayuda a asegurar la integridad de dominio, y si vamos un poco más allá, también nos ayuda a asegurar la estabilidad de relaciones en configuraciones mucho más avanzadas.

Integridad en relaciones

Este tipo de integridad, denominada integridad referencial declarativa (DRI - Declarative Referential Integrity), es el proceso por el cual SQL Server fuerza de manera automática las relaciones entre tablas. Antes de aparecer este tipo de integridad para servidores SQL Server, era necesario desarrollar códigos para aplicaciones denominadas desencadenadores para cada tabla, y estos se encargaban de ejecutar una serie de acciones que asegurasen esta integridad, y siempre bajo la supervisión del administrador.

A este tipo de integridad llegamos ahora de manera automática, de un modo muy sencillo y con un rendimiento considerable, de modo que el administrador puede dedicarse a otras tareas. Para conseguir esta integridad tenemos dos tipos de restricciones: PRIMARY KEY y FOREIGN KEY.

PRIMARY KEY

La clave principal (PRIMARY KEY) nos permite asegurar la integridad de entidad (puesto que es única en cada registro) y por otro lado nos garantiza la estabilidad de las relaciones con otras tablas.

FOREIGN KEY

La restricción FOREIGN KEY, se conoce como la clave externa o foránea que ya hemos explicado. Y como ya sabes es la pareja de la restricción PRIMARY KEY, y juntas cumplen con la integridad referencial.

Una clave externa es una copia de la clave principal de la tabla principal, se inserta en la tabla que se pretende enlazar y con esto creamos la relación entre un par de tablas. Las claves externas pueden ser varias en una misma tabla, mientra que las principales deben ser únicas.

Para que esta relación que comentamos se cumpla, la clave principal que enlaza con la externa debe cumplir obligatoriamente que las dos columnas sean del mismo tipo.

Integridad referencial en cascada

Esta tipo de integridad que Nortegió con la versión 2000 de SQL Server, permite una serie de operaciones, que sólo pueden llevarse a cabo de este modo y no de otro.

Desencadenadores

Los desencadenadores representan aplicaciones que desarrollamos en lenguaje T-SQL y que se ejecutan, o mejor dicho, se "disparan" cuando sucede algún tipo de evento en una tabla. Los desencadenadores se llaman también disparadores o triggers.
En función del tipo de evento, tenemos los siguientes grupos de desencadenadores:

- Desencadenadores de inserción. Estos desencadenadores se ejecutan cuando se añade un registro o varios.
- Desencadenadores de actualización. Se ejecutan cuando se ha actualizado uno o varios registros.
- Desencadenadores de eliminación.

Con estos desencadenadores aseguramos la lógica de negocio y definimos la integridad de OPERADOR. Antiguamente (versiones anteriores a SQL Server 2000), la integridad referencial en cascada tenía que implementarse mediante desencadenadores que permitiesen la actualización y eliminación en cascada.

Descripción de las columnas de tablas

Nombre de la Columna
Aquí escribiremos el nombre del campo (en la imagen la primera columna se llama ID Pruebas.
Tipos de Datos
Asignamos el tipod e datos del campo. Hay muchos tipos de datos como integer, numeric, char, varchar, datetime, etc. El cual elegiremos al que mas se adecue a nuestro campo.
Longitud
Longitud del campo de un tipo de datos numérico es el número de bytes utilizados para su almacenamiento de una cadena que pueda contener
El permitir valores nulos esta opción esta activada, el cual permite la asignación de valores nulos (en blanco) cuando se den alta de registros.

Los campos auto incrementales como identify no permiten que sean nulos.
Cada fila de la columna tiene una serie de opciones que se pueden modificar a gusto del OPERADOR o dependiendo de la función que realice el campo. Estas opciones se encuentran en la parte inferior de la pantalla en las ficha “columnas”.

Descripción de las opciones:
Permite poner una descripción al campo

- Valor predeterminado: inicializa el campo con un valor por defecto, de este modo cuando se crea un registro nuevo el campo torna este valor y lo inserta sin necesidad que nosotros lo pongamos.
- Precisión: Es el número de dígitos de un número.
- Escala: Es el número de dígitos situado a la derecha de la coma decimal de un número.
- Identidad: (Identity) con este campo indicamos a sql que el campo numérico es autoincremental.
- Inicialización de identidad inicia el campo identidad a un valor determinado.
- Incremento de identidad: incremento del campo identidad.
- Intercalación : Especifica el juego de caracteres y el orden de la tabla, sino se pone nada toma la intercalación por defecto de SQL SERVER.

Verificación de la definición de la tabla

SP_HELP <Nombre de la tabla>

Tipo de datos definidos por el OPERADOR

SP_ADDTYPE

EJEMPLOS PRACTICOS

EJEMPLO 031

Crear la tabla llamada ESTRELLITA, dentro de la Base de datos AMOR

```
USE AMOR
GO
CREATE TABLE ESTRELLITA (
        IdCod           CHAR ( 4 ) NOT NULL,
        Nombre          VARCHAR(30) NOT NULL,
        Precio          DECIMAL(10,2) NOT NULL,
        Edad            SMALLINT NOT NULL
Go
```

NOTA

Para realizar la verificación de la tabla empleamos SP_HELP NOMBRE DE LA TABLA

TABLA DEL SISTEMA SYSOBJECTS	
TIPO	DESCRIPCION DE LA VARIABLE TIPO
F	RESTRICCION CHECK
C	RESTRICCION CHECK
D	VALOR PREDETERMINADO O RESTRICCION DEFAUL
K	RESTRICCION PRIMARY KEY O UNIQUE
L	REGISTRO
P	PROCEDIMIENTO ALMACENADO
R	REGLA
RF	PROCEDIMIENTO ALMACENADO O FILTRO DUPLICADOS
S	TABLA DEL SISTEMA
TR	DESENCADENADOR
U	TABLA DE OPERADOR
V	VISTA
X	PROCEDIMIENTO ALMACENADO

Tipo de datos definidos por el OPERADOR

SP_ADDTYPE

EJEMPLO 032

Crear el tipo de datos DNI que no admita valores NULOS en la tabla ESTRELLA.

```
USE AMOR
Go
        EXEC SP_ADDTYPE DNI, CHAR(8), “NOT NULL”
```

EJEMPLO 033

¿Cómo eliminar un tipo de datos definidos por el OPERADOR?

```
USE AMOR
EXEC SP_DROPTYPE DNI
```

LA VISTA INFORMATION SCHEMA DOMAINSV

Contiene una fila por cada tipo de datos definido por el OPERADOR

EJEMPLO 034

Utilizando T - SQL para crear la base de datos Pais que contiene 03 bytes de tipo carácter.
Además crear el tipo de datos Dirección que contiene 60 bytes de tipo carácter.

```
USE AMOR
GO
IF EXISTS (SELECT Domain_Nombres FROM information_schema_domains
WHERE domain_schema =”dbo”  AND domanin_Nombres = ””PAIS”)
        EXEC SP_DROPTYPE Pais
GO
        EXCEC SP_Addtype Pais, “Char(3)”
Go

/*Ahora el tipo de datos DIRECCION */

IF EXISTS (SELECT Domain_Nombres FROM information_schema_domains
WHERE domain_schema =”dbo”  AND domanin_Nombres = ””DIRECCION”)
EXEC SP_DROPTYPE direccion, “varchar(30)”

GO
```

EJEMPLO 035

Ahora si deseamos verificar la creación del tipo de datos creado, pues ejecutemos el siguiente script

```
USE AMOR
GO
Select domain_Nombres FROM information_schema.domains ORDER BY
domain_Nombres

GO
```

MODIFICAR UNA TABLA

ALTER TABLE

EJEMPLO 036

Adicionar un campo llamado Manzana a la Base de datos BDEJEMPLO_01

ADICIONAR UNA COLUMNA A UNA TABLA

```
USE BDEJEMPLO_01
GO
ALTER TABLE PAIS
ADD MANZANA CHAR(10) NULL
```

EJEMPLO 037

Crear una tabla llamado OPERADOR que contenga los siguientes campos CODIGO C(10), NOMBRES C(40), FECHAINGRESO, EDAD, MONTOTOTAL N(13,2)

```
USE BDEJEMPLO_01
GO
IF EXISTS(SELECT * FROM SYSOBJECT WHERE TYPE =´U¨ AND NOMBRES = ¨OPERADOR¨)
        DROP TABLE OPERADOR
        GO

CREATE TABLE OPERADOR (
 CODIGO                 CHAR(10)            NOT NULL,
NOMBRES                  VARCHAR(40)         NULL,
FECHAINGRESO            SMALLDATETIME,
EDAD                    SMALLINT            NOT NULL,
MONTOTOTAL              DECIMAL(13,2)       NULL )
GO
```

EJEMPLO 038

En la tabla OPERADOR, añadir una columna llamada teléfono sin restricción

```
USE BDEJEMPLO_01
GO
ALTER TABLE OPERADOR
ADD TELEFONO  CHAR(10) NULL
GO
```

VERIFICAR LOS CAMBIOS DE UNA TABLA

SP_HELP

EJEMPLO 039

Revisar los cambios de la tabla anterior:

```
Sp_help OPERADOR
Go
```

EJEMPLO 040

Añadir y cambiar los valores de una columna

```
USE BDEJEMPLO_01
GO
ALTER TABLE COLUMN NOMBRES VARCHAR (40) NOT NULL
GO
```

NOTA

No se puede añadir una columna con valores NO NULOS , pero si se puede cambiar la propiedad NULL por NOT NULL.

En el ejemplo 037, cuando se creo la tabla OPERADOR el campo de la columna Nombres era NULL pues en el ejemplo 040, lo cambiamos los valores de NULL a NOT NULL.

CONSTRAINT

Especifica el comienzo de una restricción, PRIMARY KEY, UNIQUE, FOREIGN KEY o CHECK, de una definición DEFAULT

EJEMPLO 041

En la tabla OPERADOR, agregar una columna que acepta NULL con valores predeterminados

```
ALTER TABLE OPERADOR
ADD FECHASALIDA     DATETIME NULL
        CONSTRAINT FECHADFLT
        DEFAULT GETDATE() WITH VALUES
```

EJEMPLO 042

Usando SCRIPT T SQL crear la tabla producto en la base de datos BDEJEMPLO_O1

```
USE BDEJEMPLO_01
GO
IF EXISTS(SELECT * FROM SYSOBJECT WHERE TYPE = ´U´ AND NOMBRES = ¨PRODUCTO ´)
DROP TABLE PRODUCTO
GO
CREATE TABLE PRODUCTO (
IDPRODUCTO          CHAR(|0)        NOT NULL,
DETALLE             VARCHAR(30)  NULL,
PRECIO              DECIMAL(10,2) NULL,
STOCK               INT NOT         NULL,
FECHA               DATETIME)
GO
```

QUITAR UNA COLUMNA DE UNA TABLA

Para quitar una columna de una tabla empleamos DROP COLUMN

EJEMPLO 043

Quitar la columna Montototal

```
USE BDEJEMPLO_01
GO
ALTER TABLE OPERADOR
        DROP COLUMN montototal
 GO
EXEC SP_HELPDB OPERADOR
GO
```

EJEMPLO 044

Agregar una columna con una restricción.

```
 USE BDEJEMPLO_01
GO
ALTER TABLE OPERADOR
ADD DNI CHAR(10) NULL
        CONSTRAINT USU_UNIQUE UNIQUE
GO
```

RENOMBRAR A UNA TABLA SP_RENOMBRES

EJEMPLO 045

Cambiar el nombre de la tabla OPERADOR por padrón

```
USE BDEJEMPLO_01
GO
EXEC SP_RENOMBRES ´OPERADOR¨, ¨PADRON¨
```

VERIFICAR LA CREACION DE LA TABLA

EJEMPLO 046

```
USE BDEJEMPLO_01
SELECT table_Nombres FROM Information_schema.tables WHERE table_Nombres = ”producto”
GO
```

EJEMPLO 047

Crear las tablas CLIENTE y EMPLEADO haciendo uso de datos necesarios en un sistema

```
USE BDEJEMPLO_01
GO
IF EXISTS(SELECT * FROM S  ysObject WHERE type =”U” and Nombres =”CLIENTE”)
        DROP TABLE CLIENTE
        GO
CREATE TABLE CLIENTE
(id_codigo      char(08) NOT NULL,
Nombres    varchar(30) NOT NULL,
Fecha     datetime,
Pais_lugar varchar (20) NOT NULL,
Departamento varchar (30) NOT NULL
)
Go
IF EXISTS(SELECT * FROM S  ysObject    WHERE    type    =”U”    and    Nombres =”EMPLEADO”)
        DROP TABLE EMPLEADO
        GO
CREATE TABLE EMPLEADO
(id_clicod          char(10) NOT NULL,
Nomb_emp    varchar(30) NOT NULL,
Fechaingreso smalldatetime
)
```

EJEMPLO 048

Verificar la creación de las tablas, luego de ejecutarlos

```
SELECT table_Nombres FROM information_schema.tables
WHERE table_Nombres =”Cliente”


SELECT table_Nombres FROM information_schema.tables
WHERE table_Nombres =”Empleado”
```

EJEMPLO 049

Diga usted que sucede al ejecutar el siguiente script :

```
SELECT * FROM SYSOBJECT ORDER BY TYPE
```

EJEMPLO 050

Crear una restricción PRIMARY KEY sobre la columna id_codigo de la tabla cliente, que se encuentra dentro de la base de datos BDEJEMPLO01

```
USE BDEJEMPLO01
GO
ALTER TABLE CLIENTE

GO
ADD CONSTRAINT Id_codigo PRIMARY KEY
GO
```

EJEMPLO 051

Escriba los comandos necesarios para mostrar los siguientes mensajes:

Bienvenidos <<OPERADOR>>
Hoy es <<fecha del sistema>>

```
DECLARE @OPERADOR varchar(20)
DECLARE @FECHA DATETIME
SET @OPERADOR =”CESITAR”
SET @FECHA = GETDATE()
        PRINT “BIENVENIDO”+ @OPERADOR
        PRINT “HOY ES ” + @FECHA
```

EJEMPLO 052

Diga usted como crear los siguientes tipos de datos, definidos por el OPERADOR en la base de datos BDEJEMPLO02

```
Use BDEJEMPLO02
EXEC SP_addtype Apellidos, “varchar(40)”, “NOT NULL”
EXEC SP_addtype Nota, “decimal(10,2)”, “NOT NULL”
EXEC SP_addtype Promedio, “Integer”, “NOT NULL”
```

EJEMPLO 053

Diga usted como crear la siguiente estructura de la tabla llamada Alumnos, en la base de datos BDEJEMPLO02

ID CODIGO	CODIGO	NOT NULL
NOMBRES	VARCHAR(30)	NOT NULL
APELLIDOS	VARCHAR(30)	NOT NULL
EDAD	INTEGER	
FECHA	DATETIME	
SEXO	CHAR (1)	

EJEMPLO 054

Agregar un campo llamado DNI de (10) caracteres y que acepte valores NULOS en la tabla Alumnos

```
USE BDEJEMPLOS02
GO
ALTER TABLE ALUMNOS
ADD DNI CHAR (10) NULL
GO
```

EJEMPLO 055

Agregar un campo llamado Fechanacimiento de tipo fecha, y que acepte valores predeterminados GETDATE() en la tabla NOTAS

```
ALTER TABLE NOTAS
ADD fechanacimiento DATETIME NULL
CONSTRAINT FECHADFLT DEFAUL GETDATE() WITH values
```

EJEMPLO 056

Agregar un campo llamado OBSERVACION de tipo VARCHAR (20) en la tabla NOTAS

```
ALTER TABLE NOTAS
ADD observacion varchar(20) NULL
GO
```

NOTA

No se puede añadir una columna con valores NO NULOS , pero si se puede cambiar la propiedad NULL por NOT NULL.
En el ejemplo 037, cuando se creo la tabla OPERADOR el campo de la columna Nombres era NULL pues en el ejemplo 040, lo cambiamos los valores de NULL a NOT NULL

EJEMPLO 057
Cambiar el nombre de la tabla NOTAS por el nombre de EVALUACION

```
EXECE SP_RENOMBRES  “NOTAS”, “EVALUACION”
GO
```

EJEMPLO 058
Eliminar las tablas creadas en la base de datos BDEJEMPLO01

```
USE BDEJEMPLO01
DROP TABLE
```

EJEMPLO 059
Calcular la Suma de los números pares e impares comprendidos entre 1 y 10

```
Declare @num1  INT
Declare @num2  INT
Declare @suma1 INT
Declare @suma2 INT
Set @num1  =  0
Set @num2  =  0
Set @suma1 = 0
Set @suma2 = 0
WHILE (@num1<=10 AND @num2<=10)
        BEGIN
        SET @num1 = 2@num1
                PRINT @num1
        SET @num2 = 2@num2+1
                PRINT @num2
        SET @suma1 = @suma1+@num2
        SET @suma2 = @suma2+@num1
        END
PRINT ´LA SUMA DE LOS IMPARES ES:´+STR(@suma1)
PRINT ´LA SUMA DE LOS PARES ES:´+STR(@suma2)
```

MEMORIA AYUDA

Cómo saber que versión de SQL tenemos instalado en nuestra computadora

Cada vez que se libera una actualización para algún producto de Microsoft queremos instalarlo en nuestro computador para estar al corriente de las nuevas mejoras y ajustes.

Para el caso particular de SQL Server podemos efectuar una consulta muy simple que nos permite recordar la versión actual que tenemos instalada en nuestra máquina.
La sentencia es la siguiente:

```
SELECT    SERVERPROPERTY('productversion')   AS VERSION,
          SERVERPROPERTY('productlevel')     AS NIVELPRODUCTO,
          SERVERPROPERTY('edition')          AS EDICIÓN
```

INTEGRIDAD DE DATOS

Se refiere a la consistencia y exactitud de los datos que se guardan en una Base de datos.

Niveles o etapas de integridad de datos

- **Integridad de Entidad**
  - **Primary Key**
  - **Unique**
  - **Identity**
- **Integridad de Dominio**
  - **Default**
  - **Foreign Key**
  - **Check**
  - **Not Null**
- **Integridad Referencial**
  - **Foreign Key**
  - **Check**
- **Integridad definida por el OPERADOR**
  - **Todas las restricciones en columnas y de la tabla CREATE así como propiedades almacenadas.**

TIPOS DE RESTRICCIONES

Las restricciones son un método estándar ANSI para formar la integridad de los datos.

PRIMARY KEY	**Clave Primaria**
UNIQUE	**Valor no duplicado**
FOREIGN KEY	**Clave foránea**
DEFAULT	**Valor predeterminado**
CHECK	**Regla de validación**

La Clausula CONSTRAINT

Las restricciones son un método estándar ANSI para formar
Se utiliza la cláusula CONSTRAINT **en las instrucciones** ALTER TABLE **y** CREATE TABLE **para crear o eliminar índices. Existen dos sintaxis para esta cláusula dependiendo si desea Crear ó Eliminar un índice de un único campo o si se trata de un campo multiíndice. Si se utiliza el motor de datos de Microsoft, sólo podrá utilizar esta cláusula con las bases de datos propias de dicho motor. Para los índices de campos únicos:**
CONSTRAINT nombre {PRIMARY KEY | UNIQUE | REFERENCES tabla externa [(campo externo1, campo externo2)]}

Para los índices de campos múltiples:
CONSTRAINT nombre {PRIMARY KEY (primario1[, primario2 [,...]]) |
UNIQUE (único1[, único2 [, ...]]) |
FOREIGN KEY (ref1[, ref2 [,...]]) REFERENCES tabla externa
[(campo externo1 ,campo externo2 [,...])]}

nombre	Es el nombre del índice que se va a crear.
primarioN	Es el nombre del campo o de los campos que forman el índice primario.
únicoN	Es el nombre del campo o de los campos que forman el índice de clave única.
refN	Es el nombre del campo o de los campos que forman el índice externo (hacen referencia a campos de otra tabla).
tabla externa	Es el nombre de la tabla que contiene el campo o los campos referenciados en refN
campos externos	Es el nombre del campo o de los campos de la tabla externa especificados por ref1, ref2,... , refN

Si se desea crear un índice para un campo cuando se esta utilizando las instrucciones **ALTER TABLE** o **CREATE TABLE** la cláusula **CONTRAINT** debe aparecer inmediatamente después de la especificación del campo indexado.

Si se desea crear un índice con múltiples campos cuando se está utilizando las instrucciones **ALTER TABLE** o **CREATE TABLE** la cláusula **CONSTRAINT** debe aparecer fuera de la cláusula de creación de tabla.

Indice	Descripción
UNIQUE	Genera un índice de clave única. Lo que implica que los registros de la tabla no pueden contener el mismo valor en los campos indexados.
PRIMARY KEY	Genera un índice primario el campo o los campos especificados. Todos los campos de la clave principal deben ser únicos y no nulos, cada tabla sólo puede contener una única clave principal.
FOREIGN KEY	Genera un índice externo (toma como valor del índice campos contenidos en otras tablas). Si la clave principal de la tabla externa consta de más de un campo, se debe utilizar una definición de índice de múltiples campos, listando todos los campos de referencia, el nombre de la tabla externa, y los nombres de los campos referenciados en la tabla externa en el mismo orden que los campos de referencia listados. Si los campos referenciados son la clave principal de la tabla externa, no tiene que especificar los campos referenciados, predeterminado por valor, el motor Jet se comporta como si la clave principal de la tabla externa estuviera formada por los campos referenciados.

Las restricciones

Las restricciones son un método estándar ANSI para formar
Definición de claves para tablas y restricciones

Clave primaria: Primary key

Es una columna o un conjunto de columnas que identifican unívocamente a cada fila. Debe ser única, no nula y obligatoria. Como máximo, podemos definir una clave primaria por tabla.

Esta clave se puede referenciar por una columna o columnas. Cuando se crea una clave primaria, automáticamente se crea un índice que facilita el acceso a la tabla.

Formato de restricción de columna:

```
 CREATE TABLE NOMBRE_TABLA

(COL1 TIPO_DATO [CONSTRAINT NOMBRE_RESTRICCION] PRIMARY KEY
  COL2 TIPO_DATO
…
)[TABLESPACE ESPACIO_DE_TABLA];
```

Formato de restricción de tabla:

```
  CREATE TABLE NOMBER_TABLA
  (COL1 TIPO_DATO,
  COL2 TIPO_DATO,
 …
[CONSTRAINT NOMBRERESTRICCION] PRIMARY KEY (COLUMNA [,COLUMNA]),
…
)[TABLESPACE ESPACIO_DE_TABLA];
```

FOREIGN KEY

Claves ajenas: Foreign Key:

Esta formada por una o varias columnas que están asociadas a una clave primaria de otra o de la misma tabla. Se pueden definir tantas claves ajenas como se precise, y pueden estar o no en la misma tabla que la clave primaria. El valor de la columna o columnas que son claves ajenas debe ser: NULL o igual a un valor de la clave referenciada (regla de integridad referencial).

Formato de restricción de columna:

```
  CREATE TABLE NOMBRE_TABLA
  (COLUMNA1 TIPO_DATO
  [CONSTRAINT NOMBRERESTRICCION]
REFERENCES NOMBRETABLA [(COLUMNA)] [ON DELETE CASCADE]
    …
    )[TABLESPACE ESPECIO_DE_TABLA];
```

Formato de restricción de tabla:

```
  CREATE TABLE NOMBRE_TABLA
  (COLUMNA1 TIPO_DATO,
  COLUMNA2 TIPO_DATO,
  …
  [CONTRAINT NOMBRERESTRICCION]
  FOREIGN KEY (COLUMNA [,COLUMNA])
    REFERENCES NOMBRETABLA [(COLUMNA [,
    COLUMNA])]
      [ON DELETE CASCADE],
  )[TABLESPACE ESPACIO_DE_TABLA];
```

Notas:

- **En la cláusula REFERENCES indicamos la tabla a la cual remite la clave ajena.**
- **Hay que crear primero una tabla y después aquella que le hace referencia.**
- **Hay que borrar primero la tabla que hace referencia a otra tabla y después la tabla que no hace referencia.**
- **Borrado en cascada (ON DELETE CASCADE): Si borramos una fila de una tabla maestra, todas las filas de la tabla detalle cuya clave ajena sea referenciada se borraran automáticamente. La restricción se declara en la tabla detalle. El mensaje "n filas borradas" solo indica las filas borradas de la tabla maestra.**

NOT NULL: Significa que la columna no puede tener valores nulos. DEFAULT: Le proporcionamos a una columna un valor por defecto cuando el valor de la columna no se especifica en la cláusula INSERT. En la especificación DEFAULT es posible incluir varias expresiones: constantes, funciones SQL y variables UID y SYSDATE.

Verificación de restricciones: CHECK: Actúa como una cláusula where. Puede hacer referencia a una o más columnas, pero no a valores de otras filas. En una cláusula CHECK no se pueden incluir subconsultas ni las pseudoconsultas SYSDATE, UID y USER.

> **Nota: La restricción NOT NULL es similar a CHECK (NOMBRE_COLUMNA IS NOT NULL)**

UNIQUE: Evita valores repetidos en la misma columna. Puede contener una o varias columnas. Es similar a la restricción PRIMARY KEY, salvo que son posibles varias columnas UNIQUE definidas en una tabla. Admite valores NULL. Al igual que en PRIMARY KEY, cuando se define una restricción UNIQUE se crea un índice automáticamente.

Vistas del diccionario de datos para las restricciones:

Contienen información general las siguientes:

USER_CONSTRAINTS: Definiciones de restricciones de tablas propiedad del OPERADOR.
ALL_CONSTRAINTS: Definiciones de restricciones sobre tablas a las que puede acceder el OPERADOR.

DBA_CONSTRAINTS: Todas las definiciones de restricciones sobre todas las tablas.

Creación de una tabla con datos recuperados en una consulta:

CREATE TABLE: permite crear una tabla a partir de la consulta de otra tabla ya existente. La nueva tabla contendrá los datos obtenidos en la consulta. Se lleva a cabo esta acción con la cláusula AS colocada al final de la orden CREATE TABLE.

```
CREATE TABLE NOMBRETABLA
(COLUMNA [,COLUMNA]
)[TABLESPACE ESPACIO_DE_TABLA]
AS CONSULTA;
```

No es necesario especificar tipos ni tamaño de las consultas, ya que vienen determinadas por los tipos y los tamaños de las recuperadas en la consulta. La consulta puede tener una subconsulta, una combinación de tablas o cualquier sentencia select valida.

Las restricciones CON NOMBRE no se crean en una tabla desde la otra, solo se crean aquellas restricciones que carecen de nombre.

Supresión y modificación de tablas con SQL
Supresión de tablas:

DROP TABLE: suprime una tabla de la base de datos. Cada OPERADOR puede borrar sus propias tablas, pero solo el administrador o algún OPERADOR con el privilegio

"DROP ANY TABLE" puede borrar las tablas de otro OPERADOR. Al suprimir una tabla también se suprimen los índices y los privilegios asociados a ella. Las vistas y los sinónimos creados a partir de esta tabla dejan de funcionar pero siguen existiendo en la base de datos por tanto deberíamos eliminarlos.

Ejemplo:

```
DROP TABLE [OPERADOR].NOMBRETABLA [CASCADE CONSTRAINTS];
```

TRUNCATE: permite suprimir todas las filas de una tabla y liberar el espacio ocupado para otros usos sin que reaparezca la definición de la tabla de la base de datos. Una orden TRUNCATE no se puede anular, como tampoco activa disparadores DELETE.

```
TRUNCATE TABLE [OPERADOR.]NOMBRETABLA [{DROP | REUSE} STORAGE];
```

Modificación de tablas:

Se modifican las tablas de dos formas: Cambiando la definición de una columna (MODIFY) ó añadiendo una columna a una tabla existente (ADD):
Formato:

```
ALTER TABLE NOMBRETABLA
{[ADD (COLUMNA [,COLUMNA]…)]
[MODIFY (COLUMNA [,COLUMNA]…)]
[ADD CONSTRAINT RESTRICCION]
[DROP CONSTRAINT RESTRICCION]};
```

ADD= Añade una columna o mas al final de una tabla.
MODIFY= Modifica una o mas columnas existentes en la tabla.
ADD CONSTRAINT= Añade una restricción a la definición de la tabla.
DROP CONSTRAINT= Elimina una restricción de la tabla.

A la hora de añadir una columna a una tabla hay que tener en cuenta:

- Si la columna no esta definida como NOT NULL se le puede añadir en cualquier momento.
- Si la columna esta definida como NOT NULL se pueden seguir estos pasos:
    1. Se añade una columna sin especificar NOT NULL.
    2. Se da valor a la columna para cada una de las filas.
    3. Se modifica la columna NOT NULL.

Al modificar una columna de duna tabla se han de tener en cuenta:

- Se puede aumentar la longitud de una columna en cualquier momento.
- Es posible aumentar o disminuir el numero de posiciones decimales en una columna de tipo NUMBER.
- Si la columna es NULL en todas las filas de la tabla, se puede disminuir la longitud y modificar el tipo de dato
- La opción MODIFY… NOT NULL solo será posible cuando la tabla no contenga ninguna fila con valor nulo en la columna que se modifica.

Adición de restricciones:

Con la orden ALTER TABLE se añaden restricciones a una tabla.
Formato:

```
ALTER TABLE NOMBRETABLA
ADD CONSTRAINT NOMBRECONSTRAINT…
```

Borrado de restricciones:

La orden ALTER TABLE con la cláusula DROP CONSTRAINT; con la que se borran las restricciones con nombre y las asignadas por el sistema. Formato:

```
ALTER TABLE NOMBRETABLA
DROP CONSTRAINT NOMBRE_CONSTRAINT,
NOMBRE_RESTRICCION:
```

INTEGRIDAD DE ENTIDAD

Define una fila como Entidad única para una tabla determinada.

EJEMPLO 060
Crear las siguientes tablas en la Base de datos BDEJEMPLO02, y en caso de existir debemos eliminar dichas tablas.

```
IF EXISTS(Select *.FROM SYSOBJECT WHERE Type =”U” and Nombres =”OPERADOR”)
        DROP TABLE OPERADOR
GO
CREATE TABLE OPERADOR (
IdOPERADOR          char(5)          NOT NULL,
Nom_OPERADOR              varcha(20)      NOT NULL,
Ape_OPERADOR              varchar(20)     NOT NULL,
Dir_OPERADOR              varchar(60)     NULL,
Tel_OPERADOR              varchar(10)     NULL
);
```

EJEMPLO 061
Crear una restricción PRIMARY KEY sobre la columna idOPERADOR de la tabla anterior, si existe eliminar la restricción para volver a crearla.

```
USE BDEJEMPLO02
IF EXISTS(Select *.FROM SYSOBJECT WHERE Nombres =´CONS_00´)
        ALTER TABLE OPERADOR
        DROP CONSTRAINT CONS_00
GO
        ALTER TABLE OPERADOR
        ADD CONSTRAINT CONS_00 PRIMARY KEY (idOPERADOR)
GO
```

Insertar Registros a una determinada tabla

EJEMPLO 062
Insertar registros a la tabla OPERADOR de la tabla creada en el ejemplo 060

```
ALTER TABLE OPERADOR
INSERT OPERADOR VALUES(“A1000”,”HARUMI”,”PEREDA PASCAL”,”MIRAFLORES LIMA”,”999999999”);
INSERT OPERADOR VALUES(“A2000”,”MARILUISA”,”PASCAL DE PEREDA”, ”MIRAFLORES LIMA”,”999999988”);
```

Clave Primaria Primary key

EJEMPLO 063
En la tabla OPERADOR definir como clave primaria al campo o la columna IduNorteaio

```
USE BDEJEMPLO02
GO
ALTER TABLE OPERADOR
        ADD CONSTRAINT PK_IDOPERADOR PRIMARY KEY (idOPERADOR)
GO
```

EJEMPLO 064
En la tabla OPERADOR del ejemplo 060, insertar registros a determinados campos

```
USE BDEJEMPLO02
GO
INSERT INTO OPERADOR (IdOPERADOR,Nom_OPERADOR,Ape_OPERADOR) VALUES (“A3000”,”CESAR”,”PEREDA TORRES”);
```

NOTA

Cuando se insertan números no van entre comillas)

EJEMPLO PRÁCTICO

Al insertar registros tales como fecha o número lo desarrollaremos ahora:

Supongamos que tenemos una tabla con los campos fecha y Gastos

INSERT INTO PLANILLA (IdOPERADOR, Nom_OPERADOR, Ape_OPERADOR, fechainicio, Gastos) VALUES (“A3000”,”CESAR”,”PEREDA TORRES”,”27/08/2000”,5500);

EJEMPLO 065

Revisar la tabla y verificar los registros ingresados

```
USE BDEJEMPLO02
GO
SELECT * FROM OPERADOR
GO
```

CLAVE PRIMARIA COMPUESTA

EJEMPLO 066

Teniendo una tabla determinada, vamos a crear una clave primaria compuesta, por dos columnas determinadas., como ejemplo consideramos la tabla MOVIMIENTOS y vamos a enlazar con dos columnas Numdocum y codigocliente que sean las dos columnas de la tabla.

```
USE BDEJEMPLO02
GO
ALTER TABLE MOVIMIENTOS
ADD CONSTRAINT PK_MOVIMIENTOS
PRIMARY KEY (Numdocum, codigocliente)
GO
```

NOTA

Se recomienda definir la clave primaria de una tabla con el prefijo PK_ para poder identificarlo, ya que si no lo especifica SQL le asignara uno automático.

INSERT INTO

Agregar un registro en una tabla. Se la conoce como una consulta de datos añadidos. Esta consulta puede ser de dos tipo: Insertar un único registro ó Insertar en una tabla los registros contenidos en otra tabla. Para insertar un único **Registro:**

En este caso la sintaxis es la siguiente:

```
INSERT INTO Tabla (campo1, campo2, ..., campoN)
VALUES (valor1, valor2, ..., valorN)
```

Esta consulta graba en el campo1 el valor1, en el campo2 y valor2 y así sucesivamente.

Para seleccionar registros e insertarlos en una tabla nueva

En este caso la sintaxis es la siguiente:

```
SELECT campo1, campo2, ..., campoN INTO nuevatabla
FROM tablaorigen [WHERE criterios]
```

Se pueden utilizar las consultas de creación de tabla para archivar registros, hacer copias de seguridad de las tablas o hacer copias para exportar a otra base de datos o utilizar en informes que muestren los datos de un periodo de tiempo concreto. Por ejemplo, se podría crear un informe de MOVIMIENTOS mensuales por región ejecutando la misma consulta de creación de tabla cada mes.

Para insertar Registros de otra Tabla:

En este caso la sintaxis es:

```
INSERT INTO Tabla [IN base_externa] (campo1, campo2, , campoN)
SELECT TablaOrigen.campo1, TablaOrigen.campo2,,TablaOrigen.campoN FROM
Tabla Origen
```

En este caso se seleccionarán los campos 1,2,..., n de la tabla origen y se grabarán en los campos 1,2,.., n de la Tabla. La condición SELECT puede incluir la cláusula WHERE

para filtrar los registros a copiar. Si Tabla y Tabla Origen poseen la misma estructura podemos simplificar la sintaxis a:

INSERT INTO Tabla SELECT Tabla Origen.* FROM Tabla Origen

De esta forma los campos de Tabla Origen se grabarán en Tabla, para realizar esta operación es necesario que todos los campos de Tabla Origen estén contenidos con igual nombre en Tabla. Con otras palabras que Tabla posea todos los campos de Tabla Origen (igual nombre e igual tipo).

En este tipo de consulta hay que tener especial atención con los campos contadores o autonuméricos puesto que al insertar un valor en un campo de este tipo se escribe el valor que contenga su campo homólogo en la tabla origen, no incrementándose como le corresponde.

Se puede utilizar la instrucción **INSERT INTO** para agregar un registro único a una tabla, utilizando la sintaxis de la consulta de adición de registro único tal y como se mostró anteriormente. En este caso, su código especifica el nombre y el valor de cada campo del registro. Debe especificar cada uno de los campos del registro al que se le va a asignar un valor así como el valor para dicho campo. Cuando no se especifica dicho campo, se inserta el valor predeterminado o Null. Los registros se agregan al final de la tabla.

También se puede utilizar **INSERT INTO** para agregar un conjunto de registros pertenecientes a otra tabla o consulta utilizando la cláusula **SELECT... FROM** como se mostró anteriormente en la sintaxis de la consulta de adición de múltiples registros. En este caso la cláusula **SELECT** especifica los campos que se van a agregar en la tabla destino especificada.

La tabla destino u origen puede especificar una tabla o una consulta. Si la tabla destino contiene una clave principal, hay que asegurarse que es única, y con valores no nulos; si no es así, no se agregarán los registros. Si se agregan registros a una tabla con un campo Contador, no se debe incluir el campo Contador en la consulta. Se puede emplear la cláusula IN para agregar registros a una tabla en otra base de datos.

Se pueden averiguar los registros que se agregarán en la consulta ejecutando primero una consulta de selección que utilice el mismo criterio de selección y ver el resultado. Una consulta de adición copia los registros de una o más tablas en otra. Las tablas que contienen los registros que se van a agregar no se verán afectadas por la consulta de adición. En lugar de agregar registros existentes en otra tabla, se puede especificar los valores de cada campo en un nuevo registro utilizando la cláusula VALUES. Si se omite la lista de campos, la cláusula VALUES debe incluir un valor para cada campo de la tabla, de otra forma fallará INSERT.

Ejemplos:
INSERT INTO Clientes SELECT ClientesViejos.* FROM ClientesNuevos

SELECT Registros.* INTO Programadores FROM Registros WHERE Categoria = 'Programador'

Esta consulta crea una tabla nueva llamada programadores con igual estructura que la tabla empleado y copia aquellos registros cuyo campo categoria se programador

INSERT INTO Registros (Nombre, Apellido, Cargo) VALUES ('Luis' , 'Sánchez', 'Becario')

INSERT INTO Registros SELECT OPERADORes.* FROM OPERADORes WHERE
Provincia = 'Peru'

EJEMPLO 067

Crear las claves primarias de las tablas CLIENTES y MOVIMIENTOS, considerando que están relacionadas mediante la columna código de la tabla CLIENTES y la columna codcli de la tabla MOVIMIENTOS.

```
USE BDEJEMPLO02
GO
ALTER TABLE CLIENTES
        ADD CONSTRAINT PK_CLIENTES
        PRIMARY KEY (CODIGO)
GO
ALTER TABLE MOVIMIENTOS
        ADD CONSTRAINT PK_MOVIMIENTOS
        PRIMARY KEY (CODCLI)
```

CLAVE FORANEA FK_

Es el Nombre de la restricción clave, el cual se le recomienda anteponer el prefijo (FK_), ya que si no lo especifica el nombre de la restricción SQL le asigna un nombre.

EJEMPLO 068

Vamos a considerar en este ejemplo del cual vamos a Crear la clave foránea de la tabla PAGO que apunta a la calve primaria de la tabla MATRICULA

```
ALTER TABLE PAGO
        ADD CONSTRAINT FK_PAGO_MATRICULA
        FOREIGN KEY (idcurso,idalumno) REFERENCES Matricula
GO
```

EJEMPLO 069

Ahora vamos a crear la clave foránea de la tabla matricula que hace referencia a la tabla curso

```
USE BDEJEMPLO02
GO
ALTER TABLE MATRICULA
        ADD CONSTRAINT FK_MATRICULA_CURSO
        FOREIGN KEY (idcurso) REFERENCES CURSO
GO
```

FOREIGN KEY

FOREIGN KEY – Vamos a crear un ejemplo donde tengamos dos tablas, una de ellas clientes y la otra llamada MOVIMIENTOS el cual están relacionadas ambas por el campo codcliente

"CLIENTES"

CODCLIENTE	Apellidos	Nombre	Direccions	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

"MOVIMIENTOS"

ORDEN	NUMVENTA	CODCLIENTE
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Tenga en cuenta que la columna "CODCLIENTE" en la tabla "MOVIMIENTOS" indica "CODCLIENTE" tabla "CLIENTES".

"CODCLIENTE" columna de la tabla "CLEINTES" es una clave principal.

"CODCLIENTE" columna de la tabla "MOVIMIENTOS" es una clave externa.

FOREIGN KEY no permite insertar datos incorrectos en la columna "CODCLIENTE" en la tabla "MOVIMIENTOS" y "CLEINTES".

SQL KEY Restricciones FOREIGN en el CREATE TABLE

El siguiente SQL crea una FOREIGN KEY columna "CODCLIENTE", al crear la tabla "MOVIMIENTOS":

Dentro de MySQL:

```
CREATE TABLE Orders (
O_Id int NOT NULL,
OrderNo int NOT NULL,
CODCLIENTE int,
PRIMARY KEY (O_Id),
FOREIGN KEY (CODCLIENTE) REFERENCES Persons(CODCLIENTE)
)
```

Desde SQL Server / Oracle / MS Access:

```
CREATE TABLE Órdenes (
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
CODCLIENTE int FOREIGN KEY REFERENCES CLEINTES(CODCLIENTE)
)
```

Puede distribuir la restricción FOREIGN KEY en varias columnas, para usar la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
1  CREATE TABLE Órdenes(
2  O_Id int NOT NULL,
3  OrderNo int NOT NULL,
4  CODCLIENTE int,
5  PRIMARY KEY (O_Id),
6  CONSTRAINT fk_PerÓrdenes FOREIGN KEY (CODCLIENTE)
7  REFERENCES CLEINTES(CODCLIENTE)
8  )
```

SQL FOREIGN KEY Restricciones con ALTER TABLE

El siguiente SQL crea una FOREIGN KEY columna "CODCLIENTE", cuando el "Reglamento" tabla ya está creada:

MySQL / SQL Server / Oracle / MS Access:

```
1  ALTER TABLE Órdenes
2  ADD FOREIGN KEY (CODCLIENTE)
3  REFERENCES CLEINTES(CODCLIENTE)
```

Puede distribuir la restricción FOREIGN KEY en varias columnas, para usar la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
1  ALTER TABLE Órdenes
2  ADD CONSTRAINT fk_PerÓrdenes
3  FOREIGN KEY (CODCLIENTE)
```

```
4 REFERENCES CLEINTES(CODCLIENTE)
```

La eliminación de FOREIGN KEY

Para quitar el limitador de FOREIGN KEY, use el siguiente código SQL:

MySQL:

```
1 ALTER TABLE Órdenes
2 DROP FOREIGN KEY fk_PerÓrdenes
```

SQL Server / Oracle / MS Access:

```
1 ALTER TABLE Órdenes
2 DROP CONSTRAINT fk_PerÓrdenes
```

Cómo deshabilitar restricciones FOREIGN KEY con instrucciones INSERT y UPDATE (Visual Database Tools)

SQL Server 2008 R2

Puede seleccionar la opción de deshabilitar una restricción FOREIGN KEY durante transacciones INSERT y UPDATE si sabe que los nuevos datos infringirán la restricción o si la restricción sólo se aplica a los datos que ya están en la base de datos.

Para deshabilitar una restricción FOREIGN KEY de instrucciones INSERT y UPDATE

1. En el Explorador de objetos, haga clic con el botón secundario en la tabla con la restricción y, a continuación, haga clic en **Diseño**.

   La tabla se abre en el Diseñador de tablas.

2. En el menú **Diseñador de tablas**, haga clic en **Relaciones**.
3. En el cuadro de diálogo **Relaciones de clave externa**, seleccione la relación en la lista **Relación seleccionada**.
4. En la cuadrícula, haga clic en **Eliminar regla** o en **Actualizar regla** y elija una acción en el cuadro de lista desplegable situado a la izquierda de la propiedad.
   - **Sin acción** Un mensaje de error indica al OPERADOR que no se permite la eliminación y, a continuación, se revierte la eliminación.
   - **Cascada** Elimina todas las filas que contengan datos implicados en la relación de clave externa.
   - **Establecer en NULL** Establece el valor como NULL cuando todas las columnas de clave externa de la tabla aceptan valores NULL. Sólo se aplica a SQL Server 2005.
   - **Establecer como predeterminado** Establece el valor predeterminado definido para la columna cuando todas las columnas de clave externa de la tabla tienen definidos valores predeterminados. Sólo se aplica a SQL Server 2005.

NOTA

Si pretende emplear desencadenadores para implementar operaciones de base de datos, debe deshabilitar las restricciones FOREIGN KEY para que se ejecute el desencadenador

RESTRICCIONES UNIQUE VALOR A NO DUPLICARSE (U_)

EJEMPLO 070

Ahora vamos a suponer que un sistema de Matricula los códigos del alumno no se deben repetir para ello estableceremos una restricción UNIQUE

```
USE BDEJEMPLO01
GO
ALTER TABLE ALUMNOS
        ADD CONSTRAINT U_ALUMNOS_IDENTIFICACION
        UNIQUE (CODIGOALUMNO)

GO
```

EJEMPLO 071

Crear la restricción PRIMARY KEY sobre la columna IDCLIENTE de la tabla CLIENTE en la base de datos BDEJEMPLO02

```
USE BDEJEMPLO02
GO
ALTER TABLE CLIENTE
        ADD CONSTRAINT PK_CLIENTE
        PRIMARY KEY (CLIENTE)

GO
```

INTEGRIDAD DE DOMINIO

CREACION DEL VALOR PREDETERMINADO DEFAULT (DF_)

Cada columna en un registro debe contener un valor (aún si ese valor es un valor nulo). Hay situaciones en la cuales se necesita cargar una fila de datos en una tabla donde no se conocen los valores para todas las columnas (o esos valores no existen aún). Si la columna permite valores nulos, se puede cargar un valor nulo para esa fila. Dado que como vimos los valores nulos no son aconsejables, una mejor solución puede ser definir una valor por defecto para esa columna,. Por ejemplo, es común asignar un valor cero como el valor por defecto (DEFAULT) para columnas numéricas o N/A para columnas de caracteres cuando no se especifican valores.

La cláusula DEFAULT en el comando CREATE TABLE se considera una restricción aún cuando en realidad no fuerza a nada.

Cuando se carga una fila en una tabla con una definición de un valor por defecto para una columna, se le está indicando en forma implícita al SQL Server que cargue el valor por defecto en la columna en aquellos casos que no se indique valor para dicha columna.

Si una columna no permite valores nulos y no tiene una definición por defecto, se deberá explícita indicar un valor para esa columna o el SQL Server generará un mensaje de error indicando que la columna no permite valores nulos.

Se puede crear una definición de valores por defecto para una columna de dos maneras:

- Creando la definición del valor por defecto cuando se crea la tabla (como parte de la definición de la tabla)
- Agregando el valor por defecto a una tabla existente (cada columna permite solo un valor por defecto)

El siguiente ejemplo usa el comando CREATE TABLE para crear la tabla Registros. Ninguna de las tres columnas permite valores nulos; sin embargo, la columna Nombre provee la posibilidad de un nombre desconocido al agregar una definición por defecto a la definición de la columna. El comando CREATE TABLE usa la cláusula DEFAULT para definir el valor por defecto:

```
CREATE TABLE Registros
(
Emp_ID char(4) NOT NULL,
Nombre varchar(80) NOT NULL DEFAULT 'desconocido',
Apel varchar(30) NOT NULL,
)
```

Se puede modificar o eliminar una definición por defecto existente. Por ejemplo se puede modificar el valor insertado en una columna cuando no ha sido entrado ningún valor.

Cuando se utilice el Transact-SQL para modificar una definición por defecto, se debe primero borrar la cláusula DEFAULT existente y luego recrearla con la nueva definición.

El valor por defecto debe ser compatible con el tipo de dato definido para la columna. Por ejemplo para una columna con un tipo de dato entero (int) el valor por defecto deberá ser un número entero y no una cadena de caracteres.

Cuando una definición DEFAULT es agregada a una columna existente en una tabla, SQL Server (por defecto) aplica el valor por defecto solo a las filas nuevas que sean ingresadas de ahí en adelante. Las filas que tomaron el valor por defecto anterior no son afectadas. Cuando se agrega una nueva columna a una tabla existente, sin embargo, se puede especificar al SQL Server que inserte el valor por defecto (especificado en la definición de la nueva columna) en vez de un valor nulo en la nueva columna para las filas preexistentes de la tabla

CREATE DEFAULT (Transact-SQL)

Crea un objeto denominado valor predeterminado. Cuando se enlaza a un tipo de datos de columna o de alias, un valor predeterminado especifica un valor que debe insertarse en la columna a la que está enlazada el objeto (o en todas las columnas, en el caso de un tipo de datos de alias) si no se proporciona explícitamente un valor durante la inserción.

Sintaxis

```
CREATE DEFAULT [ schema_Nombres . ] default_Nombres
AS constant_expression [ ; ]
```

Argumentos

schema_Nombres

Es el nombre del esquema al que pertenece el valor predeterminado.

default_Nombres

Es el nombre del valor predeterminado. Los nombres predeterminados deben cumplir las reglas de los identificadores. Especificar el nombre del propietario del valor predeterminado es opcional.

constant_expression

Es una expresión que contiene sólo valores constantes (no puede contener el nombre de ninguna columna u otros objetos de base de datos). Se puede utilizar cualquier constante, función integrada o expresión matemática, excepto las que contienen tipos de datos de alias. No se pueden utilizar funciones definidas por el OPERADOR. Incluya las constantes de caracteres y fechas entre comillas simples ('); las constantes de moneda, de enteros y de coma flotante no necesitan comillas. Los datos binarios deben precederse de 0x y los datos de

moneda deben precederse de un signo de dólar ($). El valor predeterminado debe ser compatible con el tipo de datos de la columna.

El nombre de un valor predeterminado sólo se puede crear en la base de datos actual. En una base de datos, los nombres predeterminados deben ser únicos para cada esquema. Después de crear un valor predeterminado, utilice **sp_bindefault** para enlazarlo a una columna o a un tipo de datos de alias.

Si el valor predeterminado no es compatible con la columna a la que está enlazado, SQL Server genera un mensaje de error al intentar insertar el valor predeterminado. Por ejemplo, N/A no se puede utilizar como valor predeterminado para una columna **numeric**.

Si el valor predeterminado es demasiado largo para la columna a la que está enlazado, el valor se trunca.

Las instrucciones CREATE DEFAULT no se pueden combinar con otras instrucciones Transact-SQL en el mismo lote.

Antes de crear un nuevo valor predeterminado con el mismo nombre, es necesario quitar el anterior y quitar el enlace del valor predeterminado; para ello, ejecute **sp_unbindefault** antes de quitarlo.

Si una columna tiene un valor predeterminado y una regla asociados, el valor predeterminado no debe infringir la regla. No se insertará nunca un valor predeterminado que esté en conflicto con una regla y SQL Server genera un mensaje de error cada vez que se intente insertar el valor predeterminado.

Cuando se enlaza a una columna, un valor predeterminado se inserta cuando:

- No se ha insertado un valor explícitamente.
- Se utilizan las palabras clave DEFAULT VALUES o DEFAULT con INSERT para insertar valores predeterminados.

Si se especifica NOT NULL al crear una columna y no se crea un valor predeterminado para ésta, se generará un mensaje de error cada vez que el OPERADOR no cree una entrada en esa columna. En la tabla siguiente se ilustra la relación entre la existencia de un valor predeterminado y la definición de una columna como NULL o NOT NULL. Las entradas de la tabla muestran el resultado

VALOR PREDETERMINADO (DF_)

EJEMPLO 072

Consideraremos como ejemplo la tabla Alumnos, donde exista una columna numvacantes; el cual deseamos crear una columna que cuando se ingrese los registros de la tabla esta ingrese valores por defecto y sea el valor 20, crear la restricción DEFAULT

```
ALTER TABLE ALUMNOS
        ADD CONSTRAINT DF_ALUMNOS_NUMVACANTES
        DEFAULT 20 FOR NUMVACANTES

GO
```

EJEMPLO 079

Crear una restricción PRIMARY KEY sobre la columna IDCLIENTE de la tabla Venta en la Base de datos BDEJEMPLO02

```
USE BDEJEMPLO02
GO
IF EXISTS (Select * FROM Sysobject WHERE Nombres =¨PK_IDCLIENTE¨)
        ALTER TABLE VENTA
        DROP CONSTRAINT PK_IDCLIENTE
GO
        ALTER TABLE VENTA
        ADD CONSTRAINT PK_IDCLIENTE PRIMARY KEY (IDCLIENTE)
GO
```

SP_HELPCONSTRAINT

Devuelve la lista de todos los tipos de restricciones, el nombre definido por el OPERADOR o dado por el nombre que le dara el sistema y la expresión que define la restricción, siendo estas solo para las restricciones DEFAULT y CHECK

EJEMPLO 080

Verificar todas las restricciones del PRIMARY KEY en la tabla cliente

```
USE BDEJEMPLO02
GO
EXEC SP_helpConstraint Cliente
```

EJEMPLO 081

Crear la restricción de la tabla MOVIMIENTOS con la condición que no se duplique los códigos de los proveedores, pues para este caso emplearemos UNIQUE

```
USE BDEJEMPLO02
GO
IF EXISTS (Select * FROM Sysobject WHERE Nombres =´U_CODPROVEEDOR´)
        ALTER TABLE VENTA
        DROP CONSTRAINT U_CODPROVEEDOR
GO
        ALTER TABLE VENTA
        ADD CONSTRAINT U_CODPROVEEDOR UNIQUE  CODPROVEEDOR)
GO
```

PROPIEDADES IDENTITY

Cuando utilice la propiedad IDENTITY para definir una columna de identificadores, tenga en cuenta lo siguiente:

- Una tabla solo puede definir una columna definida con la propiedad IDENTITY y esa columna solo se puede definir con los tipos de datos: decimal, int, numeric, smallint o tinyint.
- La columna de identificadores no debe aceptar valores nulos ni contener una definición ni un objeto DEFAULT
- Es posible utilizar la función OBJECTPROPERTY para definir si una tabla tiene una columna identity y la función COLUMNPROPERTY para determinar el nombre de la columna IDENTITY.

NOTA

Cada restricción UNIQUE genera un índice y el numero de restricciones UNIQUE no puede hacer que el numero de índices de la tabla exceda de 249 índices no agrupados y 01 índice agrupado.

EJEMPLO 082

Vamos a imaginar tener una tabla llamada movimientos el cual queremos añadir e insertar un campo o columna llamada ITEM, a la tabla movimientos para esto vamos a crear una restricción IDENTITY sobre la columna ITEM de la tabla movimientos a crear

```
USE BDEJEMPLO02
GO
IF EXISTS (Select * FROM Sysobject WHERE Nombres =¨I_ITEM ¨)
        ALTER TABLE MOVIMIENTOS
        DROP CONSTRAINT I_ITEM

GO
        ALTER TABLE MOVIMIENTOS
        ADD ITEM INT IDENTITY (1,1)
         CONSTRAINTI_ITEM PRIMARY KEY (ITEM)
GO
```

EJEMPLO 083

Ejemplo practico para usted, En este ejemplo vamos a considerar que tenemos una tabla llamada TIPODOCUM donde su estructura es :
Iddocum INTEGER
Detalle VARCHAR (40)
Crear la restricción IDENTITY en la tabla TIPODOCUM considerando que el campo IDDOCUM sea auto numérico.

Soluciónelo……

NOTA

Cuando se describe este concepto es para definir una columna numérica entera en la que el valor de la columna es autogenerada a medida que se van incrementando las filas.

PROPIEDADES IDENTITY

EJEMPLO 084

Vamos a considerar que existe una tabla llamada laboratorio que tiene los siguientes campos: código y nombre; crear una restricción IDENTITY para que el campo código sea auto numérico de uno a uno.

```
USE BDEJEMPLO02
Create TABLE laboratorio (
Codigo          INT IDENTITY (1,1) PRIMARY KEY,
Nombre          varchar(20) NOT NULL
)
Go
- - vamos a considerar para insertar filas a la tabla:
INSERT INTO laboratorio (nombre) values (“TRIFARMA”);
INSERT INTO laboratorio (nombre) values (“MEDIFARMA”);
INSERT INTO laboratorio (nombre) values (“LATEX”):
INSERT INTO laboratorio (nombre) values (“QUIMICASUIZA”):
```

EJEMPLO 085

En una tabla de MOVIMIENTOS de una tienda deseamos insertar la fecha del sistema en un determinado campo cuyo nombre del campo o columna llamaremos fecharegistro, pues para esto emplearemos la función getdate(), le cual insertaremos de tipo fecha_hora.

```
USE BDEJEMPLO02
GO
INSERT INTO MOVIMIENTOS(código,razonsocial,fecha,importe,cantidad)
        VALUES(¨100A¨,´GRUPO SIABA¨,getdate(),8000,20)
GO
```

EJEMPLO 086

Tenemos una tabla llamada vivienda y una de sus columnas es de formato fecharesolucion DATETIME NOT NULL, insertar la fecha 27 de agosto del 2010 a la columna fecharesolucion.

```
USE BDEJEMPLO02
GO
        INSERT INTO vivienda(Nombres, fecharesolucion)
        VALUES(¨HARUMI PEREDA¨,27 AGO 2010¨)
GO
```

EJEMPLO 087

Tenemos el ejemplo del ejercicio anterior, ahora ingresaremos otro valor de fecha que es 28 de MARZO 2010, para esto deseamos insertar en cada columna

```
USE BDEJEMPLO02
GO
        INSERT INTO vivienda(Nombres, fecharesolucion)
        VALUES(¨MARILUISA PASCAL¨,¨03/28/2010¨)
GO
-- Observe que se ha ingresado el mes, luego el dia y posteriormente el año.
```

Tenemos el ejemplo del ejercicio anterior, ahora ingresaremos otro valor de fecha que es 08 ABRIL1996 , Pero en formato correcto día - mes y año, pues para esto emplearemos el formato SET DATEFORMAT al iniciar antes de insertar la fecha

```
SET DATE FORMAT dmy
USE BDEJEMPLO02
GO
        INSERT INTO vivienda(Nombres, fecharesolucion)
        VALUES(¨HARUMI PEREDA¨,¨08/04/1996¨)
GO
```

INTEGRIDAD DE DOMINIO

La integridad de dominio viene dada por la validez de las entradas para una columna determinada.

Restricciones CHECK
ALTER TABLE <Nombre de la tabla>
Add CONSTRAINT <Variable> CHECK <condición>

EJEMPLO 088

Vamos a considerar una tabla llamada alumno el cual vamos a crear la restricción CHECK sobre la columna código de la tabla ALUMNO
Para este ejemplo deseamos considerar que en los códigos de los alumnos solo puedan ingresar dígitos desde el 0 al 9

```
USE BDEJEMPLO02
GO
        ALTER TABLE ALUMNO
        ADD CONSTRAINT CHK_CODIGO CHECK (CODIGO NOT LIKE “%[0-9 ]%”)

GO
```

EJEMPLO 089

Crear una restricción CHECK para la columna PVENTA de la tabla MOVIMIENTOS, considerando que no puede ser el valor cero ni menor que cero.

```
USE BDEJEMPLO02
GO
        ALTER TABLE MOVIMIENTOS
        ADD CONSTRAINT CHK_PVENTA CHECK (PVENTA>0)

GO
```

EJEMPLOS PRACTICOS CON RESTRICCIONES

```
CREATE TABLE TIP_OPERADOR
(
idTipoOPERADOR int Identity(1,1),
descTipoOPERADOR varchar(20) NOT NULL,
CONSTRAINT PK_TIP_OPERADOR PRIMARY KEY(idTipoOPERADOR)
)
GO

CREATE TABLE OPERADOR
(
idOPERADOR char(8)NOT NULL,
idTipoOPERADOR int NOT NULL,
apelOPERADOR varchar(35)NOT NULL,
nomOPERADOR varchar(35) NOT NULL,
direccion varchar(50)NULL,
habilitado bit NOT NULL,
fechaExpCarnet smalldatetime NOT NULL,
fechaVencCarnet smalldatetime NOT NULL,
CONSTRAINT PK_OPERADOR_idOPERADOR PRIMARY KEY(idOPERADOR),
CONSTRAINT FK_OPERADOR_idTipoOPERADOR FOREIGN KEY
(idTipoOPERADOR) REFERENCES TIP_OPERADOR(idTipoOPERADOR)
)
GO

CREATE TABLE LIBRO_MATERIA
(
idMatBiblio varchar(20)NOT NULL,
tituloMatBiblio varchar(150)NOT NULL,
editorial varchar(50)NULL,
pais varchar(20)NULL,
año smalldatetime NULL,
nPag int NULL,
existencia int NOT NULL,
CONSTRAINT PK_LIBRO_MATERIA_idMatBiblio PRIMARY KEY(idMatBiblio)
)

CREATE TABLE PRESTAMO
(
codOper char(7)NOT NULL,
idMatBiblio varchar(20)NOT NULL,
idOPERADOR char(8)NOT NULL,
fechaP smalldatetime NOT NULL,
fechaD smalldatetime NOT NULL,
ndias int NOT NULL,
CONSTRAINT PK_PRESTAMO_codOper PRIMARY KEY(codOper),
CONSTRAINT FK_PRESTAMO_idMatBiblio FOREIGN KEY(idMatBiblio)
REFERENCES LIBRO_MATERIA(idMatBiblio)
)
GO

CREATE TABLE AUTOR
(
idAutor char(4)NOT NULL,
nomAutor varchar(50)NOT NULL,
CONSTRAINT PK_AUTOR_idAutor PRIMARY KEY(idAutor)
)
GO

CREATE TABLE MB_AUTOR
(
idMatBiblio varchar(20)NOT NULL,
idAutor char(4)NOT NULL,
CONSTRAINT PK_MB_AUTOR_idMatBiblio_idAutor
 PRIMARY KEY(idMatBiblio,idAutor),
CONSTRAINT FK_MB_AUTOR_idMatBiblio
 FOREIGN KEY(idMatBiblio) REFERENCES LIBRO_MATERIA(idMatBiblio),
CONSTRAINT FK_MB_AUTOR_idAutor
 FOREIGN KEY(idAutor) REFERENCES AUTOR(idAutor)
)
GO
```

Modificar una restricción CHECK

Recordemos que si queremos modificar una restricción CHECK en una tabla ya existente, mediante T-SQL antes debe eliminar la restricción y a continuación volver a generarla.

EJEMPLO 090

Eliminar una restricción CHECK de la tabla MATRICULA, llamada CHECK_ORDEN

```
USE BDEJEMPLO02
GO
IF EXISTS (Select Nombres FROM SYSOBJECT where NOMBRES=”CHECK_ORDEN”)
        ALTER TABLE MATRICULA
        DROP CONSTRAINT CHECK_ORDEN

GO
```

INTEGRIDAD REFERENCIAL

Protege las relaciones definidas entre tablas cuando se crean o se eliminan registros en SQL, recordemos que la integridad referencial garantiza que los valores clave son coherentes en las distintas tablas.
Cuando se fuerza la integridad referencial SQL impide:

- Agregar registros a una tabla relacionada.
- Cambiar valores en una tabla primaria o principal
- Eliminar registros de una tabla principal.

RETSRICCIONES FOREIGN KEY

Se usa para hacer referencia a otra tabla.
Las restricciones de este tipo se pueden:

- Crear cuando se crea la tabla en el momento de la definición
- Agregar a una tabla ya existente siempre que la restricción FK este vinculada a una restricción Primary Key o UNIQUE de otra o de la misma tabla.
- Una tabla puede contener varias restricciones FK
- Modificar o eliminar si ya existen FK mediante T-SQL o SLDMO antes debe eliminar la restricción existente.
- Cuando se procede a eliminar una fila de una tabla a la que apuntan claves foráneas la eliminación falla, debido a que las filas que contienen la clave foránea no pueden quedar huérfanas.

La integridad referencial en cascada permite controlar las acciones cuando se intenta actualizar o eliminar una clave primaria.

Con la restricción "foreign key" se define un campo (o varios) cuyos valores coinciden con la clave primaria de la misma tabla o de otra, es decir, se define una referencia a un campo con una restricción "primary key" o "unique" de la misma tabla o de otra.

EJEMPLO 091

Crearemos un ejemplo de integridad referencial, para ello vamos a crear diferentes restricciones

/*Vamos a crear una restricción llamada con_01, cuya tabla se llamara BODEGA*/

```
/*Si existe lo eliminamos*/
IF EXISTS (Select Nombres FROM SYSOBJECT where NOMBRES=”CON_01)
        ALTER TABLE BODEGA
        DROP CONSTRAINT CON_01
GO
/*Ahora lo crearemos e insertamos una columna ID_DET */
        ALTER TABLE BODEGA
        ADD ID_DET INT IDENTITY (1,1)
        CONSTRAINT CON_01 PRIMARY KEY (ID_DET)
GO
```

/*Vamos a crear una restricción llamada con_02, cuya tabla se llamara FARMACIA*/

```
/*Si existe lo eliminamos*/
IF EXISTS (Select Nombres FROM SYSOBJECT where NOMBRES=”CON_02)
        ALTER TABLE FARMACIA
        DROP CONSTRAINT CON_02
GO
/*Ahora lo crearemos e insertamos una columna FECHAINGRESO */
        ALTER TABLE FARMACIA
        ADD FECHAINGRESO DATETIME
        CONSTRAINT CON_02  DEFAULT GETDATE()
GO
```

/*Vamos a crear una restricción llamada con_03, cuya tabla se llamara BIENES y esta relacionada con la tabla FARMACIA mediante el campo o la columna CODBIENES*/

```
/*Si existe lo eliminamos*/
IF EXISTS (Select Nombres FROM SYSOBJECT where NOMBRES=”CON_03)
        ALTER TABLE BIENES
        DROP CONSTRAINT CON_03
GO
/*Ahora lo crearemos  */
        ALTER TABLE BIENES
        ADD CONSTRAINT CON_03 FOREIGN KEY (CODIGO)
        REFRENCES FARMACIA (CODBIENES)
GO
```

ADMINISTRADOR CORPORATIVO

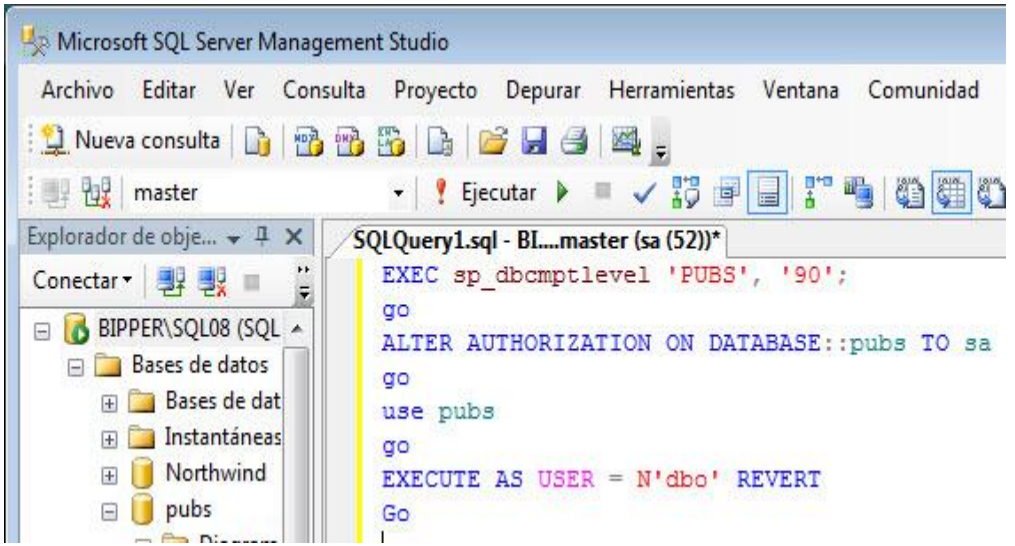
Usando el Administrador Corporativo, podemos crear una Base de datos, asimismo crear un diagrama de base de datos >nuevo diagrama de base de datos…> en la ventana activa agregaremos todas las tablas creadas disponibles; luego finalizamos y a continuación nos mostrará en una ventana el diagrama efectuado.

Cuando usted esté creando el diagrama mostrara el siguiente error el cual usted procederá a corregir

Para solucionar este problema ejecutamos las siguientes sentencias sql, seleccionado la base de datos Master.

EXEC sp_dbcmptlevel ‘PUBS’, ‘90′;
go
ALTER AUTHORIZATION ON DATABASE::pubs TO sa
go
use pubs
go
EXECUTE AS USER = N’dbo’ REVERT
Go

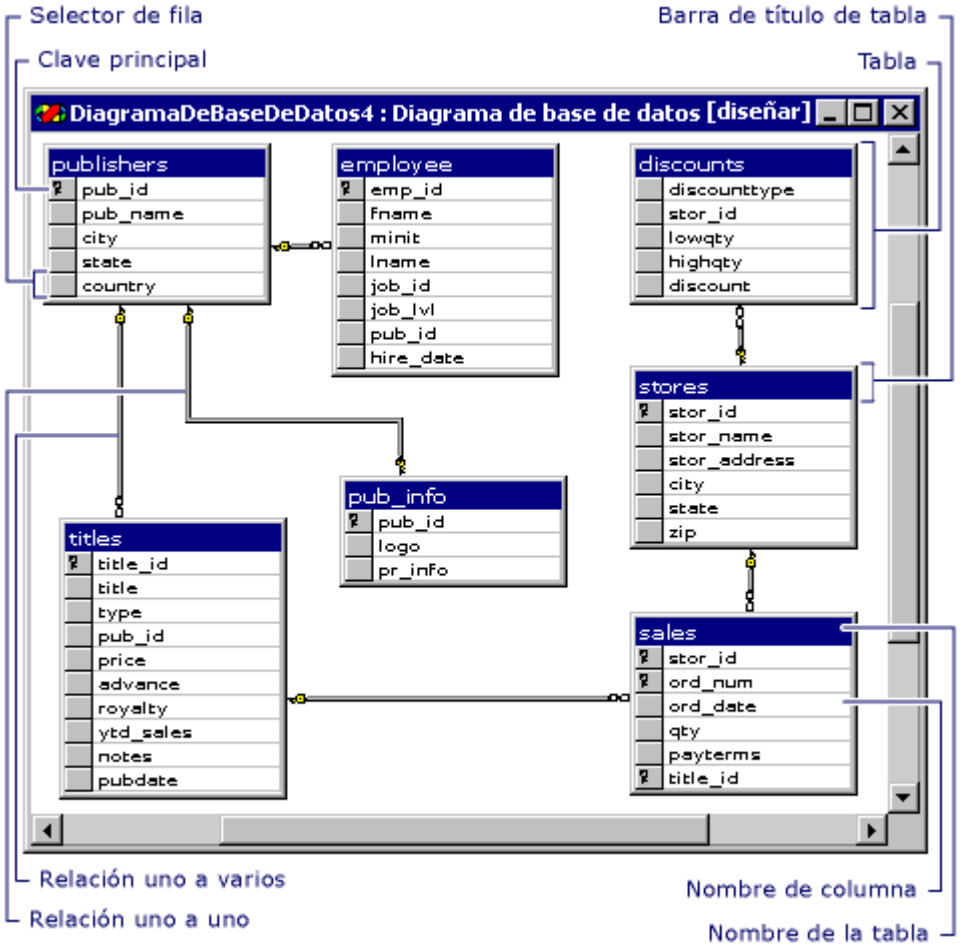
Donde pubs es la base de datos donde queremos hacer los diagramas



DISEÑAR DIAGRAMAS DE BASE DE DATOS

[Este tema pertenece a la Libroación de la versión preliminar y está sujeto a cambios en versiones futuras. Los temas en blanco se incluyen como marcadores de posición. NOTA: Con el fin de proporcionarle contenido adicional en distintos idiomas, Microsoft ofrece Libroación localizada mediante métodos de traducción alternativos. Para esta versión preliminar, parte del contenido de esta Libroación se ha traducido mediante el uso de estos métodos. Microsoft es consciente de que los Libroos traducidos de esta forma no son siempre perfectos, por lo que es posible que este artículo contenga errores de vocabulario, sintaxis o gramática. La versión final de este contenido se traducirá por los métodos tradicionales y la calidad será igual que la de las versiones anteriores.]

El Diseñador de bases de datos es una herramienta visual que permite diseñar y ver una base de datos a la que está conectado. Cuando se diseña una base de datos, se puede utilizar el Diseñador de bases de datos para crear, editar o eliminar tablas, columnas, claves, índices, relaciones y restricciones. Para ver una base de datos, puede crear uno o varios diagramas que muestren algunas o todas las tablas, columnas, claves y relaciones de la base de datos.



Para cualquier base de datos, puede crear tantos diagramas de base de datos como desee; cada tabla de base de datos puede aparecer en un número cualquiera de diagramas. De este modo, puede crear diagramas distintos para ver partes diferentes de la base de datos o resaltar diversos aspectos del diseño. Por ejemplo, puede crear un diagrama grande que muestre todas las tablas con sus columnas y también puede crear un diagrama más pequeño que muestre todas las tablas sin mostrar las columnas.

Cada diagrama de base de datos que crea se almacena en la base de datos asociada.

Tablas y columnas de un diagrama de base de datos

En un diagrama de base de datos, cada tabla puede aparecer con tres características distintas: una barra de título, un selector de fila y un conjunto de columnas de propiedades.

Barra de título La barra de título muestra el nombre de la tabla.

Si ha modificado una tabla y no la ha guardado todavía, aparecerá un asterisco (*) al final del nombre de la tabla que indica que hay cambios no guardados. Para obtener más información sobre cómo guardar tablas y diagramas modificados, vea Trabajar con diagramas de base de datos (Visual Database Tools).

Selector de fila Puede hacer clic en el selector de fila para seleccionar una columna de base de datos de la tabla. El selector de fila muestra un símbolo de clave si la columna se encuentra en la clave principal de la tabla. Para obtener información sobre las claves principales, vea Trabajar con claves (Visual Database Tools).

Columnas de propiedades El conjunto de columnas de propiedades sólo es visible en determinadas vistas de la tabla. Dispone de cinco vistas diferentes para ver una tabla, que le ayudan a controlar el tamaño y el diseño del diagrama.

Para obtener más información sobre las vistas de tabla, vea Cómo personalizar la cantidad de información mostrada en los diagramas (Visual Database Tools).

Relaciones de un diagrama de base de datos

En un diagrama de base de datos, cada relación puede aparecer con tres características distintas: los extremos, un estilo de línea y las tablas relacionadas.

Extremos Los extremos de la línea indican si la relación es de uno a uno o de uno a varios. Si una relación tiene una clave en un extremo y un símbolo en forma de ocho en el otro extremo, se trata de una relación de uno a varios. Si una relación tiene una clave en cada extremo, se trata de una relación de uno a uno.

Estilo de línea La línea en sí misma (no sus extremos) indica si el sistema de administración de bases de datos (DBMS) exige integridad referencial para la relación cuando se agregan datos nuevos a la tabla de clave externa. Si la línea aparece sólida, el DBMS exige integridad referencial para la relación cuando se agregan o modifican filas en la tabla de clave externa. Si la línea aparece punteada, el DBMS no exige integridad referencial para la relación cuando se agregan o modifican filas en la tabla de clave externa.

Tablas relacionadas Una relación de clave externa entre dos tablas se indica mediante una línea de relación entre ellas. En el caso de una relación de uno a varios, la tabla de clave externa es la tabla situada cerca del símbolo en forma de ocho de la línea. Si ambos extremos de la línea están conectados a la misma tabla, la relación es reflexiva. Para obtener más información, vea Cómo dibujar relaciones reflexivas (Visual Database Tools).

NOTA

Recordemos que cuando una determinada tabla tiene restricciones referenciales foráneas no se puede eliminar para ello primero debemos eliminar el índice de restricción.

INTEGRIDAD REFERENCIAL EN CASCADA

Se dice integridad referencial en cascada cuando un determinado registro de una tabla esta relacionado con otra tabla.

EJEMPLO 092

En este ejemplo tenemos dos tablas: una de ellas es Movimiento y la otra cabecera donde se almacenan los movimientos generados de la otra tabla en mención

Tabla Cabecera

Numdoc	**INTEGER**	**8**
Tipodocum	**VARCHAR**	**30**
FECHA	**DATETIME**	
TOTAL	**DECIMAL**	**10,2**

Tabla Movimientos

NUMDOC	INTEGER	8
FECHA	DATETIME	
TIPODOCUM	VARCHAR	30
RUC	VARCHAR	15
DETALLE	VARCHAR	60
CANT	DECIMAL	10,2
PUNIT	DECIMAL	10,2
TOTAL	DECIMAL	10,2

Y cuando deseamos eliminar un registro de la tabla movimiento entonces también debe eliminarse de la otra tabla llamada cabecera.

/*Ahora vamos a definir la clave foránea de la tabla movimientos, eliminando la clave foránea si existe, para volver a crear otra, estableciendo la eliminación en cascada*/

```
ALTER TABLE MOVIMIENTOS
        DROP CONSTRAINT FK_MOVIMIENTOS_CABECERA
GO
ALTER TABLE MOVIMIENTOS
        ADD CONSTRAINT FK_MOVIMIENTOS_CABECERA
         FOREIGN KEY (NUMDOC,TIPODOCUM)
        REFERENCES CABECERA
        ON DELETE CASCADE
GO
```

EJEMPLO 093

En el siguiente ejemplo muestra la creación de una restricción FOREIGN KEY sobre la columna IDCATEGORIA de la tabla Articulo en la Base de datos BDEJEMPLO02

```
/* Llamamos a la tabla Artículos */
IF EXISTS (Select Nombres FROM SYSOBJECTS WHERE Nombres =´FK_CATEGORIA¨)
ALTER TABLE ARTICULOS
DROP CONSTRAINT FK_CATEGORIA
GO
ALTER TABLE ARTICULOS
ADD CONSTRAINT FK_CATEGORIA FOREIGN KEY (Idcategoria)
REFERENCES Categoria (Idcategoria)
GO
```

EJEMPLO 094

Ejecutar el procedimiento almacenado SP_HELPCONSTRAINT para verificar la existencia de las restricciones en la tabla usuario.

```
EXEC SP_HELPCONSTARINT USUARIO
```

DEFINICION DEFAULT

Cuando cargue una fila en una tabla con una definición DEFAULT para una columna le esta indicando implícitamente a SQL que cargue el valor predeterminado.
Recuerda que para modificar una definición DEFAULT mediante TSQL o SQL DMO antes debe eliminar la definición DEFAULT existente y a continuación volver a crearla

EJEMPLO 095

Mostrar la creación de una restricción DEFAULT sobre la columna FECHAINGRESO de una tabla llamada movimiento, asimismo cargar la restricción en la fecha del sistema sobre la columna indicada, el cual se va crear dicha columna fechaingreso.

```
IF EXISTS (Select Nombres FRON SYSOBJECT WHERE Nombres =¨DF_FECHA¨)
        ALTER TABLE MOVIMIENTO
        DROP CONSTRAINT DF_FECHA
GO
        ALTER TABLE MOVIMIENTO
        ADD FECHAINGRESO DATETIME
        CONSTRAINT DF_FECHADEFAULT GETDATE()
```

EJEMPLO 096

En este caso ejemplo vamos a insertar una fila a algunos campos de una tabla llamada PERSONAL, para este caso ejemplo insertaremos los campos que el usuario requiera.

```
USE PADRON
GO
        INSERT INTO PERSONAL (codigo, nombre, edad, abono)
        VALUES (¨100AA¨,´CESAR PEREDA¨,40,10,000)
GO
```

NOTA

TRUNCATE TABLE es una sentencia el cual se emplea para eliminar todas las filas de una tabla.

CAPITULO V

MEMORIA AYUDA

Cerrar el archivo de log de SQL Server y el log errores del SQL Server Agent

Debemos cerrar el archivo log de SQL Server y el de errores de SQL Server Agent, ya que los archivos de log se reinician cada vez que el servicio se reinicia, es decir, se cierra el fichero de log de **SQL Server** cuando el servicio se reinicia y se abre un nuevo fichero. Igual pasa con el fichero de log de errores del **SQL server Agent**. Pero en ocasiones estos ficheros crecen mucho y son inmanejables y es necesario cerrarlo a mano y abrir uno nuevo.

Esto se puede hacer con dos instrucciones en **transact sql**.

Para cerrar el fichero de log de errores de SQL Server y abrir uno nuevo

EXEC master.sys.sp_cycle_errorlog;

Para cerrar el fichero de log de errores de SQL Server Agent y abrir uno nuevo

EXEC msdb.dbo.sp_cycle_agent_errorlog;

T-SQL

Los índices aceleran la recuperación de los datos, el cual es más fácil localizar un determinado registro.

Reemplazar un texto en SQL Server con TSQL

Supongamos que tenemos un campo con un texto y lo recuperamos con una **consulta SQL**. Supongamos también que queremos cambiar parte del contenido de ese texto y reemplazarlo por otro, en este caso realizaríamos un **replace por código (programación)**, pero y si esto no nos vale, y si necesitamos que sea por **consulta** todo. Para estos casos existe una **funcion REPLACE que se ejecuta en SQL** y es similar a la de la mayoría de lenguajes de programación.

La llamada a la **función de reemplazo (REPLACE)** se usara dentro de cualquier **código SQL o TSQL en SQL Server**. La **función REPLACE** tiene 3 parámetros, el texto original completo, el texto que se quiere buscar y por último el texto por el que se quiere sustituir como se hace en muchos lengaujes de programación.

La sintaxis sería:

```
REPLACE(TEXTO, BUSCADA, SUSTITUIDA)
```

Un ejemplo de como se incrustaría el **reemplazo en una consulta SQL** sería:

```
SELECT REPLACE(campo, 'a', 'b') FROM tabla
```

Aqui nos encontramos con el primer de los dos grandes problemas de **reemplazar texto** con esta función. El primer problema son los **campos de tipo TEXT**, que al parecer, para **SQL Server** son objetos diferentes que los **campos de tipo CHAR o VARCHAR** y para poder reemplazar un **campo de tipo TEXT** necesitamos hacer un pequeño HACK que nos **convierta de tipo TEXT a tipo STRING** como si de un **VARCHAR** se tratase.

El Hack en cuestion se trata de hacer un **SUBSTRING** desde la primera posición de la cadena hasta la última utilizando la longitud de la misma obtenida con la **funcion DATALENGTH**. De esta forma, con el **SUBSTRING convertimos el tipo TEXT en un tipo STRING o VARCHAR** y asi podemos reemplazar.

Un ejemplo seria:

```
SELECT REPLACE(
  SUBSTRING(campo, 1, DATALENGTH(campo)),
  'a', 'b') FROM tabla
```

Y es aquí cuando nos encontramos con el segundo y mayor de los problemas y es la longitud de la cadena. Cuando trabajamos con una cadena de texto realmente larga, como podría ser una página web, el **SUBSTRING** o el **REPLACE** no devuelve la cadena completa, por lo que se nos cortara la respuesta y obtendremos solo parte del texto original, eso si, ya **reemplazado**

Depurar Consultas en TSQL

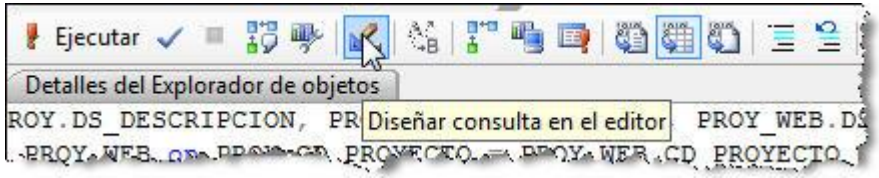
Son muchas las veces que un programador necesita depurar consultas SQL (Transact SQL, T-SQL) desde Microsoft Visual Studio: mediante la inspección rápida de variables, obtenemos el string infinitamente largo que representa nuestra consulta, que habíamos tabulado con mimo en nuestro código, y que ahora aparece en una única línea. Si queremos resolver un fallo en esta consulta, o intentar optimizarla, tenemos que pegarla en el Management Studio de Microsoft SQL Server y volver a perder nuestro tiempo en tabularla para que sea más fácil entenderla.
Con este truco evitaremos malgastar nuestro precioso tiempo: Management Studio de Microsoft SQL Server formateará esta consulta de nuevo por nosotros.

Al capturar la consulta desde la inspección rápida de variables de Microsoft Visual Studio para depurarla, podríamos obtener un string como éste (no es muy largo en este caso, pero nos sirve como ejemplo):

```
select PROY.DS_TITULO, PROY.DS_DESCRIPCION, PROY_WEB.DS_PERMALINK,
PROY_WEB.DS_TITLE, PROY.IT_CASO_EXITO from SWEB_PROYECTO PROY
inner join SWEB_PROYECTO_PAGINA_WEB PROY_WEB on PROY.CD_PROYECTO
 = PROY_WEB.CD_PROYECTO inner join SWEB_PROYECTO_LINEA_NEGOCIO
 SRV on PROY.CD_PROYECTO = SRV.CD_PROYECTO and SRV.CD_UNIDAD  is
not null  and SRV.CD_LINEA_NEGOCIO  is not null  inner join
SWEB_PROYECTO_LINEA_NEGOCIO PRD on PROY.CD_PROYECTO =
PRD.CD_PROYECTO and PRD.CD_UNIDAD  is not null  and
PRD.CD_LINEA_NEGOCIO  is not null  inner join
SWEB_SECTOR_PROYECTO SEC on PROY.CD_PROYECTO = SEC.CD_PROYECTO
and SEC.CD_SECTOR  is not null  inner join SWEB_TECNOLOGIA_PROYECTO
 TEC on PROY.CD_PROYECTO = TEC.CD_PROYECTO and TEC.CD_TECNOLOGIA  is
 not null  where PROY.IT_VISIBLE_WEB = 'S' order by
PROY.NM_PUNTUACION desc, PROY.FC_FIN desc
```

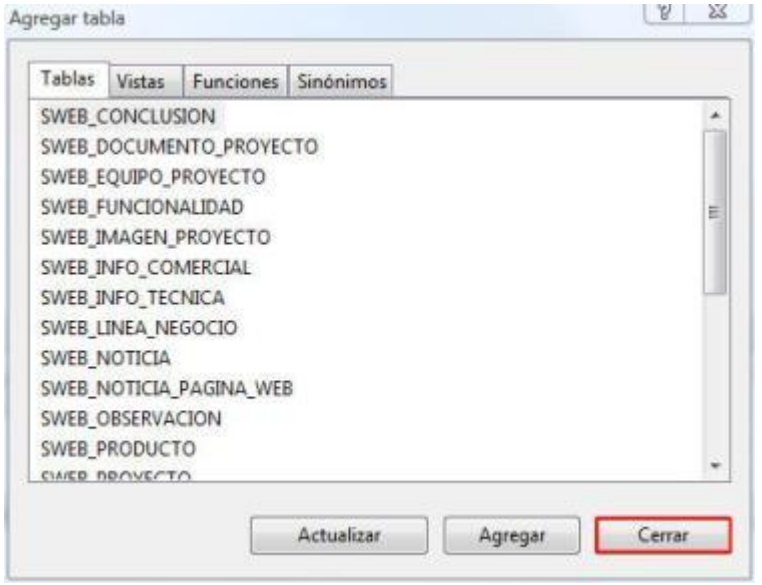
Consulta T-SQL original en una única línea

Ahora, ejecutamos el editor de consultas de Microsoft SQL Server Mangement Studio pinchando en el icono que aparece en la siguiente imagen:



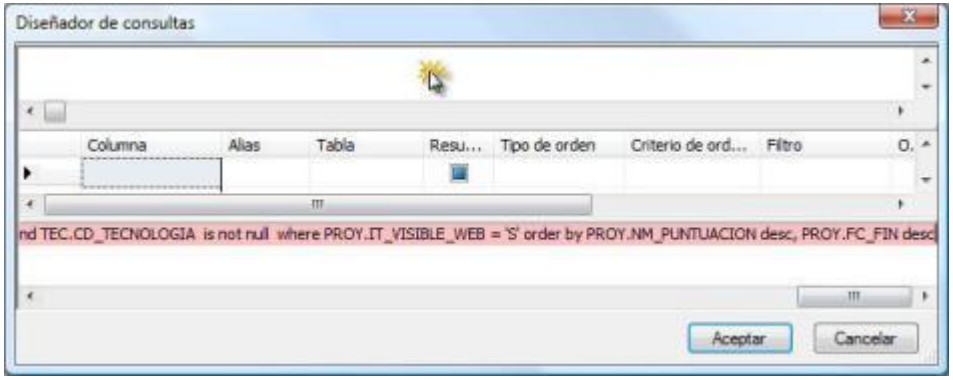
Microsoft SQL Server Mangement Studio - Diseñar consulta

Aparecerá esta ventana que ignoraremos en este caso.



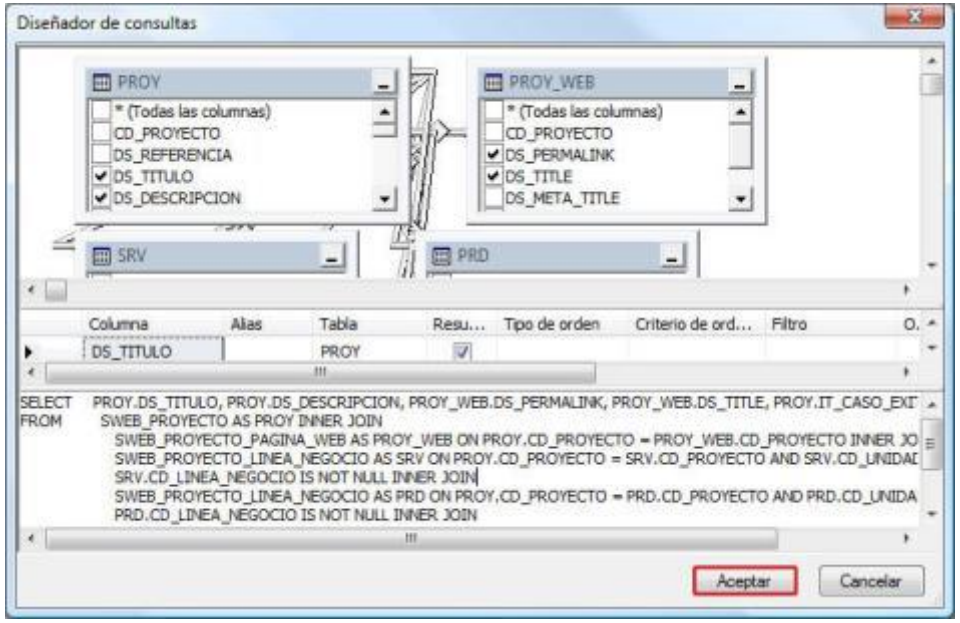
Microsoft SQL Server Mangement Studio - Agregar tabla

Pegamos la consulta que habiamos obtenido en la inspección rápida en el recuadro inferior de la pantalla (marcado en rojo en la siguiente imagen) y después quitamos el foco de ese recuadro haciendo clic en el superior, por ejemplo.



Pegamos la consulta T-SQL

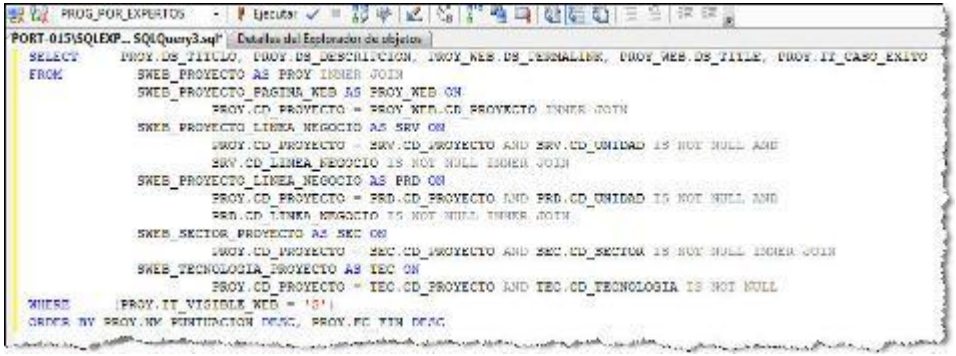
En este momento aparecerá en el recuadro superior una vista de diseño de todas las tablas que intervienen en nuestra consulta SQL, y la relación que existe entre ellas, y en la parte inferior podremos ver nuestra consulta SQL formateada automáticamente por Microsoft SQL Server Mangement Studio.



Consulta formateada

Ya sólo quedaría pulsar en “Aceptar” de la pantalla anterior para que la consulta nos sea devuelta y podamos ejecutarla.

Consulta formateada y lista para ejecutar



Anotaciones a tener en cuenta en T SQL

Para programar en **Transact SQL** es necesario conocer sus fundamentos.

Como introducción vamos a ver algunos elementos y conceptos básicos del lenguaje.

Transact SQL no es CASE-SENSITIVE, es decir, no diferencia mayúsculas de minúsculas como otros lenguajes de programación como C o Java.

Un comentario es una aclaración que el programador incluye en el código. Son soportados 2 estilos de comentarios, el de línea simple y de multilínea, para lo cual son Registros ciertos caracteres especiales como son:

-- Para un comentario de línea simple

/* ... */ Para un comentario de varias líneas

Un literal es un valor fijo de tipo numérico, caracter, cadena o lógico no representado por un identificador (es un valor explícito).

Una variable es un valor identificado por un nombre (identificador) sobre el que podemos realizar modificaciones. En **Transact SQL** los identificadores de variables deben comenzar por el caracter @, es decir, el nombre de una variable debe comenzar por @.Para declarar variables en **Transact SQL** debemos utilizar la palabra clave **declare**, seguido del identificador y tipo de datos de la variable.

Veamos algunos ejemplos:

```
/*  Este es un comentario con varias líneas.
Conjunto de Lineas.*/

declare @nombre varchar(50)
-- declare declara una variable
 -- @nombre es el identificador de la
-- variable de tipo varchar
set @nombre = 'GUNE'
 -- El signo = es un operador
print @Nombre -- Imprime por pantalla el valor de @nombre.
```

Scripts y lotes.

Un script de **Transact SQL** es un conjunto de sentencias de **Transact SQL** en formato de texto plano que se ejecutan en un servidor de **SQL Server**.

Un script está compuesto por uno o varios lotes. Un lote delimita el alcance de las variables y sentencias del script. Dentro de un mismo script se diferencian los diferentes lotes a través de la instrucción **GO.**

```
-- Este es el primer lote del script
SELECT * FROM COMENTARIOS
GO -- GO es el separador de lotes

-- Este es el segundo lote del script
SELECT getdate()  -- getdate() es una función integrada que devuelve
 -- la fecha
```

En ocasiones es necesario separar las sentencias en varios lotes, porque **Transact SQL** no permite la ejecución de ciertos comandos en el mismo lote, si bien normalmente también se utilizan los lotes para realizar separaciones lógicas dentro del script.

Dentro de los Elementos de TSQL tenemos:

- DML (Data Manipulation Language) se usa para consultas, y modificar datos(SELECT, INSERT,UPDATE,DELETE)
- DDL (Data Definition Language) se usa para crear objetos (CREATE , ALTER, DROP)
- DCL (Data control language) se usa para determinar , modificar datos(GRANT, DENY, REMOVE)

- Elementos adicionales como variables, operadores, funciones, sentencias.

Tipos de Sentencias de T-SQL.

- Sentencias de Lenguaje de Definición de Datos (LDD).
- Sentencias de Lenguaje de Control de Datos (LCD).
- Sentencias de Lenguaje de Manipulación de los datos (LMD).

Lenguaje de Definición de Datos (LDD).

- Las sentencias de Definición de datos nos ayudan a definir todos los objetos de la base de
- datos. Claro para poder hacer uso de estas sentencias se deben de tener los permisos
- adecuados.

Algunas Sentencia LDD son:

- CREATE nombre_objeto
- ALTER nombre_objeto
- DROP nombre_objeto

Lenguaje de Control de Datos (LCD)

- Estas sentencias son usadas para modificar los permisos asociados a un usuario o rol de
- la base de datos. Al igual que los LDD para poder ejecutar alguna de estas sentencias se
- deben tener los permisos adecuados.
- Sentencias LCD:
- GRANT
- DENY
- REVOKE

Lenguaje de Manipulación de Datos (LMD)

- Las sentencias LMD trabajan con los datos en la base de datos. Estas sentencias se
- utilizan para insertar, modificar, eliminar y recuperar los datos.
- Sentencias LMD:
- INSERT
- UPDATE
- DELETE

TIPOS DE DATOS DE SQL SERVER

Los tipos de datos definen los valores de los datos permitidos para cada columna en las tablas de la base de datos. SQL Server proporciona un número de tipos de datos.
Categorías de los Tipos de Datos:
Algunos tipos de datos de SQL Server 2008 se organizan en las siguientes categorías:

Instrucción CREATE

Vamos a examinar la estructura completa de la sentencia CREATE empezando con la más general. Descubrirá que las instrucciones CREATE empiezan de la misma forma y después dan paso a sus especificaciones. La primera parte de CREATE será siempre igual

EJEMPLO 097

Haciendo uso de la declaración DML, detallaremos los siguientes ejemplos:

```
USE MASTER
GO
SELECT  CODIGO, NOMBRES, DIRECCION FROM PERSONAL
GO
```

EJEMPLO 098

Haciendo uso de la declaración DDL, detallaremos los siguientes ejemplos:

```
USE MASTER
GO
CREATE TABLE PERSONAL(
Nombres char (10) PRIMARY KEY,
Telefono varhar(20) NOT NULL,
Edad int NOT NULL)
GO
```

EJEMPLO 099

Haciendo uso de la declaración DCL, detallaremos los siguientes ejemplos:

```
USE MASTER
GO
GRANT SELECT ON PERSONAL TO PUBLIC
```

Las variables locales se utilizan para almacenar valores (de cualquier tipo) con en un lote. que son locales, ya que sólo se puede hacer referencia dentro del mismo procedimiento en el que se declaran. variables locales deben ser definidos usando la instrucción declare y cada variable contiene su nombre y el tipo de datos correspondiente. Variable se hace referencia en un lote con el prefijo @ (arroba). La asignación de un valor a una variable local se realiza mediante la instrucción SELECT y la instrucción SET.

Los nombres de variable deben comenzar con una arroba (@) signo. Las variables locales se deben seguir reglas de los identificadores

Uso de las variables

```
use master
go
DECLARE @Test1 int,  @Test2 int
SET @Test1 = 3000
SET @Test2 =2000
if(@Test1 <@Test2 )
print 'test1 is greater than test2'
--select @Test1 as test1
else begin
print 'test2 is greater than test1'
--select @Test2  as test2
end
go
```

Las Variables globales son predefinidas y mantenidas por SQL SERVER **El usuario** no puede asignar o cambiar directamente los valores de las variables globales.

```
use master
go
SELECT  Codigo, nombres, apellidos  FROM PADRON
WHERE LEFT(CODIGO,4)=”1000”
SELECT @@rowcount

GO
```

Comentarios

(--) se puede emplear doble guion para los comentarios, se pueden usar en la misma línea que el código que se va a ejecutar o en una línea aparte.

(/*..*/) se puede emplear para los comentarios que contienen varios líneas es decir un comentario.

EJEMPLO 100

```
-- En este caso lo emplearemos para un comentario lineal

/* Ahora Observemos los resultados este es un gran comentario que contiene varias líneas para esto es las diagonales y asteriscos */
```

Tipos de Funciones en TSQL

Tipos de datos y funciones de fecha y hora (Transact-SQL)
Todas estas funciones son propias del SQL

Las secciones siguientes de este tema proporcionan información de todos los tipos de datos y funciones de fecha y la hora de Transact-SQL. Para obtener información y ejemplos comunes a los tipos de datos y funciones de fecha y hora, vea Usar datos de fecha y hora.

Tipos de datos de fecha y hora

En la tabla siguiente se enumeran los tipos de datos de fecha y hora de Transact-SQL.

Tipo de datos	Formato	Intervalo	Precisión
time	hh:mm:ss[.nnnnnnn]	De 00:00:00.0000000 a 23:59:59.9999999	100 nanosegundos
date	AAAA-MM-DD	De 0001-01-01 a 9999-12-31	1 día
smalldatetime	AAAA-MM-DD hh:mm:ss	De 1900-01-01 a 2079-06-06	1 minuto
datetime	AAAA-MM-DD hh:mm:ss[.nnn]	De 1753-01-01 a 9999-12-31	0,00333 segundos
datetime2	AAAA-MM-DD hh:mm:ss[.nnnnnnn]	De 0001-01-01 00:00:00.0000000 a 9999-12-31 23:59:59.9999999	100 nanosegundos
datetimeoffset	AAAA-MM-DD hh:mm:ss[.nnnnnnn]	De 0001-01-01 00:00:00.0000000 a 9999-12-31	100 nanosegundos
	[+|-]hh:mm	23:59:59.9999999 (en UTC)	

Funciones de fecha y hora

En las tablas siguientes se enumeran las funciones de fecha y hora de Transact-SQL. Para obtener más información acerca del determinismo, vea Funciones deterministas y no deterministas.

Funciones que obtienen valores de fecha y hora del sistema

Todos los valores de fecha y hora del sistema se derivan del sistema operativo del equipo en el que se ejecuta la instancia de SQL Server.

Funciones de fecha y hora del sistema de precisión elevada

SQL Server 2008 obtiene los valores de fecha y hora utilizando la API de Windows **GetSystemTimeAsFileTime()**. La exactitud depende del hardware del equipo y de la versión de Windows en las que la instancia de SQL Server se esté ejecutando. La precisión de esta API se fija en 100 nanosegundos. La precisión se puede determinar mediante la API de Windows **GetSystemTimeAdjustment()**.

Función	Sintaxis	Valor devuelto
SYSDATETIME	SYSDATETIME ()	Devuelve un valor **datetime2(7)** que contiene la fecha y hora del equipo en el que la instancia de SQL Server se está ejecutando. El ajuste de zona horaria no está incluido.
SYSDATETIMEOFFSET	SYSDATETIMEOFFSET	Devuelve un valor **datetimeoffset(7)** que contiene la fecha y hora del equipo en el

	( )	que la instancia de SQL Server se está ejecutando. El ajuste de zona horaria está incluido.
SYSUTCDATETIME	SYSUTCDATETIME ( )	Devuelve un valor **datetime2(7)** que contiene la fecha y hora del equipo en el que la instancia de SQL Server se está ejecutando. La fecha y hora se devuelven como hora universal coordinada (UTC).

Funciones de fecha y hora del sistema de precisión baja

Función	Sintaxis	Valor devuelto
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	Devuelve un valor **datetime2(7)** que contiene la fecha y hora del equipo en el que la instancia de SQL Server se está ejecutando. El ajuste de zona horaria no está incluido.
GETDATE	GETDATE ( )	Devuelve un valor **datetime2(7)** que contiene la fecha y hora del equipo en el que la instancia de SQL Server se está ejecutando. El ajuste de zona horaria no está incluido.
GETUTCDATE	GETUTCDATE ( )	Devuelve un valor **datetime2(7)** que contiene la fecha y hora del equipo en el que la instancia de SQL Server se está ejecutando. La fecha y hora se devuelven como una hora universal
		coordinada (UTC).

Funciones que obtienen partes de la fecha y hora

Función	Sintaxis	Valor devuelto
DATENOMBRES	DATENOMBRES (datepart , date)	Devuelve una cadena de caracteres que representa el datepart especificado de la fecha especificada.
DATEPART	DATEPART (datepart , date)	Devuelve un entero que representa el datepart especificado del date especificado.
DAY	DAY (date)	Devuelve un entero que representa la parte del día de date especificado.
MONTH	MONTH (date)	Devuelve un entero que representa la parte del mes de un date especificado.
YEAR	YEAR (date)	Devuelve un entero que representa la parte del año de un date especificado.

Funciones que obtienen diferencias de fecha y hora

Función	Sintaxis	Valor devuelto

DATEDIFF	DATEDIFF (datepart , startdate , enddate)	Devuelve el número de límites datepart de fecha y hora entre dos fechas especificadas.

Funciones que modifican valores de fecha y hora

Función	Sintaxis	Valor devuelto
DATEADD	DATEADD (datepart , number , date)	Devuelve un nuevo valor **datetime** agregando un intervalo al datepart especificado del dateespecificado.
SWITCHOFFSET	SWITCHOFFSET (DATETIMEOFFSET , time_Lugar)	SWITCH OFFSET cambia el ajuste de zona horaria de un valor DATETIMEOFFSET y conserva el valor UTC.
TODATETIMEOFFSET	TODATETIMEOFFSET (expression , time_Lugar)	TODATETIMEOFFSET transforma un valor datetime2 en un valor datetimeoffset. El valor datetime2 se interpreta en la hora local para el valor time_Lugar especificado.

Funciones que establecen u obtienen un formato de sesión

Función	Sintaxis	Valor devuelto
@@DATEFIRST	@@DATEFIRST	Devuelve el valor actual, para la sesión, de SET DATEFIRST.
SET DATEFIRST	SET DATEFIRST { number | **@**number_var }	Establece el primer día de la semana en un número del 1 al 7.
SET DATEFORMAT	SET DATEFORMAT { format | **@**format_var }	Determina el orden de los componentes de la fecha (mes/día/año) para escribir datos de tipo **datetime** o **smalldatetime**.
@@LANGUAGE	@@LANGUAGE	Devuelve el nombre del idioma que se está utilizando actualmente. @@LANGUAGE no es ninguna función de fecha u hora. Sin embargo, la configuración de idioma puede afectar a la salida de las funciones de fecha.
SET LANGUAGE	SET LANGUAGE { [N] 'language' | **@**language_var }	Establece el entorno del idioma de la sesión y los mensajes del sistema. SET LANGUAGE no es ninguna función de fecha u hora. Sin embargo, la configuración de idioma afecta a la salida de las funciones de fecha.

sp_helplanguage	**sp_helplanguage** [[**@language =**] 'language']	Devuelve información sobre los formatos de fecha de todos los idiomas compatibles. **sp_helplanguage** no es un procedimiento almacenado de fecha u hora. Sin embargo, la configuración de idioma afecta a la salida de las funciones de fecha.

Funciones que validan valores de fecha y hora

Función	Sintaxis	Valor devuelto	Tipo de datos devuelto	Determinismo
ISDATE	ISDATE (expression)	Determina si una expresión de entrada **datetime** o **smalldatetime** es un valor de fecha u hora válido.	**int**	ISDATE sólo es determinista si se utiliza con la función CONVERT, cuando se especifica el parámetro de estilo CONVERT y cuando el estilo no es igual a 0, 100, 9 ni 109.

Temas relacionados con la fecha y hora

Tema	Descripción
Usar datos de fecha y hora	Proporciona información y ejemplos comunes a las funciones y tipos de datos de fecha y hora.
CAST y CONVERT (Transact-SQL)	Proporciona información sobre la conversión de los valores de fecha y hora de literales de cadena y otros formatos de fecha y hora.
Escribir instrucciones Transact-SQL internacionales	Proporciona directrices para la portabilidad de bases datos y aplicaciones de bases de datos que utilizan instrucciones Transact-SQL de un idioma a otro, o que admiten varios idiomas.
Funciones escalares de ODBC (Transact-SQL)	Proporciona información sobre funciones escalares de ODBC que se pueden utilizar en instrucciones Transact-SQL. Esto incluye las funciones de fecha y hora ODBC.
Asignación de tipos de datos con consultas distribuidas	Proporciona información sobre cómo los tipos de datos de fecha y hora afectan a las consultas distribuidas entre servidores que tienen versiones diferentes de SQL Server o proveedores diferentes

Funciones de seguridad en TSQL

CURRENT_USER

Devuelve el nombre de usuario actual. Esta función es equivalente a USER_NOMBRES().

Sintaxis

```
CURRENT_USER
```

Tipos de valores devueltos Nombre del sistema
Notas

CURRENT_USER devuelve el nombre del contexto de seguridad actual. Si CURRENT_USER se ejecuta después de una llamada a EXECUTE AS, cambia el contexto; CURRENT_USER devolverá el nombre del contexto suplantado. Si una entidad de seguridad de Windows ha tenido acceso a la base de datos en concepto de pertenencia a un grupo, se devolverá el nombre de la entidad de seguridad de Windows en vez del nombre del grupo.

A. Utilizar CURRENT_USER para devolver el nombre del usuario actual

En el ejemplo siguiente se devuelve el nombre del usuario actual.

```
SELECT CURRENT_USER;
GO
```

DATABASE_PRINCIPAL_ID

Devuelve el número de Id. de una entidad de seguridad de la base de datos actual. Para obtener más información acerca de las entidades de seguridad.

```
DATABASE_PRINCIPAL_ID ( 'principal_Nombres' )
```

Argumentos
principal_Nombres

Es una expresión de tipo **sysNombres** que representa a la entidad de seguridad.
Si se omite el parámetro principal_Nombres, se devuelve el Id. del usuario actual. Es obligatorio utilizar paréntesis.

Tipos de valor devueltos

int
NULL cuando la entidad de base de datos no existe

Notas

DATABASE_PRINCIPAL_ID se puede utilizar en una lista de selección, en una cláusula WHERE o en cualquier lugar en el que se permita una expresión. Para obtener más información.

Ejemplos

A. Recuperar el Id. del usuario actual

En el siguiente ejemplo se devuelve el Id. de la entidad de seguridad de base de datos del usuario actual.

```
SELECT DATABASE_PRINCIPAL_ID();
GO
```

B. Recuperar el Id. de la entidad de seguridad de base de datos especificada

En el siguiente ejemplo se devuelve el Id. de la entidad de seguridad de base de datos de la función de base de datos db_owner.

```
SELECT DATABASE_PRINCIPAL_ID('db_owner');
GO
```

USER_NOMBRES

Devuelve un nombre de usuario de base de datos a partir de un número de identificación especificado.

```
USER_NOMBRES ( [ id ] )
```

Es el número de identificación asociado a un usuario de la base de datos. id es de tipo **int**. Es obligatorio utilizar paréntesis.

Cuando se omite id, se asume que se trata del usuario actual en el contexto actual. Cuando se llama a USER_NOMBRES sin especificar un id después de una instrucción EXECUTE AS, USER_NOMBRES devuelve el nombre del usuario representado. Si una entidad de seguridad de Windows ha tenido acceso a la base de datos en forma de miembro de un grupo, USER_NOMBRES devuelve el nombre de la entidad de seguridad de Windows en vez del nombre del grupo.

A. Usar USER_NOMBRES

En el siguiente ejemplo se devuelve el nombre de usuario del Id. de usuario

```
SELECT USER_NOMBRES(13);
GO
```

B. Usar USER_NOMBRES sin un identificador

En el siguiente ejemplo se busca el nombre del usuario actual sin especificar un identificador.

```
SELECT USER_NOMBRES();
GO
```

Este es el conjunto de resultados para un usuario que pertenezca al rol fijo de servidor sysadmin.

```
------------------------------
dbo
(1 fila afectada)
```

C. Usar USER_NOMBRES en la cláusula WHERE

En el siguiente ejemplo se busca la fila en sysusers en la que el nombre sea igual al resultado de aplicar la función del sistema USER_NOMBRES al número de identificador de usuario 1.

```
SELECT Nombres FROM sysusers WHERE Nombres = USER_NOMBRES(1);
GO
```

El conjunto de resultados es el siguiente.

```
Nombres
------------------------------
dbo
(1 fila afectada)
```

D. Llamar a USER_NOMBRES durante la suplantación con EXECUTE AS

En el siguiente ejemplo se muestra cómo se comporta USER_NOMBRES durante la suplantación.

```
SELECT USER_NOMBRES();
GO
EXECUTE AS USER = 'Zelig';
GO
SELECT USER_NOMBRES();
GO
REVERT;
GO
SELECT USER_NOMBRES();
GO
```

USER_ID

Devuelve el número de identificación para un usuario de la base de datos.

Esta característica se quitará en una versión futura de Microsoft SQL Server. Evite utilizar esta característica en nuevos trabajos de desarrollo y tenga previsto modificar las aplicaciones que actualmente la utilizan. Utilize en su Lugar

```
USER_ID ( [ 'user' ] )
```

Argumentos

user

Es el nombre de usuario que se va a utilizar. user es **nchar**. Si se especifica un valor **char**, se convierte implícitamente en **nchar**. Es obligatorio utilizar paréntesis.

Tipos de valor devueltos

int
Cuando se omite user, se da por supuesto que es el usuario actual. Cuando se llama a USER_ID después de EXECUTE AS, USER_ID devuelve el identificador del contexto suplantado.
Cuando una entidad de seguridad de Windows que no se ha asignado a un usuario específico de base de datos tiene acceso a una base de datos en forma de pertenencia a un grupo, USER_ID devuelve 0 (el identificador de público). Si este tipo de entidad de seguridad crea un objeto sin especificar un esquema, SQL Server creará un usuario implícito y un esquema asignados a dicha entidad. El usuario creado en casos como éste no se puede utilizar para conectarse a la base de datos. Las llamadas a USER_ID efectuadas por la entidad de seguridad de Windows asignada a un usuario implícito devolverán el identificador de éste.
Se puede utilizar USER_ID en una lista de selección, en una cláusula WHERE y en cualquier lugar en el que se permita una expresión. Para obtener más información

```
USE Cliente;
SELECT USER_ID('Harold');
GO
```

SYSTEM_USER

Permite insertar en una tabla un valor proporcionado por el sistema para el inicio de sesión actual cuando no se especifica ningún valor predeterminado.

```
SYSTEM_USER
```

Tipos de valor devueltos **nchar**

Funciones de tipo carácter en TSQL

Función	Propósito
CHR(n)	Nos devuelve el carácter cuyo valor en binario es n
CONCAT(cad1, cad2)	Nos devuelve cad1 concatenada con cad2
UPPER(cad)	Convierte cad a mayúsculas
LOWER(cad)	Convierte cad a minúsculas
LPAD(cad1,n[,cad2])	Con esta función añadimos caracteres a cad1 por la izquierda hasta una longitud máxima dada por n
INITCAP(cad)	Convierte la primera letra de cad a mayúscula
LTRIM(cad [,set])	Elimina un conjunto de caracteres a la izquierda de cad, siendo set el conjunto de caracteres a eliminar
RPAD(cad1, n[,cad2])	Con esta función añadimos caracteres de la misma forma que con la función LPAD pero esta vez los añadimos a la derecha
RTRIM(cad[,set])	Hace lo mismo que LTRIM pero por la derecha
REPLACE(cad,cadena_buscada [,cadena_sustitucion])	Sustituye un conjunto de caracteres de 0 o más caracteres, devuelve cad con cada ocurrencia de cadena_buscada sustituida por cadena_sustitucion
SUBSTR(cad, m[,n])	Devuelve la subcadena de cad que abarca desde m hasta el numero de caracteres dados por n.
TRANSLATE(cad1,cad2,cad3)	Convierte caracteres de una cadena en caracteres diferentes. Devuelve cad1 con los caracteres encontrados en cad2 y sustituidos por los caracteres de cad3

Ponemos algunos ejemplos de utilización de estas funciones:

Sentencia SQL que nos devuelve las letras cuyo valor ASCII es el 45 y el 23

SELECT CHR(45), CHR(23) FROM TABLA;

Sentencia SQL que obtiene el nombre de los alumnos sacando por pantalla la siguiente frase: el nombre del alumno es (nombre que esta almacenado en la tabla)

SELECT CONCAT ('el nombre de alumno es', nombre) from alumno;

Sentencia SQL que me devuelve los nombres de los alumnos en mayúsculas

SELECT UPPER(nombre) from alumno;

Sentencia sql que obtiene de un campo nombre, las 3 primeras letras

SELECT SUBSTR(nombre,0,3) from alumno;

Procesos para calcular la edad de una persona

```
ALTER FUNCTION CALEDAD(@FECHNAC SMALLDATETIME)
RETURNS TINYINT
WITH ENCRYPTION
As
BEGIN
DECLARE @EDAD TINYINT
SET @EDAD = YEAR(GETDATE()) - YEAR(@FECHNAC)
IF MONTH(GETDATE()) < MONTH(@FECHNAC)
SET @EDAD=@EDAD-1
IF MONTH (GETDATE()) = MONTH(@FECHNAC) AND
DAY(GETDATE())< DAY(@FECHNAC)
SET @EDAD=@EDAD - 1

RETURN @EDAD
END


SELECT dbo.CALEDAD('27/08/1969')
```

Como determinar si un valor es impar o par

```
create function f_parimpar(@numero int)
returns varchar(50)
as
begin

declare @mensaje varchar (50)
      if (@numero%2=0)
       set @mensaje='el numero  :'+convert(varchar,@numero)+'  es par'

      else
       set @mensaje ='el numero  :'+convert(varchar,@numero)+'  es impar'

    return @mensaje

    end
   ---2 llamada de la funcion
    go


    select dbo.f_parimpar (4)
```

como determinariamos el dia de la semana

```
create function f_Diasemana(@fecha datetime)
returns varchar(10)
as begin
declare @Dia  varchar (10)
set @Dia=case  DATEPART(WEEKDAY,@fecha)
when 1 then 'lunes'
when 2 then 'Martes'
when 3 then  'Miercoles'
when 4 then  'Jueves'
when 5 then  'Viernes'
when 6 then  'Sabado'
when 7 then  'Domingo'
end
return @Dia
end

go

declare @fec datetime
set @fec=convert(datetime,'17/11/2010',103)
select dbo.f_Diasemana (@fec) AS DIA


select DATEPART(WEEKDAY,GETDATE()) AS Nº_DIA
```

forma de calcular las edades

```
CREATE FUNCTION fn_Edades(@FecNac DATETIME, @FecHoy DATETIME)
RETURNS VARCHAR(50)
AS
BEGIN
DECLARE @ANO VARCHAR(3)
DECLARE @MES VARCHAR(3)
SET @ANO = YEAR(@FecHoy) - YEAR(@FecNac)
IF MONTH(@FecNac) > MONTH(@FecHoy)
BEGIN
SET @ANO = @ANO - 1
SET @MES = 12 - (month(@FecNac) + month(@FecHoy))
END
IF MONTH(@FecNac) < MONTH(@FecHoy)
BEGIN
SET @MES = (MONTH(@FecHoy) - MONTH(@FecNac))
END
IF MONTH(@FecNac) = MONTH(@FecHoy)
BEGIN
IF DAY(@FecNac) <= DAY(@FecHoy)
BEGIN
SET @MES = 0
END
IF DAY(@FecNac) > DAY(@FecHoy)
BEGIN
SET @ANO = @ANO - 1
SET @MES = 11
END
END
RETURN RIGHT('  ' + @ANO,2) + ' Años ' + RIGHT('  ' + @MES,2) + ' Meses'
END

DECLARE @FEC_NAC DATETIME
SET @FEC_NAC= convert(datetime,'15/05/1985',103)

SELECT dbo.fn_Edades(@FEC_NAC,getdate())


--FUNCION DARWIN (EDADES)

create function CalEdad(@fecha datetime)
returns int
as begin
return datediff(yy,@fecha,getdate())
end

declare @fech datetime
set @fech= convert(datetime,'15/05/1985',103)
```

```
select dbo.CalEdad(@fech)


--SENTENCIA SELECTIVA MULTIPLE
DECLARE @MES INT
SELECT @MES=5
SELECT CASE @MES
	WHEN 1 THEN 'ENERO'
	WHEN 2 THEN 'FEBRERO'
	WHEN 3 THEN 'MARZO'
	WHEN 4 THEN 'ABRIL'
	WHEN 5 THEN 'MAYO'
END

--PROCEDIMIENTOS ALMACENADOS
--SUMA DE DOS NUMEROS
CREATE PROCEDURE SUMA
@NUM1 INT,
@NUM2 INT,
@SALIDA INT OUTPUT
AS
SET @SALIDA = @NUM1 + @NUM2
GO
DECLARE @RES INT
EXECUTE SUMA 8,10, @RES OUTPUT
SELECT 'LA SUMA DE LOS NUMEROS ES' + CONVERT(VARCHAR, @RES)

--OPERACIONES MAT
CREATE PROCEDURE OPERACIONES
@NUM1 INT, @NUM2 INT, @OPERADOR VARCHAR(1), @SALIDA INT OUTPUT
AS
BEGIN
	IF @OPERADOR = '+'
		SET @SALIDA = @NUM1 + @NUM2
	IF @OPERADOR = '-'
		SET @SALIDA = @NUM1 - @NUM2
	IF @OPERADOR = '*'
		SET @SALIDA = @NUM1 * @NUM2
	IF @OPERADOR = '/'
		BEGIN
		IF @NUM2<>0
		SET @SALIDA = @NUM1/@NUM2
		END
END

DECLARE @RES INT
EXECUTE OPERACIONES 15,8,'*', @RES OUTPUT

SELECT @RES
```

Funciones que devuelven valores numéricos

Estas funciones nos devuelven números a modo de información.

Función	Propósito
ASCII(cad)	Devuelve el valor ASCII de la primera letra de cad
INSTR(cad1, cad2[,comienzo[,m]])	Función que busca un conjunto de caracteres dentro de una cadena. Nos devuelve la posición de cad2 en cad1 empezando a buscar en comienzo
LENGTH(cad)	Devuelve en número de caracteres de cad

EJEMPLO 101

Como con las funciones anteriores dejamos unos ejemplos para que veáis su funcionamiento.
Sentencia SQL que nos devuelve el valor ASCII de la letra ('s')

select ASCII('s') from tabla;

EJEMPLO 102

Sentencia que nos devuelve la posición de la ocurrencia 'pe' dentro de la cadena 'Los perros están bien' a partir de la posición 2

select INSTR('Los perros están bien','pe',2) from tabla;

Sentencia sql que nos devuelve el numero de caracteres de los nombres de los alumnos

select LENGTH(nombre) from alumnos;

Usar funciones matemáticas

Una función matemática realiza una operación matemática en expresiones numéricas y devuelve el resultado de la operación. Las funciones matemáticas operan sobre datos numéricos suministrados por el sistema SQL Server: **decimal**, **integer**, **float**, **real**, **money**, **smallmoney**, **smallint** y **tinyint**. De manera predeterminada, la precisión de las operaciones integradas para el tipo de datos **float** es de seis decimales.

De forma predeterminada, un número pasado a una función matemática será interpretado como un tipo de datos **decimal**. Se puede usar las funciones CAST o CONVERT para cambiar el tipo de datos a otro distinto, por ejemplo, **float**. Por ejemplo, el valor devuelto por la función FLOOR tiene el tipo de datos del valor de entrada. La entrada de la siguiente instrucción SELECT es un **decimal**. FLOOR devuelve 123. Éste es un valor decimal.

```
SELECT FLOOR(123.45);

----------------------
123

(1 row(s) affected)
```

Sin embargo, el siguiente ejemplo utiliza un valor float y FLOOR devuelve un valor float:

```
SELECT FLOOR (CONVERT (float, 123.45));

-------------------------------------
123.000000

(1 row(s) affected)
```

Cuando el resultado **float** o **real** de una función matemática es demasiado pequeño para mostrarse, se produce un error de desbordamiento negativo de punto flotante. El

resultado devuelto será 0,0 y no se mostrará ningún mensaje de error. Por ejemplo, el cálculo matemático de 2 elevado a la potencia -100,0 daría el resultado 0,0.

Los errores de dominio se producen cuando el valor proporcionado en la función matemática no es válido. Por ejemplo, los valores especificados para la función ASIN deben ser de -1,00 a 1,00. Si se especifica el valor -2, por ejemplo, se produce un error de dominio.

Los errores de intervalo se producen cuando el valor especificado se encuentra fuera de los valores permitidos. Por ejemplo, POWER(10,0, 400) excede el valor máximo (~2e+308) del intervalo para el tipo de datos **float** y POWER(-10,0, 401) es menor que el valor mínimo (~ -2e+308) del intervalo para el tipo de datos **float**.

En la siguiente tabla se muestran funciones matemáticas que producen un error de dominio o de intervalo.

Función matemática	Resultado
SQRT(-1)	Error de dominio.
POWER(10,0, 400)	Error de desbordamiento aritmético.
POWER(10,0, -400)	Valor 0,0 (desbordamiento negativo de punto flotante).

Se proporcionan capturas de error para controlar los errores de dominio o de intervalo de estas funciones. Puede utilizar lo siguiente:

- SET ARITHABORT ON. Esta opción termina la consulta y finaliza la transacción definida por el usuario. La configuración de SET ARITHABORT suplanta la configuración de SET ANSI_WARNINGS.
- SET ANSI_WARNINGS ON. Esta opción detiene el comando.
- SET ARITHIGNORE ON. Esta opción impide la visualización de un mensaje de advertencia. Tanto la configuración de SET ARITHABORT como de SET ANSI_WARNINGS suplantan la configuración de SET ARITHIGNORE.

Funciones de Metadatos

Muchas veces los desarrolladores tenemos la necesidad de consultar datos de la metadata de SQL Server, que es esto de la metadata? pues bien, la metadata es la información que guarda SQL Server en tablas sobre datos como (Stores, tablas, columnas de una tabla, las vistas, el código de los Store, etc.)

Imaginemos que deseamos saber los parámetros que tiene un Store Procedure o bien en que tablas se encuentra una columna dada, o cuales son las relaciones que tiene una tabla, para poder hacer todo este tipo de querys necesitamos consultar esta metadata.

Bien, como lo hacemos, pues hay varias formas , una de ellas es consultas las tablas internas de SQL lo cual no es muy recomendado porque pueden cambiar entre versiones del producto (hasta con la instalación de un service pack) y dejarnos de funcionar nuestras querys, a partir de 2005 se pueden usar las vistas de sistema que son una mejora importante ya que nos dan una capa intermedia de acceso sin acceder directamente a las tablas de sistema, pero solo funciona a partir de 2005.

La otra opción es usar unas vistas ANSI (Standard para aquellas bases de datos que soporten ANSI) llamadas INFORMATION_SCHEMA, estas vistas se encuentran en casi todas las versiones de SQL Server (2000, 2005 y 2008) pero como son ANSI no se puede obviamente consultar todo lo que se puede hacer con las vistas o tablas internas.

Bien, en este post les voy a mostrar distintas consultas habituales que hacemos como desarrolladores y las resolveré con las INFORMATION_SCHEMA.

Consultar información de las columnas de tablas

```
 -- TABLAS QUE CONTIENEN UNA COLUMNA

SELECT * FROM INFORMATION_SCHEMA.COLUMNS
WHERE COLUMN_NOMBRES = 'CLIENTEID'
```

Listado de Tablas

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
```

Campos que conforman una PK

```
SELECT
   TC.TABLE_NOMBRES, CU.COLUMN_NOMBRES
FROM
   INFORMATION_SCHEMA.TABLE_CONSTRAINTS TC
   INNER JOIN
   INFORMATION_SCHEMA.KEY_COLUMN_USAGE CU
   ON TC.CONSTRAINT_NOMBRES = CU.CONSTRAINT_NOMBRES
   WHERE TC.CONSTRAINT_TYPE = 'PRIMARY KEY'
   AND TC.TABLE_NOMBRES = 'CLIENTE' -- TABLA
ORDER BY TC.TABLE_NOMBRES
```

Relaciones entre Tablas

```
SELECT
    FK_Table  = FK.TABLE_NOMBRES,
    FK_Column = CU.COLUMN_NOMBRES,
    PK_Table  = PK.TABLE_NOMBRES,
    PK_Column = PT.COLUMN_NOMBRES,
    ConstraintNombres = C.CONSTRAINT_NOMBRES
FROM
   INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS C
   INNER JOIN
   INFORMATION_SCHEMA.TABLE_CONSTRAINTS FK
      ON C.CONSTRAINT_NOMBRES = FK.CONSTRAINT_NOMBRES
   INNER JOIN
   INFORMATION_SCHEMA.TABLE_CONSTRAINTS PK
      ON C.UNIQUE_CONSTRAINT_NOMBRES = PK.CONSTRAINT_NOMBRES
   INNER JOIN
   INFORMATION_SCHEMA.KEY_COLUMN_USAGE CU
      ON C.CONSTRAINT_NOMBRES = CU.CONSTRAINT_NOMBRES
   INNER JOIN
   (
     SELECT
          TC.TABLE_NOMBRES, CU.COLUMN_NOMBRES
     FROM
     INFORMATION_SCHEMA.TABLE_CONSTRAINTS TC
     INNER JOIN
     INFORMATION_SCHEMA.KEY_COLUMN_USAGE CU
     ON TC.CONSTRAINT_NOMBRES = CU.CONSTRAINT_NOMBRES
     WHERE TC.CONSTRAINT_TYPE = 'PRIMARY KEY'
   ) PT
   ON PT.TABLE_NOMBRES = PK.TABLE_NOMBRES
WHERE
PK.TABLE_NOMBRES = 'CONTENIDOSORDERHEADER' -- TU TABLA
```

Checks de una tabla dada

```
SELECT TC.TABLE_SCHEMA,TC.TABLE_NOMBRES,CU.COLUMN_NOMBRES,
CK.CHECK_CLAUSE
  FROM INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE CU
INNER JOIN
INFORMATION_SCHEMA.TABLE_CONSTRAINTS  TC
ON TC.TABLE_NOMBRES = CU.TABLE_NOMBRES
AND
TC.CONSTRAINT_NOMBRES = CU.CONSTRAINT_NOMBRES
INNER JOIN
```

```
INFORMATION_SCHEMA.CHECK_CONSTRAINTS CK
ON
CK.CONSTRAINT_NOMBRES = TC.CONSTRAINT_NOMBRES
WHERE CONSTRAINT_TYPE = 'CHECK'
AND
TC.TABLE_NOMBRES = 'CLIENTE' -- TU TABLA
```

Parámetros de un Store Procedure

```
SELECT *
FROM INFORMATION_SCHEMA.PARAMETERS
WHERE SPECIFIC_NOMBRES ='ufnGetProductDealerPrice'
```

Funciones MS SQL con ejemplos

El script de la base de datos lo adjunto como enlace a continuación: Base de datos (bd.sql) – 5,22 Kb

Mayúsculas y minúsculas:

```
SELECT UPPER(Nombre), LOWER(Apellido) FROM estudiantes
```

Eliminar espacios:

```
SELECT LTRIM('          Aijuna!')
SELECT RTRIM('Nanuk             ')
```

Longitud de un valor en un campo:

```
SELECT LEN(Nombre) FROM estudiantes
```

Función sub string: args(string, inicio, longitud)

```
SELECT SUBSTRING('Eoden',1,3)
```

Función buscar índice de caracter: args(char, string[, a partir de cuál])

```
SELECT CHARINDEX('n','Fernando', 5)
```

Función espacio:

– SET @valor = SPACE(9)

Función convertir a string:

STR(int)

Declaración de variables:

Formato:
DECLARE @NombreVar tipo

SET @NombreVar = Valor

Ejemplo:

```
DECLARE @Nombre char(8)
DECLARE @Apellido char(6)
DECLARE @valor char(50)

DECLARE @Edad smallint
SET @Edad = 41

SET @valor = 'Fernando Briano'
SET @Nombre = Substring(@valor, 1, (charindex(' ', @valor) - 1 ))
SET @Apellido = Substring(@valor, (charindex(' ', @valor) + 1), LEN(@valor))
```

SELECT @Nombre + ' ' + @Apellido + ' - edad: ' + CONVERT(char(3), @Edad) + ' años'

CONVERT:
CONVERT(Tipo, Campo o expresión, estilo)

El estilo en las FECHAS:

103 muestra: 23/05/1986

112 muestra: 19660523

114 muestra: hora

FUNCIONES CON FECHAS

DATEADD(partefecha (dd, mm, yy), número, fecha)

SELECT CONVERT(char(10), getdate(), 103)
SELECT DATEADD(dd, 3, getdate())

DATEDIFF(partefecha, fecha1, fecha2)

Resta la parte de la fecha = fecha2 – fecha1

SELECT DATEDIFF(yy, '19850707', getdate())
SELECT DATEDIFF(yy,FechaNac, getdate()) **FROM** Estudiantes

DATEPART([dd,mm,y, dw(día de la semana)], fecha)

SELECT DATEPART(yy, getdate())
SELECT DATENOMBRES(mm, getdate())
SELECT DAY(getdate())
SELECT E.Nombre, DATEDIFF(yy, E.FechaNac, getdate()) **AS** Edad,
C.Descripcion , DATEDIFF(dd, FechaIngreso, getdate()) **AS** Antiguedad
FROM estudiantes **AS** E, estudiantes_curso **AS** EC, cursos **AS** C
WHERE EC.IdEstudiante = E.Ci

Redondear valores:

SELECT ROUND(479.90, -1)

ISDATE(campo)
Devuelve 1 si el campo contiene una fecha válida, sino devuelve 0
Valor nulo no es de tipo fecha

ISNUMERIC(campo)
Devuelve 1 si el campo contiene un valor numérico, sino devuelve 0

SELECT ISDATE(FechaIngreso) **FROM** estudiantes

IDENTIFICADOR UNICO:

DECLARE @Identificador UNIQUEIDENTIFIER
SET @Identificador = NEWID()
SELECT @Identificador

La función NEWID solamente funciona para columnas que hayan sido declaradas como UNIQUEIDENTIFIER

FUNCIONES DEL SISTEMA

Generalmente llevan "@@" adelante. Para encontrarlas, en la ayuda se puede buscar el string @@ que nos va a mostrar la mayoría de las que hay.

@@**ROWCOUNT**
Devuelve el número de filas afectadas por la última instrucción Queda en cero cuando hubo error o no se realizó la instrucción.

SELECT * **FROM** estudiantes

SELECT @@rowcount **AS** Columnas
UPDATE estudiantes **SET** FechaEgreso = '07/07/2007' **WHERE** Nombre='Eustakio'
IF @@ROWCOUNT = 0 PRINT 'Santos Peruanos enchilados Batman! No ha funcionado'

@@ERROR
Devuelve 0 si no hubo error, o distinto de cero con un código específico (véalo en el manual) con el error

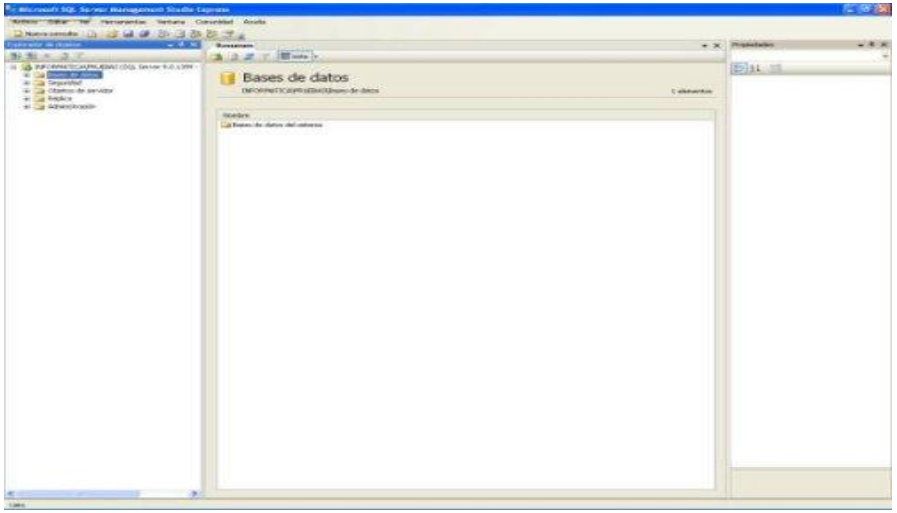
@@IDENTITY
Muestra el último identity de la tabla. O sea, si por ejemplo hacemos una columna con valor int que va autoincrementando, y es identidad, al seleccionar éste valor, nos devuelve el último insertado.

RESTAURAR UNA BASE DE DATOS DESDE UN BACKUP EN SQL

Las siguientes instrucciones están dedicadas especialmente para todos aquellos desarrolladores y administradores de bases de datos que han tenido problemas al querer restaurar bases de datos.

Primer paso

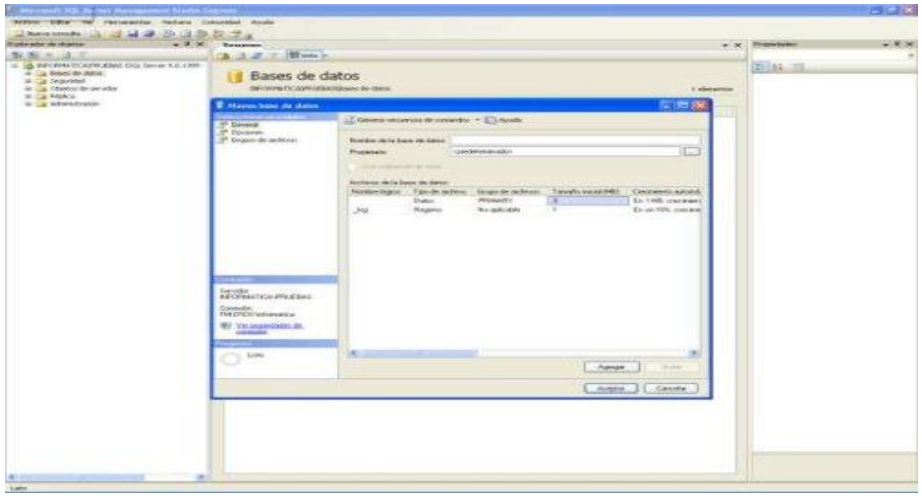
Ingresamos al Administrador de SQL Server.

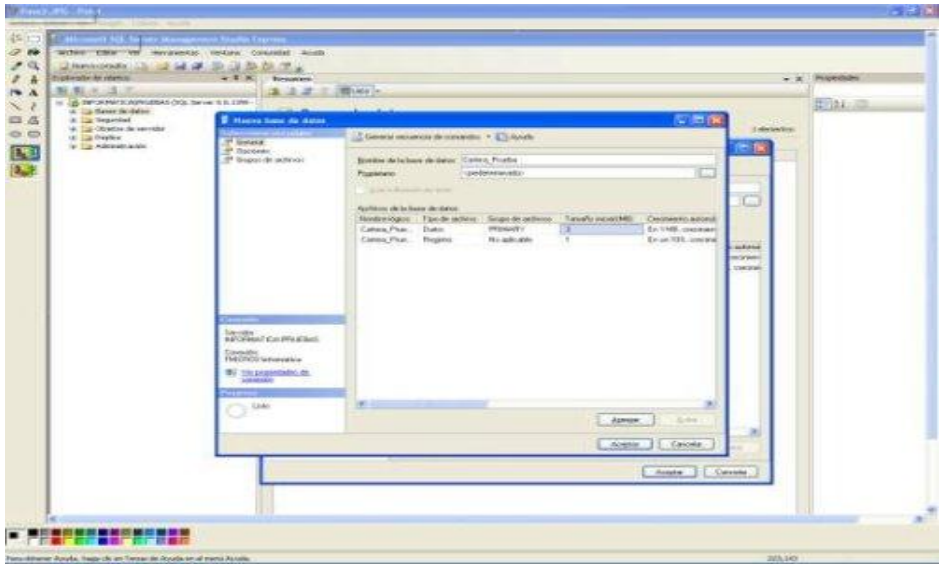


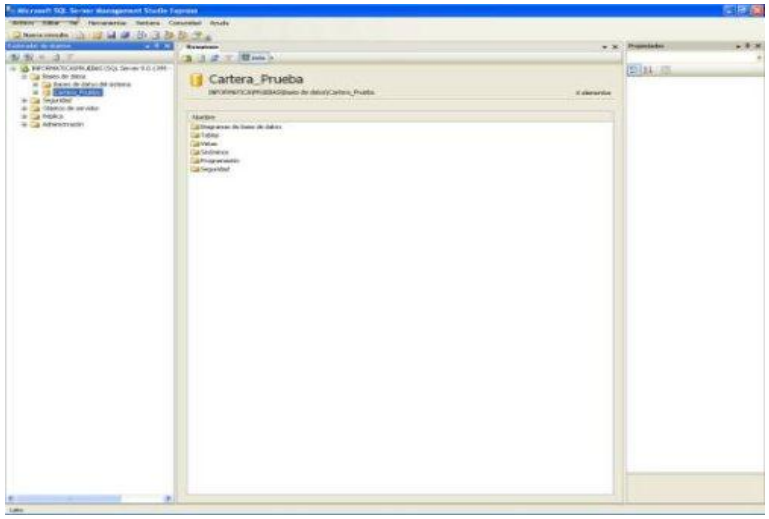
Segundo paso

Daremos Clic Derecho Sobre Base de Datos, y le damos Nueva Base de Datos, La Crearemos sin Tablas ni nada inclusive la podemos crear con diferente nombre al del Backup que queramos Restaurar

Observación: Tienen que haber hecho el Backup de su Base de datos antes, para poder hacer esto.

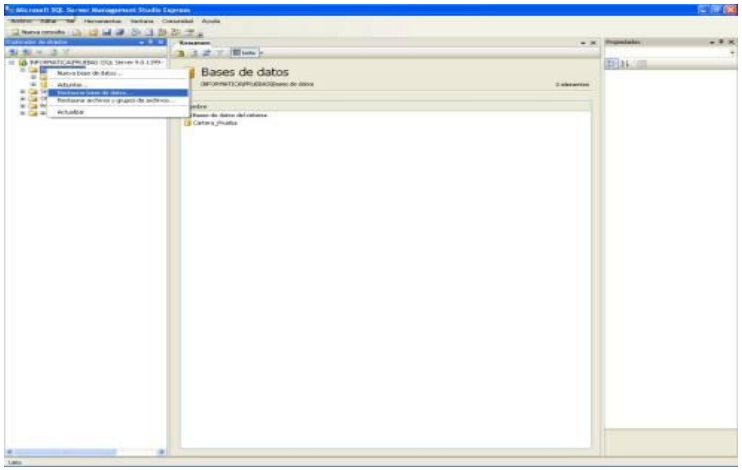






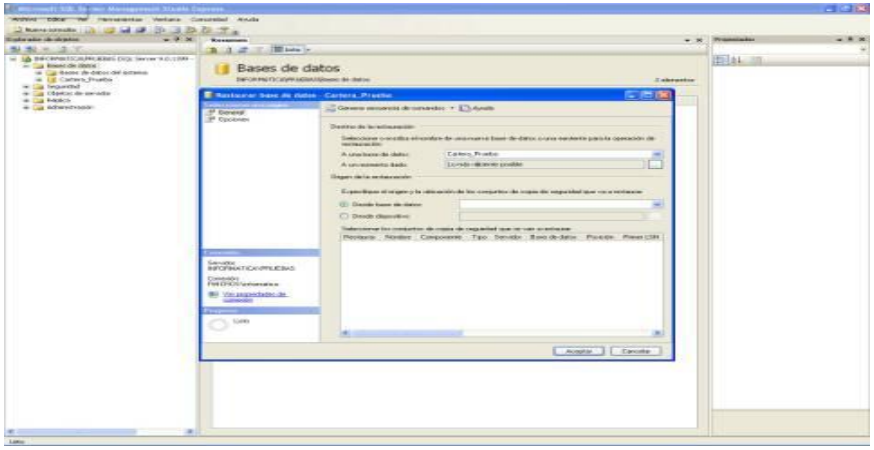
Tercer paso

Luego Damos Clic derecho sobre La Carpeta Base de Datos y damos clic en Restaurar Base de datos



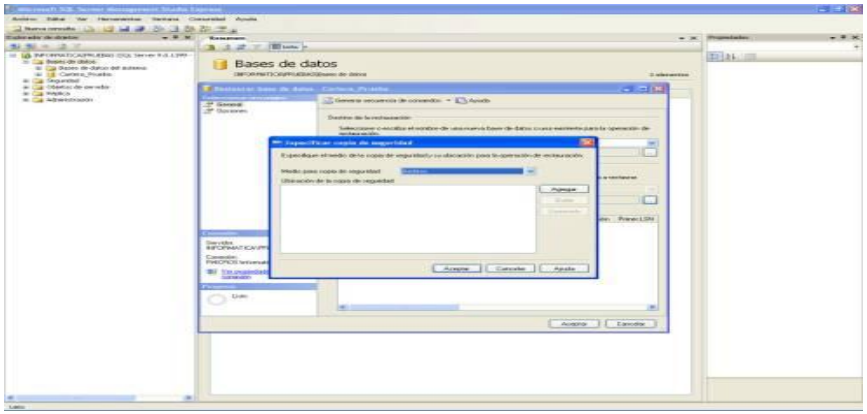
Cuarto paso

Luego Elegimos la Base de Datos que deseamos ejecutar:



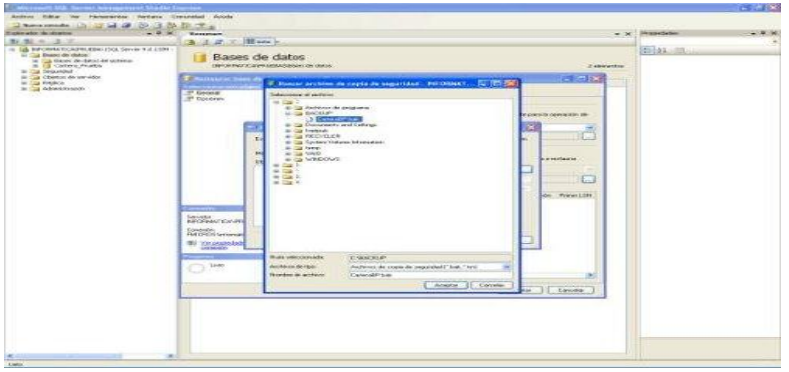
Quito paso

Seleccionamos la Opción **Desde Dispositivo** y damos Clic sobre el Botón que tiene los puntos suspensivos y nos aparecerá la siguiente pantalla.



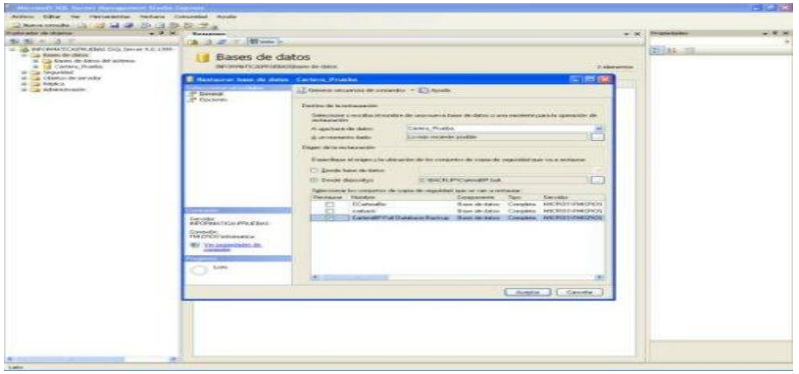
Sexto Paso

Damos clic en el Botón Agregar y Buscamos el Archivo con Extensión bak o sea el backup de nuestra base de datos.



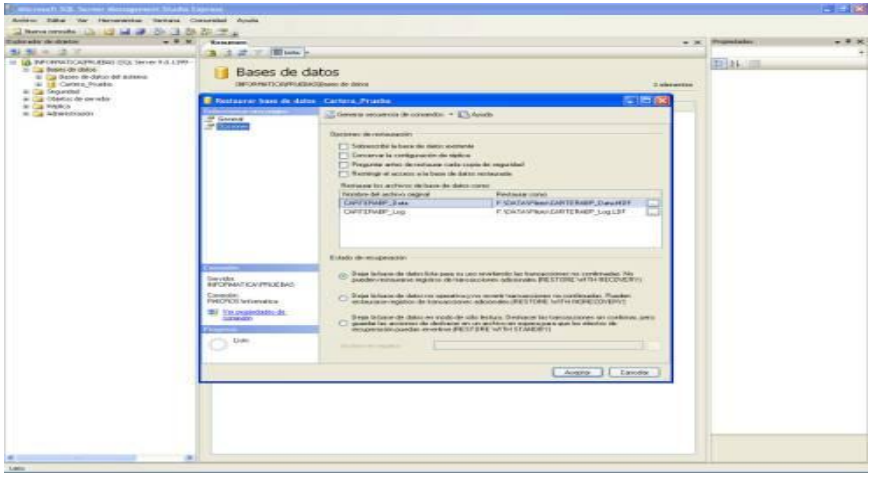
Séptimo Paso

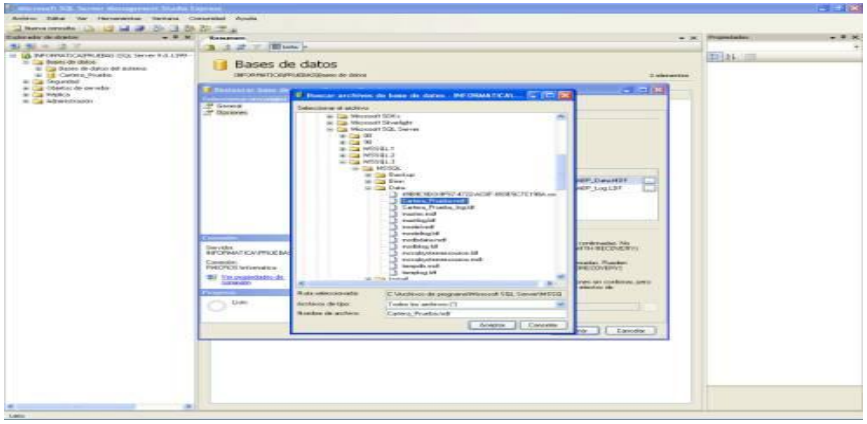
En Esta Siguiente pantalla Seleccionamos Que Backup queremos Restaurar si es que hemos hecho varios Backups de la Base de datos Original

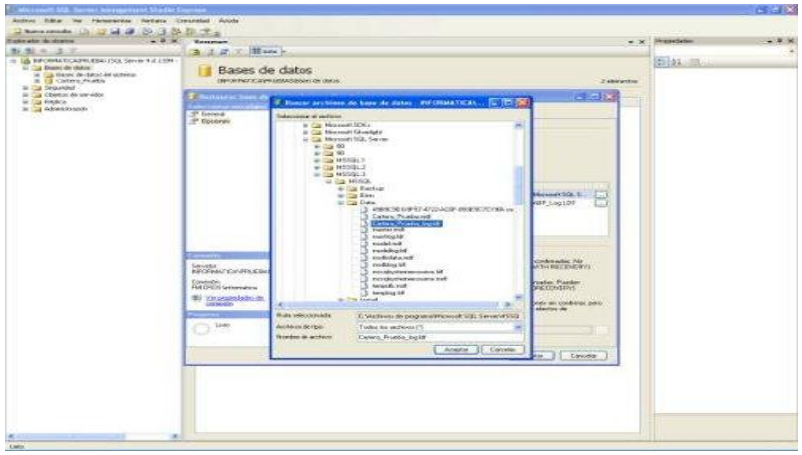


octavo Paso

En Este mismo Cuadro nos dirigimos donde se encuentran las opciones **General** y <Opciones> y Seleccionamos esta ultima y nos aparecerá esta siguiente ventana que es donde aparece la ruta por default que tiene establecido el backup, hay que cambiarlo a la dirección donde se encuentra la base de datos que creamos como se muestra a continuación.

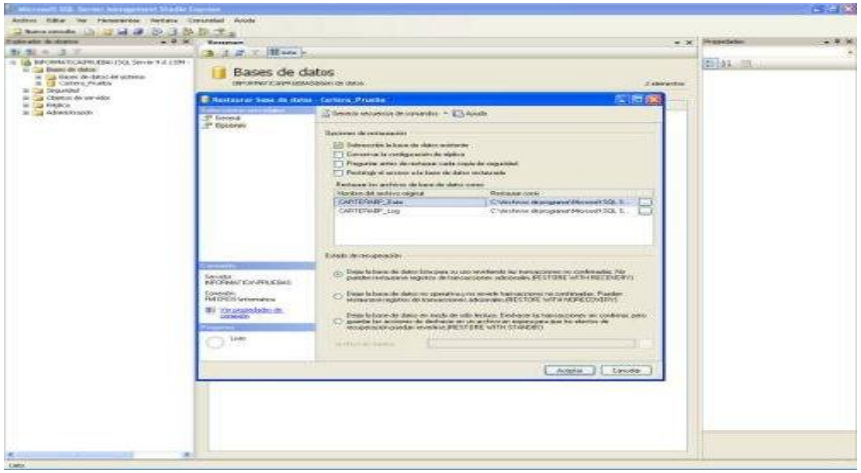






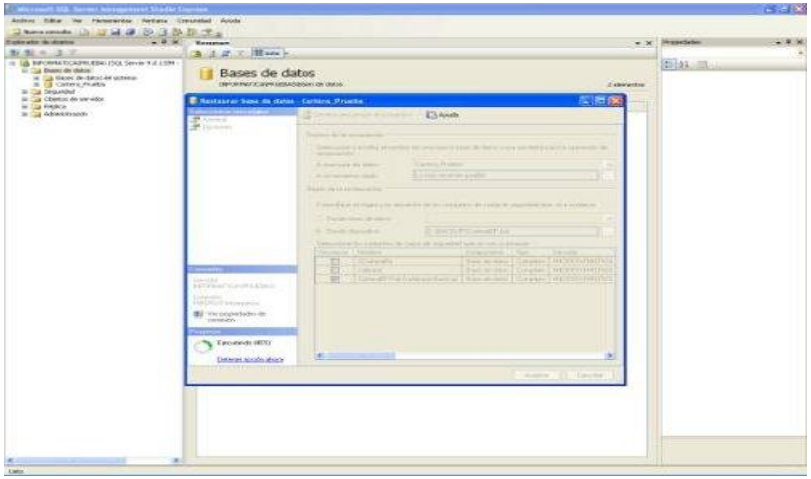
Noveno Paso

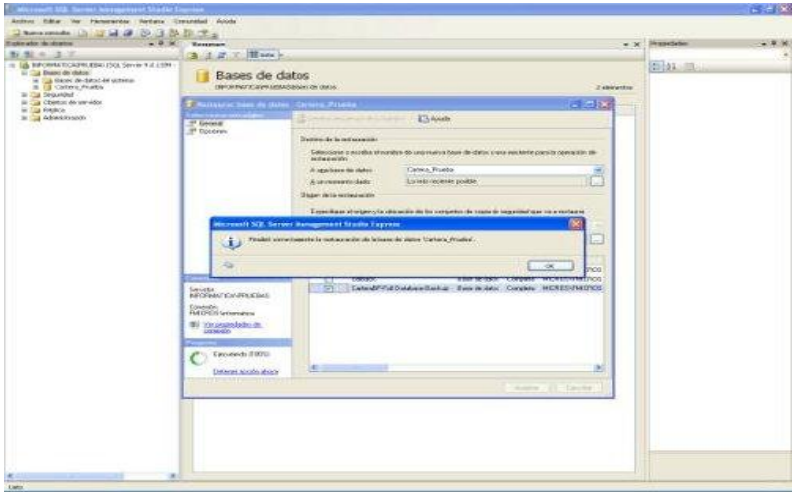
Luego que Tenemos Cambiada la Dirección de Restauración Checaremos con un clic la casilla **Sobrescribir la Base de Datos Existente**, Esto lo hacen para que no les marque ningún error al momento de Restaurar la base de datos



Decimo Paso

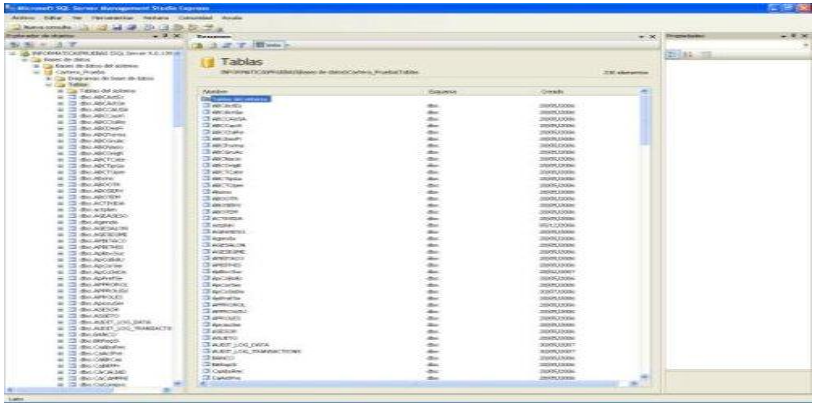
Regresamos a la Opción **General** y Presionamos el Botón Aceptar Para Que Comience la Restauración.





Onceavo Paso

verificaremos que nuestra base de datos contenga las tablas de nuestra base de datos Original y ya Podremos Trabajar Tranquilos.



CAPITULO VI

MEMORIA AYUDA

Nuevas Características de SQL SERVER 2008

FECHA/HORA

SQL Server 2008 presenta nuevos tipos de fecha y hora:

DATE – un tipo de fecha solamente

TIME – un tipo de hora solamente

DATETIMEOFFSET – una zona conciente del tipo de fecha y hora

DATETIME2 – tipo de fecha y hora con fracciones de segundos y rangos anuales más amplios que los Tipo de DATETIME

Los nuevos tipos de datos permiten que las aplicaciones posean distintos tipos de fecha y hora, además de ofrecer amplios rangos de datos y una precisión definida de usuarios para valores temporales.

Asimismo los datos NULL no consumen espacio físico, lo cual supone una manera muy eficiente de administrar datos vacíos en una base de datos. Por ejemplo, las Columnas Esparcidas admiten modelos de objetos que poseen valores NULL para ser almacenados en una base de datos de SQL Server 2005 sin ocasionar grandes costos.

CONSULTAS BASICAS

Microsoft SQL Server proporciona al usuario potentes características para realizar consultas.

Una consulta es una petición de datos almacenada el cual se puede ejecutar de diferentes modos, ya sea empleando funciones graficas de usuario para seleccionar datos de una o varias tablas.

DML
DATA MANIPULATION LANGUAGE

Se conoce con este nombre a un conjunto de sentencias de SQL que permite extraer o modificar los datos de la stablas almacenadas en una tabla, dentro de estas sentencias tenemos:
SELECT
INSERT
UPDATE
DELETE

SELECT
Recupera los datos de SQL SERVER y los presenta de nuevo al usuario en una o mas conjuntos de resultados, debiendo de recordar que existe tres componentes graficos en la instrucción SELECT

SELECT
FROM
WHERE

SINTAXIS:

SELECT [**ALL | DISTINCT**] <nombre_campo> [{,<nombre_campo>}]
FROM <nombre_tabla>|<nombre_vista>
[{,<nombre_tabla>|<nombre_vista>}]
WHERE <condicion> [{ **AND**|**OR** <condicion>}]]
[**GROUP BY** <nombre_campo> [{,<nombre_campo >}]]
[**HAVING** <condicion>[{ **AND**|**OR** <condicion>}]]
[**ORDER BY** <nombre_campo>|<indice_campo> [**ASC** | **DESC**]
[{,<nombre_campo>|<indice_campo> [**ASC** | **DESC**]}]]

	Significado
SELECT	Palabra clave que indica que la sentencia de SQL que queremos ejecutar es de selección.
ALL	Indica que queremos seleccionar todos los valores.Es el valor por defecto y no suele especificarse casi nunca.
DISTINCT	Indica que queremos seleccionar sólo los valores distintos.
FROM	Indica la tabla (o tablas) desde la que queremos recuperar los datos. En el caso de que exista más de una tabla se denomina a la consulta "consulta combinada" o "join". En las consultas combinadas es necesario aplicar una condición de combinación a través de una cláusula **WHERE**.
WHERE	Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Admite los operadores lógicos **AND** y **OR**.
GROUP BY	Especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.
HAVING	Especifica una condición que debe cumplirse para los datosEspecifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de **WHERE** pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a **GROUP BY** y la condicion debe estar referida a los campos contenidos en ella.
ORDER BY	Presenta el resultado ordenado por las columnas indicadas. El orden puede expresarse con **ASC** (orden ascendente) y **DESC** (orden descendente). El valor predeterminado es **ASC.**

WHERE

La cláusula **WHERE** es la instrucción que nos permite filtrar el resultado de una sentencia **SELECT**. Habitualmente no deseamos obtener toda la información existente en la tabla, sino que queremos obtener sólo la información que nos resulte util es ese momento. La cláusula **WHERE** filtra los datos antes de ser devueltos por la consulta.

Recordemos pues, que la selección total o parcial de una tabla se lleva a cabo mediante la instrucción Select. En dicha selección hay que especificar los campos que queremos seleccionar y la tabla en la que hacemos la selección

Ejemplo:
Select nombre, dirección From clientes

Si quisiésemos seleccionar todos los campos, es decir, **toda la tabla**, podríamos utilizar el comodín * del siguiente modo:

Select * From clientes

Resulta también muy útil el filtrar los registros mediante condiciones que vienen expresadas después de la **cláusula Where**. Si quisiésemos mostrar los clientes de una determinada ciudad usaríamos una expresión como esta:

Select * From clientes Where país Like 'Perú'

Además, podríamos **ordenar los resultados** en función de uno o varios de sus campos. Para este ultimo ejemplo los podríamos ordenar por nombre así:

Select * From clientes Where país Like 'Perú' **Order** By nombre

Teniendo en cuenta que puede haber más de un cliente con el mismo nombre, podríamos dar un segundo criterio que podría ser el apellido:

Select * From clientes Where pais Like 'Perú' Order By nombre, apellido

Tipos de Operadores :

Operadores Matemáticos

>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Distinto
=	Igual

Dentro de los operadores Lógicos tenemos : AND, OR y NOT

Otros operadores	
Like	Selecciona los registros cuyo valor de campo se asemeje, no teniendo en cuenta mayúsculas y minúsculas.
In y Not In	Da un conjunto de valores para un campo para los cuales la condición de selección es (o no) valida
Is Null y Is Not Null	Selecciona aquellos registros donde el campo especificado esta (o no) vacío.
Between...And	Selecciona los registros comprendidos en un intervalo
Distinct	Selecciona los registros no coincidentes
Desc	Clasifica los registros por orden inverso

Comodines	
*	Sustituye a todos los campos
%	Sustituye a cualquier cosa o nada dentro de una cadena
_	Sustituye un solo carácter dentro de una cadena

EJEMPLO 102

Considerar una tabla llamada proveedor, el cual se requiere visualizar todas las filas.

SELECT * FROM Proveedor ;

EJEMPLO 103

Retornar todos los valores de las columnas de la tabla cliente , teniendo los siguientes campos o columnas (código, nombres, apellidos y fechaingreso)

SELECT codigo ,nombres, apellidos, fechaingreso FROM Clientes ;

EJEMPLO 104

Reportar todas las columnas de la tabla PROVEEDOR, cuyas columnas son (id_prov,nom,dir)

SELECT id_prov AS ¨Código¨, nom AS ¨Nombres del cliente ¨,dir AS ¨Dirección¨ FROM PROVEEDOR ;

EJEMPLO 105

Retornar las columnas Id_cod, Nom_cli, Fec_in de la tabla contratos que se encuentra en la Base de datos Padron; además haga un encabezado de cada columna de la tabla.

```
USE PADRON
GO
SELECT id_cod AS Código ,nom_cli AS Nombre, fec_in  AS ¨Fecha de ingreso¨ FROM CONTRATOS ;
```

EJEMPLO 106

Ejecutaremos un ejemplo en forma similar al ejemplo anterior donde se detalle los códigos de cada columna anteponiendo un encabezado en cada una de ellas

```
USE PADRON
GO
SELECT id_cod AS ¨Código¨ ,nom_cli AS ¨Nombre del cliente ¨, fec_in  AS ¨Fecha ¨ , monto as Monto FROM CONTRATOS ;
```

EJEMPLO 107

Considerando el ejemplo anterior, diga usted como retornar todos los valores de las columnas empleando formas alternativas.

```
USE PADRON
GO
SELECT id_cod = CODIGO
        nom_cl i= ¨NOMBRE DE CLIENTE¨
        FROM CONTRATOS
GO
```

NOTA

Recordemos que si queremos colocar un alias para cada uno de los nombres de cada columna, definiendo títulos, solo debe escribir entre comillas, aquellos títulos o cabeceras que contenga espacio.

EJEMPLO 108

Ejecutaremos un ejemplo en forma similar al ejemplo anterior

```
USE PADRON
GO
SELECT id_cod = CODIGO
        nom_cl i= ¨NOMBRE DE CLIENTE¨
        Gastos_cli= GASTOS
        FROM [Datos de los Clientes]
GO
```

EJEMPLO 109

Ejecutaremos un ejemplo en forma similar al ejemplo anterior

```
USE PADRON
GO
SELECT id_cod = CODIGO
        nom_cl i= ¨NOMBRE DE CLIENTE¨
        Gastos_cli= GASTOS
        ¨Promedio de ingreso¨ =Gastos*0.01
        FROM [Datos de los Clientes]
GO
```

EJEMPLO 110

Ejecutar una consulta

```
USE PADRON
GO
SELECT id_cod = CODIGO
        nom_cl i= ¨NOMBRE DE CLIENTE¨
        Gastos_cli= GASTOS
        ¨Promedio de ingreso¨ =Gastos*0.01
        “Salario semanal” = Gastos*0.30*aporte_cli
        FROM [Datos de los Clientes]
GO
```

EJEMPLO 111

Considerando que en una tabla llamada alumnos tenemos los siguientes campos (cli_cod, nom_cli, apellidos, cli_ed, nota_1, nota_2, nota_3)
Reportar el promedio de sus notas y los nombres de los alumnos concatenados en una sola columna.

```
SELECT cli_cod AS CODIGO, Nombres =Nom_cli+” “+apellidos, cli_ed, promedio =(nota_1+nota_2+nota_3)/2 FROM Alumnos
GO
```

CLAUSULA WHERE

Esta clausula se define para las condiciones que debe cumplir cada fila, dentro del cual se pueden trabajar con operadores (=,<,>,>=,<=,<>); Así como los operadores que han sido detallados anteriormente.

EJEMPLO 112

En una determinada tabla, reportar todos los registros cuya edad sean diferentes a 30.

```
SELECT cli_cod AS CODIGO, Nombres =Nom_cli+” “+apellidos, edad FROM CLIENTES WHERE edad<>30
```

EJEMPLO 113

En una determinada tabla, reportar todos los registros cuya edad sean iguales a 30.

```
SELECT cli_cod AS CODIGO, Nombres =Nom_cli+” “+apellidos, edad FROM CLIENTES WHERE edad = 30
```

NOTA

Si el nombre de la tabla contiene espacios, se debe cerrar entre [ ].

EJEMPLO 114

En una determinada tabla, reportar todos los registros cuya edad sean menores a 30.

SELECT cli_cod AS CODIGO, Nombres =Nom_cli+" "+apellidos, edad FROM CLIENTES WHERE edad < 30

EJEMPLO 115

En una determinada tabla, reportar todos los registros cuya edad sean mayores a 40.

SELECT cli_cod AS CODIGO, Nombres =Nom_cli+" "+apellidos, edad FROM CLIENTES WHERE edad > 40

EJEMPLO 116

En una determinada tabla, reportar todos los registros cuyos nombres se encuentren antes o en la posición del cliente "JUANA TORRES", cuando la lista sea ordenada por nombres.

SELECT cli_cod AS CODIGO, Nombres =Nom_cli+" "+apellidos FROM CLIENTES WHERE Nomcli cli+" "+apellidos <="JUANA TORRES"

EJEMPLO 117

En una determinada tabla, reportar todos los registros cuyos nombres se encuentren después de la posición del cliente "JUANA TORRES", cuando la lista sea ordenada por nombres.

SELECT cli_cod AS CODIGO, Nombres =Nom_cli+" "+apellidos FROM CLIENTES WHERE Nomcli cli+" "+apellidos > = "JUANA TORRES"

CLAUSULA DISTINCT

Esta clausula permite eliminar las filas duplicadas de los resultados mostrados.

La palabra clave DISTINCT elimina las filas duplicadas de los resultados de una instrucción SELECT, con DISTINCT, puede eliminar los duplicados y ver sólo los Id. de producto que sean únicos.

Con la palabra clave DISTINCT, se considera que los valores NULL son duplicados unos de otros. Cuando se incluye DISTINCT en una instrucción SELECT, sólo se devuelve un valor NULL en los resultados, con independencia del número de valores NULL que se encuentre.

EJEMPLO 118

En una determinada tabla, reportar todas las columnas de la tabla CLIENTES, sin considerar los códigos duplicados

SELECT DISTINCT cli_cod FROM CLIENTES
GO

EJEMPLO 119

En una determinada tabla, reportar todas los registros de la tabla clientes cuyo DNI no sea duplicado

SELECT DISTINCT DNI FROM CLIENTES

GO

NOTA

Recordemos si es que no especificamos DISTINCT el reporte a ejecutar mostrara todas las filas de la tabla.

CLAUSULA INTO

La cláusula de SQL INSERT INTO se utiliza para insertar datos en una tabla de SQL. El SQL INSERT INTO se utiliza con frecuencia y tiene la sintaxis genérica:

INSERT INTO Tabla1 (Columna1, Columna2, Columna3 ...)
VALORES (ColumnValue1, ColumnValue2, ColumnValue3 ...)
La cláusula de SQL INSERT INTO tiene en realidad dos partes - la primera especificación de la tabla que se inserta en y con la lista de columnas que son la inserción de valores para, y la segunda, especificando los valores insertados en la lista de columnas de la primera parte.

EJEMPLO 120

Crear una tabla temporal llamada #lucky e insertar todas las columnas y las filas de la tabla cliente

```
SELECT * into #lucky  FROM cliente
GO
SELECT * FROM #lucky
```

EJEMPLO 121

Crear una tabla permanente llamado listado e insertar todas las columnas y las filas de la tabla clientes

```
IF EXISTS (Select table_Nombres FROM INFORMATION_SCHEMA.TABLES
 WHERE table_Nombres =”listado”)
          DROP TABLE listado
GO
SELECT * INTO listado FROM CLIENTES
-Ahora efectuamos la consulta  de la tabla listado
SLECT * FROM listado
```

NOTA

Para comprobar la existencia de dicha tabla #lucky lo llamaremos mediante la sentencia SELECT

EJEMPLO 122

Retornar todas las columnas y todas las filas de la tabla clientes cuyos apellidos sean PASCAL o PEREDA, dentro de la tabla USUARIO.

```
SELECT cli_cod, nombres, edad FROM USUARIO WHERE Apellido = ¨PEREDA¨ OR Apellido=¨PASCAL¨
GO
```

EJEMPLO 123

Retornar todas las columnas de la tabla usuario cuya edad sea igual a 20 y además IDLOCAL sea igual a 4.

```
SELECT * FROM USUARIO WHERE edad = 20 and IDLOCAL=4
GO
```

EJEMPLO 124

Reportar todas las columnas de la tabla usuario cuya edad sea mayor a 30 y el IDLOCAL sea igual a 2

```
SELECT * FROM USUARIO WHERE edad > 30 and IDLOCAL=2
GO
```

EJEMPLO 125

Retornar todas las columnas de la tabla usuario cuya columna IDLOCAL de la tabla sea 2,4 o 8

```
SELECT * FROM USUARIO WHERE IDLOCAL =2 or IDLOCAL=4 or IDLOCAL=8
GO
```

EJEMPLO 126

Empleando la función IN desarrollar el ejemplo anterior

```
SELECT * FROM USUARIO WHERE IDLOCAL IN (2,4,8)
```

EJEMPLO 127

Retornar todas las columnas de la tabla usuario cuya edad esté comprendido entre 10 y 20

```
SELECT * FROM usuario WHERE edad>=10 and edad <=20
GO
```

EJEMPLO 128

Haciendo uso de la función BETWEEN reportar todas las fechas de ingreso comprendidos entre 01-01-96 al 20-04-96

```
SELECT * FROM usuario WHERE fecha BETWEEN ¨01-01-96¨AND ¨20-04-96¨
GO
```

EJEMPLO 129

Haciendo uso de la función BETWEEN reportar todas las columnas similares en el ejemplo 127

```
SELECT * FROM usuario WHERE edad BETWEEN 10 AND 20
GO
```

EJEMPLO 130

Reportar todas las fechas de ingreso de la tabla usuario que sea 08 de Abril del año 1996

```
Set dateformat dmy
Select fecha,código,nombres FROM USUARIO WHERE fecha = ¨08/04/1996¨
GO
```

NOTA

Se emplea la Función IN si y solo sí cuando la condición se emplea OR y campo numérico, asimismo se emplea la función BETWEEN si y solo sí cuando la condición se emplea AND y el rango de valores es numérico, valores de cadenas o fechas.

CONVERT

Función de datos para tipo fecha cuyos estilos que se pueden utilizar son:

Año con 4 dígitos	Salida	Año con 2 dígitos
100 or 0	mon dd yyyy hh:miAM (or PM)	
101	mm/dd/yy	1
102	yy.mm.dd	2
103	dd/mm/yy	3
104	dd.mm.yy	4
105	dd-mm-yy	5
106	dd mon yy	6
107	Mon dd, yy	7
108	hh:mm:ss	
109 or 9	mon dd yyyy hh:mi:ss:mmmAM (or PM)	
110	mm-dd-yy	
111	yy/mm/dd	
112	yymmdd	
113 or 13	dd mon yyyy hh:mm:ss:mmm(24h)	
114	hh:mi:ss:mmm(24h)	
120 or 20	yyyy-mm-dd hh:mi:ss(24h)	
121 or 21	yyyy-mm-dd hh:mi:ss.mmm(24h)	
126	yyyy-mm-ddThh:mm:ss.mmm(no spaces)	
130	dd mon yyyy hh:mi:ss:mmmAM	
131	dd/mm/yy hh:mi:ss:mmmAM	

EJEMPLO 131

De acuerdo al ejemplo anterior, desarrollaremos considerando la columna de fecha hora.

```
Select fecha,código,nombres FROM USUARIO WHERE
        CONVERT (char(10),fecha,103)<=¨08/04/1996¨
```

--- Ahora empleando:

```
Select fecha,código,nombres FROM USUARIO WHERE
        CONVERT (char(10),fecha,102)<=¨1996.04.08¨
```

EJEMPLO 132

Ejecutar un reporte, donde muestre un rango de precios entre 30 y 40 de la tabla Productos, ordenado por detalle de articulo.

```
Select  código,detalle,precio FROM productos WHERE precio  BETWEEN 30  AND 40
ORDER BY detalle
```

EJEMPLO 133

Ejecutar un reporte, donde muestre todos los apellidos paternos enter AV y CE de la tabla Alumnos.

```
Select  codalumno,nombres,apepat AS ¨apellido paterno¨ FROM Alumnos WHERE
apepat BETWEEN ÄV¨ AND ¨CE¨ORDER BY apepat
```

NOTA

Si en el resultado no muestra ningún listado con la fecha indicada, debemos de saber que el problema es debido a que el tipo de campo de la columna debe ser fecha; pero si es fecha-hora; en este caso la fecha almacena la fecha y la hora.
Por lo que pasa a ser mas exacto en estos casos lo podemos llamar de la siguiente manera:

```
SELECT fecha, codigo, nombres FROM USUARIO WHERE fecha<=¨08/04/1996¨
```

EJEMPLO 134

Como hemos visto el desarrollo de los ejemplos anteriores haciendo uso de la función BETWEEN hemos realizado una consulta entre rangos de fecha, el cual vamos a desarrollar lo mismo empleando la sentencia DATEFORMAT entre los rangos ¨01-01-96¨al ¨20.04-96¨

```
SET DATEFORMAT dmy
GO
SELECT Idlocal,fecha FROM USUARIO WHERE fecha BETWEEN ¨01/01/1996¨ AND
¨20/04/1996¨ ORDER BY fecha

- - Otra forma de dar solición rapida seria:
SET DATEFORMAT dmy
Go
SELECT idlocal, fecha FROM usuario
WHERE fecha BETWEEN ¨01/01/1996¨ AND ¨20/04/1996 23.59:59.999 ¨
ORDER BY fecha

- - O tambien empleando la función CONVERT
SELECT idlocal, fecha FROM usuario
WHERE  CONVERT(char(10),  fecha,  103)>=¨01/01/1996¨AND  CONVERT  (char(10),
fecha,103)<=¨20/14/1996¨ORDER BY fecha
Go
```

LIKE

Esta función permite realizar busquedas usando comodines dentro de un determinado rango de caracteres:

%	Indica la posición del comodin incluyendo los valores nulos
[AB]	Indica los valores comprendidos entre A y B
c	Establece un rango de valores entre A y B
-	Indica la ubicación de un comodin
~	Indica cualquier caracter
[^]	Indica cualquier valor que no estee dentro del rango especificado.

EJEMPLO 135

Listar todos los nombres de los artículos cuya descripción sea iniciando con la letra "A"

SELECT * from Articulos WHERE descripcion_bien LIKE ¨A%¨

EJEMPLO 136

Listar todas las filas de los artículos que se encuentran en la tabla Bienes, cuya unidad de medida termine en ¨ZA¨

SELECT descripcion_bien ¨Descripcion del Bien¨, undmed from Bienes WHERE undmed LIKE ¨%ZA¨

EJEMPLO 137

Retornar todas las columnas de la tabla bienes, cuya descripción de los bienes tengan los caracteres "MANTECA"

SELECT descripcion_bien ¨Descripcion del Bien¨, undmed from Bienes WHERE descripcion_bien LIKE ¨%MANTECA%¨
GO

EJEMPLO 138

Retornar todas las columnas de la tabla bienes, cuya descripción de los bienes comiencen con P o B y a continuación la palabra "ala"

SELECT descripcion_bien ¨Descripcion del Bien¨, undmed from Bienes WHERE descripcion_bien LIKE ¨[PB]ala ¨
GO

VALORES A BUSCAR CON LIKE

LIKE	"_A%"	El segundo caracter es A
LIKE	"%A"	El ultimo caracter es A
LIKE	"A%"	El Primer caracter es A
LIKE	"%[1-9]"	Que termine con una cifra
LIKE	"%A%"	El intermedio caracter es A
LIKE	"AB[_]%"	Que comience con AB
LIKE	"[AB]"	Que comience con A o B
LIKE	"[^A]"	Que comience diferente de A

EJEMPLO 139

Retornar todas las columnas de la tabla bienes, cuya descripción de los bienes comiencen con P o B y a continuación la palabra "ala"

SELECT descripcion_bien ¨Descripcion del Bien¨, undmed FROM Bienes WHERE descripcion_bien LIKE ¨[PB]ala ¨
GO

EJEMPLO 140

Retornar todos los valores de las columnas cuyo producto sea "azúcar"

SELECT * FROM producto WHERE descripcion_bien LIKE ¨azucar¨ ORDER BY descripcion
Go

EJEMPLO 141

De acuerdo al ejemplo anterior, listar todos los artículos cuya descripción empiece con "azúcar" al iniciar

SELECT * FROM producto WHERE descripcion_bien LIKE ¨azucar%¨ ORDER BY descripcion

EJEMPLO 142

Ahora usted, liste todas las columnas de la tabla anterior cuya descripción tenga parte la palabra "azúcar"

EJEMPLO 143

Ahora listar todas las columnas de la tabla producto, cuya descripción contenga la cadena "negr" antes del ultimo carácter

SELECT * FROM producto WHERE descripcion_bien LIKE ¨%negr_¨ ORDER BY descripcion
Go

EJEMPLO 144

Listar todas las columnas de la tabla producto cuya descripción del articulo empiecen con "A, B o C"

SELECT* FROM `producto WHERE descripcion_bien LIKE ¨[ABC]% ¨
Go

EJEMPLO 145

Listar todas las columnas de la tabla producto cuya descripción del articulo comience entre los rangos de A hasta D

SELECT* FROM `producto WHERE descripcion_bien LIKE ¨[A-D]% ¨
Go

EJEMPLO 146

Listar todas las columnas cuyo primer carácter de la descripción del bien sea diferente a A,B o C

SELECT* FROM `producto WHERE descripcion_bien LIKE ¨[^ABC]% ¨
Go

EJEMPLO 147

Continuando con el ejemplo anterior ejecutar rangos, en caso de que no desamos que muestre desde la letra F hasta el carácter L, dentro del inicio de la descripción del articulo

SELECT* FROM `producto WHERE descripcion_bien LIKE ¨[^F-L]% ¨
Go

ORDER BY

La cláusula ORDER BY ordena los resultados de una consulta por una o más columnas, hasta 8.060 bytes.

A partir de SQL Server 2005, SQL Server permite especificar columnas de ordenación de las tablas de la cláusula FROM que no están especificadas en la lista SELECT. Los nombres de columna a los que se hace referencia en la cláusula ORDER BY deben corresponderse con una columna de la lista SELECT o con una columna de la tabla de la cláusula FROM sin ambigüedades. Si los nombres de columna están asociados a un alias en la lista SELECT, sólo puede utilizarse el nombre de alias en la cláusula ORDER BY. De igual modo, si los nombres de tablas están asociados a un alias en la cláusula FROM, sólo pueden utilizarse los nombres de alias para calificar sus columnas en la cláusula ORDER BY.

Una ordenación puede ser ascendente (ASC) o descendente (DESC). Si no se especifica ninguna, se supone que es ASC.

La siguiente consulta devuelve los resultados ordenados de forma ascendente por Codigo_cli.

```
USE Cliente;
GO
SELECT Codigo_cli, Nombre_ciente, Apellido_cliente
FROM Trabajo.Product
ORDER BY Codigo_cli;

USE Cliente;
```

En las columnas que tengan tipos de datos text, ntext, image o xml no se puede usar ORDER BY. Además, en la lista ORDER BY no se permiten subconsultas, agregados ni expresiones constantes. Sin embargo, en la lista de selección es posible utilizar un nombre especificado por el usuario para los agregados o expresiones.

Ejemplo

```
SELECT Color, AVG (precio) AS 'promedio de precios'
FROM Articulos
GROUP BY codigo
ORDER BY 'promedio de precios';
```

ORDER BY solamente garantiza un resultado ordenado para la instrucción SELECT más externa de una consulta. Por ejemplo, considere la definición de vista siguiente:

EJEMPLO 148

Listar todas las cantidades por cada producto o artículo en la tabla producto y ordenado por descripción

```
SELECT umedida, detabien AS ¨Detalle del bien¨, count)cantidad)  FROM producto
GROUP BY codigobien ORDER BY detabien DESC

/* también podría ser así */


SELECT umedida, detabien AS ¨Detalle del bien¨, count)cantidad)  FROM producto
GROUP BY codigobien ORDER BY 2 DESC
```

EJEMPLO 149

Explique usted, el contenido textual de la consulta:

```
SELECT umedida, detabien AS ¨Detalle del bien¨, SUM)cantidad)  FROM producto
GROUP BY codigobien ORDER BY 1 ASC
```

HAVING

Especifica una condición de búsqueda, el cual esta cláusula se emplea con GROUP BY, recordemos que cuando no se emplea esta última clausula HAVING se utiliza como WHERE.

EJEMPLO 150

Listar las columnas de la tabla producto, ordenadas por descripción que contengan los códigos, descripción y suma de cantidades vendidas con la condición que la suma de cantidades sea mayor a 120

```
SELECT codigobien, detabien AS ¨Detalle del bien¨, SUM)cantidad)  FROM producto
GROUP BY codigobien HAVING suma(cantidad)>120 ORDER BY detabien
Go
```

VALORES NULL

Cuando encontramos estos valores dentro de los campos respectivos es cuando el valor NULO es desconocido, el valor NULL es diferente a decir un registro vacío.

Tal como se detallo en el capitulo II de la stablas donde se detalla la creación de tablas con valores NULOS.

EJEMPLO 151

Vamos a crear la tabla Alumnos, dentro de la base de datos MAster

```
USE MASTER
Go
CREATE TABLE Alumnos (
        Codigo          Int             PRIMARY KEY ,
        Nombres         Varchar(40)     NOT NULL,
        Costo           Decimal (10,2)  NULL,
        Edad            Int             NULL)
Go
/* Ahora insertamos valores */
INSERT INTO ALUMNOS
        VALUES(100,¨Cesar Pereda¨,NULL,40)

INSERT INTO ALUMNOS
        VALUES(100,¨Mariluisa Pascal¨,20000.00,20)

INSERT INTO ALUMNOS
        VALUES(100,¨Harumi Pereda¨,180020.00,15)
```

EJEMPLO 152

Reportar todas las columnas de la tabla Alumnos, cuyo valor de la edad no sea nulo

```
SELECT  codigo,nombres,edad FROM Alumnos WHERE edad IS NOT NULL
Go
```

/* El caso hubiese sido si no colocamos la opción WHERE edad IS NOT NULL, mostraría todos los registros incluyendo NULOS y NO NULOS.*/

VALORES ISNULL

EJEMPLO 153

Recordemos que si tenemos valores NULOS y NO NULOS dentro de la tabla y deseamos hacer operaciones pues debemos emplear la función ISNULL.

```
SELECT  codigo,nombres,edad FROM Alumnos WHERE edad IS NULL
Go

SELECT  codigo,nombres,edad,impuesto =0.02*edad+ ISNULL(costo*0.01*edad)/10)
FROM Alumnos
Go
```

/* El caso hubiese sido si no colocamos la opción WHERE edad IS NOT NULL, mostraría todos los registros incluyendo NULOS y NO NULOS.*/

PRACTICAS DE DESARROLLO

EJEMPLO 154

Crear la tabla CONTROL, donde se pueda almacenar los datos del personal de una empresa, tales columnas son:

Codigo	**Varchar(20)**
Nombres	**Varchar(40)**
Apellidos	**Varchar(40)**
Direccion	**Varchar(120)**
Email	**Varchar(10)**

```
/* Luego con SP_TABLE visualizar las tablas existentes */
SP_TABLES @table_owner =¨dbo¨
```

SP_COLUMNS

EJEMPLO 155

Crear la tabla Alumnos con las siguientes columnas:

```
Codigo          C(10)
Nombres         C(40)
Edad            Int
Aula            Int

/*Si existe la tabla eliminemos la existente */
IF OBJECT_ID("Alumnos") IS NOT NULL
        DROP TABLE Alumnos;

/* Ahora visualicemos la estructura de la tabla */
SP_Columns Alumnos

SELECT  codigo,nombres,edad,impuesto  =0.02*edad+  ISNULL(costo*0.01*edad)/10)
FROM Alumnos
Go
```

EJEMPLO 156

Crear en la tabla Alumnos una clave primaria al código del ejemplo anterior.

EJEMPLO 157

Visualizar la estructura de la tabla Alumnos.

```
SP_COLUMNS Alumnos
```

EJEMPLO 158

Vamos a crear la tabla Farmacia que genere valores secuenciales automáticamente comenzando en 1 y se incremente de uno en uno, empleando IDENTITY.

```
CREATE TABLE Farmacia (
        CODIGO          Integer Identity (1,1),
        Ubicación       Varchar(30) NOT NULL,
        Almacen         Varchar(40) NOT NULL,
        Precio          float,
        Cantidad        Integrer
);
```

EJEMPLO 159

Vamos a crear la tabla Farmacia que genere valores secuenciales automáticamente comenzando en 2 y se incremente de tres en tres, empleando IDENTITY.

```
CREATE TABLE Farmacia (
        CODIGO          Integer Identity (2,3),
        Ubicación       Varchar(30) NOT NULL,
        Almacen         Varchar(40) NOT NULL,
        Precio          float,
        Cantidad        Integrer
);
```

NOTA

Recordemos que si deseamos insertar registros manualmente en una tabla cuya columna contenga IDENTITY, debemos emplear SET IDENT _INSERT.

EJEMPLO 160

Vamos a añadir registros manualmente en la tabla anterior creada, para esto emplearemos SET IDENTITY.

```
SET IDENTITY_INSERT Farmacia ON,

INSERT INTO Farmacia (codigo,ubicacion, almacen,precio,cantidad)
                VALUES(30,¨Central¨,¨Lima 100¨,0.20,60)
```

IDENT_SEED

EJEMPLO 161

Emplee la función IDENT_SEED para verificar el valor del inicio del campo IDENTITY de la tabla, en este caso como ejemplo de la tabla farmacia.

```
SELECT IDENT_SEED (¨Farmacia¨);
```

IDENT_INCR

EJEMPLO 162

Ahora verificaremos cual es el valor del incremento del campo IDENTITY de la tabla farmacia.

```
SELECT IDENT_INCR (¨Farmacia¨);
```

TRUNCATE

EJEMPLO 163

Ahora diga usted, como eliminaremos todos los registros de la tabla farmacia.

```
TRUNCATE table farmacia
```

SET DATEFORMAT

EJEMPLO 164

Consideramos que en la tabla farmacia hemos añadido una columna llamada Fecharegistro (datetime), el cual deseamos insertar registros, para esto en el campo fecha emplearemos el formato dia-mes-año.

```
SET DATEFORMAT ¨dmy¨
SET IDENTITY_INSERT Farmacia ON,

INSERT INTO Farmacia (codigo,ubicacion, almacen,precio,cantidad,fecharegistro)
                VALUES(30,¨Central¨,¨Lima 100¨,0.20,60,¨27/08/2010¨);


/*Ahora insertamos según formato mes-día –año*/

SET DATEFORMAT ¨mdy¨
SET IDENTITY_INSERT Farmacia ON,

INSERT INTO Farmacia (codigo,ubicacion, almacen,precio,cantidad,fecharegistro)
                VALUES(30,¨Central¨,¨Lima 100¨,0.20,60,¨08/27/2010¨);
```

EJEMPLO 165

Crear la tabla llamada medicamentos, si es que existe lo eliminaremos, considerando que el campo código sea primario

```
IF OBJECT_ID (¨ medicamentos¨) IS NOT NULL
        DROP TABLE medicamentos;
CREATE table medicamentos(
Codigo          int       Identity,
Nombre          varchar(30),
Quimico         varchar(40)  default ¨Laboratorio¨,
Precio          smallmoney,
Cantidad        tinyint   default 1,
PRIMARY KEY (codigo)
);
Go

/* Insertar registros */
INSERT INTO medicamentos (nombre,quimico,precio,cantidad)
        VALUES(Ampicilina 500 mg, antiaminico, 20, 80);

INSERT INTO medicamentos (nombre,quimico,precio,cantidad)
        VALUES(Ampicilina 500 mg, DEFAULT, 20, DEFAULT);
```

EJEMPLO 166

Ahora muestre usted todas las columnas del medicamento con la descripción química, empleando la concatenación de datos

```
SELECT Nombre+", "+quimico FROM medicamentos
```

EJEMPLO 167

Ordene usted todos los registros de la tabla en orden descendente

```
SELECT * FROM medicamentos ORDER BY nombre DESC
```

EJEMPLO 168

Liste usted todas las columnas cuyos nombres químicos y códigos estén en forma concatenada en la tabla medicamentos

EJEMPLO 169

Liste usted todas las columnas cuyos registros muestren las fechas en orden (día-mes-año) en la tabla farmacia

EJEMPLO 170

En el ejemplo de la tabla (164), diga usted como listar todos los registros que muestren el nombre del mes.

```
SELECT codigo,ubicación, dateNombres(month,fecharegistro)  Mes;
Go
```

EJEMPLO 171

En el ejemplo de la tabla (164), diga usted como listar todos los registros que muestren el nombre del día, y la hora de la fecha de registro

```
SELECT codigo,ubicación, dateNombres(day,fecharegistro)  DIA;
Go
```

EJEMPLO 172

En el ejemplo de la tabla (164), diga usted como listar todos los registros que muestren la hora de la fecha de registro

```
SELECT codigo,ubicación, dateNombres(hour,fecharegistro)  HORA;
Go
```

EJEMPLO 173

En el ejemplo de la tabla (164), diga usted como listar todos los registros que ingresaron en el mes de octubre

SELECT * FROM Farmacia WHERE
dateNombres(month,fecharegistro)=¨october¨ORDER BY codigo;
Go

EJEMPLO 174

En el ejemplo de la tabla (164), diga usted como eliminar todos los registros cuyo código sea 1000

EJEMPLO 175

En el ejemplo de la tabla (164), diga usted como eliminar todos los registros cuyo código sea 1000 y 1200

EJEMPLO 176

Tomaremos como ejemplo de la tabla (164), ahora listaremos todos los registros cuya fecha esté en el rango del 12 de septiembre del 2010 hasta el 25 de diciembre del 2010.

SELECT * FROM farmacia WHERE fecharegistro BETWEEN ¨2010-09-12¨and ¨2010-12-25¨;

EJEMPLO 177

Consideremos que en una tabla MOVIMIENTOS, tenemos una columna llamada cantidad, el cual deseamos saber el numero de registros que contengan la cantidad 4 y 6.

SELECT * FROM MOVIMIENTOS WHERE cantidad BETWEEN 4 and 6

EJEMPLO 178

Consideremos que en una tabla MOVIMIENTOS, tenemos una columna llamada Estadocivil, el cual deseamos saber el numero de registros que contengan la Viudo y Soltero

SELECT * FROM MOVIMIENTOS WHERE Estadocivil IN (¨Viudo¨,¨Soltero¨)
¨**EJEMPLO 179**

Seleccionar todos los registros cuyo número se encuentre entre 02 y 06 empleando el operador BETWEEN y luego el operador IN

SELECT * FROM MOVIMIENTOS WHERE numero IN (2,3,4,5)
SELECT * FROM MOVIMIENTOS WHERE numero BETWEEN 2 and 6
EJEMPLO 180

Supongamos que tenemos como ejemplo una tabla de empelados, donde me piden efectuar las siguientes columnas:

- **Mostrar todos los nombres de los Registros que sean similares a "MARILUISA HARUMI"**
- **Mostrar todos los Registros cuya dirección comience con la letra A y tenga entre la cadena un" 6"**
- **Listar todos los registros cuyo nombre termine en 6,8,0,4 ó 2**

  - SELECT * FROM Registros WHERE nombre LIKE "%MARILUISA HARUMI% "
  - SELECT * FROM Registros WHERE direccion LIKE "A%6%"

  - SELECT * FROM Registros WHERE nombre LIKE "%[02468]"

EJEMPLO 181

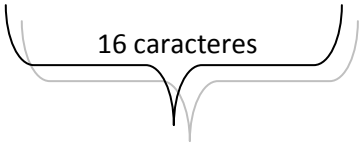
Ahora seleccione usted todas las columnas de la tabla cuyo nombre no comiencen en "CE" y cuya columna direccion termine en "se"

SELECT * FROM Registros WHERE nombre LIKE "[^CE]%" and direccion LIKE "%se"

EJEMPLO 182

Ahora muestre todas las columnas de la tabla empleado cuya direccion comience en “c” ó “e” y tengan 16 caracteres

SELECT * FROM Registros WHERE direccion LIKE “[ce]%_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _”



EJEMPLO 183

Listar todas las columnas de la tabla empleado cuya columna Gastos incluya centavos y reportar todos aquellos cuyo año de la fecha sea 2010

SELECT * FROM empleado WHERE Gastos NOT LIKE “%.00” and fecha LIKE “%2010”

EJEMPLO 184

Listar todas los registros de la tabla empleado usando la función count_big(), luego contar la cantidad de direcciones diferentes o distintas, asimismo luego contar la cantidad de la tabla empleado que tiene el Gastos distinto a NULL

SELECT COUNT_BIG (*)
SELECT COUNT_BIG(DISTINCT direccion) FROM empleado
SELECT COUNT_BIG (Gastos) AS MENSUAL FROM empleado

EJEMPLO 185

Listar todas las columnas de la tabla empleado con una condición que muestre a la fecha el numero de días hasta hoy, con relación a la fecha de ingreso

SELECT nombres,numerodias = DATOS: == (day,fecha,getdate()) FROM empleado

NOTA

Recordemos que si deseamos calcular la longitud de una cadena también podemos emplear la función LEN

EJEMPLO 186

Consideremos como ejemplo y añadir a la columna fecha 80 días adicionales

SELECT nombre, Actualmente = DATEADD(day,80,fecha) FROM empleado

EJEMPLO 187

Ahora Considerando que trabajamos con las funciones de fecha, queremos devolver el número y el nombre del mes de la fecha del sistema

SELECT DATEPART(month,GETDATE())
SELECT DATENOMBRES (Month,GETDATE())
_ _ **ahora calcular el día, mes y año de la fecha del sistema**
SELECT DAY(GETDATE( ))
SELECT MONTH(GETDATE( ))
SELECT YEAR(GETDATE( ))

TABLA DE VALORES DE FECHAS

Función	Propósito
SYSDATE	Devuelve la fecha del sistema
ADD_MONTHS (fecha, n)	Devuelve la fecha incrementada en n meses
LAS_DAY (fecha)	Devuelve la fecha del último día del mes que contiene fecha
MONTHS_BETWEEN (fecha1, fecha2)	Devuelve la diferencia en meses entre la fecha1 y la fecha2
NEXT_DAY (fecha, cad)	Devuelve la fecha del primer día de la semana indicado por cad después de la fecha indicada por fecha. Cad será siempre un día de la semana escrito con letras, por ejemplo Monday.

EJEMPLO 188

Supongamos que tenemos una tabla llamada MOVIMIENTOS, donde exista una columna llamada fechaingreso y deseamos sumar 03 meses a la fecha SELECT fechaingreso, add_months(fechaingreso,3) as fechanueva from MOVIMIENTOS;

EJEMPLO 189

Deseamos obtener el último dia del mes para cada una de las fechas de la tabla MOVIMIENTOS, de cada cliente

```
Select fechaingreso, last_day(fechaingreso) from MOVIMIENTOS;
```

EJEMPLO 190

Ahora diga usted, del ejemplo en mención como obtener el día que sera el proximo domingo

```
Select next_day(sysdate, 'Sunday') "siguiente domingo" from MOVIMIENTOS
```

EJEMPLOS ADICIONALES

Listar el total de articulos vendidos en la tabla MOVIMIENTOS

```
SELECT COUNT(*) FROM MOVIMIENTOS
```

Listar cuantos articulos se han vendido sin mostrar los articulos duplicados

```
SELECT DISTINCT idcodigo FROM MOVIMIENTOS ORDER BY idcodigo
```

Ahora mostrar el total

```
SELECT COUNT(DISTINCT idcodigo) FROM MOVIMIENTOS
```

Ahora listaremos todas las columnas de la tabla clientes

```
SELECT CODIGO = cod_cli,
        “descripción del Articulo” = desc_bien,
        “Precio de venta” = pventa,
        Stock = stock_bien
        FROM CLIENTES;
```

También puede ser:

```
SELECT cod_cli CODIGO, desc_bien DESCRIPCION, pventa ¨PRECIO VENTA¨, stock_bien STOCK   FROM Master.dbo.CLIENTES;
```

MEMORIA AYUDA

Una sub consulta es una sentencia SELECT que es incrustada en una cláusula de otra sentencia SQL, llamada sentencia padre, asimismo podemos decir que una sub consulta (consulta interna) obtiene un valor que es usado por la sentencia padre. Usar una sub consulta anidada es equivalente a ejecutar dos Consultas secuenciales y utilizar el resultado de la consulta interna como valor de búsqueda en la consulta externa (consulta principal).

Las sub consultas pueden ser usadas para los siguientes propósitos:
• Proveer valores para condiciones en cláusulas WHERE, HAVING y START WITH de sentencias SELECT

Se pueden construir sentencias poderosas utilizando sub consultas. Las sub consultas pueden ser muy útiles cuando necesites seleccionar filas de una tabla con una condición que dependa de los datos de la misma u otra tabla.
Las sub consultas son muy útiles para escribir sentencias SQL que necesiten valores de un o más valores condicionales desconocidos.
Donde:
operator incluye un operador de comparación como >, = o IN

Nota: los operadores de comparación se encuentran en dos clases:
Operadores de fila única (>, =, >=, <, <>, <=) y operadores de múltiples filas (IN, ANY, ALL)

AGRUPAMIENTO DE DATOS, CONSULTAS, SUB CONSULTAS Y REGLAS

GROUP BY

Su mismo nombre lo especifica, nos sirve para agrupar datos, considerando que esta claúsula lo emplearemos con los operadores CUBE, ROLLUP y otros en relación.

SELECT [**ALL | DISTINCT**]

<nombre_campo> [{,<nombre_campo>}]
FROM <nombre_tabla>|<nombre_vista>
[{,<nombre_tabla>|<nombre_vista>}]
[**WHERE** <condicion> [{ **AND**|**OR** <condicion>}]]
[**GROUP BY** <nombre_campo> [{,<nombre_campo >}]]
[**HAVING** <condicion>[{ **AND**|**OR** <condicion>}]]
[**ORDER BY** <nombre_campo>|<indice_campo> [**ASC** | **DESC**]
[{,<nombre_campo>|<indice_campo> [**ASC** | **DESC**]}]]

GROUP BY	Especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.
HAVING	Especifica una condición que debe cumplirse para los datosEspecifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de **WHERE** pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a **GROUP BY** y la condicion debe estar referida a los campos contenidos en ella.
ORDER BY	Presenta el resultado ordenado por las columnas indicadas. El orden puede expresarse con **ASC** (orden ascendente) y **DESC** (orden descendente). El valor predeterminado es **ASC.**

EJEMPLOS:

Listar todos los registros de nombres de clientes, mostrando la cantidad de pedidos por cada cliente.

SELECT codigo,nombres,count(codigo) AS Pedidos FROM MOVIMIENTOS GROUP BY codigo

Listar todos los registros de la tabla MOVIMIENTOS agrupados por codigo y ordenados por el total de MOVIMIENTOS

SELECT codigo,nombres, pventa, cantidad, (pventa*cantidad) As Total FROM MOVIMIENTOS GROUP BY codigo ORDER BY Total
Go

GROUP BY con el operador ROLLUP

Permite resumir los valores de grupo. El operador ROLLUP se usa normalmente para producir promedio y sumas acumuladas.

SELECT codbien,deta_bien AS Detalle, AVG(Pventa) FROM Articulos GROUP BY codbien, deta_bien WITH ROLLUP

COMPUTE..BY

Agrupa las filas en base a una o mas columnas lo mas importante es que permite mostrar información mas detallada por cada grupo.
Sintaxis:

SELECT columna1, columna2,…. FROM table WHERE <condicion> ORDER BY columnax, columnay ….. COMPUTE función <Columna Z> BY <columnax>,<columnay>

Ejemplos:

Listar todas las columnas de la tabla MOVIMIENTOS luego calcular el promedio del precio de venta y lo muestra por cada grupo en este caso por grupos de codigo y nombre y en cada uno su promedio de precio venta.

SELECT codigo,nombre_bien AS Producto, pventa FROM MOVIMIENTOS ORDER BY codigo,nombre COMPUTE AVG(pventa) BY codigo,nombre

En este caso vamos a listar todas las columnas de la tabla Almacen ordenados por codigo, donde me va crear grupos por codigo calculando los precios promedios, maximo y minimo por cada grupo.

SELECT idcodigo, deta_bien AS Descripción, Umedida,precio,cant, total FROM Almacen ORDER BY idcodigo COMPUTE AVG(precio), MAX(precio),MIN(PRECIO) BY idcodigo

Empleando ALIAS como refrencias en las tabla ahora vamos a listar las columnas de las tabla MOVIMIENTOS

SELECT DISTINCT MN.codbien,PQ.detabien FROM venta MN INNER JOIN almacen PQ ON MN.codbien=PQ.codbien ORDER BY MN.codbien

Empleando ALIAS como refrencias en las tabla ahora vamos a listar todos los productos agrupados por codigo considerando que el numero de factura sea 40.

SELECT MN.codbien,PQ.detabien FROM venta MN INNER JOIN almacen PQ ON MN.codbien = PQ.codbien GROUP BY MN.codbien, MN.fecha HAVING MN.numfactura = 40

Considerando que tenemos una tabla clientes ya creada, ahora vamos a listar el primer y el último nombre del regsitro de la tabla cientes

SELECT MIN(nombre_cli),MAX(nombre_cli) FROM cliente

Ahora diga usted, como contar la cantidad de regsitros de la tabla clientes

SELECT COUNT(*) FROM cliente GROUP BY codbien

CONSULTAS Y AGRUPAMIENTOS DE DATOS

Una de las funciones mas comunes dentro de las bases de datos es la agrupación de datos, obteniendo reportes consolidados el cual es necesario trabajar con la instrucción SELECT y la cláusula GROUP BY

FUNCIONES

AVG	Retorna el promedio de los valores
COUNT	Retorna la cuenta de las columnas en la consulta
MAX	Retorna el valor maximo
MIN	Retorna el minimo valor
SUM	Retorna la suma de los valores

EJEMPLO 191

Supongamos que en la tabla personal, deseamos calcular el total de los Gastoss agrupados por sexo

```
SELECT sexo,SUM(Gastos) FROM personal GROUP BY sexo
```

EJEMPLO 192

Se necesita saber el maximo y minimo valor de los Gastos agrupados por sexo y ciudad

```
SELECT sexo, ciudad, MAX(Gastos) AS MAXIMO , MIN(Gastos) AS MINIMO FROM
personal GROUP BY sexo, ciudad
```

EJEMPLO 193

Ahora calcular el promedio del valor de los Gastoss agrupados por ciudad

```
SELECT  ciudad, AVG(Gastos) AS promedio FROM personal GROUP BY ciudad
```

EJEMPLO 194

Contar y agrupar por ciudad todos los registros sin considerar los que no tienen telefono

```
SELECT  ciudad, COUNT(*) AS CANTIDAD FROM personal WHERE telefono IS NOT
NULL AND telefono <>¨no tiene¨GROUP BY ciudad
```

EJEMPLO 195

Obtener el total de los registros agrupados por ciudad y sexo

```
SELECT  ciudad, sexo, COUNT(*) AS CANTIDAD FROM personal GROUP BY
        ciudad, sexo
```

EJEMPLO 196

Obtener el total de los registros agrupados por ciudad y sexo sin considerar los que tienen menos de dos registros en la misma ciudad

```
SELECT  ciudad, sexo,COUNT(*) AS CANTIDAD FROM personal GROUP BY
        ciudad, sexo HAVING count(*)>1
```

EJEMPLO 197

Obtener el total de los registros de la tabla personal que viven en la calle (LOS NARANJOS) agrupados por ciudad, teniendo en cuenta los registros de aquellas ciudades que tengan menos de dos registros o personas y omitiendo la fila correspondiente a la Ciudad de LIMA

```
SELECT ciudad, domicilio, count(*) FROM personal WHERE domicilio LIKE
        "%LOS NARANJOS"GROUP BY all ciudad HAVING count(*)<2  AND
        ciudad <>¨LIMA¨
```

EJEMPLO 198

Calcular la cantidad de registros por ciudad y la cantidad total de personas en una sola columna

```
SELECT  ciudad, COUNT(*) AS CANTIDAD FROM personal GROUP BY
        Ciudad WITH ROLLUP
```

EJEMPLO 199

Reportar la cantidad de registros agrupados por sexo y ciudad incluyendo resultados parciales

```
SELECT  ciudad, COUNT(*) AS CANTIDAD FROM personal GROUP BY sexo, ciudad
        WITH ROLLUP
```

NOTA

Cuando usamos la clausula GROUP BY con el operador ROLLUP para resumir los valores de grupo recordemos que ROLLUP se utiliza para producir promedios acumulados empleando la función GROUP BY de izquierda a derecha.

EJEMPLO 200

En una Historia Clínica se desea crear un control de pacientes atendidos considerando que si son por primera vez o ya han sido atendidos (P=primera vez y R= Atención regular o continua)

```
CREATE TABLE clinica(
Numero          int             IDENTITY,
Pago            decimal         (10,2),
Tipopaciente    char(1), - - p=primera vez, R=regular
Nombre          Varchar(40)     NOT NULL,
Fecharegistro   datetime,
Ciudad          Varchar(20),
Sexo            Varchar(1), - - M=Masculino, f=Femenino
Provincia       Varchar(20),
PRIMARY KEY (numero)
)
```

- - Después de crear la tabla pues agruparemos por tipo de paciente y nombre, asimismo la cantidad.

```
SELECT nombre,tipopaciente,count(*) AS cantidad FROM clinica GROUP BY
nombre,tipopaciente WITH ROLLUP
```

EJEMPLO 201

Agrupar por tipo de paciente y nombre, asimismo cuente la cantidad de atenciones empleando CUBE

```
SELECT nombre,tipopaciente, count(*) AS cantidad FROM clinica GROUP BY nombre,
tipopaciente WIITH CUBE
```

EJEMPLO 202

Contar la cantidad de atenciones agrupados por sexo y tipos de paciente.

```
SELECT sexo, tipopaciente, count(*) AS Cantidad FROM clinica GROUP BY
sexo,tipopaciente WITH ROLLUP
```

EJEMPLO 203

Realizar la misma consulta del ejemplo anterior pero ahora empleando la función GROUPING para los datos.

```
SELECT sexo, tipopaciente, count(*) AS Cantidad, grouping(sexo)
        AS ¨Resumen por tipo de sexo¨, GROUPING(tipopaciente) AS ¨Resumen por
        tipo de paciente¨FROM clinica GROUP BY sexo, tipopaciente WITH ROLLUP
```

EJEMPLO 204

Ahora hagamos la misma consulta anterior pero usando CUBE

```
SELECT sexo, tipopaciente, count(*) AS Cantidad, grouping(sexo)
        AS ¨Resumen por tipo de sexo¨, GROUPING(tipopaciente) AS ¨Resumen por
        tipo de paciente¨FROM clinica GROUP BY sexo, tipopaciente WITH CUBE
```

EJEMPLO 205

Diga usted como reportar los nombres de los pacientes sin repetir las anteriores

```
SELECT DISTINCT nombre FROM clinica
```

EJEMPLO 206

Ahora cuente todos los pacientes atendidos sin repetir si fueron por primera vez o atención regular

```
SELECT COUNT(DISTINCT nombre) AS  Cantidad FROM clinica
```

EJEMPLO 207

Listar todos los nombres de los socios sin repetir

```
SELECT DISTINCT nombre FROM clinica
```

EJEMPLO 208

Listar todas las cantidades de los nombres

```
SELECT COUNT(DISTINCT nombre) FROM clinica
```

EJEMPLO 209

Combinar con WHERE para obtener los distintos nombres de los pacientes atendidos con el tipo de atención “R” Regular o continua

```
SELECT DISTINCT nombre FROM clinica WHERE tipopaciente=¨R¨
```

EJEMPLO 210

Diga usted como contar los diferentes pacientes por cada empleado, usando GROUP BY

```
SELECT DISTINCT nombre FROM clinica WHERE tipopaciente=¨R¨
```

EJEMPLO 211
Listar de la tabla clinica los 10 primeros registros
SELECT TOP 10 FROM clinica

EJEMPLO 212
Mostrar los 06 primeros registros ordenados por nombres
SELECT TOP 6 nombre,sexo FROM clinica ORDER BY nombre

EJEMPLO 213
Desarrollando la misma operación incluya todos los registros que tengan el mismo tipo de paciente
SELECT TOP 6 WITH TIES nombre,sexo FROM clinica ORDER BY tipopaciente

EJEMPLO 214
Muestre los nombres, sexo y tipo de paciente de los 06 primeros pacientes ordenados por sexo y por nombres
SELECT TOP 6 nombre,sexo FROM clinica ORDER BY sexo,nombre

EJEMPLO 215
Realice la misma operación anterior pero ahora incluya todos los valores iguales al ultimo registro
SELECT TOP 6 WITH TIES nombre, sexo, tipopaciente FROM clinica ORDER BY sexo, nombre

EJEMPLO 216
Vamos a definir que tenemos una tabla llamada Registro y cuya columna CIUDAD presenta por defecto un valor el cual es "UCAYALI" por defecto en caso de no ingresar valores para dicho campo.

ALTER TABLE Registro
ADD CONSTRAINT DF_Regsistro_ciudad
DEFAULT Ücayali¨FOR ciudad

EJEMPLO 217
Ahora vamos a suponer que tenemos una determinada tabla llamada ESTUDIO el cual vamos a definir una restricción DEFAULT para la columna (TOTAL) que almacena el valor cero (0) en caso de no ingresar un valor

ALTER TABLE Estudio
ADD CONSTRAINT DF_Estudio_total
DEFAULT 0
FOR TOTAL

EJEMPLO 218
Diga usted como visualizar las restricciones creadas en una tabla
SP_HELPCONSTRAINT Estudio

EJEMPLO 219
Supongamos que tenemos una tabla llamada Resultados y deseamos agregar una restricción CHECK en la columna GASTOS, para asegurarnos que no se ingresen valores negativos
ALTER TABLE Resultados
ADD CONSTRAINT CHECK_Resultados_Gastos CHECK(GASTOS>0);

EJEMPLO 220
Ahora agregue otra restricción CHECK al campo Gastos, con la condición que el Gastos maximo sea 2000
ALTER TABLE Resultados
ADD CONSTRAINT CHECK_Resultados_Gastosmaximo CHECK(GASTOS<=2000);

EJEMPLO 221
Ahora diga usted, como eliminaria un registro infractor para luego volver a crear la restricción
ALTER TABLE Resultados
DELETE FROM Resultados WHERE Gastos=3000
ALTER TABLE Resultados
ADD CONSTRAINT CHECK_Resultados_Gastosmaximo CHECK(GASTOS<=2000);

EJEMPLO 222
Ahora efectuemos listar el promedio de la cantidad total de registros de la tabla Resultados.
SELECT AVG(Cantidad) FROM Resukltados
Go

NOTA

Distinct indica que debe de conraerse los valores duplicados de la expresión numerica.

EJEMPLO 223

Reportar el total de la cantidad de los registros de la tabla ejemplo Resultados

```
SELECT COUNT(DISTINCT numero) FROM Resultados
Go
```

EJEMPLO 224

Diga usted como deshabilitar una restricción NO CHECK creada, supongamos que la tabla se llama Resultados.

```
ALTER TABLE Resultados
NOCHECK CONSTRAINT CHK_Resultado_Gastos
```

EJEMPLO 225

Supongamos que deseamos mostrar el codigo, el nombre y el apellido de una determinada tabla

```
Select CodigocliID as 'Codigo' , FirstNombres +' '+ LastNombres as 'Nombre y Apellido'
from Codigoclis
```

Ahora mostrar el precio de venta mas el 10%

```
Select Detalle_bien 'Nombre del Producto' , Pventa as 'Precio Unitario' ,
Pventa*0.1+Pventa 'Aumento en 10%' from Articulos
```

Ahora mostrar en forma ascendente, en un ejemplo de de consulta

```
Select Empresa as 'Nombre de la Compañia', Pais _empresa as 'Pais' from
ClientesOrder By pais_empresa ASC
```

Ahora mostrar en forma descendente, en un ejemplo de de consulta– Se desea Visualizar

```
Select Empresa as 'Nombre de la Compañia', Pais _empresa as 'Pais' from
ClientesOrder By pais_empresa DESC
```

NOTA

Recordemos que el operador CUBE devuelve las mismas filas que ROLLUP pero a diferencia es que agrupa de acuerdo a la segunda columna de GROUP BY sin considerar los valores de la primera columna.

```
SELECT codigo,nombre, sum(total), AVG(precio) FROM Venta GROUP BY codigo,nombre WITH CUBE

/*En este caso lo agrupa por codigo y nombre pero lo clasifica CUBE solo por nombre*/
```

EJEMPLO 226

Mostrar todo los productos q pertenezcan a la categoria 1,2 y 3 y que esten descontinuados = 1 vigentes 0

```
select * from products where (categoryid=1 or categoryid=2 or categoryid=3)
and discontinued=1

select * from products where Categoryid IN(1,3,5) and discontinued=1
```

Ahora mostrar todo los pedidos que realizo el cliente durante el primer trimestre del año 2010 y la forma de envio sea 2

```
select*from orders where (year(orderdate)=2010 and month(orderdate)<4)
and Clienteid='bergs' and envio=2

select*from orders where orderdate BETWEEN '01/01/2010' AND '31/12/2010'
and envio=2
```

REPASO

EJEMPLO

Vamos a considerar que vamos a diseñar una tala llamada Empleados el cual agregaremos restricciones CHECK dentro de la estructura

```
        IF OBJECT_ID("Empleados") IS NOT NULL
                DROP Tale Empleados;
-- Ahora vamos a crear la table con una nuea estructura
        CREATE TABLE Empleados(
        Tipodocm        Varchar(40),
        Nombres         Varchar(40),
        Ubicacion       Varchar(20),
        Pagomes         Decimal(10,2)
        );
```

```
--  Insertar Valores
        INSERT INTO Empleados
                VALUES(“Boleta”,”Cesitar”,”Lima”,500,00.00);
--Ahora agregamos las restricciones CHECK para asegurarse que no se deen ingresar
valores negatives
USE Empleados
ALTER TABLE Empleados
        Add Constraint CHK_Empleados_seldo CHECK (Sueldo>=0);

-- Ahora intentemos ingresar una opción WITH NOCHECK
        Add Constraint CHK_Empleados_sueldomas CHECK (Sueldo>=0);
        Alter table Empleados WITH NOCHECK
```

EJEMPLO

Deshabilite la restricción del ejemplo anterior

```
ALTER TABLE Empleados
        NOCHECK Constraint CHK_Empleados_sueldomas;
```

EJEMPLO

Estalezca una restricción CHECK en la columna “Uicacion” ya que solo se pueden insertar estos alores por defecto(I CiclomII Ciclom III Ciclo)

```
ALTER TABLE Empleados
        Add Constraint CHKEmpleadoubicate CHECK  (Ubicacion IN(“I CICLO”,”II
        CICLO”,”III CICLO”);
```

EJEMPLO

Ahora visualicemos las restricciones de la tabla Empleadossi es que están o no habilitados

```
SP_HELPCONSTRAINT Empleados;
```

Nuevas caracteristicas de grouping sets en sql server 2008

Una característica piola que tiene la nueva versión de SQL Server, es el nuevo operador GROUPING SETS, que ya existía en otros motores como Oracle hace tiempo (es un estándar ANSI 2006) y permite combinar consultas de agrupación distintas en una sola consulta. El operador GROUPING SETS es una extensión de la cláusula estándar GROUP BY.

Cuando no se requieren todas las agrupaciones posibles que se generan utilizando un operador ROLLUP o CUBE (que ya existían en SQL Server 2005), se debe utilizar GROUPING SETS para especificar sólo las agrupaciones que se deseen. O sea, gracias a GROUPING SETS obtenemos los niveles de agrupación deseados y además nos devuelve el subtotal para cada subconjunto de agrupación.
Podríamos decir que GROUPING SET es mas abarcativo y genérico que ROLLUP o CUBE. Generalmente consultas que usen este tipo de operadores están relacionadas con el análisis de datos, reporting y todo lo relacionado con el mundo BI. Este nuevo operador no permite hacer nada nuevo que antes no se pudiese, solo simplifica y optimiza determinadas consultas, lo ya de por si, es un factor importante. Veamos un ejemplo de donde utilizarlo:
Supongamos que tenemos la siguiente tabla de Clientes con datos ya cargados:

```
CREATE TABLE [dbo].[Client]
(
[ID] [int] IDENTITY(1,1) NOT NULL,
[Nombres] [varchar](50) NULL,
[Pais] [varchar](50) NULL,
[pais] [varchar](50) NULL,
[año] [int] NULL,
[Precio] [numeric](12, 4) NULL,
)
```

Y ahora supongamos que queremos traer la suma del campo Precio, agrupada por los Campos Pais y Lugar, luego solo por Lugar y luego por el total y todo en una sola misma sentencia de tal manera que el resultado de la consulta fuese el siguiente:

Total	Lugar	pais
3,400.00	Milian	colombia
3,466.00	Las Brisas	España
678.00	Coronel Portillo	Peru
7,899.00	Huanuco	Peru
1,244.00	Paucarbamba	Peru

La primera opción que uno piensa, es hacer una consulta usando UNION ALL con tres consultas diferentes, una por cada agrupación:

```
SELECT SUM(Value) AS Total, Lugar, Pais FROM dbo.Client
GROUP BY Lugar,Pais

UNION ALL
SELECT SUM(Value), NULL, Pais FROM dbo.Client
GROUP BY Pais

UNION ALL
SELECT SUM(Value), NULL, NULL FROM dbo.Client
ORDER BY Lugar, Pais
```

Sin embargo, podemos escribir una consulta equivalente y acá entra GROUPING SETS de SQL Server 2008 para simplificarnos la vida:

```
SELECT SUM(value) AS Total, Lugar, Pais FROM dbo.Client
GROUP BY
GROUPING SETS
(
(Lugar, Pais),
(Pais),
()
)
ORDER BY GROUPING(Lugar), GROUPING(Pais)
```

Ahora como podrán ver en este último ejemplo, además de usar el operador GROUPING SETS, hago uso de la función GROUPING en el ORDER BY, para ordenar el resultado por Zona y Pais.

Esta función (que ya existía en SQL Server 2005), nos indica si una expresión de columna especificada en una lista GROUP BY es agregada o no. GROUPING devuelve 1 para agregado y 0 para no agregado, en el conjunto de resultados.

GROUPING SETS in SQL Server 2008

Con el nuevo grupo operador fija incluidos en SQL Server 2008, usted tiene máscontrol sobre lo que se agrega. Aquí está un ejemplo práctico del uso de GROUPING SETS de SQL Server 2008With the new GROUPING SETS operator included in SQL Server 2008, you have more control over what is aggregated. Here's a practical example of using GROUPING SETS in SQL Server 2008

```
--DROP TABLE #Alumnos
CREATE TABLE #Alumnos
(
Codalumno int ,
Cursoaño smallint,
Semestre smallint,
Promedio float
)
INSERT INTO #Alumnos VALUES (1, 2008, 1, 5.6)
INSERT INTO #Alumnos VALUES (1, 2008, 2, 6.5)
INSERT INTO #Alumnos VALUES (1, 2008, 3, 8.9)
INSERT INTO #Alumnos VALUES (1, 2008, 4, 9.1)
INSERT INTO #Alumnos VALUES (1, 2009, 1, 4.4)
INSERT INTO #Alumnos VALUES (1, 2009, 2, 7.9)
INSERT INTO #Alumnos VALUES (1, 2009, 3, 8.5)
INSERT INTO #Alumnos VALUES (1, 2009, 4, 8.7)
INSERT INTO #Alumnos VALUES (2, 2008, 1, 5.4)
INSERT INTO #Alumnos VALUES (2, 2008, 2, 9.9)
INSERT INTO #Alumnos VALUES (2, 2008, 3, 8.5)
INSERT INTO #Alumnos VALUES (2, 2008, 4, 4.7)
INSERT INTO #Alumnos VALUES (2, 2009, 1, 6.4)
INSERT INTO #Alumnos VALUES (2, 2009, 2, 7.9)
INSERT INTO #Alumnos VALUES (2, 2009, 3, 7.4)
INSERT INTO #Alumnos VALUES (2, 2009, 4, 9.7)
```

Vamos a efectuar algunas operaciones

Calcular el total de puntuaciones obtenidas por los estudiantes cada año

```
SELECT Codalumno, Cursoaño, SUM(Promedio) AS TotalPromedio
FROM #Alumnos
GROUP BY Codalumno, Cursoaño
ORDER BY Codalumno
```

	codalumno	cursoaño	Totalpromedio
1	1	2010	24.6
2	1	2010	15.9
3	2	2008	14.6
4	2	2008	32.3

Calcular el total de puntuaciones obtenidas por los estudiantes agrupados por cada semestre y para cada año

```
SELECT Codalumno,semestre, SUM(Promedio) AS TotalPromedio
FROM #Alumnos
GROUP BY Codalumno,semestre
ORDER BY Codalumno
```

Sin embargo, con el operador GROUPING SETS, se pueden definir diferentes gruposde agregado en una sola consulta, como se muestra a continuación

```
SELECT Codalumno, Cursoaño,semestre, SUM(Promedio) AS TotPromedio
FROM #Alumnos
GROUP BY GROUPING SETS((Codalumno, Cursoaño), (Codalumno,semestre))
ORDER BY Codalumno,semestre
```

	codalumno	Semestre	Totalpromedio
1	1	1	10
2	1	2	14.4
3	1	3	17.4
4	1	4	17.8
5	2	1	11.8
6	2	2	17.8
7	2	3	15.9
8	2	4	14.4

UNION ALL

Si en SQL Server 2005, estabas cansado de usar escribir, escribir y escribir UNION ALL para realizar varias operaciones de inserción consecutivas o en bloques pues esto en SQL Server 2008 de alguna manera se ha solucionado con la implementación de una nueva característica llamada Constructores de Fila. Ahora podemos usar esta nueva funcionalidad y ahorrarnos el trabajo es esta escribiendo cientos de UNION ALL. Me explico con un ejemplo bastante práctico.

Primero: Crear una tabla.

```
CREATE TABLE [Trabajo].[Libro](
   [LibroID] [int] IDENTITY(1,1) NOT NULL,
   [Titulo] [nvarchar](50) COLLATE Latin1_General_CS_AS NOT NULL,
   [Ubicacion] [nvarchar](400) COLLATE Latin1_General_CS_AS NOT NULL,
   [Nombre_conocido] [nvarchar](8) COLLATE Latin1_General_CS_AS NOT NULL,
   [Numguia] [nchar](5) COLLATE Latin1_General_CS_AS NOT NULL,
   [Nuevo_numero] [int] NOT NULL CONSTRAINT [DF_Libro_Nuevo_numero]
DEFAULT ((0)),
   [Status] [tinyint] NOT NULL,
   [LibroSummary] [nvarchar](max) COLLATE Latin1_General_CS_AS NULL,

   [ModifiedDate] [datetime] NOT NULL CONSTRAINT [DF_Libro_ModifiedDate]
DEFAULT (getdate()),
 CONSTRAINT [PK_Libro_LibroID] PRIMARY KEY CLUSTERED
(
   [LibroID] ASC
)
) ON [PRIMARY]
```

Siguiente paso: Insertar Datos. , podemos hacerlo de dos maneras.

La más trabajoso es esta:

```
INSERT INTO
[Trabajo].[Libro]([Titulo],[Ubicacion],[Nombre_conocido],[Numguia],[Nuevo_numero],[Status],[LibroSummary])
VALUES ('Installing Replacement Pedals','C:\Libros\Installing Replacement Pedals.doc','.doc','0',32,2,'Detailed instructions ...')

INSERT INTO
[Trabajo].[Libro]([Titulo],[Ubicacion],[Nombre_conocido],[Numguia],[Nuevo_numero],[Status],[LibroSummary])
VALUES ('Introduction 1','C:\Libros\Introduction 1.doc','.doc','4',28,2,NULL)

INSERT INTO
[Trabajo].[Libro]([Titulo],[Ubicacion],[Nombre_conocido],[Numguia],[Nuevo_numero],[Status],[LibroSummary])
VALUES ('Lubrication Maintenance','C:\Libros\Lubrication Maintenance.doc','.doc','2',11,1,'Guidelines and recommendations...')

INSERT INTO
[Trabajo].[Libro]([Titulo],[Ubicacion],[Nombre_conocido],[Numguia],[Nuevo_numero],[Status],[LibroSummary])
VALUES ('Seat Assembly','C:\Libros\Seat Assembly.doc','.doc','8',55,2,'Worn or damaged seats...')
```

Este problema (en SQL Server 2005) se soluciona usando UNION ALL, asi:

```
INSERT INTO
[Trabajo].[Libro]([Titulo],[Ubicacion],[Nombre_conocido],[Numguia],[Nuevo_numero],[Status],[LibroSummary])
SELECT 'Installing Replacement Pedals','C:\Libros\Installing Replacement Pedals.doc','.doc','0',32,2,'Detailed instructions ...'
UNION ALL
SELECT 'Introduction 1','C:\Libros\Introduction 1.doc','.doc','4',28,2,NULL
UNION ALL
SELECT 'Lubrication Maintenance','C:\Libros\Lubrication Maintenance.doc','.doc','2',11,1,'Guidelines and recommendations...'
UNION ALL
SELECT 'Seat Assembly','C:\Libros\Seat Assembly.doc','.doc','8',55,2,'Worn or damaged seats...'
```

Para SQL Server 2008, usando Constructores de Fila, podemos hacerlo asi:

```
INSERT INTO [Trabajo].[Libro](
    [Titulo]
    ,[Ubicacion]
    ,[Nombre_conocido]
    ,[Numguia]
    ,[Nuevo_numero]
    ,[Status]
    ,[LibroSummary]
)
VALUES ('Installing Replacement Pedals','C:\Libros\Installing Replacement Pedals.doc','.doc','0',32,2,
      'Detailed instructions ...'),
('Introduction 1','C:\Libros\Introduction 1.doc','.doc','4',28,2,NULL),
('Lubrication Maintenance','C:\Libros\Lubrication Maintenance.doc','.doc','2',11,1,
      'Guidelines and recommendations...'),
('Seat Assembly','C:\Libros\Seat Assembly.doc','.doc','8',55,2,'Worn or damaged seats...')
```

Fijate que no tenemos que usar UNION ALL, ahorrándonos (como ya dije, y vuelvo a repetir...) mucho trabajo de estar escribiendo cientos de UNION ALL en el caso de desear insertar masivamente. Está demás decir, que la performance mejorará ya que será inserción directa.

Kill "UNION ALL" para este tipo de operaciones!

Auditoría de permisos y Roles del servidor (SQL Server)

Dado que SQL Server implemento un nuevo modelo de seguridad y se pueden asignar permisos, hay varias vistas del catálogo que nos interesa para realizar una Auditoría.

sys.server_principals : Contiene una fila por cada entidad de seguridad del servidor.
sys.server_permissions : Devuelve una fila por cada permiso a nivel de servidor.
sys.server_role_members : Devuelve una fila por cada miembro de cada función fija de servidor.

Para obtener los miembros de la función de servidor vamos a combinarsys.server_principals sys.server_role_members :

```
SELECT SP1.[Nombres] AS ‘Login’, SP2.[Nombres] AS ‘ServerRole’
FROM sys.server_principals SP1
JOIN sys.server_role_members SRM
ON SP1.principal_id = SRM.member_principal_id
JOIN sys.server_principals SP2
ON SRM.role_principal_id = SP2.principal_id
ORDER BY SP1.[Nombres], SP2.[Nombres];
```

Para obtener los permisos, usaremos sys.server_principals y sys.server_permissions :

```
SELECT SP.[Nombres] AS ‘Login’ , SPerm.state_desc + ‘ ‘ +
SPerm.permission_Nombres AS‘ServerPermission’
FROM sys.server_principals SP
JOIN sys.server_permissions SPerm
ON SP.principal_id = SPerm.grantee_principal_id
ORDER BY [Login], [ServerPermission];
```

Tener en cuenta que si usted no está interesado en el permiso CONNECT SQL, puedes realizar un filtro utilizando una cláusula WHERE apropiada (SPerm.type = ‘COSQ’ Y SPerm.state = ‘G’) – que eliminará del conjunto de resultados. Por supuesto, podemos combinar los dos para generar una consulta para un único informe. Ya que sabemos que no habrá duplicaciones entre las dos instrucciones SELECT, podemos utilizar UNION ALL y una ligera modificación a la primera consulta (para indicar el ROL) para obtener los permisos para cada login.

```
SELECT SP1.[Nombres] AS ‘Login’, ‘Role: ‘ +
SP2.[Nombres] COLLATE DATABASE_DEFAULT AS‘ServerPermission’
FROM sys.server_principals SP1
JOIN sys.server_role_members SRM
ON SP1.principal_id = SRM.member_principal_id
JOIN sys.server_principals SP2
ON SRM.role_principal_id = SP2.principal_id
UNION ALL
SELECT SP.[Nombres] AS ‘Login’ , SPerm.state_desc + ‘ ‘ +
SPerm.permission_Nombres COLLATEDATABASE_DEFAULT AS ‘ServerPermission’ F
ROM sys.server_principals SP
JOIN sys.server_permissions SPerm
ON SP.principal_id = SPerm.grantee_principal_id
ORDER BY [Login], [ServerPermission];
```

Como saber si existe un metodo en un objeto – PEMSTATUS()

Emplearemos PEMSTATUS para saber si existe una propiedad :

* tcObject : Nombre del Objeto a evaluar
* tcMethod : Nombre de la propiedad a comprobar

```
tcObject = “Thisform”
tcMethod = “NombredelMetodoAVerificar”
 IF !PEMSTATUS(&tcObject,&tcMethod,5) THEN
     WAIT WINDOW “No existe Método” NOWAIT
ENDIF
```

EJEMPLO 227

Establezca una restricción CHECK en la columna “ubicacion” que solamente se pueden ingresar estos valores (I CICLO, II CICLO y III CICLO)

```
ALTER TABLE  Alumnos
ADD CONSTRAINT CK_ALUMNOS CHECK ubicación IN (“I CICLO”,”II CICLO”, “III CICLO”);
```

EJEMPLO 228

Visualizaremos la restricción de la tabla empleando :

```
SP_HELPCONSTARINT Alumnos
```

EJEMPLO 229

Ahora diga usted como establecriamos una restricción PRIMARY KEY en una determinada tabla , para que la columna TIPODOCUM no se repita ni admita valores nulos

```
ALTER TABLE USUARIOS
ADD CONSTRAINT PK_USUARIOS_TIPODOCUM
PRIMARY KEY (TIPODOCUM);
```

EJEMPLO 230

Ahora modificaremos un registro de la tabla anterior, cuyo tipo de documento es Boleta Militar por Libreta Militar

```
UPDATE Usuarios SET TIPODOCUM =”Libreta Militar” WHERE tipodocum=”Boleta Militar”;
```

EJEMPLO 231

Considerando que estamos estableciendo restricciones, pues ahora ejecutaremos un ejemplo de restricción por defecto que sea Partida de Nacimiento

```
ALTER TABLE Usuario
ADD CONSTRAINT DF_Usuarios_documento DEFAULT “Partida de Nacimiento”
FOR TIPODOCUM;
```

EJEMPLO 232

Ahora dentro de una tabla Control tenemos una columna denominada “Tipo_movilidad” pues debemos de crear una restricción DEFAULT

```
CREATE TABLE CONTROL(
Numdoc          Tinyint identity,
Detalle         char(10) not null,
Tipo_movilidad  char(1), _ _ “a” =auto, “m”= moto
Horallegada     datetime not null,
Horasalida      datetime
);
```

EJEMPLO 233

De la misma manera ahora procederemos a crear un ejemplo con una restricción CHECK que pueda almacenar una determinada columna primero 3 digitos numericos y luego 2 caracteres alfabeticos.

```
ALTER TABLE CONTROL
ADD CONSTRAINT CK_CONTROL_TIPOK CHECK (TIPOK LIKE “[0-9] 0-9][0-9][A-Z]
[A-Z] “);
```

EJEMPLO 234

Ahora diga usted como restablecer una restricción CHECK que admita los valores H y M para el campo o columna denominado UBICACIÓN

```
ALTER TABLE CONTROL
ADD CONSTRAINT CK_CONTROL_TIPO CHECK (UBICACION IN (¨H¨, ¨M¨);
```

COMBINACIONES

Existen 03 tipos de combinaciones y se pueden clasificar en :

- Combinaciones internas INNER JOIN
- Combinaciones externas OUTER JOIN
- Combinaciones cruzadas CROSS JOIN

Las vinculaciones entre tablas se realizan mediante la cláusula INNER que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es:

```
SELECT campos FROM tb1 INNER JOIN tb2 ON tb1.campo1 comp tb2.campo2
```

En donde:

tb1, tb2	Son los nombres de las tablas desde las que se combinan los registros.
campo1, campo2	Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.
Como	en cualquier operador de comparación relacional: =, <,<>, <=, =>, ó >.

Se puede utilizar una operación **INNER JOIN** en cualquier cláusula **FROM**. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones equivalentes son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores **en u**n campo común a ambas tablas.

Se puede utilizar **INNER JOIN** con las tablas Departamentos y Registros para seleccionar todos los Registros de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea **LEFT JOIN** o todos los Registros (incluso si alguno no está asignado a ningún departamento), en este caso **RIGHT JOIN**.

Si se intenta combinar campos que contengan datos Memo u Objeto OLE, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad Size de su objeto Field está establecida como Entero, y un campo Contador.

El ejemplo siguiente muestra cómo podría combinar las tablas **Categorías** y **Productos** basándose en el campo **IDCategoria**:
SELECT NombreCategoria, NombreProducto FROM Categorias INNER JOIN Productos ON Categorias.IDCategoria = Productos.IDCategoria

En el ejemplo anterior, **IDCategoria** es el campo combinado, pero no está incluido en la salida de la consulta ya que no está incluido en la instrucción SELECT. Para incluir el campo combinado, incluir el nombre del campo en la instrucción SELECT, en este caso, **Categorias.IDCategoria**.

También se pueden enlazar varias cláusulas **ON** en una instrucción **JOIN**, utilizando la sintaxis siguiente:
SELECT campos FROM tabla1 INNER JOIN tabla2 ON (tb1.campo1 comp tb2.campo1 AND ON tb1.campo2 comp tb2.campo2) OR ON (tb1.campo3 comp tb2.campo3)

También puede anidar instrucciones **JOIN** utilizando la siguiente sintaxis:
SELECT campos FROM tb1 INNER JOIN (tb2 INNER JOIN [( ]tb3
[INNER JOIN [( ]tablax [INNER JOIN ...)]
ON tb3.campo3 comp tbx.campox)] ON tb2.campo2 comp tb3.campo3)
ON tb1.campo1 comp tb2.campo2

Un **LEFT JOIN** o un **RIGHT JOIN** puede anidarse dentro de un INNER JOIN, pero un INNER JOIN no puede anidarse dentro de un LEFT JOIN o un RIGHT JOIN.

Ejemplo:
SELECT DISTINCT Sum(PrecioUnitario * Cantidad) AS Contenidos, (Nombre + ' ' + Apellido) AS Nombres FROM Registros INNER JOIN(Pedidos INNER JOIN DetallesPedidos
ON Pedidos.IdPedido = DetallesPedidos.IdPedido) ONRegistros.IdEmpleado = Pedidos.IdEmpleado GROUP BY Nombre + ' ' + Apellido

(Crea dos combinaciones equivalentes: una entre las tablas Detalles de pedidos y Pedidos, y la otra entre las tablas Pedidos y Registros. Esto es necesario ya que la tabla Registros no contiene datos de MOVIMIENTOS y la tabla Detalles de pedidos no contiene datos de los Registros. La consulta produce una lista de Registros y sus MOVIMIENTOS totales.)

Si empleamos la cláusula **INNER** en la consulta se seleccionarán sólo aquellos registros de la tabla de la que hayamos escrito a la izquierda de **INNER JOIN** que contengan al menos un registro de la tabla que hayamos escrito a la derecha. Para solucionar esto tenemos dos cláusulas que sustituyen a la palabra clave **INNER**, estas cláusulas son LEFT y RIGHT. LEFT toma todos los registros de la tabla de la izquierda aunque no tengan ningún registro en la tabla de la izquierda. RIGHT realiza la misma operación pero al contrario, toma todos los registros de la tabla de la derecha aunque no tenga ningún registro en la tabla de la izquierda.

La sintaxis expuesta anteriormente pertenece a ACCESS, en donde todas las sentencias con la sintaxis funcionan correctamente. Los manuales de SQL-SERVER dicen que esta sintaxis es incorrecta y que hay que añadir la palabra reservada OUTER: LEFT OUTER JOIN y RIGHT OUTER JOIN. En la práctica funciona correctamente de una u otra forma.

No obstante, los **INNER JOIN ORACLE** no es capaz de interpretarlos, pero existe una sintaxis en formato **ANSI** para los INNER JOIN que funcionan en todos los sistemas. Tomando como referencia la siguiente sentencia:

SELECT Facturas.*, Albaranes.* FROM Facturas INNER JOIN Albaranes
ON Facturas.IdAlbaran = Albaranes.IdAlbaran WHERE
Facturas.IdCliente = 325

La transformación de esta sentencia a formato **ANSI** sería la siguiente:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran AND
Facturas.IdCliente = 325 ;
```

Como se puede observar los cambios realizados han sido los siguientes:

Todas las tablas que intervienen en la consulta se especifican en la cláusula FROM. Las condiciones que vinculan a las tablas se especifican en la cláusula WHERE y se vinculan mediante el operador lógico AND.

Referente a los OUTER JOIN, no funcionan en ORACLE y además conozco una sintaxis que funcione en los tres sistemas. La sintaxis en ORACLE es igual a la sentencia anterior pero añadiendo los caracteres (+) detrás del nombre de la tabla en la que deseamos aceptar valores nulos, esto equivale a un **LEFT JOIN**:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran (+)
AND Facturas.IdCliente = 325
```

Y esto a un **RIGHT JOIN**:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran (+) = Albaranes.IdAlbaran
AND Facturas.IdCliente = 325
```

En **SQL-SERVER** se puede utilizar una sintaxis parecida, en este caso no se utiliza los caracteres (+) sino los caracteres =* para el LEFT JOIN y *= para el RIGHT JOIN.

COMBINACIONES INTERNAS INNER JOIN

Este tipo de operaciones se emplea con operadores comunes tales como (= ó <>), recordemos que las combinaciones internas usan un operador de comparación para hacer coincidir las filas de las tablas

COMBINACIONES EXTERNAS OUTER JOIN

Este tipo de operaciones emplea:

- LEFT OUTER JOIN (Externa Izquierda)
- RIGHT OUTER JOIN(Externa derecha)
- FULL OUTER JOIN (Completa)

COMBINACIONES CRUZADAS CROSS JOIN

Este tipo de operaciones devuelve las filas de las tablas combinadas llamadas también productos cartesianos .

EJEMPLO 235

Obtener el nombre de los clientes de la tabla clientes y de la tabla articulos los nombres de los articulos cuyos registros estan ordenados por nombres

```
SELECT AM.nomcliente, OR.productos FROM CLIENTES AS AM INNER JOIN
ARTICULO AS OR ON AM.Idcodcli = OR.Idcodin ORDER BY 1
```

EJEMPLO 236

Ahora desarrollaremos otro ejemplo, considerando que deseamos retornar todas las columnas de una tabla llamada CLIENTE donde la columna Idcodin esté relacionada con la tabla MOVIMIENTOS utilizando la combinación externa izquierda con respecto a MOVIMIENTOS (Cli_cod) y de esta manera obtener las columnas Fechaventa y Cantidad.

Select A.Idcodin, A.nombres,B.Fechaventa,B.cantidad FROM CLIENTE AS A LEFT OUTER JOIN MOVIMIENTOS AS B ON B.Cli_cod = A.Idcodin ORDER BY B.Cli_cod

EJEMPLO 237

Obtener todos los registros comprendos entre las tablas CLIENTES y MOVIMIENTOS

SELECT A.Codigocli AS CODIGO,A.Nomcli AS NOMBRES, B.Fechaventa, B.Cant, B.Precio,B.Total FROM CLIENTES AS A INNER JOIN MOVIMIENTOS AS B ON A.Codigocli = B.Codigo

EJEMPLO 238

Ahora vamos a considerar que en la practica real cuando el operador trabaje con tres tablas en relacion, donde las tablas son CLIENTES, MOVIMIENTOS y ALMACEN

SELECT A.Clicod, A.Nombres,B.Fechaventa,B.Cantidad, C.stock FROM CLIENTES AS A INNER JOIN MOVIMIENTOS AS B ON A.Cli_cod = B.Cli_cod INNER JOIN ALMACEN AS C ON A.Codbien = C.codbien

EJEMPLO 239

Ejecutar una consulta donde retorne todas las columnas codigocliente, nombre de cliente, telefono de la tabla CLIENTES utilizando una combinación externa izquierda con la tabla MOVIMIENTOS

SELECT A.Codigocliente,Anombreclie AS “Nombre cliente”, A.telefono,B.factura, B.Fechaventa, B.total FROM CLIENTES AS A LEFT OUTER JOIN MOVIMIENTOS AS B ON A.codigocliente = B.codigocliente ORDER BY B.Codigocliente

EJEMPLO 240

En este ejemplo vamos a trabajar con dos tablas llamadas CLIENTE y MOVIMIENTOS donde nos piden retornar todas las columnas de la tabla CLIENTE que no efectuaron movimientos, para esto podemos emplear IS NULL

SELECT A.Codigocli, A.Nombre,B.Numdoc, B.Fecha, b.Cant FROM CLIENTE AS A LEFT OUTER JOIN MOVIMIENTOS AS B ON B.Codigocli = A.codigocli WHERE B.Cant IS NULL

EJEMPLO 241

Ahora vamos a reportar todas las columnas de una tabla denominada MOVIMIENTO y LISTADO DE PROVEEDORES, para lo cual vamos a emplear una combinación externa como ejemplo

SELECT A.Numdoc,A.Cantidad,B.Codprov FROM CLIENTE AS A RIGHT OUTER JOIN [LISTADO DE PROVEEDORES] AS B ON A.Rucprov = B.Ruc_prov ORDER BY 3
GO

EJEMPLO 242

Ahora deseamos retornar todas las columnas de las tablas MOVIMIENTOS y CLIENTES, empleando la combinación externa completa

SELECT A.Rucprov,A.Nomprov as PROVEEDOR, B.Cliente FROM PROVEEDOR AS A FULL OUTER JOIN CLIENTES AS B ON A.codigocli = B.Codigocli ORDER BY 2

EJEMPLO 243-244-245-246

En este caso como ejemplo vamos a obtener diferentes resultados, considerando los siguiente: En una determinada empresa tiene registros de todos sus clientes cuya tabla se llama CLIENTES tambien tiene una tabla llamada UNIDADES donde se encuentra registrados los sitios u organos donde trabaja cada uno de ellos, asimismo tenemos una tabla llamada DEPENDENCIAS, donde se almacenan todas las dependencias organicas en la que laboran.

1. **Eliminemos las tablas CLIENTES y DEPENDENCIAS si es que existen**

2. **Crear las estructuras de las tablas**
3. **Ahora obtener algunos registros**
4. **Luego listar los datos de ambas tablas en forma ordenada**
5. **Reportar los clientes que laboran en la Dependencia "NORTE ORIENTE CUSCO"**

```
IF (Object_id("CLIENTES")) IS NOT NULL
        DROP table clientes;
IF (Object_id("DEPENDENCIAS")) IS NOT NULL
        DROP table dependencias;
------------
CREATE TABLE CLIENTES(
Codigocli               INT     IDENTITY,
Nombrecli               VARCHAR(60),
Telefono                VARCHAR(40),
Dependenciacod          TINYINT NOT NULL,
PRIMARY KEY (Codigocli)
);
CREATE TABLE DEPENDENCIAS (
Codigodep               TINYINT IDENTITY,
Nombredep               VARCHAR(20),
PRIMARY KEY (Codigodep)
);
-------------
```

Para ingresar registros a la tabla cuya columna se encuentre IDENTITY, recordemos que debemos activar manualmente la opción SET ON

```
SET IDENTITY _INSERT DEPENDENCIAS ON
INSERT INTO DEPENDENCIAS (Nombredep) Values("Norte Oriente Cusco");
INSERT INTO DEPENDENCIAS (Nombredep) Values("Oficina Regional Lima");
INSERT INTO DEPENDENCIAS (Nombredep) Values("Region Oriente Pucallpa");
INSERT INTO DEPENDENCIAS (Nombredep) Values("región centro Huancayo");
-------------
```

SELECT A.Nombrecli AS CLIENTE, A.Telefono, B.NombreDep AS Dependencias FROM CLIENTES AS A INNER JOIN DEPENDENCIAS AS B ON A.Dependenciacod = B.Codigodep;

SELECT A.Nombrecli AS NOMBRES, B.Nombredep as Dependencias FROM CLIENTES AS A INNER JOIN DEPENDENCIAS AS B ON A.Dependenciacod = B.Codigodep WHERE B.Nombredep ="Norte Oriente Cusco";

COMBINACION EXTERNA IZQUIERDA LEFT JOIN

EJEMPLO 247

Una Empresa tiene registrados sus clientes en una tabla llamada CLIENTES y otra tabla llamada DEPENDENCIAS, similar al ejemplo anterior, ahora diga usted como desarrollaria las siguientes consultas :

- Muestre todos los datos de los CLIENTES incluido el nombre de las dependencias.
- Muestre solamente los clientes de las Dependencias que existen en Cada Dependencia
- Muestre todas las columnas de los clientes cuya dependencia no se contemple y ordenado por nombres.
- Muestre todos los datos de los clientes de la Dependencia NORTE ORIENTE

----- Ahora vamos a la solución grupal

```
SELECT C.Nombres,c.Telefono,D.Dependencia FROM CLIENTES AS C
        LEFT OUTER JOIN DEPENDENCIAS AS D ON C.Coddep = D.Codigo;

SELECT C.Nombres, C.Telefono, D.Dependencia FROM CLIENTES AS C
        LEFT OUTER JOIN Dependencias AS D  ON C.Coddep = D.Codigo ORDER
        BY Nombres;
SELECT c.Nombres, C.Telefono,D.Dependencias AS D ON C.Coddep =D.codigo
WHERE D.Nomdep ="NORTE ORIENTE";
```

COMBINACION EXTERNA DERECHA RIGHT OUTER JOIN

EJEMPLO 248

Mostrar todos los datos de los clientes incluido el nombre de la Dependencia usando la clausula RIGHT

Mostrar lo mismo del ejemplo anterior pero en este caso vamos a emplear la clausula LEFT

SELECT C.Nombre,c.Telefono,D.Dependencia FROM Dependencia AS D
RIGHTOUTER JOIN CLIENTES AS C ON C.Coddep = D.Codigo;

SELECT C.Nombre, C.Telefono, D.Dependencia FROM CLIENTES AS C LEFT
OUTER JOIN Dependencias AS D ON C.Coddep =D.codigo,

COMBINACION EXTERNA COMPLETA FULL JOIN

Este tipo de operador se utiliza para devolver todas las filas de una combinación tengan o no correspondencia. Es el equivalente a la utilización de **LEFT JOIN** y **RIGHT JOIN** a la misma vez. Mediante este operador se obtendrán por un lado las filas que tengan correspondencia en ambas tablas y también aquellas que no tengan correspondencia sean de la tabla que sean.

EJEMPLO 249

1. **Mostrar toda la información de la tabla de los alumnos que se han inscritos para estudiar en la universidad y obtener de la tabla CURSOS el nombre de cada curso**
2. **Empleando LEFT JOIN con la tabla CURSOS obtenga todos los registros**
3. **Efectuar el mismo reporte emplenado RIGHT JOIN**
4. **Liste todos los cursos que no hay inscritos, empleando la clausula LEFT JOIN**
5. **Liste todos los registros de los alumnos matriculados que no contengan y no contemplen cursos en la tabla CURSOS**
6. **Empleando la clausula FULL JOIN obtener todos los datos de ambas tablas.**

/* Dando solución a lo expuesto */

```
SELECT B.Nombres,B.CodigoAlumno, C.Curso FROM ALUMNOS AS C
        INNER JOINCursos AS C ON B.Codcurso = c.codigo;

SELECT B.Nombres, B.codigocurso, C.Curso FROM ALUMNOS AS B
        LEFT OUTER JOIN CURSOS AS C ON B.Codcurso = C.codigo;

SELECT B.Nombres, B.Codigocurso,C.Curso FROM CURSOS AS C
        RIGHT OUTER JOIN ALUMNOS AS B ON B.Codcurso = c.Codigo;

SELECT C.Curso FROM ALUMNOS AS B LEFT OUTER JOIN CURSOS AS C
        ON B.Codcurso = C.Codigo WHERE B.Codcurso IS NULL

SELECT B.Nombres FROM ALUMNOS AS B LEFT OUTER JOIN Cursos AS C
        ON B.Codcurso = C.Codigo WHERE C.Codigo IS NULL;
SELECT B.Nombres, B.aula, C.curso, B.horario FROM Alumnos AS B FULL JOIN
        Cursos  AS C ON B.Codcurso = C.codigo;
```

COMBINACION CRUZADA CROSS JOIN

Se utiliza en SQL-SERVER para realizar consultas de unión. Supongamos que tenemos una tabla con todos los autores y otra con todos los libros. Si deseáramos obtener un listado combinar ambas tablas de tal forma que cada autor apareciera junto a cada título,

EJEMPLO 250

Una Agencia de Empleos almacena información de todos los postulantes o clientes inscritos de sexo Femenino e una tabla llamada VARONESA y en otra tabla de Varones llamada VARON los inscritos de sexo Masculino.

- **Listar la combinación de todas las personas de sexo femenino con la de sexo Masculino**
- **Realice la misma combinación, pero considerando solamente las personas cuya edad sea mayor a 40 años**
- **Luego agrupe las parejas pero teniendo en cuenta que no tengan una diferencia superior a 10 años**

/*Dando solución */

- **SELECT M.Nombre,M.Edad,M.Sexo,V.nombre,V.edad, V.sexo FROM VARONESA AS M CROSS JOIN VARON AS V;**
- **SELECT M.Nombre,M.Edad,M.Sexo,V.Nombre,V.Edad,V.Sexo FROM VARONESA AS M CROSS JOIN VARON AS V WHERE M.Edad>40 and V.edad>40;**

- **SELECT M.Nombre,M.Edad,V.Nombre,V.Edad FROM VARONESA AS M CROSS JOIN VARON AS V WHERE M.Edad-V.Edad BETWEEN -10 AND 10;**

AUTOCOMBINACION (En la misma tabla)

La autocombinación se utiliza para unir una tabla consigo misma, comparando valores de dos columnas con el mismo tipo de datos. La sintaxis en la siguiente:

SELECT alias1.columna, alias2.columna, ... FROM tabla1 as alias1, tabla2 as alias2 WHERE alias1.columna = alias2.columna AND otras condiciones

EJEMPLO 251

En este ejemplo vamos a considerar que tenemos una agencia donde se va almacenar la información de todos sus clientes en una tabla llamada CLIENTES.

Se necesita la combinación de todas las personas de sexo femenino con las de sexo masculino y obtengamos:

- El mismo reporte pero realizando la operación JOIN
- Listar el mismo reporte empleando JOIN pero asimismo que las parejas no tengan una diferencia superior entre edades a 5 años entre cada una de ellas.

SELECT CM.Nombres, CM.Edad, CV.Nombres,CV.Edad FROM CLIENTES AS CM CROSS JOIN CLIENTES AS CV WHERE CM.SEXO = “F” AND CV.SEXO=”M”

SELECT CM.Nombres, CM.Edad, CV.Nombres,CV.Edad FROM CLIENTES AS CM CROSS JOIN CLIENTES AS CV WHERE CM.SEXO = “F” AND CV.SEXO=”M” AND CM.Edad-CV.Edad BETWEEN -5 AND 5 ;

COMBINACIONES Y FUNCIONES DE AGRUPAMIENTO

EJEMPLO 252

Crear una tabla llamada CLIENTES y otra llamada DEPARTAMENTOS donde se encuentra los nombres de los departamentos

Ahora en base a lo titulado en combinaciones de agrupamiento, vamos a considerar estas preguntas y consultas practicas para el operador :

- **Contar la cantidad de clientes por cada departamento, mostrando el nombre de cada departamento**
- **Mostrar el promedio de gastos por cada cliente agrupados por ciudad y sexo**
- **Liste la cantidad de visitas agrupados por cada departamento**
- **Liste el monto mas alto de cada departamento, considerando que para cad una de estas consultas tenemos estas dos tablasCLIENTES y DEPARTAMENTO, con la siguiente estructura.**

```
CREATE TABLE CLIENTES (
Nombre          VARCHAR (40),
Edad            Tinyint,
Sexo            CHAR(1) DEFAULT ¨v¨,
Domicilio       Varchar (60),
Codigodep       Tinyint  NOT NULL,
Totalmonto      Decimal(10,2)
);

CREATE TABLE DEPARTAMENTO (
Codigo          Tinyint identity,
Nombre          Varchar(40)
);
```

- **SELECT C.Nombre, COUNT(*) AS Cantidad FROM DEPARTAMENTO AS C JOIN CLIENTES AS V ON V.Codigodep = C.Codigo GROUP BY C.Nombre;**

- **SELECT C.Nombre, AVG (V.Totalmonto) AS ¨Promedio total¨ FROM DEPARTAMENTO AS C JOIN CLIENTES AS V ON V.Codigodep = C.codigo GROUP BY C.Nombre, V.Sexo;**
- **SELECT C.Nombre, COUNT(Totalmonto) AS TOTAL FROM DEPARTAMENTO AS C JOIN CLIENTES AS V ON V.Codigodep = C.Codigo GROUP BY C.Nombre;**
- **SELECT C.Nombre, MAX(Totalmonto) FROM CLIENTES AS V JOIN DEPARTAMENTO AS C ON V.Codigodep = C.codigo GROUP BY C.Nombre;**

COMBINACIONES ENTRE DOS O MAS TABLAS

Haciendo un repaso breve a lo detallado, sobre combinaciones de tablas vamos a resaltar una vez mas el uso de los alias en los nombres de las tablas, asicomo la combinación de varios conjuntos de resultados mediante el operador UNION.
Asi que debemos recordar que una clave primaria es un campo o varios que se identifica en un solo registro dentro de una tabla y la clave foranea se define como un campo cuyo valor coincide con la clave primaria, haciendo referencia a un campo con una restricción Primari Key o Unique.
Dando inicio a la introducción de la misma, consideramos estos ejemplos anexos para el operador

Ejemplo
Se tienen las tablas MOVIMIENTOS y OPERADOR, dichas tablas deberán relacionarse por el idOPERADOR que es la llave primaria en la tabla OPERADOR y es foránea en la tabla MOVIMIENTOS.

```
--Creación de BD
CREATE Database MOVIMIENTOS
GO
USE MOVIMIENTOS
GO
---tabla OPERADOR
CREATE TABLE OPERADOR(
idOPERADOR INT PRIMARY KEY ,
nombre VARCHAR(20),
apellido VARCHAR(10),
);
GO
---tabla MOVIMIENTOS
CREATE TABLE MOVIMIENTOS(
Idventa INT PRIMARY KEY ,
nombre VARCHAR(20),
apellido VARCHAR(10),
idOPERADOR INT
foreign key (idOPERADOR) references OPERADOR(idOPERADOR))
```

Cuando creamos relaciones en SQL Server, este nos permite crear un diagrama relacional de las tablas de la base de datos,

El uso de alias en los nombres de tablas mejora la legibilidad de las secuencias de comandos, facilita la escritura de combinaciones complejas y simplifica el mantenimiento de Transact-SQL. al escribir secuencias de comandos, puede sustituir un nombre de tabla descriptivo largo y complejo por un alias sencillo y abreviado. El alias se utiliza en lugar del nombre completo de la tabla.

Sintaxis parcial: SELECT * FROM servidor.baseDeDatos.esquema.tabla AS aliasTabla

En este ejemplo se muestran los nombres de los clientes, el identificador del cliente y la cantidad vendida de las tablas **Regis001** y **Contenidos**. Esta consulta no utiliza alias en las tablas de la sintaxis de JOIN.

```
USE Padrondb
SELECT Regis001_Nombres, Contenidos.Regis001_id, qty FROM Regis001 INNER JOIN Contenidos ON Regis001.Regis001_id = Contenidos.Regis001_id
GO
```

En este ejemplo se muestran los nombres de los clientes, el identificador del cliente y la cantidad vendida de las tablas **Regis001** y **Contenidos**. Esta consulta utiliza alias en las tablas de la sintaxis de JOIN.

```
USE Padrondb
SELECT Regis001_Nombres, s.Regis001_id, qty FROM Regis001 AS b INNER JOIN Contenidos AS s ON b.Regis001_id = s.Regis001_id
GO
```

Algunas veces, la compleja sintaxis de JOIN y las subconsultas deben usar alias en los nombres de tablas. Por ejemplo, al combinar una tabla consigo misma deben utilizarse

alias.
Ahora la tabla Regis001 se le podrá conocer con el alias b y la tabla Contenidos se podrá conocer con el alias s

Combinación de datos de varias tablas:

Una combinación es una operación que permite consultar dos o más tablas para producir un conjunto de resultados que incorpore filas y columnas de cada una de las tablas. Las tablas se combinan en función de las columnas que son comunes a ambas tablas.
Cuando se combinan tablas, Microsoft® SQL Server™ compara los valores de las columnas especificadas fila por fila y, después, utiliza los resultados de la comparación para combinar los valores que cumplan los criterios especificados en nuevas filas.
Hay tres tipos de combinaciones: combinaciones internas, combinaciones externas y combinaciones cruzadas. Adicionalmente, en una instrucción SELECT se pueden combinar **más de dos tablas** mediante un conjunto de combinaciones o se puede combinar una tabla consigo misma mediante una autocombinación.

JOIN:
Es una operación que combina registros de dos tablas en una base de datos relacional que resulta en una nueva tabla (temporal) llamada tabla de JOIN.
Las tablas se combinan para producir un único conjunto de resultados que incorpore filas y columnas de dos o más tablas.
Sintaxis Parcial
SELECT columna [, columna …] FROM {<tablaOrigen >} [**,** ...n] <tipoCombinación > ::= [INNER | { { LEFT | RIGHT | FULL } [OUTER] }] [<sugerenciaCombinación>] JOIN <tablaCombinada> ::= <tablaOrigen > <tipoCombinación > <tablaOrigen > ON <condiciónBúsqueda> | <tablaOrigen > CROSS JOIN <tablaOrigen > | <tablaCombinada>

Selección de columnas específicas de varias tablas:
Una combinación permite seleccionar columnas de varias tablas al expandir la cláusula FROM de la instrucción SELECT. En la cláusula FROM se incluyen dos palabras clave adicionales: JOIN y ON.
La palabra clave JOIN especifica qué tablas se van a combinar y cómo.
La palabra clave ON especifica las columnas que las tablas tienen en común.
Consultas de dos o más tablas para producir un conjunto de resultados:
Una combinación permite consultar dos o más tablas para producir un único conjunto de resultados. Al implementar combinaciones, tenga en cuenta los siguientes hechos e instrucciones:

1. Especifique la condición de combinación en función de claves principales y externas.
2. Si una tabla tiene una clave principal compuesta, cuando combine tablas debe hacer referencia a toda la clave en la cláusula ON.
3. Para combinar tablas, utilice columnas comunes a las tablas especificadas. Dichas columnas deben tener tipos de datos iguales o similares.
4. Haga referencia al nombre de la tabla si las columnas de las tablas que va a combinar tienen el mismo nombre. Califique los nombres de las columnas con el formato tabla.columna.
5. Limite el número de tablas de las combinaciones porque cuantas más tablas combine, mayor será la duración del proceso de la consulta.
6. Puede incluir varias combinaciones en una instrucción SELECT.

Uso de combinaciones internas:
Las combinaciones internas combinan tablas mediante la comparación de los valores de las columnas que son **comunes** a ambas tablas. SQL Server sólo devuelve las filas que cumplen las condiciones de la combinación.
• **Por qué se utilizan combinaciones internas:**
Utilice combinaciones internas para obtener información de dos tablas independientes y combinar dicha información en un conjunto de resultados. Al utilizar combinaciones internas, tenga en cuenta los siguientes hechos e instrucciones:

1. Las combinaciones internas son el tipo predeterminado de SQL Server.Puede abreviar la cláusula INNER JOIN como JOIN.
2. Para especificar las columnas que desea presentar en el conjunto de resultados, incluya los nombres calificados de las columnas en la lista de selección.
3. Incluya una cláusula WHERE para restringir las filas que se devuelven en el conjunto de resultados.
4. No utilice valores NULL como condición de combinación, ya que no se evalúan como iguales entre sí.

Supongamos que tenemos dos tablas: una de **trabajos** y otra de **Alumnos** relacionadas entre sí:

Si quiero obtener todas las trabajosy el nombre de su director haría lo siguiente:

```
USE TABLA_TRABAJOS
SELECT m.titulodetrabajo, d.Nombres FROM TABLA_TRABAJOS m INNER JOIN directors d ON m.director = d.id
```

Donde los campos a seleccionar son titulodetrabajo que está en la tabla TABLA_TRABAJOS y nombres que está en la tabla Alumnos, como TABLA_TRABAJOS tiene el alias m, podemos anteponer ese alias a los campos que queramos seleccionar de la tabla TABLA_TRABAJOS, esto se hace para no causar ambigüedad en caso que hallan 2 campos con nombres iguales en cada tabla.

Este ejemplo devuelve los valores **Regis001_nombres**, **Regis001_id** y **qty** de los clientes que han adquirido algún producto. Los clientes que no hayan adquirido nada no se incluyen en el conjunto de resultados. Los clientes que han adquirido más de un producto aparecen una vez por cada compra realizada.
Las columnas **Regis001_id** de las dos tablas pueden especificarse en la lista de selección.

```
USE Padrondb
SELECT Regis001_nombres, Contenidos.Regis001_id, qty FROM Regis001
INNER JOIN Contenidos ON Regis001.Regis001_id = Contenidos.Regis001_id
GO
```

Combinaciones Externas

Una combinación interna (**JOIN**) encuentra registros de la primera tabla que se correspondan con los registros de la segunda, es decir, que cumplan la condición del "**ON**" y si un valor de la primera tabla no se encuentra en la segunda tabla, el registro no aparece.
Si queremos saber qué registros de una tabla NO encuentran correspondencia en la otra, es decir, no existe valor coincidente en la segunda, necesitamos otro tipo de combinación, "**OUTER JOIN**" (combinación externa).

Las combinaciones externas combinan registros de dos tablas que cumplen la condición, más los registros de la segunda tabla que no la cumplen; es decir, muestran todos los registros de las tablas relacionadas, aún cuando no haya valores coincidentes entre ellas.

Este tipo de combinación se emplea cuando se necesita una lista completa de los datos de una de las tablas y la información que cumple con la condición. Las combinaciones externas se realizan solamente entre 2 tablas.
Hay tres tipos de combinaciones externas: "**LEFT OUTER JOIN**", "**RIGHT OUTER JOIN**" y "**FULL OUTER JOIN**"; se pueden abreviar con "**LEFT JOIN**", "**RIGHT JOIN**" y "**FULL JOIN**" respectivamente.

Se emplea una combinación externa izquierda para mostrar todos los registros de la tabla de la izquierda. Si no encuentra coincidencia con la tabla de la derecha, el registro muestra los campos de la segunda tabla seteados a "null".

En el siguiente ejemplo solicitamos el título y nombre de la editorial de los libros:

```
SELECT titulo,nombre FROM editoriales AS e LEFT JOIN libros AS l
ON codigoeditorial = e.codigo;
```

El resultado mostrará el título y nombre de la editorial; las editoriales de las cuales no hay libros, es decir, cuyo código de editorial no está presente en "libros" aparece en el resultado, pero con el valor "null" en el campo "titulo".

Es importante la posición en que se colocan las tablas en un "left join", la tabla de la izquierda es la que se usa para localizar registros en la tabla de la derecha.

Entonces, un "left join" se usa para hacer coincidir registros en una tabla (izquierda) con otra tabla (derecha); si un valor de la tabla de la izquierda no encuentra coincidencia en la tabla de la derecha, se genera una fila extra (una por cada valor no encontrado) con todos los campos

Sintaxis:

```
SELECT CAMPOS FROM TABLAIZQUIERDA LEFT JOIN TABLADERECHA
ON CONDICION;
```

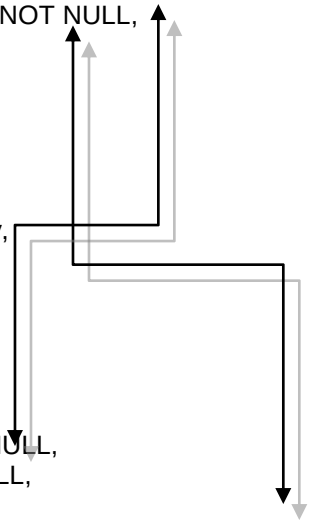
EJEMPLO 254

En una universidad se inscriben varios alumnos cuya tabla es llamado ALUMNOS, en otra tabla se encuentra la relación de los cursos y en otra existente están todos los alumnos llamado GENERAL donde estan todos los alumnos inscritos.

```
CREATE TABLE ALUMNOS (
Id_codigo       Char(10)        NOT NULL,
NombreVarchar(40),
Domicilio       Varchar(60),
PRIMARY KEY (Id_codigo)
);

CREATE TABLE CURSOS (
Codigo          Tinyint  identity,
NombreVarchar (40),
Profesor        Varchar(60),
PRIMARY KEY (codigo)
);

CREATE TABLE GENERAL (
Id_codigo       Char(10) NOT NULL,
Codigocurso     Tinyint NOT NULL,
Anio            char(8),
Pago            Char(1), _ _ “S”=Pago, “N” = No pago
PRIMARY KEY (Id_codigo,codigocurso,anio)
);
```



Ahora reporte el nombre del Alumno, el nombre del curso en que se encuentra matriculado y el año empleando JOIN

```
SELECT S.Nombre AS ALUMNO, D.Nombre AS CURSO, Anio FROM CURSOS AS D
RIGHT JOIN GENERAL AS I ON I.Codigocurso = D.codigo LEFT JOIN ALUMNOS AS S
ON I.Id_codigo = S.Id_codigo;
```

EJEMPLO 255

Ahora diga usted como listar todos los datos de los alumnos matriculados incluyendo los cursos que no contemplen en la tabla.

```
SELECT S.Nombre, D.Nombre AS CURSO, I.Anio,I.Pago FROM CURSOS AS D FULL
JOIN GENERAL AS I ON I.Codigocurso =D.codigo FULL JOIN ALUMNOS AS S ON
S.Id_Codigo =I.Id_codigo
```

Recordemos que dentro de las funciones de agregación podemos emplear la sentencia SELECT, en muchas veces con la clausula GROUP BY

EJEMPLO 256 - REPASO

Reportar el valor promedio del precio de venta de los articulos que están en la tabla Almacen

```
SELECT AVG(Pventa) FROM ARTICULOS
```

MAX( )

Expresa el maximo valor

```
SELECT MAX(Edad) FROM Personal
```

MIN( )

Expresa el minimo valor

```
SELECT MIN(Edad) FROM Personal
```

SUM( )

Suma los valores

```
SELECT SUM(Precio*Cantidad) FROM ARTICULOS
```

WITH ROLLUP

EJEMPLO 257

Agrupar todas las filas de la tabla ARTICULOS en base a la combinación de valorees de las columnas CODIGOARTICULO y UBICACIÓN, luego empleando ROLLUP agrupamos los registros en base a CODIGOARTICULO

```
SELECT Codigoarticulo,ubicación,AVG(pventa) AS Promedio FROM ARTICULOS
GROUP BY codigoarticulo, ubicación WITH ROLLUP
```

COMPUTE … BY

Permite que se agrupen todas las filas de las tablas en relación a los valores de una o mas columnas y luego efectuar una relación de resumen sobre cada grupo.
La diferencia con GROUP BY es que permite mostrar la información mas detallada de cada grupo

EJEMPLO 258

Retornar las columnas Id_codigo, nombres, numerodocumento de la tabla personal y ademas el tipo de documento de la tabla documento, empleando la combinación interna.

SELECT B.Id_codigo,B.nombres,c.Tipodocum FROM PERSONAL AS B INNER JOIN Tipodocum AS C ON B.Codigodocum = C.Codigo ORDER BY 2

EJEMPLO 259

Del ejemplo anterior reportar todas las columnas Id_codigo, nombres y el tipo de documento empleando la combinación externa izquierda

SELECT B.Id_codigo,b.Nombres,C.Tipodocum FROM PERSONAL AS B LEFT OUTER JOIN Tipodocum AS C ON B.Codigodocum = C.Codigo ORDER BY 1

CROSS JOIN

EJEMPLO 260

Retornar todas las columnas: idcodigo y nombres de la tabla personal, asi como el codigo de tipo de documento de la tabla documento empleando una combinacion externa completa

SELECT C.Idcodigo, C.Nombres, B.Tipodocumento FROM PERSONAL AS C FULL OUTER JOIN DOCUMENTO AS B ON C.Iddocum = B.Coddocum ORDER BY 2

EJEMPLO 261

Reportar todas las columnas Idcodigo y nombres de la tabla personal y tipodocume de la tabla documento empleando una combinación cruzada

SELECT C.Idcodigo, C.Nombres, B.Tipodocumento FROM PERSONAL AS C CROSS JOIN DOCUMENTO AS B ORDER BY 2

SUB CONSULTAS

Una sub consulta es una sentencia SELECT que es incrustada en una cláusula de otra sentencia SQL, llamada sentencia padre.
La sub consulta (consulta interna) obtiene un valor que es usado por la sentencia padre. Usar una sub consulta anidada es equivalente a ejecutar dos consultas secuenciales y utilizar el resultado de la consulta interna como valor de búsqueda en la consulta externa (consulta principal).

Las sub consultas pueden ser usadas para los siguientes propósitos:

- Proveer valores para condiciones en cláusulas WHERE, HAVING y START WITH de sentencias SELECT
- Definir el conjunto de filas a ser insertadas en una tabla de una sentencia INSERT o CREATE TABLE
- Definir el conjunto de filas a ser incluidas en una vista o snapshot en una sentencia CREATE VIEW o CREATE SNAPSHOT
- Definir uno o más valores para ser asignados a filas existentes en una sentencia UPDATE
- Definir una tabla para ser operada por el contenido de una consulta.
  (Esto se hace colocando la sub consulta en la cláusula FROM. Esto puede hacerse también en sentencias INSERT, UPDATE y DELETE

Sub consultas

Se pueden construir sentencias poderosas utilizando sub consultas. Las sub consultas pueden ser muy útiles cuando necesites seleccionar filas de una tabla con una condición que dependa de los datos de la misma u otra tabla. Las sub consultas son muy útiles para escribir sentencias SQL que necesiten valores de un o más valores condicionales desconocidos.
Donde:
o*perator* incluye un operador de comparación como >, = o IN

NOTA

los operadores de comparación se encuentran en dos clases: operadores de fila única (>, =, >=, <, <>, <=) y operadores de múltiples filas

SUB CONSULTAS DE MÚLTIPLES COLUMNAS

Hasta ahora hemos escrito sub consultas de filas únicas y sub consultas de múltiples filas donde solo una columna es obtenida por la sentencia SELECT interna y estas son usadas para evaluar la expresión en la sentencia SELECT padre. Si quieres comparar dos o más columnas, debemos escribir una cláusula WHERE compuesta usando operadores lógicos. El uso de sub consultas de múltiples columnas, puede combinar condiciones WHERE duplicadas en una simple cláusula WHERE.

Una sub consulta de múltiples columnas puede también ser una comparación no par. En una comparación no par, cada columna de la cláusula WHERE de la sentencia SELECT padre es individualmente comparada con múltiples valores recuperados por la sentencia SELECT interna. Las columnas individuales pueden corresponder con algunos de los valores recuperados por la sentencia SELECT interna. Pero en conjunto, todas las condiciones múltiples de la sentencia SELECT principal deben ser satisfechas para las filas a ser desplegadas.

Usando una sub consulta dentro de la cláusula FROM

Se puede usar una sub consulta en la cláusula FROM de una sentencia SELECT, el cuál es muy similar al manejo de las vistas que hemos usado. Una sub consulta en la cláusula FROM de una sentencia SELECT es también llamada una *vista en línea*. Una sub consulta en una cláusula FROM de una sentencia SELECT define un origen de datos para esa sentencia SELECT en particular, y solo esa sentencia SELECT.

Sub consultas escalares en SQL

Una sub consulta que obtiene exactamente un valor de una columna de una fila es también llamada sub consulta escalar. Sub consultas de múltiples columnas escritas para comparar dos o más columnas, usando una cláusula WHERE compuesta y operadores lógicos, no pueden ser calificados como sub consultas escalares.

El valor de una expresión en una sub consulta escalar es el valor del elemento de la lista seleccionado de la sub consulta. Si la sub consulta obtiene 0 filas, el valor de la expresión de la sub consulta escalar es nulo. Si la sub consulta obtiene más de una fila, el servidor de Oracle muestra un error. El servidor de Oracle siempre tiene el apoyo para usar una sub consulta escalar en una sentencia SELECT. El uso de una sub consulta escalar ha sido mejorado en Oracle9*i*. Ahora se pueden usar sub consultas escalares en:

- Condiciones y parte de expresiones de funciones DECODE y CASE
- Todas las cláusulas del SELECT excepto GROUP BY
- En el lado izquierdo del operador en una cláusula SET y WHERE de una sentencia UPDATE

Sin embargo, las sub consultas escalares no son expresiones válidas en los siguientes lugares:

- Como valor por defecto para columnas y expresiones para clusters
- En la cláusula RETURNING de sentencias DML
- Como base de una función base indexada
- En la cláusula GROUP BY, *constraints* CHECK, condiciones WHEN
- Cláusulas HAVING
- En cláusulas START WITH y CONNECT BY
- En sentencias que no son relacionados con consultas, como CREATE PROFILE

Sub consulta correlacionada

El servidor de Oracle ejecuta sub consultas correlacionadas cuando la sub consulta se relaciona con una columna de una tabla referida en la sentencia padre. Una sub consulta correlacionada es evaluada una vez para cada fila procesada por la sentencia padre. La sentencia padre puede ser una sentencia SELECT, UPDATE o DELETE.

Sub consultas anidadas contra sub consultas correlacionadas

Con una sub consulta anidada normal, la consulta SELECT interna corre primero y se ejecuta una sola vez, obteniendo valores para ser usados en la consulta principal. Una sub consulta correlacionada, sin embargo, se ejecuta una vez para cada fila candidata considerada por la consulta externa. En otras palabras, la consulta interna es la guía para la consulta externa.

Funcionamiento de sub consultas anidadas

- La consulta interna se ejecuta primero y encuentra un valor
- La consulta externa se ejecuta una vez, usando el valor de la consulta interna

Funcionamiento de sub consultas correlacionadas

- Recibe una fila candidata (obtenida por la consulta externa)
- Ejecuta la consulta interna usando el valor de la fila candidata
- Usa los valores resultantes de la consulta interna para calificar o descalificar la fila candidata
- Se repite hasta terminar con las filas candidatas

Una sub consulta correlacionada es un camino de lectura de cada fila en una tabla y la comparación de valores en cada fila contra los datos relacionados. Es usado cuantas veces una sub consulta deba retornar un resultado diferente o conjunto de resultados para cada fila candidata considerada por la consulta principal. En otras palabras, utilice una consulta correlacionada para resolver una pregunta de múltiples partes cuya respuesta dependa del valor de cada fila procesada por la sentencia padre. El servidor de Oracle ejecuta una sub consulta correlacionada cuando la sub consulta hace referencia a una columna de la consulta padre.

Nota: Puedes usar los operadores ANY y ALL en una sub consulta correlacionada.

Operador EXISTS

Cuando anidamos sentencias SELECT, todos los operadores lógicos son válidos. En suma, se puede usar el operador EXISTS. Este operador es frecuentemente usado en sub consultas correlacionadas para verificar cuando un valor recuperado por la consulta externa existe en el conjunto de resultados obtenidos por la consulta interna. Si la sub consulta obtiene al menos una fila,
el operador obtiene el valor TRUE. Si el valor no existe, se obtiene el valor FALSE. Consecuentemente, NOT EXISTS verifica cuando un valor recuperado por la consulta externa no es parte del conjunto de resultados obtenidos por la consulta interna.

Operador NOT EXISTS como solución alternativa
Un operador NOT IN puede ser utilizado como una alternativa para el operador NOT EXISTS, como se muestra en el siguiente ejemplo:
Sin embargo, NOT IN evalúa a FALSE si algún miembro del conjunto de resultados es un valor nulo. Por consiguiente, las consultas pueden no obtener algunas filas si estas filas en la tabla DEPARTMENTS no satisfacen la condición WHERE.

UPDATE Correlacionado
En el caso de la sentencia UPDATE, se puede usar una sub consulta correlacionada para actualizar filas en una tabla con base a las filas de otra tabla.

DELETE Correlacionado
En el caso de la sentencia DELETE, se puede usar una sub consulta correlacionada para eliminar solo aquellas filas que también existan en otra tabla. Si decides que debes mantener solo los últimos cuatro registros históricos de la tabla JOB_HISTORY, entonces cuando un empleado sea transferido a su quito puesto, debes de eliminar las filas mas antiguas.

Cláusula WITH
Usando la cláusula WITH, se puede definir un bloque de una consulta antes de que esta sea usada. La cláusula WITH (formalmente conocida como "*subquery_factoring_clausula*" cláusula de sub consulta factorizada) habilita la reutilización del mismo bloque de la consulta en una sentencia SELECT cuando esto ocurre en mas de una ocasión en una consulta compleja. Esto es particularmente útil cuando una consulta tiene muchas referencias al mismo bloque de una consulta y se tienen asociaciones y agrupaciones.

Lo bueno de la cláusula WITH
- Hace que la consulta sea fácil de leer
- Evalúa una cláusula una sola vez, aun si esta aparece muchas veces en la consulta, por esta razón aumenta el desempeño

EJEMPLO 262
Reportar todas las columnas Idcodigo, fecha de ingreso y el precio minimo unitario de lo adquirido

SELECT B.Idcodigo, B.Fechaingreso, (SELECT MIN(C.Preciounitario) FROM VENTAS AS C WHERE B.Cdobien = C.Codbien) FROM MOVIMIENTO AS B

EJEMPLO 263
Reportar todas las columnas de los productos con su respectivo precio que sean iguales o mayor que el precio del catalogo de bienes.

SELECT * FROM ARTICULOS AS B WHERE Preciounitario = (SELECT MAX(Preciounitario) FROM ARTICULOS AS C)

EJEMPLO 264

Reportar todas las columnas de los Alumnos inscritos en la universidad que se encuentran cursando el mismo ciclo de estudios de la facultad de Ingenieria del III Ciclo o semestre academico.

SELECT Nombres, Apellidos, Carnet FROM ALUMNOS WHERE Ciclo IN (SELECT Ciclo FROM Alumnos WHERE Ciclo =”III”)

EJEMPLO 265 (Emplearemos NOT IN)

Listar del ejemplo anterior todos los alumnos que curcen ciclos diferentes al III Ciclo

SELECT Nombres, Apellidos, Carnet FROM ALUMNOS WHERE Ciclo NOT IN (SELECT Ciclo FROM Alumnos WHERE Ciclo =”III”)

Recuerda, pues que cuando presenta la clausula EXISTS dentro de una sub consulta funciona como una existencia.

EJEMPLO 266 (Emplearemos NOT IN)

Listar todas las columnas Idcodigo, Nombres de todos los clientes que hayan efectuado al menos una compra o movimiento.

SELECT Idcodigo, Nombres FROM CLIENTES AS B WHERE EXISTS (SELECT * FROM CLIENTES AS CINNER JOIN VENTAS AS D ON (C.Codigo = D.Idcodigocli) WHERE C.Codigo = D.Idcodigocli)

EJEMPLO 267

De acuerdo al emplo anterior listar todos los clientes que no efectuaron compra alguna.

SELECT Idcodigo, Nombres FROM CLIENTES AS B WHERE NOT EXISTS (SELECT * FROM CLIENTES AS CINNER JOIN VENTAS AS D ON (C.Codigo = D.Idcodigocli) WHERE C.Codigo = D.Idcodigocli)

EJEMPLO 268 (Emplearemos DISTINCTROW)

Listar todas las columnas de los clientes con el mismo nombre y visualizar su codigo respectivo.

SELECT DISTINCTROW Clientes.Nombres,Clientes.Apellidos FROM CLIENTES WHERE Clientes.Nombres IN (SELECT Nombres HAVING COUNT(*)>1) ORDER BY 1

EJEMPLO 269

Recuperar todos los registros de la tabla que no contengan en la otra tabla:
Listaremos todos los productos del almacén que no se han vendido dentro de un rango determinado de fechas.

SET DATE FORMAT dmy
SELECT DISTINCTROW Articulos.Idcodigo, Articulos.Nombres FROM ARTICULOS LEFT JOIN VENTAS ON ARTICULOS.Idcodigo = VENTAS.Codigo WHERE (VENTAS.codigo IS NULL) AND (VENTAS.Fechaventa BETWEEN # 27-08-2010 and 30-08-2010 #)

Recordemos que las consultas de referencias cruzadas son aquellas que nos permite visualizar los datos en la sfilas y en la columna.

FUNCION AGREGADA

Es una función de SQL que se opera sobre los datos seleccionados.

SELECT	Es una isntrucción y el campo PIVOT (expresión) debe emplearse para crear cabeceras de las columnas en el resultado de la columna.
Valor1…	son los valores Registros para crear cabeceras de la columna.
TRANSFORM	Es opcional pero si incluye es la primera instrucción de una cadena SQL, procede a la instrucción SELECT que especifica los campos utilizados como encazados de fila y una clausula GROUP BY que especifica el agrupamiento de las filas.

Sintaxis :

TRANSFROM Funcionagregada.Instruccion
SELECT PIVOT Campo PIVOT {[IN VALOR1…..)]

EJEMPLO 270

Crear una consulta de tabla de referencias cruzadas que muestra las ventas de los productos por cada semestre.

SET DATE FORMAT dmy
TRANSFORM SUM(Cantidad) AS VENTAS SELECT ARTICULO FROM MOVIMIENTO WHERE Fecha BETWEEN # 01-01-2010# AND #31-12-2010# group by Articulo ORDER BY Articulo PIVOT "Semestre" &Datepart ("q", Fechaven) IN ("Semestre1","Semestre2","Semestre3,"Semestre4","Semestre5","Semestre6")

ARTICULO	Semestre 1	Semestre 2	Semestre 3	Semestre 4	Semestre 5	Semestre 6
Zapatillas	200.00	120.00	48.00	30.00	23.00	100.00
Zapatos	340.00	120.00	57.00	567.00	59.00	23.00
Toallas	500.00	45.00	69.00	900.00	90.00	340.00

EJEMPLO 271 (Emplearemos PIVOT)
Crear una tabla d ereferencias cruzadas similar al ejemplo anterior por cada mes.

SET DATE FORMAT dmy
TRANSFORMT SUM(Cantidad) AS VENTAS SELECT Articulo FROM MOVIMIENTO WHERE Fecha BETWEEN #01-01-2010# AND #31-12-2010# GROUP BY Articulo ORDER BY Articulo PIVOT DATEPART ("M" ,Fcehaventa)
EJEMPLO 272 (Emplearemos PIVOT)
Crear una tabla d ereferencias cruzadas listando e total de productos vendidos en un año.

SET DATE FORMAT dmy
TRANSFORMT SUM(VENTAS.Cantidad) AS TOTALVENTAS SELECT Articulo.Nombre AS ARTICULO, VENTAS.Codigobien AS CODIGO, SUM(VENTAS.Cantidad) AS RESULTADO, AVG(VENTAS.Cantidad) AS PROMEDIO FROM VENTAS INNER JOIN ARTICULOS ON VENTAS.Codigobien = ARTICULOS.Codigo GROUP BY VENTAS.Codigobien,ARTICULOS.Nombre PIVOT YEAR(Fecha)

ARTICULO	codigo	Resultado	Promedio	2000	2001	2002	2003	2004	2010
Zapatillas	A100	40	46.00	30	23	100	30	23	100
Zapatos	A200	20	89.00	567	59	23	567	59	23
Toallas	A300	80	100.20	900	90	340	900	90	340

Recordemos que la clausula TRANSFORM indica el valor que deseamos visualizar en las columnas que realmente pertenecen a la consulta.

PIVOT **Indica el nombre de las columnas no opcionales**
GROUP BY **Especifica el agrupamiento de todos los registros**

OCASIONES PARA USO DE PIVOT

Por Trimestre	**PIVOT "TRIMESTRE" & DATEPART("q",FECHA)**
Por Meses	**PIVOT FORMAT (FECHA,"mmmm") IN ("ENERO","FEBRERO","MARZO",…."DICIEMBRE")**
Por días	**PIVOT FORMAT(FECHA,"SHORT DATE")**

SUB CONSULTAS

Como sabemos es una instrucción SELECT anidada dentro de una instrucción SELECT, SELECT INTO, DELETE o UPDATE.
Dentro de los terminos Registros en una sub consulta tenemos:

COMPARACION Es una expresión que compara la expresión con el resultado de una sub consulta.

EXPRESION Es una expresión por la que se busca el conjunto resultante de la sub consulta.

INSTRUCCIÓN **E**s una expresión que sigue el mismo formato y reglas que cualquierotra instrucción.

Se puede utilizar una sub consulta en lugar de una expresión en la lista de campos de una isntrucciòn SELECT o una cluausula WHERE o HAVING, el cual debemos saber que se puede emplear el predicado ANY o SOME los cuales son sinónimos para recuperar registros de la consulta principal que satisfahgan la compraraciòn con cualquier otro registro recuperado.

SELECT * FROM VENTAS WHERE Preciounitario>ANY(SELECT Preciounitario FROM MOVIMIENTO WHERE Preciounitario>=20)

SELECT * FROM VENTAS WHERE Codigo IN (SELECT Codigo FROM WHERE Preciounitario>20)

IN

El predicado IN se emplea para recuperar los registros de la consulta principal

NOT IN

Se emplea para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la sub consulta que contenga un valor igual.

EJEMPLO 273

Listar todos los nombres yprecios de todos los artículos cuyo preciounitario es mayor que los del laboratorio TRIFARMA Y MEDIFARMA.

SELECT Nombres,unidamedida AS MEDIDA,preciounitario AS PRECIO FROM ARTICULOS WHERE Laboratorio LIKE “TRIFARMA*” AND Preciounitario>ALL (SELECT Preciounitario FROM ARTICULOS WHERE (Laboratorio LIKE “*MEDIFARMA*”))

EJEMPLO 274

Obtener una lista con el nombre y el precio de venta de todos los productos con el mismo precio que el del medicamento llamado HEPABIONTA.

SELECT DISTINCTROW Nombre,Precioventa FROM PRODUCTOS WHERE (Precioventa = (SELECT Precioventa FROM PRODUCTOS WHERE Nombre=”HEPABIONTA”)

EJEMPLO 275

Ahora nos piden listar todas las empresas proveedoras y todas las unidades de cada área que han realizado un pedido en el segundo trimestre del año 2010.

SET DATE FORMAT dmy
SELECT DISTINCTROW Nombrearea AS AREA, Nombreprov AS PROVEEDOR, Telefono FROM DEPENDENCIAS WHERE (Codarea IN (SELECT DISTINCTROW Codarea FROM VENTAS WHERE Fechaventa>=#04-01-2010# AND fechaventa<#07-01-2010#)

EJEMPLO 276

Seleccionar el nombre de todos los Registros que han reservado por lo menos un pedido de un artículo en una venta.

SELECT Nombres, Apellidos FROM REGISTROS AS B WHERE EXISTS (SELECT * FROM VENTAS AS C WHERE C.Codempleado = B.Codigo)

EJEMPLO 277

Recuperar el código del producto y la cantidad vendida de la tabla VENTAS detallando el nombre del producto de la tabla ARTICULOS.

SELECT DISTINCTROW VENTAS.Codbien, VENTAS.Cantidad, (SELECT DISTINCTROW ARTICULOS.Nombre FROM ARTICULOS WHERE ARTICULOS.Codigo = VENTAS.Codbien) AS DESCRIPCION FROM VENTAS ORDER BY VENTAS.Codbien

EJEMPLO 278

Recuperar todos los datos de los clientes cuya edad supera el promedio entre su talla de zapato.

SELECT Nombres,código FROM CLEINTES CLI WHERE Edad> (SELECT AVG(Talla) FROM CLIENTES WHERE Talla = CLI.Talla)

EJEMPLO 279

Reportar las piezas que se encuentran en la tabla de existencias similar al ejemplo anterior.

SELECT Tipo,modelo,Precioventa FROM Piezas P WHERE EXISTS (SELECT Tipo, Modelo FROM VENTAS WHERE Tipo = P.Tipo AND Modelo = P.modelo)

CONSULTAS CON PARAMETROS

Las consultas con parámetros son aquellas cuyas condiciones de búsqueda se definen mediante parámetros. Si se ejecutan directamente desde la base de datos donde han sido definidas aparecerá un mensaje solicitando el valor de cada uno de los parámetros. Si deseamos ejecutarlas desde una aplicación hay que asignar primero el valor de los parámetros y después ejecutarlas. Su sintaxis es la siguiente:

PARAMETERS nombre1 tipo1, nombre2 tipo2, ... , nombreN tipoN Consulta
En donde:

Parte	Descripción
nombre	Es el nombre del parámetro
tipo	Es el tipo de datos del parámetro
consulta	Una consulta SQL

Puede utilizar nombre pero no tipo de datos en una cláusula WHERE o HAVING.

```
PARAMETERS Precio_Minimo Currency, Fecha_Inicio DateTime;
  SELECT IDPedido, Cantidad FROM Pedidos WHERE Precio > Precio_Minimo  AND
FechaPedido >= Fecha_Inicio;
```

El ejemplo siguiente muestra como utilizar los parámetros en el programa de Visual Basic:

```
Public Sub GeneraConsulta()
Dim SQL As String
Dim Qd As QueryDef
Dim Rs As Recordset
   SQL = "PARAMETERS Precio_Minimo Currency,  Fecha_Inicio DateTime; "
   SQL = SQL & "SELECT IDPedido, Cantidad FROM Pedidos WHERE Precio >
"
   SQL = SQL & "Precio_Minimo AND FechaPedido >= Fecha_Inicio; "
   Set Qd = BaseDatos.CreateQueryDef(MiConsulta, SQL)
   Qd.Parameters!Precio_Minimo = 2
   Qd.Parameters!FechaInicio = #31/12/95#
   Set Rs = Qd.OpenRecordset()
End Sub
```

Ejemplo:

```
PARAMETERS [Escriba los Apellidos:] Text; SELECT * FROM Registros
WHERE [Escriba los Apellidos:] = [Apellidos];
```

La ejecución desde la base de datos solicita al usuario los apellidos del empleado y después muestra los resultados

CONSULTAS DE UNIONES EXTERNAS

Se utiliza la operación UNION para crear una consulta de unión, combinando los resultados de dos o más consultas o tablas independientes. Su sintaxis es:

[TABLE] consulta1 UNION [ALL] [TABLE]
consulta2 [UNION [ALL] [TABLE] consultan [...]]

En donde:

consulta 1,consulta 2, consulta n	Son instrucciones SELECT, el nombre de una consulta almacenada o el nombre de una tabla almacenada precedido por la palabra clave TABLE.

Puede combinar los resultados de dos o más consultas, tablas e instrucciones SELECT, en cualquier orden, en una única operación UNION. El ejemplo siguiente combina una tabla existente llamada Nuevas Cuentas y una instrucción SELECT:

TABLE NuevasCuentas UNION ALL SELECT * FROM Clientes WHERE CantidadPedidos > 1000

Si no se indica lo contrario, no se devuelven registros duplicados cuando se utiliza la operación UNION, no obstante puede incluir el predicado ALL para asegurar que se devuelven todos los registros. Esto hace que la consulta se ejecute más rápidamente. Todas las consultas en una operación UNION deben pedir el mismo número de campos, no obstante los campos no tienen porqué tener el mismo tamaño o el mismo tipo de datos.

Se puede utilizar una cláusula GROUP BY y/o HAVING en cada argumento consulta para agrupar los datos devueltos. Puede utilizar una cláusula ORDER BY al final del último argumento consulta para visualizar los datos devueltos en un orden específico.

SELECT NombreCompania, Ciudad FROM Proveedores WHERE Pais = 'Peru'
UNION SELECT NombreCompania, Ciudad FROM Clientes WHERE Pais = 'Peru'
(Recupera los nombres y las ciudades de todos proveedores y clientes de Peru)

SELECT NombreCompania, Ciudad FROM Proveedores WHERE Pais = 'Peru'
UNION SELECT NombreCompania, Ciudad FROM Clientes WHERE Pais = 'Peru'
ORDER BY Ciudad

(Recupera los nombres y las ciudades de todos proveedores y clientes radicados en Peru, ordenados por el nombre de la ciudad)

SELECT NombreCompania, Ciudad FROM Proveedores WHERE Pais = 'Peru'
UNION SELECT NombreCompania, Ciudad FROM Clientes WHERE Pais = 'Peru'
UNION SELECT Apellidos, Ciudad FROM Registros WHERE Region = 'América del Norte'

(Recupera los nombres y las ciudades de todos los proveedores y clientes de Peru y los apellidos y las ciudades de todos los Registros de América del Norte)

Se utiliza la operación UNION para crear una consulta de unión, combinando los resultados de dos o más consultas o tablas independientes. Su sintaxis es:

consulta1 UNION consulta2 UNION consulta N

En donde:
Consulta1, consulta2, consulta n Son instrucciones SELECT
Puede combinar los resultados de dos o más consultas, instrucciones SELECT, en cualquier orden, en una única operación UNION. Todas las consultas en una operación UNION deben pedir el mismo número de campos, no obstante los campos no tienen porqué tener el mismo tamaño o el mismo tipo de datos.

Se puede utilizar una cláusula GROUP BY y/o HAVING en cada argumento consulta para agrupar los datos devueltos. Puede utilizar una cláusula ORDER BY al final del último argumento consulta para visualizar los datos devueltos en un orden específico.

CRITERIOS DE SELECCIÓN
Antes de comenzar el desarrollo de este apartado hay que recalcar tres detalles de vital importancia. El primero de ellos es que cada vez que se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples; la segunda hace referencia a las fechas se hace necesario particularizarlas según el banco de datos.

Referente a los valores lógicos True o False; en estos sistemas se utilizan los campos BIT que permiten almacenar valores de 0 ó 1, 0 para los valores FALSE, y 1 para los valores TRUE, se puede utilizar la sintaxis siguiente que funciona en todos los casos: si se desea saber si el campo es falso "... CAMPO = 0" y para saber los verdaderos "CAMPO <> 0" o “CAMPO = 1”.

Operadores Lógicos
Los operadores lógicos soportados por SQL son: AND, OR, NOT. A excepción del último todos poseen la siguiente sintaxis:
<expresión1> **operador** <expresión2>
En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:
<expresión1> Operador <expresión2> Resultado
Verdad AND Falso Falso
Verdad AND Verdad Verdad
Falso AND Verdad Falso
Falso AND Falso Falso
Verdad OR Falso Verdad
Verdad OR Verdad Verdad
Falso OR Verdad Verdad
Falso OR Falso Falso

Si a cualquiera de las anteriores condiciones le anteponemos el operador NOT el resultado de la operación será el contrario al devuelto sin el operador NOT.
Ejemplos:

EJEMPLO 280

SELECT * FROM Registros **WHERE** Edad > 25 AND Edad < 50
SELECT * FROM Registros **WHERE** (Edad > 25 AND Edad < 50) OR Gastos = 100
SELECT * FROM Registros **WHERE** NOT Estado = 'Soltero'
SELECT * FROM Registros **WHERE** (Gastos > 100 AND Gastos < 500) OR (Municipio = 'Chimbotano' AND Estado = 'Casado')

Valores Nulos
En muchas ocasiones es necesario emplear como criterio de selección valores nulos en los campos. Podemos emplear el operador IS NULL para realizar esta operación. Por

ejemplo:
SELECT * **FROM**Registros **WHERE** DUI IS NULL

Intervalos de Valores
Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es:

campo [Not] Between valor1 And valor2 (la condición Not es opcional)

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

SELECT * FROM Registros **WHERE** Codigo Between 1154 And 1524

El Operador Like

Se utiliza para comparar un campo con un modelo en una expresión SQL. Su sintaxis es:
campo Like modelo
En donde campo se compara con el modelo expresión. Se puede utilizar el operador Like para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (Ana María), o se puede utilizar una cadena de caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (An%,).

El operador Like se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduce Like 'C*' en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

La cláusula WHERE

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. Después de escribir esta cláusula se deben especificar las condiciones expuestas en los apartados anteriores. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. WHERE es opcional, pero cuando aparece debe ir a continuación de FROM o INNER, LEFT RIGHT JOIN.

EJEMPLO 281

SELECT Apellidos, Salario **FROM**Registros **WHERE** Salario = 21000
SELECT IdProducto, Existencias **FROM**Productos **WHERE** Existencias <= NuevoPedido
SELECT * FROMPedidos **WHERE** FechaEnvio = '19943005'
SELECT Apellidos, Nombre **FROM** Registros **WHERE** Apellidos = 'King'
SELECT Apellidos, Nombre **FROM** Registros **WHERE** Apellidos Like 'S%'
SELECT Apellidos, Salario **FROM** Registros **WHERE** Salario Between 200 And 300
SELECT Apellidos, Salario **FROM** Registros **WHERE** Apellidos Between 'Lon' And 'Tol'
SELECT IdPedido, FechaPedido **FROM** Pedidos **WHERE** FechaPedido Between '19940101' And '19941231'
SELECT Apellidos, Nombre, Ciudad **FROM** Registros **WHERE** Ciudad In ('Sevilla', 'Los Angeles', 'Barcelona')

EJEMPLO 282
Seleccionar todos los nombres de los proveedores, número de ruc y que sean del país de PERU, asi como los de la tabla CLIENTES que sean filtrados por el mismo país.

SELECT Nombre, Ruc, Pais FROM PROVEEDORES WHERE Pais = "PERU" UNION SELECT Clientes, Ruc, Pais FROM CLIENTES WHERE Pais ="PERU"

EJEMPLO 283
Seleccionar todos los nombres y las ciudades de todos los proveedores y clientes radicados en PERU, ordenados por ciudad

SELECT [Datos de Proveedores],ciudad FROM PROVEEDORES WHERE Pais= "PERU" UNION SELECT [Datos de los proveedores], ciudad FROM CLIENTES WHERE Pais ="PERU" ORDER BY Ciudad

EJEMPLO 284
Seleccionar todos los nombres y las ciudades de todos los proveedores y clientes de PERU y los apellidos y ciudades de todos los Registros de AMERICA

SELECT [Nombre de la empresa],Ciudad FROM PROVEEDORES WHERE Pais="PERU" UNION SELECT [Apellidos], Ciudad FROM REGISTROS WHERE Region ="PERU"

Volviendo a dar el repaso respectivo a **CONSULTAS DE UNIÓN INTERNAS, debemos considerar que** as vinculaciones entre tablas se realizan mediante la cláusula INNER que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es:

SELECT campos **FROM** tb1 **INNER JOIN** tb2 **ON** tb1.campo1 comp tb2.campo2

En donde:

tb1, tb2 Son los nombres de las tablas desde las que se combinan los registros.
campo1, campo2 Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos,pero no tienen que tener el mismo nombre.
comp Es cualquier operador de comparación relacional: =, <,<>, <=, =>, ó >.
Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones equivalentes son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar INNER JOIN con las tablas Departamentos y Registros para seleccionar todos los Registros de cada departamento.

Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea LEFT JOIN o todos los Registros (incluso si alguno no está asignado a ningún departamento), en este caso RIGHT JOIN.

También se pueden enlazar varias cláusulas ON en una instrucción JOIN, utilizando la sintaxis
siguiente:

SELECT campos FROM tabla1 INNER JOIN tabla2 ON (tb1.campo1 comp tb2.campo1 AND tb1.campo2 comp tb2.campo2) OR tb1.campo3 comp tb2.campo3

Si empleamos la cláusula INNER en la consulta se seleccionarán sólo aquellos registros de la tabla de la que hayamos escrito a la izquierda de INNER JOIN que contengan al menos un registro de la tabla que hayamos escrito a la derecha. Para solucionar esto tenemos dos cláusulas que sustituyen a la palabra clave INNER, estas cláusulas son LEFT y RIGHT. LEFT toma todos los registros de la tabla ubicada en FROM aunque no tengan ningún registro en la tabla del JOIN. RIGHT realiza la misma operación pero al contrario, toma todos los registros de la tabla del JOIN aunque no tenga ningún registro en la tabla del FROM.

Consultas de Auto combinación
La auto combinación se utiliza para unir una tabla consigo misma, comparando valores de dos
columnas con el mismo tipo de datos. La sintaxis en la siguiente:
SELECT alias1.columna, alias2.columna, ... FROMtabla1 as alias1, tabla2 as alias2
WHERE alias1.columna = alias2.columna AND otras condiciones

Por ejemplo, para visualizar el número, nombre y puesto de cada empleado, junto con el número,nombre y puesto del supervisor de cada uno de ellos se utilizaría la siguiente sentencia:
SELECT t.num_emp, t.nombre, t.puesto, t.num_sup,s.nombre, s.puesto FROMRegistros AS t, Registros AS s WHERE t.num_sup = s.num_emp

AGRUPAMIENTO DE REGISTROS

GROUP BY
Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro. Para cada registro se crea un valor sumario si se incluye una función SQL agregada, como por ejemplo Sum o Count, en la instrucción SELECT. Su sintaxis es:

SELECT campos **FROM** tabla **WHERE** criterio **GROUP BY** campos del grupo
GROUP BY es opcional. Los valores de resumen se omiten si no existe una función SQL agregada en la instrucción SELECT. Los valores Null en los campos GROUP BY se agrupan y no se omiten. No obstante, los valores Null no se evalúan en ninguna de las funciones SQL agregadas.

Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar, y la cláusula HAVING para filtrar los registros una vez agrupados.
Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada.
SELECT IdFamilia, Sum(Stock) AS StockActual **FROM** Productos **GROUP BY** IdFamilia

Una vez que GROUP BY ha combinado los registros, HAVING muestra cualquier registro agrupado

por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING.
HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.
SELECT IdFamilia, Sum(Stock) AS StockActual **FROM**Productos **GROUP BY** IdFamilia

HAVING StockActual > 100 AND NombreProducto Like ‘BOS%’

AVG
Calcula el promedio de un conjunto de valores contenidos en un campo especificado de una consulta. Su sintaxis es la siguiente
Avg (expr)
En donde expr representa el campo que contiene los datos numéricos para los que se desea calcular el promedio o una expresión que realiza un cálculo utilizando los datos de dicho campo. El promedio para Avg se calcula asi: la suma de los valores dividido por el número de valores). La función Avg no incluye ningún campo Null en el cálculo.

SELECT Avg(Gastos) AS Promedio **FROM** Pedidos **WHERE** Gastos > 100

Count
Calcula el número de registros devueltos por una consulta. Su sintaxis es la siguiente
Count(expr) En donde expr contiene el nombre del campo que desea contar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.

Aunque expr puede realizar un cálculo sobre un campo, Count simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función Count no cuenta los registros que tienen campos null a menos que expr sea el carácter comodín asterisco (*). Si utiliza un asterisco, Count calcula el número total de registros, incluyendo aquellos que contienen campos null. Count(*) es considerablemente más rápida que Count(Campo). No se debe
poner el asterisco entre dobles comillas ('*').

SELECT Count(*) AS Total **FROM**Pedidos

Max, Min
Devuelven el máximo o el mínimo de un conjunto de valores contenidos en un campo especifico de una consulta. Su sintaxis es:
Min(expr)
Max(expr)

En donde expr es el campo sobre el que se desea realizar el cálculo. Expr puede incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

SELECT Min(Gastos) AS ElMin **FROM** Pedidos **WHERE** Pais = 'España'
SELECT Max(Gastos) AS ElMax **FROM** Pedidos **WHERE** Pais = 'España'

Sum
Devuelve la suma del conjunto de valores contenido en un campo específico de una consulta. Su sintaxis es:
Sum(expr)
En donde expr representa el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

SELECT Sum(PrecioUnidad * Cantidad) AS Total **FROM** DetallePedido

SUBCONSULTAS
Una subconsulta es una instrucción SELECT anidada dentro de una instrucción SELECT,
SELECT...INTO, INSERT...INTO, DELETE, o UPDATE o dentro de otra subconsulta. Puede utilizar dos formas de sintaxis para crear una subconsulta:
expresión [NOT] IN (instrucción sql) [NOT] EXISTS (instrucción sql)

En donde:
Expression Es una expresión por la que se busca el conjunto resultante de la subconsulta.

Es una instrucción SELECT, que sigue el mismo formato y reglas que cualquier
otra instrucción SELECT. Debe ir entre paréntesis.
Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT o en una cláusula WHERE o HAVING. En una subconsulta, se utiliza una instrucción SELECT para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula WHERE o HAVING.

El predicado IN se emplea para recuperar únicamente aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual. El

ejemplo siguiente devuelve todos los productos vendidos con un descuento igual o mayor al 25 por ciento:

SELECT * FROM Productos **WHERE** IDProducto IN (SELECT Improducto FROM DetallePedido WHERE Descuento = 0.25)
Inversamente se puede utilizar NOT IN para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro. Supongamos que deseamos recuperar todos aquellos clientes que hayan realizado al menos un pedido:
SELECT Clientes.Compañía, Clientes.Teléfono **FROM** Clientes **WHERE EXISTS (SELECT * FROM** Pedidos **WHERE** Pedidos.IdPedido = Clientes.IdCliente **)**

Esta consulta es equivalente a esta otra:
SELECT Clientes.Compañía, Clientes.Teléfono **FROM** Clientes **WHERE** IdClientes IN (SELECT Pedidos.IdCliente FROM Pedidos)

Se puede utilizar también alias del nombre de la tabla en una subconsulta para referirse a tablas listadas en la cláusula FROM fuera de la subconsulta. El ejemplo siguiente devuelve los nombres de los Registros cuyo salario es igual o mayor que el salario medio de todos los Registros con el mismo título. A la tabla Registros se le ha dado el alias T1:

SELECT Apellido, Nombre, Titulo, Salario **FROM** Registros AS T1 **WHERE** Salario = (SELECT Avg(Salario) FROM Registros WHERE T1.Titulo = Registros.Titulo)
ORDER BY Titulo

En el ejemplo anterior, la palabra reservada AS es opcional.

SELECT Apellidos, Nombre, Cargo, Salario **FROM** Registros **WHERE** Cargo LIKE 'Agente Ven%' AND Salario > (SELECT Salario FROM Registros WHERE Cargo LIKE '%Jefe%' OR Cargo LIKE '%Director%')

(Obtiene una lista con el nombre, cargo y salario de todos los agentes de ventas cuyo salario es mayor que el de todos los jefes y directores.)

SELECT DISTINCT NombreProducto, Precio_Unidad **FROM** Productos **WHERE** PrecioUnidad = (**SELECT** PrecioUnidad **FROM** Productos **WHERE** NombreProducto = 'Almíbar anisado')

(Obtiene una lista con el nombre y el precio unitario de todos los productos con el mismo precio que el almíbar anisado.)

EJEMPLO 285
SELECT DISTINCT NombreContacto, NombreCompania, CargoContacto, Telefono **FROM** Clientes **WHERE** IdCliente IN (SELECT DISTINCT IdCliente FROM Pedidos

WHERE FechaPedido < '19930701')
(Obtiene una lista de las compañías y los contactos de todos los clientes que han realizado un
pedido en el segundo trimestre de 1993.)

SELECT Nombre, Apellidos **FROM** Registros AS E **WHERE EXISTS (SELECT * FROM** Pedidos AS O **WHERE** O.IdEmpleado = E.IdEmpleado **)**

(Selecciona el nombre de todos los Registros que han reservado al menos un pedido.)

SELECT DISTINCT Pedidos.Id_Producto, Pedidos.Cantidad, (SELECT roductos.Nombre FROM Productos WHERE Productos.IdProducto = edidos.IdProducto) AS ElProducto **FROM** Pedidos **WHERE** Pedidos.Cantidad = 150 **ORDER BY** Pedidos.Id_Producto

(Recupera el Código del Producto y la Cantidad pedida de la tabla pedidos, extrayendo el nombre del producto de la tabla de productos.)

SELECT NumVuelo, Plazas **FROM** Vuelos **WHERE** Origen = 'Peru' AND Exists (SELECT T1.NumVuelo FROM Vuelos AS T1 WHERE T1.PlazasLibres > 0 AND T1.NumVuelo=Vuelos.NumVuelo)

(Recupera números de vuelo y capacidades de aquellos vuelos con destino Peru y plazas libres

EJEMPLO 286

Acceder a la base de datos externa llamada REGISTRO.MDB, y seleccionar los registros de la tabla clientes cuyo codigo empiece con A

SELECT Codigo FROM CLIENTES IN REGSITRO.MDB WHERE Codigo LIKE “A*”

EJEMPLO 287

Recuperar los datos de la tabla PARADOX Version 3.X , con las mismas condiciones del ejemplo anterior

Pues, debemos saber que para recuperar los datos de 3.X hay que sustituir por PARADOX 4.X

SELECT Codigo FROM CLIENTES IN “C:\PARADOX\BD\VENTAS” “PARADOX 4.X; “ WHERE Codigo LIKE “A*”

OMITIR PERMISOS DE EJECUCION

En entornos de base de datos con permisos de seguridad para grupos de trabajo se pueden emplear la clausula WITH OWNERACCES OPTION para que el usuario actual adquiera los derechos a la hora de actualizar y ejecutar la consulta.
Pues en este caso ejemplo el cual vamos a considerar es que primero vamos a declarar el acceso al fichero de grupo de trabajo (generalmente SYSTEM.MDA o SYSTEM.MDW) de la base de datos vigente.

INSTRUCCION SQL WITH OWNERACCES OPTION
SELECT Apellido, Nombre,Gastos FROM REGISTROS ORDER BY Apellido WITH OWNERACCES OPTION

PROCEDURE

Esta clausula es poco usual y se usa para crear una consulta a la misma vez que se ejecuta.
SINTAXIS:
PROCEDURE Nombre de la consulta Parametro1 Tipo1, Tipo2……tipo N

¿Cómo vamos, hasta el momento entiendes. Entonces ejemplo para el operador?

Reportar todos los productos y los datos de los proveedores de cualquier combinación de partes suministradas por la empresa cuyo nombre empiece por la letra “H” y el precio de venta sea mayor a 10,000.

EJEMPLO 288

Devolver un conjunto de resultados con el valor NULL, de la tabla llamada DEPARTAMENTO, especificando en las SUB CONSULTAS, que sigue evaluando como TRUE al emplear EXISTS

Use Master
SELECT Codigo, Nombres FROM RRHH.Departamento WHERE EXISTS (SELECT NULL) ORDER BY 2

EJEMPLO 289

Comparar consultas mediante EXISTS e IN, el cual se sugiere que comparemos dos consultas que sean semanticamente equivalente y en la primera consulta se emplee EXISTS y en la segunda IN, teniendo en cuenta que tenemos dos tablas lladas PERSONAL y DETALLE, cuyas tablas estan contenidas dentro de la base de datos RRHH, y a esta busqueda debemos considerar que liste solo los de apellidos PEREDA

EMPLEANDO EXISTS

SELECT A.Nombres,A.Apellidos FROM RRHH.PERSONAL AS A WHERE EXISTS (SELECT * FROM RECURSOS.Detalle AS B WHERE A.Ubicacion =B.Codubicacion AND A.Apellidos =”PEREDA”

AHORA EMPLEANDO IN

SELECT A.Nombres,A.Apellidos FROM RRHH.PERSONAL AS A WHERE A. Apellidos IN (SELECT A.Apellidos FROM RECUROS.DETALLE AS B WHERE A.Ubicacion =B.codubicacion AND A.Apellidos =”PEREDA”

EJEMPLO 290

Ahora en el ejemplo siguiente debemos mostrar dos consultas para lozalizar calles cuyo nombre sea el mismo que la de un proveedor.

SELECT DISTINCT B.Nombre, FROM GUIAS.CALLE AS B WHERE EXISTS (SELECT * FROM CONTROL.REGISTRO AS D WHERE B.Nombre = D.Nombre

Observemos que hemos trabajado en el ejemplo anterior con dos bases de datos diferentes donde en cada base de datos esta cada una de las tablas que hemos trabajado.

EJEMPLO 291
Localizar todos los Registros de las diferentes áreas que comiencen con M

SELECT B.Nombres,B.apellidos,C.area FROM PERSONAL.PADRON AS B JOIN DEPARTAMENTO AS C ON C.AREA =B.AREA WHERE EXISTS (SELECT * FROM DEPARTAMENTO.Ubicacion AS D JOIN DEPARTAMENTO.DEPART AS EDH WHERE C.Codigo =EDH.Codigo AND D.Nombre LIKE “M%”

DATEDIFF (Transact TSQL)

Se utiliza la operación UNION para crear una consulta de uni

Del capitulo V sobre las funciones de fecha tenemos :
Devuelve el recuento (entero con firma) de los límites *datepart* que se han cruzado entre los valores *startdate* y *enddate* especificados.

Sintasix:

DATEDIFF (datepart , startdate , enddate)
datepart = Obtener la diferencia en (Años, Meses, Dias, Horas, Minutos o Segundos) entre dos fechas.
startdate = Fecha Inicial
enddate = Fecha Final
Ejemplos en sql:
select DateDiff(minute, '2009/08/01', '2009/08/05') as DiferenciaMinutos
select DateDiff(hour, '2009/08/01', '2009/08/05')as DiferenciaHoras
select DateDiff(day, '2009/08/01', '2009/08/05') as DiferenciaDias
select DateDiff(second, '2009/08/01', '2009/08/05') as DiferenciaSegundos
select DateDiff(month, '2009/08/01', '2009/08/05') as DiferenciaMes
select DateDiff(year, '2008/08/01', '2009/08/05') as DiferenciaAnnio

REPASO

EJEMPLO

Recupre todos los registros de los proveedores con sus países respectivos, considerando que sean exclusivamente de PERU, asimismo los clientes de PERU, considerando qe ambos están relacionados

SELECT Nombre, Ruc, Pais FROM PROVEEDORES WHERE Pais=”PERU” UNIO SELECT Clientes,Ruc,Pais FROM CLIENTES WHERE Pais=”PERU”;
GUNE

SQL DINAMICO T-SQL

Transact SQL (*TSQL*) es el lenguaje que usamos para escribir : Store Procedures – Triggers – Querys – Etc.

Sin dudas que TSQL no dispone de las mismas habilidades y potencia que puede tener un lenguaje como C# o VB.NET.

NUMERANDO REGISTROS (ROWLD)

En muchas ocasiones es necesario poder obtener una columna con el número de registro o también poder generar un ranking. Hasta SQL2000 este tipo de operaciones no eran tan simples de realizar y no disponíamos de instrucciones directas. Apartir de TSQL 2005 disponemos de una serie de instrucciones las cuales nos hacen el trabajo mucho mas simple y eficiente. Veamos de qué se trata ello.

Row_Number: Esta nueva función de TSQL nos permitirá numerar los resultados de una query.

A las funciones de numeración como Row_number() le podemos agregar la cláusula ***Partition*** la cual nos permitirá numerar pero haciendo un corte y reiniciando el numerador a partir de ese corte. En el siguiente ejemplo hacemos uso de ello:

Control de Errores:

Apartir de Tsql 2005 incorpora al control de errores los bloques TRY..CATH. La administración de errores de esta manera es conocida por los desarrolladores de la actualidad ya que .NET administra de la misma manera.

ERROR_LINE() : Retorna el número de línea donde se genero el error. El resultado es NULL cuando el error se genero fuera del bloque Try.

ERROR_MESSAGE(): Retorna el texto del error.

ERROR_PROCEDURE(): Retorna el nombre del Procedimiento Almacenado o el Trigger desde donde se ha generado el error en el bloque Try Catch

ERROR_SEVERITY(): Retorna la severidad del error

MAX para las columnas dinámicas

Recordemos que en SQL se encuentra una diversidad de caracteristicas tales en etse caso como MAX que son Registros para los tipos de datos VARCHAR, NVARCHAR y VARBINARY. La idea de esta característica es poder ampliar la capacidad de estos tipos de datos. En sql2000 estos tipos de datos disponían un máximo de 8000 y 4000 (para los Nvarchar), con MAX se puede almacenar hasta 2GB de información en estos tipos de datos. La idea seria reemplazar donde se pueda este tipo de datos por los viejos IMAGE , TEXT y NTEXT.

DDL TRIGGERS

En la versión 2000 de MS SQL-Server solo disponíamos de triggers (Desencadenadores) DML para las instrucciones Insert – Update y Delete. Apartir de SQL 2005 incorpora un nuevo tipo de triggers llamados DDL Triggers, los cuales pueden tener efecto sobre las instrucciones DDL como por ej: CREATE TABLE.

Los DDL triggers son una nueva herramienta muy poderosa de verdad, la cual nos permitirá desde ahora poder tener control sobre las sentencias DDL como así también poder realizar auditorias a las mismas.

TRIGGERS SQL SERVER 2008

Un ejemplo de triggers para el control de Stock

```
USE master
GO
IF EXISTS (SELECT NOMBRES FROM sys.databases
        WHERE nombres = 'StockArticulos')
  BEGIN
    DROP DATABASE StockArticulos
  END
CREATE DATABASE StockArticulos
GO
USE StockArticulos
GO
```

```
CREATE TABLE dbo.Articulos (ID INT PRIMARY KEY,
NOMBRE VARCHAR(100),STOCK DECIMAL(18,2))
GO
CREATE TABLE dbo.Movimientos
(TRANSACCION INT,FECHA DATEDEFAULT(GETDATE()),ARTICULO_ID INT FOREI
GN KEY
REFERENCES DBO.ARTICULOS(ID),CANTIDAD DECIMAL(18,2),
TIPO CHAR(1) CHECK (TIPO ='I' OR TIPO = 'O'))
GO
– Insertamos registros a la tabla Articulos
INSERT
INTO dbo.Articulos(ID,NOMBRE,STOCK) VALUES (1,'Monitores',0),(2,'CPU',0),(3,'Mou
se',0)
GO
– Creamos los triggers para tener actualizado los articulos
CREATE TRIGGER dbo.MovimientosInsert ON dbo.Movimientos
FOR INSERT
AS
BEGIN
  — No retorna el mensaje de cantidad de registros afectados
  SET NOCOUNT ON
  UPDATE DBO.ARTICULOS
  SET STOCK = STOCK + T.PARCIAL
  FROM DBO.ARTICULOS A
  INNER JOIN
  ( SELECT ARTICULO_ID,
    SUM(CASE WHEN TIPO='I' THEN CANTIDAD ELSE -CANTIDAD END)
    AS PARCIAL FROM INSERTED
    GROUP BY ARTICULO_ID
  ) T
  ON
  A.ID = T.ARTICULO_ID
END
GO
CREATE TRIGGER dbo.MovimientosDelete ON dbo.Movimientos
FOR DELETE
AS
BEGIN
  — No retorna el mensaje de cantidad de registros afectados
  SET NOCOUNT ON
  UPDATE dbo.Articulos
  SET STOCK = STOCK – T.PARCIAL
  FROM dbo.Articulos A
  INNER JOIN
  ( SELECT ARTICULO_ID,
    SUM(CASE WHEN TIPO='I' THEN CANTIDAD ELSE -CANTIDAD END)
    AS PARCIAL FROM DELETED
    GROUP BY ARTICULO_ID
  ) T
  ON
  A.ID = T.ARTICULO_ID
END
GO
– Probemos el ejercicio
– Mostremos el Stock actual
SELECT A.ID,A.NOMBRE,A.STOCK FROM dbo.Articulos A
```

– Insertemos un registro para el articulo 1

```
INSERT INTO dbo.Movimientos
(TRANSACCION,ARTICULO_ID,FECHA,CANTIDAD,TIPO)
VALUES (1,1,GETDATE(),100,'I')
```

– Mostremos el Stock actual para el ID 1

```
SELECT A.ID,A.NOMBRE,A.STOCK FROM dbo.Articulos A WHERE A.ID = 1
```

– Insertemos otros registros

```
INSERT INTO dbo.Movimientos
(TRANSACCION,ARTICULO_ID,FECHA,CANTIDAD,TIPO)
VALUES (2,1,GETDATE(),10,'I'), (3,1,GETDATE(),5,'O'), (4,2,GETDATE(),5,'I')
```

– Mostremos el Stock actual para el ID 1

```
SELECT A.ID,A.NOMBRE,A.STOCK FROM dbo.Articulos A WHERE A.ID = 1
```

– Eliminemos la transaccion (1) de cantidad = 100

```
DELETE FROM dbo.Movimientos WHERE TRANSACCION = 1
```

– Eliminemos la transaccion (3) de cantidad = 5

```
DELETE FROM dbo.Movimientos WHERE TRANSACCION = 3
```

– Mostremos el stock actual de la tabla Articulos

```
SELECT A.ID,A.NOMBRE,A.STOCK FROM dbo.Articulos A
```

– Eliminamos todos los movimientos realizados

```
DELETE FROM dbo.Movimientos
```

– Deshabilitar los triggers

```
ALTER TABLE dbo.Movimientos DISABLE TRIGGER ALL
```

– Mostremos lo que pasa se insertamos un registro en la tabla

– Movimientos que tiene deshabilitados los triggers

```
INSERT INTO dbo.Movimientos
(TRANSACCION,ARTICULO_ID,FECHA,CANTIDAD,TIPO)
VALUES (1,1,GETDATE(),100,'I')
```

– Mostremos el stock actual de la tabla Articulos

```
SELECT A.ID,A.NOMBRE,A.STOCK FROM dbo.Articulos A
```

EJEMPLO 292

Diga usted como mostrar una cadena de caracteres con la isntrucción EXEC

```
DECLARE @SQL NVARCHAR(100)
        SET @SQL =''SELECT Codigo,nombre FROM REGISTRO"
        Exec (@SQL)
```

También con SQL dinamico podemos ejecutar sentencias de tipo DDL como CREARE TABLE

```
DECLARE @SQL ="CREATE TABLE PERSONAL ( ID INT IDENTITY, DETALLE VARCHAR(40))"
        EXEC (@SQL)
        SET @SQL ="SELECT * FROM PERSONAL"
        EXEC (@SQL)
```

El principal inconveniente de trabajar con la instrucción EXEC es que no permite el uso de parametros, además el uso de la instrucción EXEC es menos eficiente en terminos de rendimiento que SP_ECECUTESQL.

EL PROCEDIMIENTO ALMACENADO SP_EXECUTESQL

Para ejecutar SQL Dinamico se recomienda utilizar el procedimiento almacenado SP_EXECUTESQL en lugar de una isntrucción EXECUTE.

```
DECLARE @SQL NVARCHAR(100)
        SET @SQL = "SELECT Codigo, nombre,ciudad FROM REGISTRO"
        EXEC SP_EXECUTE@SQL
```

EJEMPLO 293

Ahora como ya hemos visto el uso y empleo de TRIGGERS, diseñaremos la forma que avise a la base WEB con un email cuando el usuario se da de alta en nuestra WEB.

Pues lo primero es declarar dos variables, una de ellas es para el mensaje que se enviara al email y la otra sera para obtener el ID del registro para luego conectarla al mensaje y enviarlo.

EJEMPLO 294 SUB CONSULTAS ANIDADAS

Mostrar todas las ventas que se hayan ejecutado y adquirido el articulo "PERNOS 3/8", para esto vamos a considerar que tenemos tres tablas las cuales son : VENTAS, MOVIMIENTOS DE VENTAS y PRODUCTOS.

SELECT IdCodigo,fecha, UnidadMedida FROM VENTAS WHERE Idcodigo IN (SELECT Idocidog FROM [MOVIMIENTOS DE VENTAS] WHERE Idcodigo IN (SELECT Codigo FROM PRODUCTOS WHERE Descripcion ="PERNOS 3/8"

EJEMPLO 295 SUB CONSULTAS ANIDADAS

Diga usted ahora como mostrar todos los clientes que compraron los pernos de 3/8

SELECT Nombres,apellidos, FROM CLIENTES WHERE Idcodcli IN (SELECT Idcodcli FROM VENTAS WHERE Idcodbien FROM ARTICULOS WHERE Descripcion ="PERNOS 3/8"

EJEMPLO 296 (REPASO)

Listar cuantos articulos existen por categoria, debiendo considerar que tenemos dos tablas llamadas ALMACEN y VENTAS

SELECT b.Categoria, COUNT(c.Codigo) AS CANTIDAD FROM ALMACEN AS b INNER JOIN VENTAS AS c ON (b.Idcateg = c.Codcategoria) GROUP BY c.Codcategooria, c.Nombrecategoria

EJEMPLO 297

Listar la cantidad de ventas efectuadas por cada producto que son del almacen ALMACEN SIABA, considerando qque tenemos dos tablas denominadas VENTAS y ARTICULO.

SELECT b.Idcodigo AS "CODIGO DE PRODUCTO", COUNT(NUMDOC) AS CANTIDAD FROM VENTAS AS b INNER JOIN ARTICULO AS c ON b.Idcodigo = c.Codigo AND c.Almacen ="ALMACEN SIABA" GROUP BY b.Idcodigo

EJEMPLO 298

Listar todos los nombres de los vendedores con sus respectivos montos de cantidades de ventas efectuadas por cada uno, para esto damos a conocer dos tablas llamadas OPERARIOS y VENTAS.

SELECT b.Nombres, COUNT(c.Codvendedor) AS "NUMERO DE VENTAS" FROM OPERARIOS AS b INNER JOIN VENTAS AS c ON (b.Idcod = c.Codvendedor) GROUP BY c.Codvendedor, b.Nombres ORDER BY 1

EJEMPLO 299

Mostrar todos los datos de los alumnos y de los cursos que llevan por cada ciclo semestral de estudios dentro de la universidad.

SELECT b.Nombres, b.Apellidos, COUNT(c.Codcuros) AS "CANTIDAD DE CURSOS" FROM MATRICULADOS AS c INNER JOIN ALUMNOS AS b

CREATE RULE (Transact-SQL)

Crea un objeto denominado regla. Cuando se enlaza a una columna o a un tipo de datos de alias, la regla especifica los valores aceptables que se pueden insertar en esa columna.

Argumentos

schema_Nombres

Es el nombre del esquema al que pertenece la regla.

rule_Nombres

Es el nombre de la nueva regla. Los nombres de las reglas deben ajustarse a las reglas de los identificadores. La especificación del propietario de la regla es opcional.

condition_expression

Es la condición o condiciones que definen la regla. Una regla puede ser cualquier expresión válida en una cláusula WHERE y puede incluir elementos como operadores aritméticos, operadores relacionales y predicados (por ejemplo, IN, LIKE, BETWEEN). Una regla no puede hacer referencia a columnas u otros objetos de base de datos. Se pueden incluir funciones integradas que no hagan referencia a objetos de base de datos. No es posible utilizar funciones definidas por el usuario.

condition_expression incluye una variable. El carácter arroba (@) precede a cada variable local. La expresión hace referencia al valor especificado con la instrucción UPDATE o INSERT. Se puede utilizar cualquier nombre o símbolo para representar el valor cuando se crea la regla, pero el primer carácter debe ser la arroba (@).

Nota:

Evite crear reglas en expresiones que utilicen tipos de datos de alias. Aunque es posible crear reglas en expresiones que utilicen tipos de datos de alias, después de enlazar las reglas a las columnas o a los tipos de datos de alias, cuando se hace referencia a las expresiones, éstas no se compilan.

Notas

CREATE RULE no se puede combinar con otras instrucciones Transact-SQL en un único lote. Las reglas no se aplican a los datos ya existentes en la base de datos en el momento en que se crean las reglas y no se pueden enlazar a los tipos de datos del sistema. Para obtener más información,

Una regla sólo se puede crear en la base de datos actual. Una vez creada la regla, ejecute **sp_bindrule** para enlazarla a una columna o a un tipo de datos de alias. Una regla debe ser compatible con el tipo de datos de la columna. Por ejemplo, "@value LIKE A%" no se puede utilizar como regla para una columna numérica. Una regla no se puede enlazar a una columna con un tipo de datos **text**, **ntext**, **image**, **varchar(max)**, **nvarchar(max)**, **varbinary(max)**, **xml**, definido por el usuario CLR o **timestamp**. Una regla no se puede enlazar a una columna calculada.

Incluya las constantes de fecha y de caracteres entre comillas simples (') y preceda las constantes binarias de 0x. Si la regla no es compatible con la columna a la que se ha enlazado, el Motor de base de datos de SQL Server devuelve un mensaje de error cuando se inserta un valor, pero no cuando se enlaza la regla.

Una regla enlazada a un tipo de datos de alias sólo se activa cuando se intenta actualizar o insertar un valor en una columna de la base de datos del tipo de datos de alias. Dado que las reglas no prueban las variables, no asigne un valor a una variable de tipo de datos de alias que sería rechazada por una regla enlazada a una columna del mismo tipo de datos.

Para obtener un informe sobre una regla, utilice **sp_help**. Para que se muestre el texto de una regla, ejecute **sp_helptext** con el nombre de la regla como parámetro. Para cambiar el nombre de una regla, utilice **sp_reNombres**.

Una regla debe quitarse mediante DROP RULE antes de crear una nueva con el mismo nombre y debe cancelarse el enlace mediante **sp_unbindrule** antes de quitarla. Utilice **sp_unbindrule** para cancelar el enlace de una regla a una columna.

Una nueva regla se puede enlazar a una columna o tipo de datos sin cancelar el enlace de la anterior; la nueva regla anula la anterior. Las reglas enlazadas a columnas siempre tienen prioridad sobre las enlazadas a tipos de datos de alias. Enlazar una regla a una columna sustituye una regla ya enlazada al tipo de datos de alias de esa columna. Sin embargo, el enlace de una regla a un tipo de datos no sustituye una regla enlazada a una columna de ese tipo de datos de alias. La tabla siguiente muestra la prioridad cuando se enlazan reglas a columnas y a tipos de datos de alias en los que ya existen reglas.

EJEMPLO 300

Crear una tabla llamada ESTACION donde ingresen los vehiculos en una playa de estacionamiento, donde el cual vamos a asociar reglas dentro de ellas.

```
IF OBJECT_Id (“VEHICULOS”) IS NOT NULL
        DROP TABLE VEHICULOS;
- - Eliminamos las reglas creadas en la tabla
IF OBJECT_Id (“RG_PATENTE_PATRON”) IS NOT NULL
        DROP RULE RG_PATENTE_PATRON;
IF OBJECT_Id(“RG_FECHA_HORA”) IS NOT NULL
        DROP RULE RG_FECHA_HORA;
```

Ahora creamos la tabla:

```
CREATE TABLE VEHICULOS(
Patente         char(10) NOT NULL,
Tipo            char(1), - - “a” = auto, “m”= moto
Domicilio       varchar(30) NOT NULL,
Fechaing        datetime NOT NULL
);
```

EJEMPLO 301

Crear una regla en una tabla llamada VEHICULOS para restingir los valores que se pueden ingresar en un campo PATENTE que contenga tres letras seguidas de tres digitos

```
CREATE RULE RG_PATENTE_PATRON AS @patente LIKE “[A-Z][A-Z][A-Z][0-9][0-9][0-9]”
```

EJEMPLO 302

Ahora asociaremos la regla creada al campo PATENTE de la tabla anterior llamada VEHICULOS

```
SP_BRINRULE RG_PATENTE, “Vehiculos.patente”;
```

EJEMPLO 303

Crear una regla que controle los valores para el campo TIPO para que solamente puedan ingresar las letras M y H

```
CREATE RULE RG_VEHICULOS_TIPO AS @TIPO IN (“M”, “H”);
```

EJEMPLO 304

Ahora asociaremos la regla creada en la tabla anterior

```
SP_BRINRULE RG_VEHICULOS_TIPO, “Vehiculo.tipo”;
```

EJEMPLO 305

Crear una regla llamada RG_VEHICULOS_TIPO4 que controle los valores para el campo TIPO para que solamente puedan ingresar los caracteres M,H y C

```
CREATE RULE RG_VEHICULOS_TIPO4 AS @tipo IN (“M”,”H”,”C”);
```

EJEMPLO 306

Si la asociamos a un campo que ya tiene asociada otra regla, la nueva regla reemplaza a la asociacion anterior. Asocie la regla creada en el punto anterior al campo TIPO

```
SP_BRINRULE RG_VEHICULOS_TIPO4, “Vehiculos.Tipo”;
```

EJEMPLO 307

Crear una regla que permita ingresar ingresar fechas menores o iguales a la del dia.

```
CREATE RULE RG_menor_fecha AS @Fecha <=GETDATE();
```

EJEMPLO 308

Ahora intente establecer una restricción CHECK que asegure que la fecha y hora de llegada no sea posterior a la hora actual.

```
ALTER TABLE VEHICULOS
ADD CONSTRAINT CHK_VEHICULOS_LLEGADA CHECK (Fechaing< = GETDATE());
SP_HELPCONSTRAINT VEHICULOS;
```

EJEMPLO 309

Crear una regla para establecer un patron para los valores que se ingresan en el campo PATENTE dos letras seguidas de dos cifras

CREATE RULE RG_Vehiculos_Patente AS @Valor LIKE “[A-Z][A-Z][0-9][0-9]”;

EJEMPLO 310
Asocie la regla anterior al campo patente

EXEC SP_BRINDRULE RG_Vehiculos_patente, “Vehiculos.patente”;

EJEMPLO 311
Crear un valor predeterminado para el campo PATENTE (“MHPP”) llamado VP_VEHICULO_PATENTE

CREATE DEFAULT VP_VEHICULO_PATENTE AS “MHPP”;

EJEMPLO 312
Asociar al campo patente

EXEC SP_BRINDEFAULT VP_Vehiculo_patente, “Vehiculo.patente”;

EJEMPLO 313
Crear un valor predeterminado con la cadena ?? llaamdo VP_datoincognito

CREATE DEFAULT VP_Datoincognito AS “**??**”;

EJEMPLO 314
Asocie el valor predeterminado anterior al campo domicilio

EXEC SP_BRINDEFAULT
VP_datoincognito, “Vehiculo.domicilio”

ELIMINAR VALORES PREDETERMINADOS

EJEMPLO 315
Recordemos que si eliminamos una tabla, las asociaciones de reglas y valores predeterminados de sus campos desaparecen, pero las reglas y valores predeterminados siguen existiendo. Entonces ¿Cómo eliminamos un valor predeterminado?

IF OBJECT_ID(“Vp_vehiculo.patente”) IS NOT NULL
DROP DEFAULT Vp_vehiculo_patente;

EJEMPLO 316
Supongamos que tenemos una tabla llamada LIBRERÍA donde se nos piden crear una regla para impedir que se ingresen valores negativos en la columna PRECIO y luego asociemos la regla al campo precio

CREATE RULE RG_POSOTIVO
AS @valor>=0;
EXEC SP_BRINDRULE RG_POSITIVO, “librería.precio”;

EJEMPLO 316
Ahora del mismo ejemplo anterior asociaremos al campo cantidad

EXEC SP_BRINDRULE RG_POSITIVO, “librería.cantidad”;

EJEMPLO 317-318
En este ejemplo crearemos un valor predeterminado para que almacene el valor cero llamado VP_CERO en la columna precio para luego asociarlo

CREATE DEFAULT VP_CERO AS 0;
EXEC SP_BRINDEFAULT VP_CERO, “librería.precio”;

EJEMPLO 319 -320
De la misma manera ahora crearemos un valor predeterminado con la cadena MOVIMIENTO llamado VP_MOVIMIENTO, ahora asocielo al campo AUTOR y luego al campo PAGINAS

CREATE DEFAULT VP_MOVIMIENTO AS “Movimiento”;
EXEC SP_BRINDEFAULT VP_MOVIMIENTO, “librería.autor”;
EXEC SP_BRINDEFAULT VP_MOVIMIENTO, “librería.paginas”;

EJEMPLO 321
Ahora visualice usted y vea las reglas y valores predeterminados con SP_HELP

SP_HELP;

EJEMPLO 322
Diga usted como visualizar las reglas y valores predeterminados asociados a la tabla librería.

SP_HELPCONSTRAINT librería;

EJEMPLO 323
Ahora diga usted como quitar la asociacion del valor predeterminado VP_CERO al campo precio que se encuentra en la tabla libreria

EXEC SP_UNBNIDEFAULT “librería,precio”;

EJEMPLO 324
Verificar que el valor predeterminado VP_CERO existe aun

SP_HELP Vp_cero;

EJEMPLO 325
Quite la asociacion del valor predeterminado VP_CERO al campo CANTIDAD y luego verifique que ya no existe la asociacion de este valor predeterminado en la base de datos.

EXEC SP_UNBINDEFAULT “librería.cantidad”;
SP_HELPCONSTRAINT librería;

CONSULTAS RECURSIVAS USANDO SQL SERVER

Las jerarquías son muy comunes en los sistemas de información, se utilizan para organizar elementos por orden de importancia o por tamaño.

Para almacenar este tipo de estructuras en la base de datos se utiliza lo que se conoce como lista de adyacencia, en donde cada nodo de la jerarquía guarda el Id del nodo padre.

Table - dbo.Registro

Id	Nombres	codigoubica	Gastos
100	Harumi Pereda	A100	230.00
101	Mariluisa Pascal	E300	56.00
102	Cesar Pereda	F560	800.00
103	Carlos Lopez	H600	127.00
104	Wilmer Pereda	A100	890.00
105	Maria Torres	A100	234.00
106	Juana Torres	A200	657.00
107	Cintia Pereda	F560	233.00
108	Julia Lopez	E300	978.00
109	Maria Lopez	A100	123.00
110	Gune Pereda	H600	456.00

Esta forma de almacenar es bastante eficiente en cuanto a almacenamiento y flexibilidad, puesto que no tiene limitaciones en cuanto a la profundidad del árbol, ni tampoco en la cantidad de hijos que puede tener cada nodo. Las listas de adyacencia no contienen información repetida entre los nodos lo que permite realizar modificaciones a partes de la jerarquía sin afectar al resto de los nodos.

Las desventajas aparecen cuando se trata de recuperar la información de la jerarquía, puesto que no es posible realizar una sola consulta a la base de datos que devuelva toda o parte de la jerarquía.

Definiendo la siguiente función:

```
CREATE FUNCTION [dbo].[GetIdRegistros](@Codigoubica int)
RETURNS @retRegistros TABLE (Id int)
AS
BEGIN
   DECLARE @RegistrosDirectos_Id int

   DECLARE RegistrosDirectos CURSOR STATIC LOCAL FOR
```

```
    SELECT Id FROM dbo.Registro WHERE Codigoubica=@Codigoubica

  INSERT INTO @retRegistros VALUES(@Codigoubica)

  OPEN RegistrosDirectos

  FETCH NEXT FROM RegistrosDirectos
    INTO @RegistrosDirectos_Id

  WHILE (@@FETCH_STATUS = 0)
  BEGIN
    INSERT INTO @retRegistros
      SELECT * FROM dbo.GetIdRegistros(@RegistrosDirectos_Id)

    FETCH NEXT FROM RegistrosDirectos
      INTO @RegistrosDirectos_Id
  END

  CLOSE RegistrosDirectos
  DEALLOCATE RegistrosDirectos

  RETURN
END
```

Esta función es capaz de devolver todos los identificadores de los Registros que reportan directa o indirectamente a una persona. Utilizando el resultado de esta función es posible responder a la consulta “Cual es el Gasto total de Mariluisa Pascal”, primero buscando el Id de Mariluisa Pascal, luego llamando a la función GetIdRegistros con ese Id como parámetro, realizando un join entre el resultado de la función la tabla empleado, para finalmente obtener la suma de los Gastos de los Registros.

```
elect sum(Gastos) as GastosTotal From GetIdRegistros(
(select Id from dbo.Registro WHERE Nombre='Mariluisa Pascal')
) emp inner join dbo.Registro e on emp.Id = e.Id
```

A partir de SQL Server 2005 se introdujo una nueva funcionalidad al motor que facilita aun más el trabajo con jerarquías. Esta nueva funcionalidad se llama Recursive Common Table Expressions (Recursive CTE) y permite realizar una consulta recursiva sin tener que definir una función para realizar dicha operación.

```
WITH CTE_Registros (Id, Gastos)
AS
(
   SELECT Id, Gastos FROM Empleado WHERE Nombre = 'Mariluisa Pascal'
UNION ALL
   SELECT e.Id, e.Gastos FROM Empleado e
   INNER JOIN CTE_Registros cte ON e.Codigoubica = cte.Id
)

SELECT sum(Gastos) FROM CTE_Registros
```

Esta consulta se resuelve de forma mucho más eficiente pues el motor entiende de mejor manera que es lo que se quiere hacer, con lo que se obtienen mejoras considerable. Algunas pruebas realizadas en SQL 2005 utilizando las 2 técnicas descritas mostraron que las consultas que utilizaban los Recursive CTEs demoraban 4 a 10 veces menos que las que utilizaban las técnicas descritas para la versión para 2000.

La versión de SQL Server 2008 trae aun más mejoras al desempeño y a la facilidad de uso de las consultas sobre jerarquías. En un próximo post escribiré sobre el nuevo tipo de datos llamado HierarchyID, explicaré su funcionamiento y como impacta al almacenamiento y manipulación de jerarquías en la base de datos.

OrdPath

Los OrdPaths son una forma de codificar el camino que hay desde un nodo hasta la raíz del árbol al que pertenece. En esta codificación se guarda la posición que tiene cada hijo en el árbol, obteniendo algo similar a esto: /1/2, lo que significa que este es un nodo que está en el tercer nivel de la jerarquía (contando al nodo raíz), es el segundo hijo del primer hijo de la raíz.

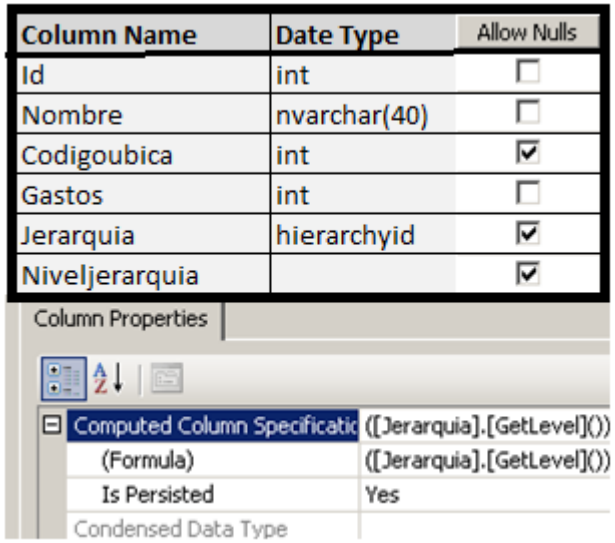
Otra ventaja de esta codificación es que al ordenar los nodos del árbol utilizando los OrdPaths se obtienen los nodos del árbol ordenados en profundidad primero (Depth-

first) y si se ordenan utilizando el nivel y el OrdPath en un índice compuesto, se obtienen los nodos ordenados en orden ancho primero (Breadth-first).

Pero esta codificación retorna una cadena de caracteres, lo que no es muy eficiente en cuanto a almacenamiento y distribución. Es por eso que se diseñó una forma de guardar los OrdPaths en formato binario el cual es realmente eficiente en cuanto a requerimientos de espacio y tiene gran ventaja de mantener todas las propiedades antes mencionadas.

Como trabajar con los HierarchyId

Siguiendo con el ejemplo del artículo anterior, se toma la tabla empleado y se le agrega una columna de tipo HierarchyId y opcionalmente una columna calculada que describe el nivel de la jerarquía.



Ahora se deben asignar los valores correspondientes al campo “Jerarquia”. Para esto se deben utilizar los métodos definidos GetRoot y GetDecendent.

Lo primero es obtener el valor correspondiente a la raíz de la jerarquía.

```
update empleado set jerarquia=hierarchyid::GetRoot() where id=1
```

Luego se asigna el valor del primer hijo de la raíz

```
update empleado set jerarquia=hierarchyid::GetRoot()
      .GetDescendant(null, null) where id=2
```

Luego se asigna el valor del segundo hijo de la raíz. Para el segundo hijo es necesario indicar si el hijo va a ir antes o después del hijo ya ingresado, puesto que los HierarchyId mantienen el orden entre los hijos, algo que no se garantiza con la técnica de “padre-hijo”.

```
update empleado set jerarquia=hierarchyid::GetRoot()
      .GetDescendant((select jerarquia from empleado where id=2), null) where id=3

update empleado set jerarquia=hierarchyid::GetRoot()
      .GetDescendant((select jerarquia from empleado where id=3), null) where id=4

update empleado set jerarquia=hierarchyid::GetRoot()
      .GetDescendant((select jerarquia from empleado where id=4), null) where id=5
```

Para agregar un hijo a un nodo no raíz, el proceso es el mismo.

```
update empleado set jerarquia=(select jerarquia from empleado where id=5)
      .GetDescendant(null, null) where id=6

update empleado set jerarquia=(select jerarquia from empleado where id=2)
      .GetDescendant(null, null) where id=7

update empleado set jerarquia=(select jerarquia from empleado where id=2)
      .GetDescendant((select jerarquia from empleado where id=7), null) where id=8
```

```
update empleado set jerarquia=(select jerarquia from empleado where id=7)
      .GetDescendant(null, null) where id=9
```

Es importante destacar que todas las llamadas a los métodos del HierarchyId nunca acceden a los datos de la tabla para contestar, por lo que para los mismos parámetros de entrada siempre se obtendrá el mismo resultado. Esto puede parecer muy obvio pero es importante ya que durante la creación de los HierarchyId no se garantiza la correcta operación en ambientes concurrentes, por lo cual es responsabilidad de la aplicación controlar esta situación.

Como hacer consultas utilizando los HierarchyId

Es en esta parte del proceso donde se obtienen los mayores resultados, puesto que el optimizador de consultas es capaz de transformar las llamadas a IsDecendent y GetAncestor en operaciones mucho más eficientes.

Esto:

```
WHERE @value.IsDescendant(Jerarquia)
```

Se transforma en:

```
WHERE Jerarquia >= @Value AND Jerarquia <= @Value.DescendantLimit()
```

DescendantLimit es el valor máximo que puede tener un hijo de este nodo, lo que es solo un cálculo aritmético.

Como se ve en esta transformación si existe el índice sobre el campo Jerarquia es posible resolver la consulta de forma muy eficiente.

En el caso del método GetAncestor ocurre algo similar,

Esto:

```
WHERE Jerarquia.GetAncestor(1) = @value
```

Se transforma en:

```
WHERE Jerarquia >= @Value AND Jerarquia <= @value.DescendantLimit()

    AND Jerarquia.GetAncestor(1) = @value
```

Esta consulta utiliza el índice sobre Jerarquia para filtrar la cantidad de filas que pueden ser ancestros de @value. Pero si existe un índice sobre jararquia.GetLevel() + Jerarquia, conocido como índice de ancho primero (Breadth-first), la consulta se transforma en lo siguiente:

```
WHERE Jerarquia >= @value AND Jerarquia <= @Value.DescendantLimit()
    AND @value.GetLevel()+1 = Jerarquia.GetLevel()
```

Es importante destacar que esta consulta utiliza los índices de forma eficiente y no requiere de recursividad para resolverse.

Para comparar la forma de recuperar información jerárquica disponible en SQL Server 2005 (Recursive Common Table Expression) con los HierarchyId de SQL Server 2008, se muestran 2 consultas que obtienen el total de Gastoss de un área de la empresa:

SQL Server 2005

```
WITH CTE_Registros (Id, Gastos)
AS
(
   SELECT Id, Gastos FROM Empleado WHERE Nombre = 'Mariluisa Pascal'
UNION ALL
   SELECT e.Id, e.Gastos FROM Empleado e
   INNER JOIN CTE_Registros cte ON e.Codigoubica = cte.Id
)

SELECT sum(Gastos) FROM CTE_Registros
```

SQL Server 2008

```
SELECT sum(Gastos) FROM Empleado
  WHERE (SELECT jerarquia FROM Empleado WHERE Nombre = 'Mariluisa Pascal')
      .IsDescendant(Jerarquia) = 1
```

O esto que es equivalente a la consulta anterior

```
DECLARE @Jefe HIERARCHYID

SELECT @Jefe=jerarquia FROM Empleado
  WHERE Nombre = 'Mariluisa Pascal'

SELECT nombre, jerarquia.ToString() Jerarquia, jerarquia.GetLevel() Nivel FROM
Empleado
  WHERE @Jefe.IsDescendant(Jerarquia) = 1
```

Limitaciones

Pero lamentablementes, no todo puede ser tan bueno. Los HierarchyId no representan un árbol automáticamente, lo que significa que cada fila almacena la información sobre la posición que tiene en el árbol, algo que no garantiza que se pueda generar un árbol al tomar todas las filas, puesto que puede haberse borrado un nodo padre o puede existir más de un nodo con el mismo HierarchyId.

También es necesario controlar la concurrencia y la unicidad de los HierarchyId, puesto que la base de datos no controla estos aspectos, sin embargo, siempre se puede definir un índice o un constraint que garanticen la unicidad.

Por último, a diferencia de las referencias a otras entidades tradicionales (foreign key), los HierarchyId no mantienen la integridad referencial, lo que permitiría eliminar un nodo que era padre de otros nodos sin que la base de datos lance un error.

Conclusiones

Los HierarchyId son una nueva y poderosa herramienta que estará disponible para todas las aplicaciones que requieran almacenar datos jerárquicos, como por ejemplo un organigrama, una lista de materiales, las tareas de un proyecto o una estructura de directorios.

EJEMPLO 326 -329

En la universidad se tiene dos tablas y una es llamada ALUMNOS y la otra es llamada SEMESTRE, diga usted :

- © **Eliminar si existe**
- © **Crear las tablas**
- © **Establecer una restricción FORIGN_KEY especificando la acción en cascada para actualizaciones y NO_ACTION para eliminaciones**

```
IF OBJECT_ID("ALUMNOS") IS NOT NULL
        DROP TABLE ALUMNOS
IF OBJECT_ID("SEMESTRE") IS NOT NULL
        DROP TABLE SEMESTRE
CREATE TABLE ALUMNOS(
Codigo          Int     Identity,
Nombres         Varchar (40),
Domicilio       Varchar(20),
Semestre        Tinyint,
PRIMARY KEY (Codigo)
);
CREATE TABLE SEMESTRE(
Codigo          Tinyint,
Descripcion     Varchar(40),
PRIMARY KEY (Codigo)
);
ALTER TABLE ALUMNOS
ADD CONSTRAINT FK_Alumnos_semestre
        FOREIGN KEY (Semestre) REFERENCES Semestre (codigo)
ON UPDATE CASACADE
ON DELETE NO ACTION;
```

EJEMPLO 330

En una empresa existen registros de datos, de los cuales tenemos las siguientes tablas

```
CREATE TABLE PROFESORES(
Documento     char(6)          NOT NULL,
Nombre        Varchar(40),
Domicilio     Varchar(60),
CONSTRAINT  CHK_Profesores_documento CHECK (documento LIKE “[0-9] [0-9] [0-9] [0-9] [0-9] [0-9]”),
CONSTRAINT PK_Profesores_documento
PRIMARY KEY (documento)
);
CREATE TABLE DEPORTES(
Codigo        Tinyint          Identity,
Nombre        varchar(40)      NOT NULL,
Dias          Varchar(40) CONSTRAINT DF_Deportes_dia DEFAULT (“Lunes”),
Profesor      char(10),        - documento del profesor
CONSTRAINT CHK_Deportes_dia_lista
CHECK(dia IN (“Lunes”,”Martes”,”Miercoles”,”Jueves”, viernes”)),
CONSTRAINT PK_deportes_codigo
PRIMARY KEY (Codigo)
);

CREATE TABLE SOCIOS(
Numero        int Identity,
Documento     char(6),
Nombre        Varchar(40),
Domicilio     varchar(60),
CONSTRAINT CHK_documento_orden CHECK(documento LIKE “[0-9] [0-9] [0-9] [0-9] [0-9] [0-9]”),
CONSTRAINT PK_SOCIOS_Numero
PRIMARY KEY NONCLUSTERED (Numero),
CONSTRAINT UQ_Sociso_documento
UNIQUE CLUSTERED (Documento)
);
```

EJEMPLO 331

En una empresa existen registros de datos, los cuales existen tablas de profesores, deportes, socios, inscritos, para esto diseñaremos las tablas con restricciones.

```
CREATE TABLE PROFESORES(
Documeno      Cha(6) OT NULL,
Nombre        Varchar(40),
Domicilio     Varchar(60),
CONSTRAINT CH_PROFESORES_DOC CHECK (Documento LIKE “[0-9] [0-9] [0-9]
        [0-9][0-9][0-9]”),
CONSTRAINT PK_Profesores_docmento PRIMAR KEY(Documento)
);
CREA TALE DEPORTES(
Codigo        Identit,
Nombre        Varchar(40) Not Null,
Días          Vachar(40)
        CONSTRAINT DF_Deportes_dia DEAULT (“Lunes”),
Profesor      Char(10), - documento del profesor-
CONSTRAINT CHK_DEPORTES_DIA CHECK (dia IN (“Lunes”,”Martes”,
        ”Miercoles”,”Jueves”,”Viernes),
CONSTRAINT PK_DEPORTES_CODIGO PRIMARY KEY (Codigo)
):
CREATE TABLE SOCIOS(
Numero        Int IDENTITY,
Documento     Char(6),
Nombre        Varchar(40),
Domicilio     Varchar(60),
CONSTRAINT CHK_DOCMENTO_ORDEN CHEC (Documento LIKE “[0-9] [0-9] [0-9]
        [0-9][0-9][0-9]”),
CONSTRAINT PK_SOCIOS_NUMERO PRIMAR KEY NONCLUSTERED (Numero),
CONSTRAINT UQ_SOCISO_DOCUMENTO UNIQUE CLUSTERED (Docmeto)
);
CREATE TABLE INSCRITOS(
Numerosocio   Int NOT NULL,
Codigodeporte Tynyint,
Matricula     Char(1),
CONSTRAINT  PK_INSCRITOS_NUMERODEPORTE  PRIMARY  KEY  CLUSTERED
(Numerosocio,Codigodeporte),
```

```
CONSTRAINT FK_INSCRITOS_DEPORTE FOREIGN KEY(Codigodeporte)
REFRENCES Deportes(Codigo) ON UPDATE CASCADE,
CONSTRAIT CHK_MATRICULA_VALORES CHECK (Matricula IN("S","N"))
);
```

EJEMPLO 332

Ejecute un JOIN para reportar todos los datos de los socios junto con el nombre de los deportes en las cuales esta inscito, asicomo el día que tiene que asistir y el nomre del profesor.

```
Select b.*, d.Nombre AS Deporte, d.dia, c.Nombre AS Profesor FROM SOCIOS AS b
JOIN inscritos as i ON numero=i.numerosocio JOIN deporte as d ON d.codigo =
i.codigodeporte LEFT JOIN Profesores AS c ON d.Profesor = c.Documento;
```

EJEMPLO 333

Ejecutar la misma operación anterior, pero ahora incluyamos los socios que no están inscritos en ningún deporte.

```
Select b.*, d.Nombre AS Deporte, d.dia, c.Nombre AS Profesor FROM SOCIOS AS b
FULL JOIN inscritos as i ON numero=i.numerosocio LEFT JOIN deporte as d ON
d.codigo = i.codigodeporte LEFT JOIN Profesores AS c ON d.Profesor = c.Documento;
```

EJEMPLO 334

Considerando el mismo ejemplo anterior, diga usted como mostrar todos los datos de los profesores incluyendo el deporte que dicta y el día incluido; asimismo reportar a los profesores que no tienen asignado ningún tipo de deporte.

```
Select c.*, d.Nombre AS Deporte, d.dia FROM Profesores AS c LEFT JOIN Deportes
AS d ON d.Profesor =c.documento;
```

EJEMPLO 335

Ahora mostrar todos los deportes y la cantidad de inscritos, inclusive los que no hay inscritos

```
Select d.Nombre, COUNT(i.codigodeporte) AS Cantidad FROM Deportes AS d LEFT
JOIN Inscritos AS I ON d.codigo = i.codigodeporte GROUP BY d.Nombre;
```

EJEMPLO 336

Como repaso, ahora explique usted como mostrar las restricciones de toda la tabla SOCIOS

```
SP_Helpconstraint SOCIOS;
```

REPASO

Agregar y Eliminar Campos en una tabla

EJEMPLO 337-338-339-340

Crear una tabla llamada REGISTRO y si existiese lo eliminemos; posteriormente agregaremos el campo ID_COD de tipo INT, asimismo agregaremos el campo Monto con tipo Decimal(10,2), al terminar agregaremos el campo Tip_docum Char(10) para finalmente concluir verificando la estructura de la tabla.

```
IF Object_Id ("Registro") IS NOT NULL
        DROP Table Registro;
CREATE TABLE Registro(
Nombres         Varchar(40),
Telefono        Varchar(20),
Domicilio       Varchar(60),
Fechacontrol    Datetime2
);

ALTER TABLE Registro
        Add Monto       decimal(10,2);
ALTER TABLE Registro
        Add Id_cod      INT IDENTITY;
ALTER TABLE Registro
        Add tip_docum  Char(10) NOT NULL DEFAULT "DNI"

SP_COLUMNS Resgistro;
```

EJEMPLO 341

De acuerdo al ejemplo anterior diga usted como modificar el campo Nombres extendiendo la longitud

```
ALTER COLUMN Nombres Varchar(60);
```

EJEMPLO 342

De acuerdo al ejemplo anterior cuya tabla registro, modificar el campo Monto, para que no admita valores nulos

```
ALTER COLUMN Monto Decimal(10,2) NOT NULL;
```

EJEMPLO 343-344-345

Elaborar una tabla llamada CONTROL, para luego agregar el Campo ORDEN de Tipo INT IDENTITY y una restricción PRIMARY KEY; luego agregar el campo EST_CIVIL de tipo TINYINT y en la misma restricción CHECK que solo permita ingresar valores de 0 a 4.

```
IF Object_id("CONTROL") IS NOT NULL
        DROP TABLE Control;
CREATE TABLE CONTROL (
Nombres         Varchar(40),
Direccion       Varchar(60),
Documento       char(10) NOT NULL,
Tipodocum       Varchar(16) DEFAULT "DNI"
);
ALTER TABLE CONTROL
ADD Orden INT identity CONSTRAINT _ORDEN PRIMARY KEY;

ALTER TABLE CONTROL
ADD Est_civil TINYINT CONSTRAINT CHK_CONTROL_ESTCIVIL CHECK
(EST_CIVIL LIKE "[0-4]");
```

EJEMPLO 342

Verificar la estructura si es que hubo cambios

```
SP_COLUMNS CONTROL;
EXEC SP_HELPCONSTRAINT CONTROL;
```

CAPITULO IIX

MEMORIA AYUDA

SENTENCIAS ANIDADAS
SELECT anidados a sentencias INSERT, UPDATE, SELECT

En ocasiones es muy útil realizar sentencias de lectura anidadas ya sea para realizar consultas o filtros y he aquí un ejemplo de cómo realizar SELECT en cascada

SELECT * FROM Categoria where tipocategoriaId in(select tipocategoriaId from TipoCategoria where tipocategoriaNombre like '%admin%')

INSERT anidados:
Para realizar migraciones es útil el uso de INSERT y SELECT anidados.
insert into Categoria(nombre,tipoId) select tipoNombre,tipoId from TipoCategoria where tipoId<12

UPDATE anidados:
Para el mantenimiento de datos puede ser útil reconstruir información.
update siteCategoria
set categoriaTitulo =(select categoriaTitulo from dbo.siteCategoria where categoriaId=1)
where categoriaId=6

DATOS DEFINIDOS POR EL USUARIO Y GESTION DE INDICES – TSQL – PROCEDIMIENTOS ALMACENADOS

Indices en SQL Server

Un índice es una estructura de datos definida sobre una columna de tabla (o varias) y que permite localizar de forma rápida las filas de la tabla en base a su contenido en la columna indexada además de permitir recuperar las filas de la tabla ordenadas por esa misma columna.

Los índices funcionan igual que un Libro, veamos, si queremos buscar un tema específico tenemos 3 Opciones:

La primera consiste un ir pagina por pagina buscando la información hasta encontrar el tema que requerimos; esto nos tomara mucho tiempo y esfuerzo.

La Segunda nos vamos al Indice del Libro y buscamos lo que necesitamos, eso nos dará el numero físico exacto del pagina del tema que buscamos nos dirigimos a él y la búsqueda seria rápida, a esto lo llamamos índices Clustered, serian por defecto las llaves primarias (Primary key).

La tercera opción siempre en el ejemplo del libro, seria nos vamos al Glosario y ahí encontraremos un conjunto de paginas donde buscar información sobre el tema a investigar, a esto lo llamamos índices NON-Clustered.

Entonces una definición más ortodoxa seria:

Los Clustered Indexes son índices que controlan el orden físico de las filas en la tabla, por lo cual solo puede existir uno para cada tabla.

Los Non-Clustered indexes son índices que mantienen un sub conjunto de las columnas de la tabla en orden. Estos índices no modifican el orden de las filas de la tabla, en lugar de esto mantienen una lista ordenada de referencias a filas de la tabla original.

Ventajas

La utilización de índices puede mejorar el rendimiento de las consultas, ya que los datos necesarios para satisfacer las necesidades de la consulta existen en el propio índice. Es decir, sólo se necesitan las páginas de índice y no las páginas de datos de la tabla o el índice agrupado para recuperar los datos solicitados; por tanto, se reduce la E/S global en el disco. Por ejemplo, una consulta de las columnas **a** y **b** de una tabla que dispone de un índice compuesto creado en las columnas **a**, **b** y **c** puede recuperar los datos especificados del propio índice.

Los índices en vistas pueden mejorar de forma significativa el rendimiento si la vista contiene agregaciones, combinaciones de tabla o una mezcla de agregaciones y combinaciones.

Inconvenientes

Las tablas utilizadas para almacenar los índices ocupan espacio.
Los índices consumen recursos ya que cada vez que se realiza una operación de actualización, inserción o borrado en la tabla indexada, se tienen que actualizar todas las tablas de índice definidas sobre ella (en la actualización sólo es necesaria la actualización de los índices definidos sobre las columnas que se actualizan

Recomendaciones para crear indices

Las columnas que se aconseja indexar son:
- Las que son clave primaria o ajena
- Aquellas que se usan frecuentemente en búsquedas de rangos de valores con **BETWEEN**
- Aquellas que se usan frecuentemente en *ordenaciones* con **ORDER BY**
- Aquellas que se usan frecuentemente en *cruces de tabla* o **JOIN**
- Aquellas que se usan frecuentemente en *agrupaciones* con **GROUP BY**

Procedimientos Importantes

```
EXEC sp_helpindex  Clientes
```

Mostrara los índices que contiene una tabla, en el ejemplo Clientes de la base Northwind

```
SET STATISTICS IO ON
```

Habilitara

La importancia de los índices clustered

Una de las recomendaciones sobre rendimiento de SQL Server más simple y útil es que 'toda tabla debe tener un índice clustered'. Esto no es 100% cierto y como casi toda norma tiene sus excepciones, pero son pocas.

Las ventajas de tener un índice clustered son varias pero cabe destacar algunos de los motivos por los que influyen en el rendimiento (simplificando algo el tema dicho sea de paso):

- Los registros están fisicamente ordenados según el índice clustered de la tabla (solo puede haber uno por tabla). Esto hace que el acceso a rangos de registros utilizando los campos del índice como filtro sea extremadamente rápido. Tambien es extremadamente rápida la ordenación y el filtrado sobre un índice clustered. Por lo tanto, debemos elegir indicés clustered adecuados para soportar este tipo de consultas, sobre todo cuando se realicen con mucha frecuencia.
- El resto de índices de una tabla que tenga un índice clustered se apoyan en este índice para guardar su información. Por ello debemos tratar elegir índices clustered sobre campos o combinaciones de campos del menor tamaño posible.
- Al final de un índice clustered se encuentra fisicamente los datos de los campos que forman parte del índice, de manera que si nuestra consulta solo necesita campos que se encuentran dentro del índice clustered no necesitará hacer ninguna lectura adicional una vez buscados los registros usando el índice.

A la hora de elegir un índice clustered debemos tener en cuenta las siguientes recomendaciones:

- Se sea usado por el mayor número posible de consultas, sobre todo por aquellas que devuelven un rango de registros seleccionados por el índice o se ordenan o agrupan por los campos del índice. También se benefician aquellas consultas que realizan JOINs sobre los campos cubiertos por el índice.
- No debemos elegir campos que cambian con mucha frecuencia o que almacenan mucha información. Cuanto más pequeño en cuanto a tamaño de los campos sea nuestro índice clustered mejor. Las columnas autonuméricas suelen ser unas exelentes candidatas a índice clustered. Por defecto las claves primarias son índices clustered, suele ser una buena opción, salvo que la clave primaria cambie con mucha frecuencia.
- Cuanto más exclusivos sean los valores del índice clustered mejor. La situación ideal es que los valores combinados de los campos que componen el índice sean únicos.

Por último os dejo un pequeño script T-SQL que nos dice que tablas de nuestra base de datos no tienen un índice clustered, y que por lo tanto requieren nuestra atención:

```
select t.Nombres from sys.tables t
where t.Nombres not in
(
    select t.Nombres from sys.tables t
    join sys.indexes i
    on t.object_id = i.object_id
    where
        t.type = 'U' --Solo nos interesan las tablas de usuario
        and i.type = '1' --1 == Índice clustered
)
```

TIPOS DE INDICE

Ahora platiquemos sobre los tipos de índice, del cual tenemos:

Agrupado
Un tipo de índice agrupado y tiendas de las filas de datos de la tabla o vista en el orden basado en la clave del índice agrupado. El índice agrupado se implementa como una estructura de árbol b que admite la recuperación rápida de las filas, en función de sus valores de índice agrupado clave.

No agrupado
Un índice no agrupado se puede definir en una tabla o vista con un índice agrupado o en un montón. Cada fila de índice en el índice no agrupado contiene el valor de clave no agrupada y un localizador de fila. Este localizador apunta a la fila de datos en el índice agrupado o el montón que el valor de clave. Las filas en el índice se almacenan en el orden de los valores de clave de índice, pero las filas de datos no están garantizados

para estar en cualquier orden particular, a menos de un índice agrupado se crea sobre la mesa.

Único
Un índice único garantiza que la clave del índice no contiene valores duplicados y por lo tanto, cada fila de la tabla o la vista es de alguna manera único. Tanto los índices agrupados y no agrupados pueden ser únicos.

Un índice no agrupado que se amplía para incluir columnas sin clave además de las columnas de clave.

Un tipo especial de token de índice funcional basado en que está construido y mantenido por Microsoft motor de texto completo para SQL Server. Proporciona un apoyo eficiente a la búsqueda de palabras complejas en datos de cadenas de caracteres.

Un índice espacial proporciona la capacidad de realizar determinadas operaciones de manera más eficiente en objetos espaciales (datos espaciales) en una columna del tipo de geometría de datos. El índice espacial reduce el número de objetos en los que las operaciones espaciales relativamente costosos deben aplicarse.

Un índice no clúster optimizado,
Especialmente indicado para cubrir consultas que seleccionan a partir de un subconjunto bien definido de los datos. Se utiliza un predicado de filtro para indizar una parte de las filas de la tabla. Un índice filtrado bien diseñado puede mejorar el rendimiento de las consultas, reducir los costos de mantenimiento y de reducir los costos de almacenamiento del índice en comparación con los índices de tabla completa.

XML
Un rallado, y se mantuvo, la representación de los objetos binarios grandes (BLOB XML) en la columna tipo de datos xm

EJEMPLO 343
Reportar todos los datos de la tabla VENTAS y de la tabla ARTICULOS

```
SELECT A.CodArt,A.Unid_med,V.Cant,V.Descripcion_bien,v.PVenta FROM
Master.dbo.articulos A, Master.dbo.ventas V WHERE A.Cod_art =V.Codigo_art;
```

EJEMPLO 344
Empleando la misma consulta en sintaxis ANSI:

```
SELECT A.cod_art,A.Unid_med, V.Cant, V.Descripcion_bien,V.Pventa FROM
Master.dbo.Articulos A INNER JOIN Master.dbo.Ventas V ON A.Cod_art =
V.codigo_art;
```

EJEMPLO 345
Tenemos cuatro tablas de movimientos cuyos registros se almacenan en ellas, siendo:

Movimcta (Mov)	**Tabla de los movimientos VENTAS donde solo almacena las cabeceras de la factura**
Itemvta (Ite)	**Tabla donde almacena cada ítem de las ventas de la tabla Movimvta**
Clientes (Cli)	**Tabla de los clientes**
Bienes (Bie)	**Tabla de los stock de almacén**

```
SELECT Mov.Num_fact, CONVERT(Char(10),Fec_ven,103) AS Fecha,
SUBSTRING(Nombres,1,20) Nombres, LOWER(Guia), Ite.Referen_doc
“Documento de Referencia”, Cod_cli “Codigo del Cliente”, cli_direccion,
Bie.stock FROM CLIENTES CLI INNER JOIN Movimcta MOV ON
Mov.clientescod = Cli.codcli INNER JOIN Itemvta ITEON Ite.Numfac =
Mov.factura INNER JOIN Bie.codbien = Ite.codbie;
```

COMBINACIONES EXTERNAS

Recordemos que cuando empleamos Consultas externa, es por que nos permiten extraer datos de las tablas relacionadas.

Sintaxis:
Nombre_tabla1 LEFT OUTER JOIN Nombre_tabla2 ON Nombre_tabla1.col1 = Nombre_tabla2.col2
O también puede ser:
Nombre_tabla1 RIGHT OUTER JOIN Nombre_tabla2 ON Nombre_tabla1.col1 = Nombre_tabla2.col2

O también puede ser:

Nombre_tabla1 FULL OUTER JOIN Nombre_tabla2 ON Nombre_tabla1.col1 = Nombre_tabla2.col2

EJEMPLO 346

Vamos a considerar como ejemplo que tenemos dos tablas en una empresa que son clientes y Ventas; el cual nos piden ver los resultados de las ventas, solamente de todos los clientes a quienes se haya movimiento.

SELECT A.clicod AS Codigo, A.Fecha, A.Producto, A.Pventa, A.Cantidad,A.Factura, B.Nom_cli AS “Nombre del Cliente” FROM Ventas A LEFT OUTER JOIN Clientes B ON A.clicod = B.codigocli;

Observemos que nos mostrara solamente los clientes que han tenido movimiento, pero si queremos ver todos los clientes que no tienen movimiento, hubiésemos empleado RIGHT.

AUTOCOMBINACION

Es posible asociar filas de una tabla a otras filas de la misma tabla realizando un SELF_JOIN recordando que es obligatorio el uso de las ALIAS.

ORDER BY

Permite ordenar todos los registros de la consulta efectuada .

UNION

Este operador permite obtener un conjunto de filas proveniente de varias consultas, todas las columnas deben proporcionar el mismo numero de columnas.

EXCEPT

Este operador permite ejecutar la diferencia entre dos resultados con la misma estructura.

EJEMPLO 347

Listar todos los alumnos de la facultad de Medicina, pero que no estén desaprobados.

```
SELECT * FROM ALUMNOS WHERE Facultad LIKE “Medicina%” EXCEPT
SELECT * FROM ALUMNOS WHERE Nota>10;
```

INTERSECT

Este operador corresponde a la traducción de TSQL Intersección.

EJEMPLO 348

Seleccionar los diez primeros registros de una consulta, considerando que tenemos tres tablas y son:
Movimientos, Itemmovimiento y Articulos.

La tabla Movimientos es donde se almacena todos las facturas de Ventas, mas no los Items de cada venta; en la tabla Itemmovimiento se graba los item de cada venta y en la tabla Articulos se graba todo el Stock de Articulos.

SELECT TOP(10) M.Factura,M.total,M.Fecha,I.codigobien,I.Pventa,I.Total,A.stock, A.Descripcion FROM Movimientos M INNER JOIN Itemmovimiento I ON M.Factura = I.Numfact INNER JOIN AARTICULO A ON A.codbien = I.Codigobien GROUP BY M.Factura ORDER BY 1 DESC;

EJEMPLO 349

Listar todas las columnas de las tablas Movimientos y Articulos que se encuentran relacionadas mediante la columna CODIGOBIEN.

SOLUCION COMUN

SELECT Mov.Codigobien,Mov.Precio,Art.Detalle, Art.Stock FROM Master.dbo.Movimientos Mov, Master.dbo.Articulo Art WHERE Art.codigobien = Mov.Codigobien

SOLUCION ANSI

SELECT Mov.Codigobien,Mov.Precio,Art.Detalle, Art.Stock FROM Master.dbo.Movimientos Mov INNER JOIN Master.dbo.Articulo Art ON Art.codigobien = Mov.Codigobien

NOTA

WITH TIES Esta opción se puede emplear si se especifica una cláusula ORDER BY en la consulta.

EJEMPLO 350

Tenemos 04 tablas los cuales se detallan sus ALIAS en la siguiente manera:

- **LIN Tabla de ítem de pedidos**
- **PDO Tabla de Pedidos**
- **CLI Tabla de Clientes**
- **ART Tabla de Articulos**

El Alias de la tabla LIN se relaciona con el Alias de la tabla PDO mediante la columna Numero_pdo; el Alias de la tabla LIN se relaciona con el alias de la tabla ART mediante la columna Referencia_art; el Alias de la tabla PDO se relaciona con El Alias de la tabla CLI mediante la columna Numero_cli.

Ahora observamos que tenemos 04 tablas relacionadas entre si, el cual efectuaremos una combinación de tablas para obtener como resultado la siguiente consulta.

```
SELECT PDO.Numero_pdo, CONVERT(Char(10),fecha,103) AS Fecha_inicio,
SUBSTRING(Apellido,1,15) APELLIDO, LIN.Referencia_art,Precio,Cantidad FROM
Clientes CLI INNER JOIN Pedidos PDO ON PDO.Numero_cli = CLI.Numero_cli INNER
JOIN Lineas_pdo LIN ON LIN.Numero_pdo = PDO.Numero_pdo INNER JOIN Articulos
ART ON Art.Referencia_art = LIN.Referencia_art;
```

EJEMPLO 351

Como ejemplo vamos ahora considerar que queremos listar todos los clientes de la tabla VENTAS que estén en relación con la tabla CLIENTES mediante la columna CODIGOCLI.

Observaremos que de acuerdo al resultado va mostrar todas las columnas de los clientes inclusive de aquellos que no se han generado movimientos.

```
SELECT Clientes = C.Codigocli, apellidos, V.numfactura, V.Fechaventa FROM
CLIENTES C LEFT OUTER JOIN VENGTAS V ON C.Codigocli = V.Codigocli
```

NOTA

Recordemos que las combinaciones externas completa (FULL OUTER JOIN) permite mostrar los datos definidos de estas dos tablas aunque no sea posible establecer correspondencia.

EJEMPLO 352

Una empresa almacena sus productos en un almacén por lo que es necesario crear una tabla de productos llamados ARTICULOS y luego insertar registros adicionales

```
IF OBJECT_ID (“Articulos”) NOT NULL
        DROP TABLE ARTICULOS;
CREATE TABLE ARTICULOS(
Codigo INT      IDENTITY;
Descripcion     Varchar(40),
Pventa          Decimal(13,2) NOT NULL,
Cantidad SMALLINT NOT NULL DEFAULT 0,
Monto  AS Pventa∗Cantidad
);
```

Ahora insertemos un registro con valor para el campo calculado:

```
SET IDENTITY_INSERT ARTICULOS ON
```

INSERT INTO ARTICULOS VALUES(“Farmacon”,120.00,40,500);

Observemos que no lo permite ya que hemos creado un campo MONTO automatico.

EJEMPLO 353

Actualicemos la tabla del ejemplo anterior

```
UPDATE  Articulos SET Pventa=120 WHERE Descripcion = “Farmacon”;
```

EJEMPLO 354

Actualice una cantidad y veamos el resultado

```
UPDATE  Articulos SET Pventa=100 WHERE Descripcion = “Farmacon”;
```

NOTA

Si deseamos ingresar manualmente los registros a una tabla, donde exista la columna IDENTITY, ya sabemos que debemos emplear SET IDENTITY_INSERT TABLA ON

Recordemos que la importancia de los INDICES. T-SQL Server.

La creación de índices útiles es uno de los métodos más importantes para lograr un mejor rendimiento de las consultas. Los índices útiles ayudan a encontrar los datos con menos operaciones de E/S de disco y un menor uso de los recursos del sistema.

Para crear índices útiles, debe comprender cómo se utilizan los datos, los tipos y las frecuencias de ejecución de las consultas y cómo el procesador de consultas puede utilizar los índices para encontrar los datos con rapidez.

Una vez elegidos los índices que creará, examine las consultas más importantes, cuyo rendimiento es el factor que más afecta a la experiencia del usuario. Cree los índices específicamente para ayudar a estas consultas. Después de agregar un índice, vuelva a ejecutar la consulta para comprobar si el rendimiento ha mejorado. En caso negativo, quite el índice.

Al igual que en la mayoría de las técnicas de optimización del rendimiento, existen ventajas e inconvenientes. Por ejemplo, con más índices, es probable que las consultas **SELECT** se ejecuten con mayor rapidez. Sin embargo, las operaciones DML (**INSERT**, **UPDATE** y **DELETE**) reducirán su velocidad porque se deben mantener más índices con cada operación. Por consiguiente, si las consultas son principalmente instrucciones **SELECT**, el uso de más índices puede ser positivo. Si su aplicación lleva a cabo muchas operaciones DML, el número de índices que cree debería ser más moderado.

En definitiva, puede mejorar el rendimiento de la aplicación SQL Server, optimizando las consultas que utiliza. En los próximos días publicaré en mi blog (este mismo) algunas técnicas que pueden aplicarse para optimizar el rendimiento de las consultas.

Los índices en las columnas utilizadas en la cláusula WHERE de las consultas importantes normalmente mejoran el rendimiento. Sin embargo, esto depende del grado de selectividad del índice. La selectividad es la proporción de filas resultantes respecto al total de filas. Si la proporción es baja, significa que el índice es muy selectivo, ya que puede deshacerse de la mayoría de las filas y reducir en gran medida el tamaño del conjunto de resultados. Por consiguiente, se trata de un índice muy útil. En cambio, un índice que no es selectivo no es tan útil.

Los índices únicos son los más selectivos. Sólo puede coincidir una fila, lo que es realmente útil para las consultas que pretenden exactamente devolver una fila. Por ejemplo, un índice en una sola columna de Id. servirá de ayuda para encontrar con rapidez una fila concreta.

Los índices de varias columnas son extensiones naturales de los índices de una sola columna. Los índices de varias columnas son útiles para evaluar expresiones de filtro que coinciden con un conjunto de prefijos de columnas de clave. Por ejemplo, el índice compuesto CREATE INDEX Idx_Emp_Nombres ON Employees ("Last Nombres" ASC, "First Nombres" ASC) ayuda a evaluar las siguientes consultas:

- ... WHERE "Last Nombres" = 'Doe'
- ... WHERE "Last Nombres" = 'Doe' AND "First Nombres" = 'John'
- ... WHERE "First Nombres" = 'John' AND "Last Nombres" = 'Doe'

Sin embargo, no será útil para esta consulta:

- ... WHERE "First Nombres" = 'John'

Al crear un índice de varias columnas, las columnas más selectivas deberían ubicarse en la parte izquierda de la clave. De este modo, el índice es más selectivo cuando coincide con varias expresiones.

Una tabla pequeña es aquella cuyo contenido cabe en una o pocas páginas de datos. Evite indizar tablas muy pequeñas porque normalmente es más eficaz realizar una exploración de tablas. De este modo, se evita tener que cargar y procesar las páginas de índices. Si no crea un índice en las tablas muy pequeñas, está eliminando la posibilidad de que el optimizador seleccione una.

Es recomendable que siempre cree índices en las claves principales. También suele ser muy útil crear índices en claves externas, puesto que tanto las claves principales como las externas se utilizan con frecuencia para combinar tablas. Los índices de estas claves permiten al optimizador calcular los algoritmos de combinación de índices más eficaces. Si la consulta combina tablas utilizando otras columnas, a menudo es útil crear índices en esas columnas por la misma razón.

Cuando se crean las restricciones de claves principales y externas, SQL Server crea automáticamente índices para ellas y las utiliza para optimizar las consultas. Recuerde que es aconsejable crear claves principales y externas lo más pequeñas posible, ya que las combinaciones son más rápidas.

Los índices pueden utilizarse para acelerar la evaluación de ciertos tipos de cláusulas de filtro. Si bien todas las cláusulas de filtro reducen el conjunto de resultados final de una consulta, algunas de ellas también ayudan a reducir la cantidad de datos que se deben explorar.

Un argumento de búsqueda (SARG) limita una búsqueda porque especifica la coincidencia exacta, un intervalo de valores o una conjunción de dos o más elementos combinados con AND. Presenta uno de los siguientes formatos:

- Columna operador <constante o variable>
- <constante o variable> operador Columna

Entre los operadores SARG se incluyen =, >, <, >=, <=, IN, BETWEEN y, en ocasiones, LIKE (en casos de coincidencia de prefijos, tales como LIKE 'John%'). Un argumento SARG puede incluir varias condiciones combinadas con un AND. Los argumentos SARG pueden ser consultas que coinciden con un valor específico, por ejemplo:

- "Cliente ID" = 'ANTON'
- 'Doe' = "Last Nombres"

Un argumento SARG también pueden ser una consulta que coincide con un intervalo de valores, por ejemplo:

- "Order Date" > '1/1/2002'
- "Cliente ID" > 'ABCDE' AND "Cliente ID" < 'EDCBA'
- "Cliente ID" IN ('ANTON', 'AROUT')

Una expresión que no utilice operadores SARG no mejorará el rendimiento porque el procesador de consultas de SQL Server debe evaluar cada fila para determinar si cumple la cláusula de filtro. Por consiguiente, un índice no es de utilidad en expresiones que no utilizan operadores SARG. Entre los operadores que no son SARG se incluyen NOT, <>, NOT EXISTS, NOT IN, NOT LIKE y funciones intrínsecas.

Las operaciones **ORDER-BY**, **GROUP-BY** y **DISTINCT** son todas tipos de ordenación. El procesador de consultas de SQL Server implementa la ordenación de dos modos distintos. Si los registros ya están ordenados por un índice, el procesador sólo tiene que usar el índice.

De lo contrario, el procesador debe utilizar una tabla de trabajo temporal para ordenar primero los registros. Esta ordenación preliminar puede provocar retrasos iniciales considerables en dispositivos con una CPU lenta y memoria limitada, y debería evitarse si el tiempo de respuesta es importante.
En el contexto de índice con varias columnas, para que **ORDER-BY** o **GROUP-BY** tengan en cuenta un índice concreto, las columnas **ORDER-BY** o **GROUP-BY** deben coincidir con el conjunto de prefijos de columnas de índice en el orden exacto. Por ejemplo, el índice CREATE INDEX Emp_Nombres ON Employees ("Last Nombres" ASC, "First Nombres" ASC) puede ayudar a optimizar las siguientes consultas:

- ... ORDER BY / GROUP BY "Last Nombres" ...
- ... ORDER BY / GROUP BY "Last Nombres", "First Nombres" ...

No ayudará a optimizar:

- ... ORDER BY / GROUP BY "First Nombres" ...
- ... ORDER BY / GROUP BY "First Nombres", "Last Nombres" ...

Para que una operación DISTINCT tenga en cuenta un índice de varias columnas, la lista de proyección debe coincidir con todas las columnas de índice, aunque no es necesario que estén en el orden exacto. El índice anterior puede ayudar a optimizar las siguientes consultas:

- ... DISTINCT "Last Nombres", "First Nombres" ...
- ... DISTINCT "First Nombres", "Last Nombres" ...

No ayudará a optimizar:

- ... DISTINCT "First Nombres" ...
- ... DISTINCT "Last Nombres" ...

Si la consulta siempre devuelve filas únicas, no especifique la palabra clave DISTINCT, ya que sólo aumenta la sobrecarga.

Hay que tener en cuenta que cada vez que ponemos agrupaciones del tipo **ORDER-BY**, **GROUP-BY** y **DISTINCT,** podemos estar penalizando el rendimiento si no tenemos los índices adecuados y muchas veces estas agrupaciones que usamos no tienen sentido ya que se pueden hacer desde el reporte o la aplicación. En ocasiones es mejor no dar una sobrecarga el motor, ya que he visto como quedan resuelto algunos temas de performance en querys simplemente por eliminar estas agrupaciones, recordemos que las agrupaciones las podemos hacer muchas veces en la grilla de nuestra aplicación sin castigar al motor de base.

DBCC DBREINDEX

Para reconstruir los índices de modo que los datos ya no estarán fragmentados. Los datos fragmentados pueden causar que SQL Server realice lecturas de datos innecesarias, decrementando el rendimiento de SQL Server, por lo que es necesario hacer una reorganización en una tabla con índices agrupados, algunos índices no-agrupados en la misma tabla serán automáticamente reconstruidos.

La reorganización de la base de datos puede ser realizada usando el Maintenance Wizard, o corriendo tu propio script a través del SQL Server Agent

El comando DBCC DBREINDEX no reconstruirá automáticamente todos los índices de las tablas en la base de datos; solo puede funcionar de a una tabla por vez. Pero si corres el siguiente script, puedes indexar todas las tablas en una base de datos con facilidad.

CREAR Y ASOCIAR REGLAS

En SMO, el objeto Rule representa las reglas. La propiedad TextBody, que es una cadena de texto que contiene una expresión de condición que utiliza operadores o predicados, como IN, LIKE o BETWEEN, define la regla. Una regla no puede hacer referencia a columnas u otros objetos de base de datos. Se pueden incluir funciones integradas que no hagan referencia a objetos de base de datos.

La definición en la propiedad TextBody debe contener una variable que haga referencia al valor de datos escrito. Se puede utilizar cualquier nombre o símbolo para representar

REGLAS: Regla se utiliza para enlazar una o más columnas de la base de datos actual.
Sintaxis:

CREATE RULE Regla como condición

EJEMPLO 355
Definamos un tipo de dato llamado PVENTA en una tabla CLIENTES, que permita valores nulos.

```
EXEC SP_ADDTYPE  PVENTA, “INT”,”NULL”;
```
Pero que hubiese sido si existiera dicha columna, pues primero lo hubiésemos eliminado
```
IF EXISTS(Select * FROM SYSTYPES WHERE Nombres =”PVENTA”)
        EXEC SP_DROPTYPE PVENTA;
```

EJEMPLO 356
Supongamos que existiese una Regla llamada RG_FECHA, en la tabla CLIENTES, eliminemos dicha regla
```
IF OBJECT_ID(“RG_FECHA”) IS NOT NULL
        DROP RULE RG_FECHA
```

EJEMPLO 357
Conseiderando que tenemos una columna llamada inicio (Columna Tipo año), dentro de la tabla VENTAS, donde debe considerar ingresar los valores de los años, pero con la condición que sean entre 2000 y hasta la fecha, crear una regla.

```
CREATE RULE RG_INICIO
        AS @inicio BETWEEN 2000 AND DATEPART(YEAR,GETDATE());
```

EJEMPLO 358
En el caso que hemos creado dicha regla anterior en la tabla VENTAS, y ya exista registros ingresados, pero deseamos considerar que dicha regla se aplique para los futuros

```
Exec SP_BRINDRULE RG_INICIO,”INICIO”,”FUTURE ONLY”;
```

EJEMPLO 359
Ahora quite la asociación de la regla creada ante rior mente

Exec SP_UNBINDRULE RG_INICIO;

EJEMPLO 360
Ahora veamos si ya se quito la asociacion
SP_HELP CONSTRAINT VENTAS;

EJEMPLO 361
Crear una regla en la tabla VENTAS, considerando que tenga una columna llamada CONTROL, el cual admita solo valores entre -40 y -1

CREATE RULE RG_NEGATIVO
AS @CONTROL BETWEEN -40 AND -1
-- Ahora lo asociamos
EXEC SP_BRINDRULE RG_ **NEGATIVO**, **"VENTAS.Control";**

EJEMPLO 362
Efectuando un repaso a los capítulos anteriores, diga usted como agregar una restricción DEFAULT a la tabla VENDEDORES

ALTER TABLE VENDEDORES
ADD CONSTRAINT DF_VENDEDORES_AÑO DEFAULT 2000 FOR AÑO;

Recordemos que cuando hablamos de actualización de datos es diferente de hablar de actualización de inidices pero en ambos casos emplearemos UPDATE

EJEMPLO 363
Modificar las columnas SEMESTRE y FACULTAD del alumno cuyo código es 112127766 de la tabla universidad cuya base de datos sea PADRON

UPDATE Padron.dbo.Universidad SET SEMESTRE = X, FACULTAD ="Ingenieria" WHERE Codigo=112127766;

EXCEPT

La operacion EXCEPT es una extensión del lenguaje SQL introducida en Microsoft SQL-Server a partir de la version 2005, y resulta muy útil en diversas labores.

Consideremos la situación en que tenemos dos tablas del mismo nombre en dos bases de datos diferentes. Por ejemplo, la misma tabla en una base de datos de pruebas y otra en la base de datos en producción.

Ahora, para comprobar que los datos de prueba están actualizados, deseamos saber si hay CUALQUIER diferencia en los valores de CUALQUIERA de las columnas de las dos tablas.

Dado que EXCEPT retorna "cualquier valor distinto de la consulta izquierda que no se encuentre en la consulta derecha", parece una buena situación para usar EXCEPT.

SELECT * FROM pruebas.dbo.Clientes
EXCEPT
SELECT * FROM produccion.dbo.Clientes

Recordamos la especificacion del EXCEPT veremos que solo vamos a obtener los registros que estan en la tabla de pruebas para los cuales hay diferencias con respecto a la tabla en producción, pero NO veremos los registros en producción que no están en la tabla de pruebas.

Para ello, podemos realizar una UNION como esta:

SELECT * FROM pruebas.dbo.Clientes
EXCEPT
SELECT * FROM produccion.dbo.Clientes

UNION

SELECT * FROM produccion.dbo.Clientes

EXCEPT
SELECT * FROM pruebas.dbo.Clientes

Pera debido a que el operador UNION tiene precedencia sobre el EXCEPT, la operacion de UNION se realizar ANTES del EXCEPT y de nuevo obtendremos solo una parte de las diferencias.

Corrijamos entonces esa construcción de esta forma:

```
SELECT * FROM

(SELECT * FROM pruebas.dbo.Clientes
EXCEPT
SELECT * FROM produccion.dbo.Clientes) AS IZQUIERDA

UNION

SELECT * FROM

(SELECT * FROM produccion.dbo.Clientes
EXCEPT
SELECT * FROM pruebas.dbo.Clientes) AS DERECHA
```

Ahora si tendremos la lista completa de diferencias entre una y otra tabla. Tendremos tanto las filas que tienen valores con diferencias, como las filas que están en una tabla pero no en la otra.

Asi que esta consulta funciona correctamente, pero al ejecutarla notaremos que tenemos las diferencias, pero la consulta no nos indica de donde proviene.

De modo que hacemos un último ajuste:

```
SELECT 'pruebas' AS origen,* FROM
(SELECT * FROM pruebas.dbo.Clientes
EXCEPT
SELECT * FROM produccion.dbo.Clientes) AS IZQUIERDA

UNION
SELECT 'produccion' AS origen,* FROM
(SELECT * FROM produccion.dbo.Clientes
EXCEPT
SELECT * FROM pruebas.dbo.Clientes) AS DERECHA
```

Finalmente, tenemos una lista de todas las diferencias entre ambas tablas, debidamente identificadas.

PRECAUCIONES:

El uso de tablas que tengan una llave primaria mejora el desempeño de la sentencia EXCEPT.

Además, las tablas deben tener la misma estructura (es decir, los campos deben tener los mismos nombres, ser del mismo tipo y estar en el mismo orden en ambas tablas). Aunque hay pequeáas excepciones (por ejemplo, dos campos de tipo VARCHAR con el mismo nombre y diferente longitud pueden compararse sin problemas), es preferible garantizar esta igualdad.

Para una absoluta garantía, tal vez valga la pena no utilizar "*" sino especificar la lista de campos de ambas tablas.

Note que esto nos lleva a un punto interesante: el EXCEPT compara TODOS los campos incluidos en la consulta, pero no aquellos que se omitan. De manera que podemos utilizar una consulta similar a la indicada solo para garantizar cosas como por ejemplo: que todos los códigos de clientes tengan el mismo nombre y dirección (aunque otros campos que existan en las tablas puedan ser diferentes).

Esperamos que este artÃculo los motive a aprovechar y explotar las nuevas posibilidades de sentencias como EXCEPT en SQL-Server 2005 y posteriores.

INTERSECT

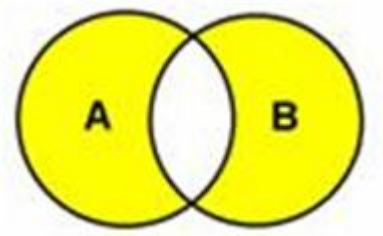
Este operador consiste en identificar en una sola consulta las filas que están presentes en dos resultados, pero con la misma estructura.
Ejemplo:

```
SELECT * FROM ALUMNOS WHERE Apellidos LIKE “%Pereda%”
INTERSECT
SELECT * FROM ALUMNOS WHERE Profesion NOT “PSICOLOGOS”;
```

Recordemos que los operadores INTERSECT, EXCEPT y UNION son un set de operadores que ejecutan operaciones entre 2 o más set de datos. UNION ha estado disponible en T-SQL desde las primeras versiones, mientras INTERSECT y EXCEPT fueron introducidos en SQL 2005. Los tres operadores tienen requerimientos similares:

Requieren un mínimo de 2 set de datos.
Cada set de datos debe tener el mismo número de columnas.
Cada columna con su relativa columna deben ser tipos de datos compatibles.
La cláusula ORDER BY puede usarse únicamente al final de la consulta.

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.

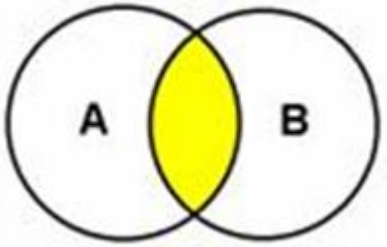
```
SELECT A, B, C  FROM TABLA1
UNION
SELECT A, B, C  FROM TABLA2
```

Una variante es UNION ALL que agrega todas las filas a los resultados. Incluye las filas duplicadas. Si no se especifica, las filas duplicadas se quitan.

```
SELECT A, B, C  FROM TABLA1
UNION ALL
SELECT A, B, C  FROM TABLA2
```

INTERSECT

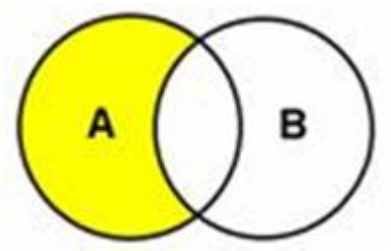
Devuelve los valores distintos de la consulta inicial que corresponda con la siguiente consulta. Es decir los valores en común que tengan ambas consultas.




```
SELECT A, B, C  FROM TABLA1
INTERSECT
SELECT A, B, C  FROM TABLA2
```

EXCEPT

Devuelve los valores distintos de la consulta inicial que no se devuelven desde la consulta siguiente. Es decir en dependencia de la ubicación de la consulta nos regresara los valores que no se encuentren la siguiente.



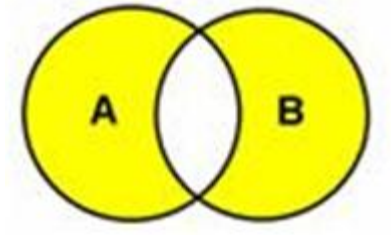

```
SELECT * FROM t1
EXCEPT
SELECT * FROM t2
```

Sera distinto los resultados para la siguiente consulta:

```
SELECT * FROM t2
EXCEPT
SELECT * FROM t1
```

DIFERENCIA SIMETRICA

Como lograriamos esta consulta?



Pudiésemos usar NOT IN para ello

```
        SELECT A
    FROM Tabla1
    WHERE A NOT IN(SELECT A FROM Tabla2)
    UNION
    SELECT A
    FROM Tabla2
    WHERE A NOT IN(SELECT A FROM Tabla1)
```

O bien lo hariamos con la combinacion de los operadores antes vistos:

```
SELECT A FROM TABLA1
   UNION
        SELECT A FROM TABLA2
   EXCEPT
        SELECT A FROM TABLA1
   INTERSECT
        SELECT A FROM TABLA2
```

EJEMPLO 364

Supongamos que tenemos una tabla donde se elabora un sistema de control, donde su registro de almacenamientos de datos se llama ALUMNOS, para esto tenemos una estructura determinada y nos piden crear un índice "Agrupado unico" para la columna FACULTAD, donde vamos a observarque no lo permite por que existen valores duplicados ya que diversos alumnos tienen la misma facultad.

```
CREATE UNIQUE CLUSTERED
INDEX I_Alumos_nombres ON Alumnos(Nombres);
```

LOS INDICES NO AGRUPADOS UNICOS – INDICES AGRUPADOS

EJEMPLO 365

Crear una tabla clientes, donde usted debe defini un índice no agrupado único en ella, para luego crear dos índices no agrupados, donde uno de ellos es único y el otro no es único.

```
CREATE TABLE Clientes
(
ClienteID INT NOT NULL,
ClienteNombres CHAR (100) NOT NULL,
ClienteDireccions CHAR (100) NOT NULL,
Comentarios CHAR (189) NOT NULL
)
GO
```

- Crear un índice único no agrupado en la tabla anterior

```
CREATE CLUSTERED idx_Clientes índice en los clientes (ClienteID)
GO
```

- Colocar 80.000 registros

```
DECLARE @ i INT = 1
WHILE (@ i <= 20000)
COMENZAR
DECLARE @ j = 1 INT

INSERT INTO valora a los clientes
(
@ I,
''ClienteNombres '+ CAST (@ i AS CHAR) + CAST (@ j AS CHAR),
'ClienteDireccions' + CAST (@ i AS CHAR),
'Comentarios' + CAST (@ i AS CHAR)
)

SET @ j + = 1;

INSERT INTO valora a los clientes
(
@ I,
''ClienteNombres '+ CAST (@ i AS CHAR) + CAST (@ j AS CHAR),
'ClienteDireccions' + CAST (@ i AS CHAR),
'Comentarios' + CAST (@ i AS CHAR)
)

SET @ j + = 1;

INSERT INTO valora a los clientes
(
@ I,
''ClienteNombres '+ CAST (@ i AS CHAR) + CAST (@ j AS CHAR),
'ClienteDireccions' + CAST (@ i AS CHAR),
'Comentarios' + CAST (@ i AS CHAR)
)

SET @ j + = 1;

INSERT INTO valora a los clientes
(
@ I,
''ClienteNombres '+ CAST (@ i AS CHAR) + CAST (@ j AS CHAR),
'ClienteDireccions' + CAST (@ i AS CHAR),
'Comentarios' + CAST (@ i AS CHAR)
)

SET @ i + = 1
FIN
GO
```

- Crear un índice único no agrupado en la tabla agrupada

```
CREATE UNIQUE idx_UniqueNCI_ClienteID índice no agrupado
En los Clientes (ClienteNombres)
GO
```

- Crear un no único índice no agrupado en la tabla agrupada CREAR índice no agrupado idx_NonUniqueNCI_ClienteID En los Clientes

```
(ClienteNombres)
GO
```

Aplicando ahora sys.dm_db_index_physical_stats, se puede ver que el índice único no agrupado tiene 107 bytes por fila de índice en los niveles de navegación, donde la única no índice no agrupado toma 117 bytes en el promedio (mínimos de 111 bytes, máximo 117 bytes). Vamos a analizar las diferencias y volcar la página raíz de índice del único índice no agrupado a través del comando **DBCC PAGE:**

```
DBCC PAGE (NonUniqueClusteredIndexStructure_NonClusteredIndex, 1, 4529, 3)
GO
```

Ahora vamos a volcar la raíz de índice del no único índice no agrupado definido en nuestro no único índice agrupado:

Esto es ahora una salida muy interesante! La clave del registro de índice debe ser por un diseño único. ¿Cómo se puede hacer un SQL Server no única clave de índice no agrupado único? Fácil - SQL Server sólo se suma la clave del índice agrupado (4 bytes). Pero la clave del índice agrupado no es único también por defecto, por lo tanto, SQL Servidor también añade la uniquifier (4 bytes), por lo que tiene una sobrecarga resultante de 8 bytes por fila de índice, cuando el uniquifier no es igual a 0. Cuando el uniquifier es igual a 0, se obtiene una sobrecarga de 4 bytes, ya que en este caso, el uniquifier no se almacena físicamente en el registro de índice, y un 0 es asumido por SQL Server de forma automática. Al analizar el nuevo byte por byte de la representación se puede ver los siguientes bytes:

- 1 Byte: los bits de estado
- n Bytes: clave única de índice no agrupado - en este caso de 100 bytes
- n Bytes: no único clave de índice agrupado - en este caso de 4 bytes
- 4 bytes: pageID
- 2 Bytes: FileID
- 4 bytes: Algunos bytes utilizados por el uniquifier
- 4 bytes: El valor uniquifier sí mismo, cuando no es igual a 0

La longitud mínima del registro de índice es 111 bytes, y por lo tanto, la longitud máxima es de 117 bytes que ya se encuentran a cabo anteriormente por el DMV **sys.dm_db_index_physical_stats.** Cuando finalmente volcar la hoja a nivel de la no único índice no agrupado, se obtiene el siguiente resultado:

EJEMPLO 366
CREAR UN INDICE AGRUPADO SOBRE DATOS YA ORDENADOS

```
CREATE UNIQUE CLUSTERED INDEX PK-LINPDO
ON LINEAS-FACT(NUMFACT,ITEM)  WITH FILLFACTOR =100;
-- Ahora eliminemos un indice:
```

Recordemos que los indices definidos con la instrucción CREATE INDEX pueden ser eliminados con DROP INDEX

```
DROP INDEX Nombre_index  ON Nombre_tabla;
```

EJEMPLO 367
Ahora si usted desea reconstruir un índice, emplearemos DBCC DBREINDEX, para ello emplearemos ALTER INDEX

```
ALTER INDEX ALL
ON CLIENTES
REBUILD WITH (FILLFACTOR=50);
```

INDEXAR LAS TABLAS DE DESARROLLO

Cuando tus tablas y la base de datos está activa y en producción no ejecutarla, ya que puede bloquear los recursos y hacer que sus usuarios tengan problemas. La Re-indización debe programarse en los momentos en que la BD esté abajo (desactivado), o en el peor de los casos, mientras el uso de la misma sea reducido.

Si utilizas el comando CREATE INDEX para crear o reconstruir tus índices, la opción FILLFACTOR tiene su propia sub-opción llamada PAD_INDEX. Si no se especifica la opción PAD_INDEX, entonces el FILLFACTOR sólo se aplica a la hoja de páginas en el índice, no en el índice de páginas intermedias. Sin embargo, si especificas PAD_INDEX a lo largo de la opción FILLFACTOR, cuando el índice se crea, el FILLFACTOR se aplicará a las páginas de índice intermedio.

Si deseas reconstruir un índice cluster utilizando el comando CREATE INDEX, y suponiendo que la tabla también tiene índices non cluster, los mejores resultados se obtienen cuando se utiliza también la opción DROP_EXISTING, junto con el comando CREATE INDEX. La opción DROP_EXISTING incluye optimizaciones que impiden la reconstrucción de alguno de los índices no agrupados dos veces.

SQL Server 2000 tiene un comando llamado **DBCC INDEXDEFRAG,** que se utiliza para desfragmentar índices cluster y non cluster en una tabla o en vistas indexadas. Esto se logra mediante la compactación y la desfragmentación de la hoja nivel del índice a fin de que el orden físico de las páginas del índice coincide con el orden lógico de los nodos, lo que aumenta el rendimiento. Utilizar DBCC INDEXDEFRAG en lugar de DBCC DBREINDEX suele ser beneficioso ya que este comando no incluye bloqueos de la BD por largos períodos de tiempo como DBCC DBREINDEX. Esto significa que se pueden

ejecutar durante el periodo que la BD está en producción sin afectar significativamente el rendimiento, a pesar de que cualquier tarea de mantenimiento de este tipo debería, idealmente, ser programada durante los tiempos de parada o de bajo uso de la misma.
En el lado negativo, DBCC INDEXDEFRAG tarda más tiempo en ejecutarse que DBCC REINDEX, y las estadísticas no se actualizan automáticamente. Esto significa que si utiliza DBCC INDEXDEFRAG, también tendrá que ejecutar UPDATE STATISTICS.

Y para acelerar el indexado de tus bases de datos es necesario estar seguro de que tu base de datos SQL Server y los archivos de log se encuentran físicamente desfragmentador, antes de reindexar tu base de datos. Al asegurarte que tu base de datos y archivos de registro son contiguos (desfragmentados), el re-indexado no sólo será más rápido, sino que requerirá menos recursos de E/S, ayudando al rendimiento global de SQL Server. Si utiliza Windows 2000 o 2003, una utilidad de desfragmentación está disponible para este fin, aunque solo desfragmentará la base de datos SQL Server y los archivos de log cuando estén cerrados. Idealmente, debes usar una utilidad diseñada para desfragmentar bases de datos SQL Server y archivos de log abiertos.

Según Microsoft, **el número total de páginas en una tabla afecta el rendimiento** de **SQL Server como la fragmentación.** Por ejemplo, si una tabla tiene menos de 100 páginas de datos, reindexar la misma para eliminar la fragmentación no va a beneficiar el rendimiento. Esto se debe a que existen otras cosas, tales como las caches por hardware, el cacheo de SQL Server y la funcionalidad de lectura adelantada de SQL Server , las cuales ocultaran el efecto negativo de la fragmentación. Por otra parte, grandes tablas pueden verse beneficiadas con el reindexado, pues debido a su tamaño la fragmentación puede afectar negativamente a la E/S de disco, perjudicando el rendimiento.

CREATE INDEX

Crea un índice relacional en una tabla especificada o una vista de una tabla especificada. Se puede crear un índice antes de que la tabla posea datos. Los índices relacionales se pueden crear en tablas o vistas de otra base de datos especificando un nombre completo de base de datos.

Sintaxis

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
    ON <object> ( column_name [ ASC | DESC ] [ ,...n ] )
    [ WITH <backward_compatible_index_option> [ ,...n ] ]
    [ ON { filegroup_name | "default" } ]

<object> ::=
{
    [ database_name. [ owner_name ] . | owner_name. ]
    table_or_view_name
}

<backward_compatible_index_option> ::=
{
    PAD_INDEX
  | FILLFACTOR = fillfactor
  | SORT_IN_TEMPDB
  | IGNORE_DUP_KEY
  | STATISTICS_NORECOMPUTE
  | DROP_EXISTING
}
```

Argumentos

UNIQUE

Crea un índice único en una tabla o una vista. Un índice único es aquel en el que no se permite que dos filas tengan el mismo valor de clave del índice. El índice clúster de una vista debe ser único.

Motor de base de datos no admite la creación de un índice único sobre columnas que ya contengan valores duplicados, independientemente de si se ha establecido o no IGNORE_DUP_KEY en ON. Si se intenta, Motor de base de datos muestra un mensaje de error. Se deben quitar los valores duplicados para poder crear un índice único en la columna o columnas. Las columnas que se utilizan en un índice único se deben establecer en NOT NULL, dado que varios valores NULL se consideran duplicados cuando se crea un índice único.

CLUSTERED

Crea un índice en el que el orden lógico de los valores de clave determina el orden físico de las filas correspondientes de la tabla. El nivel inferior, u hoja, de

un índice clúster contiene las filas de datos reales de la tabla. Una tabla o vista permite un índice clúster al mismo tiempo.

Una vista con un índice clúster único se denomina vista indizada. La creación de un índice clúster único en una vista materializa físicamente la vista. Es necesario crear un índice clúster único en una vista para poder definir otros índices en la misma vista.

NONCLUSTERED

Crea un índice que especifica la ordenación lógica de una tabla. Con un índice no clúster, el orden físico de las filas de datos es independiente del orden indizado.

Cada tabla puede tener hasta 999 índices no clúster, independientemente de cómo se crean: de forma implícita con las restricciones PRIMARY KEY y UNIQUE, o explícita con CREATE INDEX.

Para las vistas indizadas, solo se pueden crear índices no clúster en una vista que ya tenga definido un índice clúster único.

El valor predeterminado es NONCLUSTERED.

index_name

Es el nombre del índice. Los nombres de índice deben ser únicos en una tabla o vista, pero no es necesario que sean únicos en una base de datos. Los nombres de índice deben seguir las reglas de los identificadores.

column

Es la columna o columnas en las que se basa el índice. Especifique dos o más nombres de columna para crear un índice compuesto sobre los valores combinados de las columnas especificadas. Enumere las columnas que desee incluir en el índice compuesto (en orden de prioridad) entre paréntesis después de table_or_view_name.

Se pueden combinar hasta 16 columnas en la clave de un único índice compuesto. Todas las columnas de una clave del índice compuesto deben encontrarse en la misma tabla o vista. El tamaño máximo permitido de los valores de índice combinado es 900 bytes.

[ASC | DESC]

Determina la dirección ascendente o descendente del orden de la columna de índice determinada. El valor predeterminado es ASC.

INCLUDE (**column [** ,**... n]**)

Especifica las columnas que no son de clave que se agregarán en el nivel hoja del índice no clúster. El índice no clúster puede ser único o no único.

Los nombres de columna no se pueden repetir en la lista INCLUDE y no se pueden utilizar simultáneamente como columnas de clave y que no son de clave. Los índices no clúster siempre contienen las columnas de índice clúster si se define un índice clúster en la tabla.

WHERE <filter_predicate>

Crea un índice filtrado especificando qué filas se van a incluir en el índice. El índice filtrado debe ser un índice no clúster en una tabla. Crea las estadísticas filtradas para las filas de datos en el índice filtrado.

El predicado de filtro utiliza la lógica de comparación simple y no puede hacer referencia a una columna calculada, a una columna UDT, a una columna de tipo de datos espacial o a una columna de tipo de datos hierarchyID. Las comparaciones que utilizan literales NULL no se admiten con los operadores de comparación. En su lugar, use los operadores IS NULL e IS NOT NULL.

A continuación, se muestran algunos ejemplos de predicados de filtro para la tabla Trabajo.BillOfMaterials:

WHERE StartDate > '20000101' AND EndDate <= '20000630'

WHERE ComponentID IN (533, 324, 753)

WHERE StartDate IN ('20000404', '20000905') AND EndDate IS NOT NULL

Los índices filtrados no se aplican a los índices XML ni a los índices de texto completo. Para los índices UNIQUE, solo las filas seleccionadas deben tener valores de índice únicos. Los índices filtrados no admiten la opción IGNORE_DUP_KEY.

ON partition_scheme_name(column_name)

Especifica el esquema de partición que define los grupos de archivos a los que se asignarán las particiones de un índice con particiones. El esquema de partición debe existir dentro de la base de datos mediante la ejecución de CREATE PARTITION SCHEME o de ALTER PARTITION SCHEME. column_name especifica la columna en la que se van a crear las particiones de un índice con particiones. Esta columna debe coincidir con el tipo de datos, la longitud y la precisión del argumento de la función de partición que partition_scheme_name emplea. column_name no está limitado a las columnas de la definición del índice. Se pueden especificar todas las columnas de la tabla base, excepto en el caso de partición de un índice UNIQUE en el que se debe elegir un valor para column_name entre las columnas utilizadas como clave única. Esta restricción permite que Motor de base de datos compruebe la unicidad de los valores de clave en una única partición solamente.

ON filegroup_name

Crea el índice especificado en el grupo de archivos indicado. Si no se ha especificado una ubicación y la tabla o vista no tiene particiones, el índice utiliza el mismo grupo de archivos que la tabla o vista subyacente. El grupo de archivos debe existir previamente.

ON "default"

Crea el índice especificado en el grupo de archivos predeterminado. El término predeterminado (default), en este contexto, no es una palabra clave. Es un identificador para el grupo de archivos predeterminado y debe delimitarse, como en ON **"**default**"** o en ON **[**default**]**. Si se especifica "default", la opción QUOTED_IDENTIFIER debe ser ON para la sesión actual. Ésta es la configuración predeterminada

[FILESTREAM_ON { filestream_filegroup_name | partition_scheme_name | "NULL" }]

Especifica la posición de datos FILESTREAM para la tabla cuando se crea un índice clúster. La cláusula FILESTREAM_ON permite mover los datos FILESTREAM a otro esquema de partición o a otro grupo de archivos FILESTREAM.

filestream_filegroup_name es el nombre de un grupo de archivos FILESTREAM. El grupo de archivos debe tener un archivo definido para el grupo de archivos, utilizando para ello las instrucciones CREATE DATABASE o ALTER DATABASE; de lo contrario, se producirá un error.

Si se crean particiones de la tabla, la cláusula FILESTREAM_ON deberá incluirse y especificar un esquema de partición de grupos de archivos FILESTREAM que utilice la misma función de partición y columnas de partición que el esquema de partición para la tabla. En caso contrario, se produce un error.

Si la tabla no tiene particiones, no se pueden crear particiones en la columna FILESTREAM. Los datos FILESTREAM para la tabla deben estar almacenados en un grupo de archivos único que se especifica en la cláusula FILESTREAM_ON.

NULL de FILESTREAM_ON se puede especificar en una instrucción CREATE INDEX si se va a crear un índice clúster y la tabla no contiene una columna FILESTREAM.

PAD_INDEX = { ON | OFF }

Especifica el relleno de índice. El valor predeterminado es OFF.

ON
El porcentaje de espacio disponible especificado por fillfactor se aplica a las páginas de nivel intermedio del índice.
No se especifica OFF ni fillfactor.

La opción PAD_INDEX solamente resulta útil si también se especifica FILLFACTOR, porque PAD_INDEX utiliza el mismo porcentaje especificado por FILLFACTOR. Si el porcentaje especificado para FILLFACTOR no es lo suficientemente grande como para admitir una fila, Motor de base de datos invalida internamente el porcentaje para permitir el valor mínimo. El número de filas de una página de nivel intermedio del índice no es nunca inferior a dos, independientemente de lo bajo que sea el valor de fillfactor.
En la sintaxis compatible con versiones anteriores, WITH PAD_INDEX es equivalente a WITH PAD_INDEX = ON.

FILLFACTOR =**fillfactor**

Especifica un porcentaje que indica cuánto debe llenar el Motor de base de datos el nivel hoja de cada página de índice durante la creación o nueva generación de los índices. fillfactor debe ser un valor entero comprendido entre 1 y 100. Si fillfactor es 100, el Motor de base de datos crea índices con páginas hoja rellenas en toda su capacidad.

El valor FILLFACTOR solo se aplica cuando se crea o se regenera el índice. Motor de base de datos no mantiene dinámicamente el porcentaje especificado de espacio disponible de las páginas.

La creación de un índice clúster con un valor de FILLFACTOR menor que 100 afecta a la cantidad de espacio de almacenamiento que ocupan los datos, porque Motor de base de datos vuelve a distribuir los datos cuando crea el índice clúster.

SORT_IN_TEMPDB = { ON | OFF }

Indica si deben almacenarse resultados temporales de orden en **tempdb**. El valor predeterminado es OFF.

ON

Los resultados de ordena intermedios utilizados para generar el índice se almacenan en **tempdb**. Esto puede reducir el tiempo necesario para crear un índice si **tempdb** y la base de datos de usuarios están en conjuntos de discos distintos. Sin embargo, esto aumenta la cantidad de espacio en disco utilizado durante la creación del índice.

OFF

Los resultados de orden intermedios se almacenan en la misma base de datos que el índice.

Además del espacio necesario en la base de datos del usuario para crear el índice, **tempdb** debe tener la misma cantidad de espacio adicional para almacenar los resultados de orden intermedio..

En la sintaxis compatible con versiones anteriores, WITH SORT_IN_TEMPDB es equivalente a WITH SORT_IN_TEMPDB = ON.

IGNORE_DUP_KEY = { ON | OFF }

Especifica la respuesta de error cuando una operación de inserción intenta insertar valores de clave duplicados en un índice único. La opción IGNORE_DUP_KEY se aplica solamente a operaciones de inserción realizadas tras crear o volver a generar el índice. La opción no tiene efecto cuando se ejecutan CREATE INDEX, ALTER INDEX o UPDATE. El valor predeterminado es OFF.

ON

Se producirá un mensaje de advertencia cuando se inserten valores de clave duplicados en un índice único. Solo las filas que infrinjan la restricción de unicidad darán error.

OFF
Se producirá un mensaje de error cuando se inserten valores de clave duplicados en un índice único. Toda la operación INSERT se revertirá.

IGNORE_DUP_KEY no se puede establecer en ON para los índices creados en una vista, los índices que no sean únicos, los índices XML, los índices espaciales y los índices filtrados.

En la sintaxis compatible con versiones anteriores, WITH IGNORE_DUP_KEY es equivalente a WITH IGNORE_DUP_KEY = ON.

STATISTICS_NORECOMPUTE = { ON | OFF}

Especifica si se vuelven a calcular las estadísticas de distribución. El valor predeterminado es OFF.
ON

Las estadísticas obsoletas no se vuelven a calcular automáticamente.

OFF

Se habilita la actualización automática de las estadísticas.

Para restaurar la actualización automática de estadísticas, establezca STATISTICS_NORECOMPUTE en OFF o ejecute UPDATE STATISTICS sin la cláusula NORECOMPUTE.

Deshabilitar el cálculo automático de estadísticas de distribución puede impedir que el optimizador de consultas elija los planes de ejecución óptimos de las consultas relativas a la tabla.

En la sintaxis compatible con versiones anteriores, WITH STATISTICS_NORECOMPUTE es equivalente a WITH STATISTICS_NORECOMPUTE = ON.

DROP_EXISTING = { ON | OFF }

Especifica que el índice clúster o no clúster preexistente mencionado debe quitarse y volver a generarse. El valor predeterminado es OFF.

ON

El índice existente se quita y se vuelve a generar. El nombre de índice especificado debe ser el mismo que el de un índice actualmente existente; sin embargo, la definición se puede modificar. Por ejemplo, puede especificar columnas, criterio de ordenación, esquema de particionamiento u opciones de índice diferentes.

OFF

Se muestra un error si ya existe el nombre de índice especificado.

El tipo de índice no puede cambiarse utilizando DROP_EXISTING.

En la sintaxis compatible con versiones anteriores, WITH DROP_EXISTING es equivalente a WITH DROP_EXISTING = ON.

ONLINE = { ON | OFF }

Especifica si las tablas subyacentes y los índices asociados están disponibles para realizar consultas y modificar datos durante la operación de indización. El valor predeterminado es OFF.

La instrucción CREATE INDEX se optimiza como cualquier otra consulta. Para guardar en operaciones de E/S, el procesador de consultas puede elegir examinar otro índice en lugar de realizar un recorrido de tabla. La operación de orden se puede eliminar en algunos casos. En equipos con varios procesadores, CREATE INDEX puede utilizar

más procesadores para realizar las operaciones de examen y orden asociadas a la creación del índice, al igual que hacen otras consultas.

La operación de creación de índices se registra al mínimo si el modelo de recuperación de base de datos se establece en Registro masivo o Sencillo.

Los índices se pueden crear en una tabla temporal. Cuando se quita la tabla o finaliza la sesión, se quitan los índices.

Los índices admiten propiedades extendidas.

Índices clúster

La creación de un índice clúster en una tabla (montón) o la eliminación y nueva creación de un índice clúster existente requiere área de espacio adicional disponible en la base de datos para acomodar la ordenación de datos y una copia temporal de la tabla original o datos del índice clúster existente.

Índices únicos

Cuando existe un índice único, Motor de base de datos comprueba si hay valores duplicados cada vez que se agregan datos con una operación de inserción. Las operaciones de inserción que generarían valores de clave duplicados se revierten y el Motor de base de datos muestra un mensaje de error. Esto se cumple incluso si la operación de inserción cambia muchas filas pero crea un único duplicado. Si se intenta indicar datos donde existe un índice único y se ha especificado la cláusula IGNORE_DUP_KEY en ON, solo causarán un error las filas que infrinjan el índice UNIQUE.

Índices con particiones

La creación y el mantenimiento de los índices con particiones son similares a los de las tablas con particiones pero, al igual que en índices ordinarios, éstos son tratados como objetos de base de datos independientes. Puede tener un índice con particiones en una tabla que carezca de particiones, y puede tener un índice sin particiones en una tabla que tenga particiones.

Si crea un índice en una tabla con particiones y no especifica un grupo de archivos en el que desea ubicar el índice, se crean particiones en el índice de la misma manera que en la tabla subyacente. Esto se debe a que, de manera predeterminada, los índices se ubican en los mismos grupos de archivos que sus tablas subyacentes, y en una tabla con particiones del mismo esquema de partición que usa las mismas columnas de partición. Cuando el índice usa el mismo esquema y columna de partición que la tabla, el índice está alineado con la tabla.

Advertencia

La creación y regeneración de índices no alineados en una tabla con más de 1.000 particiones es posible, pero no se admite. Si se hace, se puede degradar el rendimiento o consumir excesiva memoria durante estas operaciones. Se recomienda usar solo índices alineados cuando el número de particiones sea superior a 1.000.

Cuando se crean particiones en un índice clúster no único, el Motor de base de datos agrega de forma predeterminada las columnas de partición a la lista de claves del índice clúster, en caso de que no se hubieran especificado aún.

Especificar opciones de índice

SQL Server 2005 incluye opciones de índice nuevas y también modifica el modo en que se especifican las opciones. En la sintaxis compatible con versiones anteriores, WITH option_name es equivalente a WITH **(** <option_name> **=** ON **)**. Al establecer opciones de índice, se aplican las siguientes reglas:

- Solo se pueden especificar nuevas opciones de índice mediante WITH **(**option_name**=** ON | OFF**)**.
- Las opciones no se pueden especificar utilizando la sintaxis compatible con versiones anteriores y la nueva sintaxis en la misma instrucción. Por ejemplo, al especificar WITH **(**DROP_EXISTING**,** ONLINE **=** ON**)**, se genera un error en la instrucción.
- Cuando se crea un índice XML, las opciones se deben especificar mediante WITH **(**option_name**=** ON | OFF**)**.

Cláusula DROP_EXISTING

Puede utilizar la cláusula DROP_EXISTING para volver a generar el índice, agregar o quitar columnas, modificar opciones, modificar el criterio de ordenación de las columnas o cambiar el grupo de archivos o el esquema de partición.

Si el índice exige una restricción PRIMARY KEY o UNIQUE, y la definición de índice no se ha modificado en absoluto, se quita el índice y se vuelve a crear conservando la restricción existente. Sin embargo, si se ha modificado la definición de índice, se genera un error en la instrucción. Para cambiar la definición de una restricción PRIMARY KEY o UNIQUE, quite la restricción y agregue una restricción con la nueva definición.

DROP_EXISTING mejora el rendimiento cuando se vuelve a crear un índice clúster (con el mismo conjunto de claves o con uno distinto) en una tabla que también tiene índices no clúster. DROP_EXISTING reemplaza la ejecución de una instrucción DROP INDEX en el antiguo índice clúster seguida de la ejecución de una instrucción CREATE INDEX para el nuevo índice clúster. Los índices no clúster se vuelven a generar una vez, siempre que la definición de índice haya cambiado. La cláusula DROP_EXISTING no vuelve a generar los índices no clúster cuando la definición de índice posee los mismos nombres de índice, clave y columnas de partición, atributo de unicidad y criterio de ordenación que el índice original.

Independientemente de si se vuelven a generar o no los índices no clúster, éstos siempre permanecen en sus esquemas de partición o grupos de archivos originales, y utilizan las funciones de partición originales. Si un índice clúster se vuelve a generar en un esquema de partición o grupo de archivos diferente, los índices no clúster no se mueven para coincidir con la nueva ubicación del índice clúster. Por lo tanto, es posible que incluso los índices no clúster alineados previamente con el índice clúster no se puedan alinear con éste. Para obtener más información sobre la alineación de índices con particiones, vea.

La cláusula DROP_EXISTING no volverá a ordenar los datos si se utilizan las mismas columnas de clave de índice en el mismo orden y con la misma disposición ascendente o descendente, a menos que la instrucción del índice especifique un índice no clúster y la opción ONLINE se establezca en OFF. Si se deshabilita el índice clúster, se debe establecer ONLINE en OFF para la operación CREATE INDEX WITH DROP_EXISTING. Si se deshabilita un índice no clúster y no se asocia con un índice clúster deshabilitado, se puede establecer ONLINE en OFF u ON para la operación CREATE INDEX WITH DROP_EXISTING.

Cuando se quitan o se vuelven a generar índices con 128 o más extensiones, el Motor de base de datos aplaza las cancelaciones de asignación de página reales y los bloqueos asociados, hasta después de que se confirme la transacción.

Opción ONLINE

Las directrices siguientes se aplican para el desarrollo de operaciones de índice en línea:

- La tabla subyacente no se podrá alterar, truncar ni quitar mientras haya una operación de índice en línea en curso.
- La operación de índice requiere un espacio en disco temporal adicional.
- Las operaciones en línea se pueden realizar en índices con particiones e índices que contienen columnas calculadas persistentes, o columnas incluidas.

Opciones de bloqueo de fila y página

Si ALLOW_ROW_LOCKS = ON y ALLOW_PAGE_LOCK = ON, se permiten los bloqueos de nivel de fila, página y tabla cuando se tiene acceso al índice. Motor de base de datos elige el bloqueo apropiado y puede cambiar de escala el bloqueo: de un bloqueo de fila o página a un bloqueo de tabla.

Si ALLOW_ROW_LOCKS = OFF y ALLOW_PAGE_LOCK = OFF, solo se permiten los bloqueos de nivel de tabla cuando se tiene acceso al índice.

Ver información de índice

Para devolver información sobre índices, puede utilizar vistas de catálogo, funciones del sistema y procedimientos almacenados del sistema.

Compresión de datos

- La compresión puede permitir que se almacenen más filas en una página, pero no cambia el tamaño máximo de la fila.
- Las páginas no hoja de un índice no tienen compresión de página pero pueden tener compresión de fila.
- Cada índice no clúster tiene una configuración de compresión individual y no hereda la configuración de compresión de la tabla subyacente.
- Cuando se crea un índice clúster en un montón, el índice clúster hereda el estado de compresión del montón, a menos que se especifique otro estado de compresión.

Las restricciones siguientes se aplican a los índices con particiones:

- No se puede cambiar la configuración de compresión de una partición única si la tabla tiene índices no alineados.
- La sintaxis ALTER INDEX <index> ... La sintaxis REBUILD PARTITION ... vuelve a generar la partición especificada del índice.
- La sintaxis ALTER INDEX <index> ... La sintaxis REBUILD WITH ... vuelve a generar todas las particiones del índice.

EJEMPLO 368
Tomaremos como caso ejemplo crer un índice agrupado no único, para la colmna acultad en la tabla Alumnos

CREATE CLUSTERED INDEX_I_Alumnos ON ALUMOS (Nomres);

EJEMPLO 369
Considerando qe tenemos una tabla llamda Alumnos , nos piden como restablecer una restricción PRIMARY KEY al campo Codigo, resaltando qe debemos considerar un índice agrupado.

Recordemos que cuando generamos un índice agrupado único y no único , la diferencia esta en UNIQUE
INDICE AGRUPADO UNICO
CREATE UNIQUE CLUSTERED Variale ON tabla COLUMNA

INDICE AGRUPADO NO UNICO
CREATE CLUSTERED Variable ON Tabla COLUMNA

Y en una tabla solo pueden existir solamente un índice agrupado, además un indie no agrupado seria NONCLUSTERED

ALTER TABLE ALUMNOS
ADD CONSTRAINT PK_Alumnos PRIMARY KEY CLUSTERED(Codigo);

Obseramos que no se va poder ejecutar ya que existe un índice agrupado y solamente puede haber no por tabla

EJEMPLO 370
Establecer la restricción PRIMARY KEY a la columna Codigo, de la tabla ALUMNOS, especificando que cree un índice no agrupado

ALTER TABLE ALUMNOS
ADD CONSTRAINT PK_Alumnos_cod PRIMARY KEY (Codigo);

Luego revisamos los índices
SP_HELPINDEX Alumnos;

EJEMPLO 371
Crear un una restricción o digamos un índice no agrupado para la clumna facultad de la tabla ALUMNOS

CREATE UNIQUE NONCLUSTERED IDEX I_Alumos_facultad ON Alumnos(Facultad);

EJEMPLO 372
Crear un índice compuesto para la columna NOMRES y la column FACULTAD.

CREATE INDEX I_Alumnos_nomresfac ON Alumnos(Nombres,Facultad);

Observemos que se creo uno agrupado por que no especificamos el ipo, además ya existe uno arupado y o puede existir dos en una misma tabla

Ahora si deseamos visualizar todos los índices creados en la tabla Alumnos, debemos ejecutar:

SELECT NAME FROM SYS INDEXES WHERE NAME LIKE “%ALUMNOS%”

EJEMPLO 373
Crear una restricción UNIQUE para la colmna FACULTAD de la tabla ALMNOS
ALTER TABLE alumnos
Add constraint U_Alumnos_acultad UNIQUE (FACULTAD);

EJEMPLO 374
Ahora visualize todos los índice de la tala Alumos, lugo empleando SYS INDEXES

SP_HELPINDEX ALUMNOS;

SELECT NAME ROM SYSINDEXES WHERE NAME LIKE “%ALUMNOS%”;

EJEMPLO 375
Considerando que tenemos una tabla llamada MATRICULA, y nos piden ejecutar las siguientes respuestas:

- **Genere un índice no agrupado para la columna APELLIDOS**
- **Establezca una restricción PRIMARY KEY para la columna código y especifique que sea un índice agrupado**
- **Luego intente eliminar el índice Index_matricula_apellidos, son especiicar el nombre de la tabla.**
- **Asimismo intente eliminar el índice PK con DROP INDEX**
- **Por ultimo ejecutemos eliminar el índice empleando DROP INDEX**

ALTR TABLE MATRICULA
CREATE NONCLUSTERED INDEX_MATRICULA_APELLIDOS ON MATRICULA (Apellidos);

ADD CONSTRAINT PK_MATRICULA_CODIGO PRIMARY KEY CLUSTERED (Codigo);

DROP INDEX INDEX_MATRICULA_APELLIDOS;

DROP INDEX PK_MATRICULA_CODIGO;

DROP INDEX MATRICULA_INDEX_AMATRICULA_APELLIDOS;

EJEMPLO 376
Como erificar si se eleimino el índice de la tabla anterior

SP_HELPINDEX MATRICULA;

Recordemos que todos los elementos diseñados se les detalla a cada uno por su respectivo nombre.

Identificadores Regulares	**Es el mas común y utilizado no identificado entre mayúsculas y minúsculas**
Identificadores delimitados	**Es aquello que permite conservar los caracteres especiales en identificadores distinguiendo entre mayúsculas y minúsculas y variables de uso paa los identificadores.**

EJEMPLO 377 REPASO DEL CAPITULO II
Como renombrar una base de datos por medio de la sentencia ALTER DATABASE

ALTER DATAASE Nombre_base_datos MODIFY NAME=Nombre_base nuevo;

EJEMPLO 378
Mediante el uso de variables, creareos una tabla y luego insertar registros

```
DECLARE @Tabla table(
        Codigo          INT     PRIMARY KEY,
        Nombres         VARCHAR(60);

        INSERT INTO @Tabla VALUES(1,”MARILUISA PASCAL”);
        INSERT INTO @Tabla VALUES(2,”MARILUIS HARUMI PEREDA”);
```

ANEXO VARIABLES E INIDICES SQL SERVER 2008

SQL Server 2008 introduce algunas mejoras de T-SQL que se utiliza para mejorar el rendimiento de base de datos y mantenimiento.

Declarar e inicializar las variables:

En SQL Server 2008, podemos declarar e inicializar las variables

```
Use master
DECLARE @intA      INT = 100
SELECT SELECT   @intA
```

Resultado:

100

Se puede declarar e inicializar las variables en líneas separadas, de lo contrario, a través de un error

```
Use master
DECLARE @intA   INT = 100
SELECT @intA
```

Error:

```
Msg 139, Level 15, State 1, Line 0
No se puede asignar un valor predeterminado a una variable local.
Msg 137, Level 15, State 2, Line 3
Debe declarar la variable escalar "@ INTA".
DECLARE @intA    INT
SELECT @ IntA = 100 - (o) SET   @intA = 100
SELECT   @intA
```

Resultado

100

Operadores de asignación compuestos:

Los operadores de asignación compuestos son: + =, - =, * =, / =,% =

```
DECLARE @intA    INT = 100
SELECT @intA += 1000  --(or) SELECT @intA = @intA + 1000
SELECT @intA [@intA]
@intA
----- -----
1100
DECLARE @intA  INT = 100
SELECT @intA -= 1000    --(or) SELECT @intA = @intA - 1000
SELECT @intA [@intA]
@intA
----- -----
-900 -900
DECLARE @intA INT = 100
SELECT @intA *= 1000    --(or) SELECT @intA = @intA * 1000
SELECT @intA [@intA]
@intA
------ ------
100000 100000
DECLARE @intA INT = 100
SELECT @intA /= 1000   --(or) SELECT @intA = @intA / 1000
SELECT @intA [@intA]
@intA
----- -----
0 0
DECLARE @intA INT = 100
SELECT @intA %= 1000  --SELECT @ @% = inta inta 1000
SELECT @intA [@intA]
@intA
----- -----
100 100
```

TABLA constructor de valor:

Al insertar el valor de la tabla podemos usar el constructor como la estructura.

```
IF OBJECT_ID ( 'TBL_TABLE1' , 'U' ) IS NOT NULL
DROP TABLE TBL_TABLE1
GO
CREATE TABLE TBL_TABLE1
(
ID Identificación          INT T    IDENTITY ( 1 , 1 ) IDENTIDAD (1, 1)      , ,
COL1 COL1   VARCHAR ( 10 )
COL2 COL2   VARCHAR ( 15 )
)
GO
```

```
INSERT TBL_TABLE1 ( COL1 , COL2 )
VALUES
( 'A'   , 'AA' )
( 'B'   , 'BB' )
( 'C'   , 'CC' )

 GO
INSERT TBL_TABLE1 ( COL1 , COL2 ) VALUES ( 'A' , 'AA' )
INSERT TBL_TABLE1 ( COL1 , COL2 ) VALUES ( 'B' , 'BB' )
INSERT TBL_TABLE1 ( COL1 , COL2 ) VALUES ( 'C', 'CC' )
GO
SELECT ID , COL1 , COL2 FROM TBL_TABLE1
```

Resultado:

ID Identificación	COL1 COL1	COL2 COL2
11	A	AA
22	B	BB
33	C	CC

(3 row(s) affected)

Fecha y tipo de datos TIME:

SQL Server 2008 introduce cuatro FECHA y HORA Los tipos de datos: fecha, hora, datetime2 DATATIMEOFFSET. Proporciona el valor de la splited FECHA y HORA. DATETIME2, es una versión mejorada del tipo de datos DATETIME.. Proporciona los datos de gran tamaño y precisión. DATATIMEOFFSET es el DATETIME2 como un tipo de datos, además de con el componente de zona horaria.

SQL Server 2008

DATE FECHA 3 Bytes à January-01-0001 to December-01-9999 (YYYY-MM-DD)
1947-08-15 à 08/15/1947

EL TIEMPO 3 to 5 Bytes (hh:mm:ss.nnnnnnn)
12:02:13.1234567

DATETIME2 DATETIME2 6 to 8 Bytes à January-01-0001 to December-01-9999 (DATE + TIME) (YYYY-MM-DD (Fecha + Hora)

DATETIMEOFF 8 to 10 Bytes à January-01-0001 to December-01-9999

SQL Server 2005

DATETIME 4 Bytes à January-01-1753 to December-01-9999

SMALLDATETIME 2 Bytes à January-01-1900 to June-06-2079

El formato por defecto y / o máxima es de 7 nanosegundos.

```
DECLARE @ A     DATETIME2 ( 7 )
SELECT @A = '1947-08-15 12:02:13.9370000'
SELECT @ A
```

Resultado:

1947-08-15 12:02:13. **9370000**

```
DECLARE @ A     DATETIME2 ( 4
SELECT   @A = '1947-08-15 12:02:13.9370000'
SELECT @ A
```

Resultado:

1947-08-15 12:02:13. **9370**

Si el nanosegundo no es necesario

```
DECLARE @ A     DATETIME2 ( 0 )
SELECT     @A = '1947-08-15 12:02:13.9370000'
SELECT   @A
```

Resultado:

1947-08-15 12:02:14

Functiones : Fecha y hora:

SYSDATETIME ()

```
SELECT SYSDATETIME ()   [SYSDATETIME()]
```

SYSDATETIME()

2010-04-13 23:49:29.2343750

Al igual que en SQL Server 2005: GETDATE (), excepto en nanosegundos 7 CURRENT_TIMESTAMP formato.

GETDATE () 2010-04-13 23:49:29.230

CURRENT_TIMESTAMP 2010-04-13 23:49:29.230 2010-04-13 23:49:29.230

SYSUTCDATETIME ()

```
SELECT SYSUTCDATETIME () SYSUTCDATETIME()]
```

SYSUTCDATETIME()

2010-04-13 18:23:02.6718750

Al igual que en SQL Server 2005: GETUTCDATE () excepto en nanosegundos 7 Formato.

GETUTCDATE ()2010-04-14 05:19:29.230 (Actual DateTime + Timezone(05:30))

SYSDATETIMEOFFSET ()

```
SELECT SYSDATETIMEOFFSET ()
[SYSDATETIMEOFFSET()]
```

SYSDATETIMEOFFSET()

2010-04-13 23:54:46.9843750 + **05:30**

SWITCHOFFSET (DATETIMEOFFSET, time_zone)

```
SELECT SWITCHOFFSET ( SYSDATETIMEOFFSET (), '-02:12')
```
[SWITCHOFFSET()] [SWITCHOFFSET ()]
Se ajustará y devolver el datetimeoffset con zona horaria (-02:12) dado.
SWITCHOFFSET() SWITCHOFFSET ()
2010-04-14 13:48:41.8906250 -02:12

TODATETIMEOFFSET (expression , time_zone)

```
SELECT TODATETIMEOFFSET ( SYSDATETIMEOFFSET (), '-02:12' )
```
[TODATETIMEOFFSET()]
Se devolverá el DatatimeOffset con la zona horaria (-02:12) dado.
TODATETIMEOFFSET()
2010-04-14 21:43:08.5937500 -02:12

XML

Este tipo de datos permite almacenar un documento XML en la columna de una tabla relacional.

Tipo de datos definidos por el usuario

Se pued definir tipos de datos propios atraves de MANAGEMNT STUDIO o empleando CREATE TYPE

Los procedimiento almacenados SP_ADDTYPE y SD_DROPTYPE, no son recomendables en emplearlos ya que no están presentes en el SQL2008 para adelante.

EJEMPLO 379

Como eliminar un tipo de datos

DROP TYPE [esquema] TYPE Nombre;

EJMEPLO 380

En este ejemplodetallaremos la elaboración de códigos en SQL 2008
Anteriormente escribíamos de esta manera:

```
declare
@fecha datetime = getdate(),
@edad int = 36,
@nombre varchar(100) = Juan;

-- Muestro los valores...
select @fecha, @edad, @nombre
```

Ahora, en SQL Server 2008 es completamente válido.
Veamos esta sintaxis:

```
-- Incremento i, agrego algo al nombre....
  select @edad += 1, @nombre += ' Perez';
-- Muestro los valores...
select @edad, @nombre;
```

Estos operadores funcionan incluso con sentencias DML y columnas. Veamos la forma de incrementar en 100 la columna de una tabla:

```
update tabla set columna1 += 100;
```

o realizar la operación entre dos columnas:

```
update tabla set columna1 += columna2;
```

Para la insercion de datos :

```
INSERT INTO Clientes

    VALUES('Bernardo', 2, 0),

    ('Jose', 3, 1)

    ('Maria', 8, 1)

    ('Isabella', 5, 0)
```

TABLAS PERSISTENTES

EJMEPLO 381

```
IF EXISTS (SELECT * FROM sys.table_types WHERE name = 'Perros')
  DROP TYPE dbo.Perros
CREATE TYPE Perros AS TABLE
(
    Nombre NVARCHAR(20) NOT NULL,
    Peso INT NULL,
    Vacunado BIT NOT NULL
)

GO
DECLARE @Perros AS Perros
INSERT INTO @Perros (Nombre, Peso, Vacunado)
 VALUES
        ('Hachi', 62, 0),
        ('Lucy', 47, 1),
        ('Popy', 59, 1),
        ('Rambo', 42, 1),
        ('Ranger', 34, 0)

SELECT * FROM @Perros

SELECT * FROM (VALUES

                ('Hachiko', 62, 0),

                ('Laica', 47, 1),

                ('Sunday', 59, 1),

                ('Peluza', 42, 1),

                ('Peluche', 34, 0))

    AS Perros(Nombre, Peso, Vacuna)
```

FECHA Y HORA

EJMEPLO 382

Extraer los nuevos tipos de datos de Fecha y Hora haciendo uso de la función GetDate()

```
-- Ejemplo con Hora
SELECT CAST(GETDATE() AS TIME(6)) AS 'Hora'

-- Ejemplo con Fecha
SELECT CAST(GETDATE() AS Date) AS 'Fecha'

-- Ejemplo con DateTime2
SELECT CAST(GETDATE() AS DateTime2) AS 'DateTime2'

-- Ejemplo con DateTimeoffset
SELECT CAST(GETDATE() AS DateTimeoffset(7)) AS 'DateTimeOfSet'

-- Comparativa de los tipos de datos
SELECT CAST(GETDATE() AS TIME(6)) AS 'Hora',
CAST(GETDATE() AS Date) AS 'Fecha'  ,
CAST(GETDATE() AS DateTime2) AS 'DateTime2',
          CAST(GETDATE() AS DateTimeoffset(7)) AS 'DateTimeOfSet'
```

HEARCHYID

EJMEPLO 383

Almacenar la siguiente estructura en una tabla, haciendo uso del tipo de datos, HierarchyID

```
-- CREAMOS LA TABLA
CREATE TABLE Organigrama
(Nodo  hierarchyID PRIMARY KEY,
 Nivel AS Nodo.GetLevel(),
 Empleado INT,
 Nombre VARCHAR(100),
 Titulo VARCHAR(100) )
```

```
-- INSERTAR LA RAIZ
INSERT INTO Organigrama
VALUES(hierarchyid::GetRoot(), 16, 'Juanita Torres', 'Gerente General')

-- VISUALIZAR EL NODO EN TIPO TEXTO
SELECT Nodo.ToString() AS Nodo_Texto, Nivel, Empleado, Nombre, Titulo FROM
Organigrama o

--- INSERTAR EL SIGUIENTE NIVEL
INSERT INTO Organigrama
VALUES( (hierarchyID::GetRoot()).GetDescendant(NULL, NULL),
25, 'Pedro Martinez', 'Gerente de Operaciones' )

-- VISUALIZAMOS LOS NODOS INSERTADOS
SELECT Nodo.ToString() AS Nodo_Texto, Nivel, Empleado, Nombre, Titulo FROM
Organigrama o


-- CREAMOS UN PROCEDIMIENTO ALMACENADO

CREATE PROC Agregar_Organigrama(@Jefe INT, @Empleado INT, @Nombre
VARCHAR(100), @Titulo VARCHAR(100))
AS

DECLARE @JefeNodo hierarchyID, @EmpleadoNodo hierarchyID

SELECT @JefeNodo = Nodo  FROM Organigrama o WHERE o.Empleado = @Jefe

SELECT @EmpleadoNodo = MAX(Nodo) FROM Organigrama WHERE
Nodo.GetAncestor(1) = @JefeNodo

INSERT INTO Organigrama
VALUES(@JefeNodo.GetDescendant(@EmpleadoNodo, NULL), @Empleado,
@Nombre, @Titulo)
```

```
---------- INSERTAR PRIMER REGISTRO ----------
Agregar_Organigrama 25, 12, 'Amanda Juarez', 'Asistente de Gerencia Operaciones';
Agregar_Organigrama 16, 63, 'Mario Torrez', 'Gerente de Mercadeo';
Agregar_Organigrama 63, 5, 'Maria Jiron', 'Ejecutiva de Ventas';
Agregar_Organigrama 16, 71, 'Carlos Aguirre', 'Asistente de Gerencia General';


SELECT Nodo.ToString() AS Nodo_Texto, Nodo, Nivel, Empleado, Nombre, Titulo
FROM Organigrama o
```

HEARCHYID

EJMEPLO 384

Realizar consultas sobre HierarchyID

Aquí una lista de los métodos disponibles en el motor de SQL Server 2008 que dan soporte a HiearchyID:

1. **GetAncestor**, recibe un entero que permite buscar el ancestro n de un nodo hijo.
2. **GetDescendant**, devuelve un nodo hijo que es descendiente de su padre.
3. **GetLevel** , indica el nivel del nodo corriente
4. **GetRoot** , devuelve el nodo raíz del árbol jerárquico
5. **IsDescendantOf**, devuelve verdadero o falso dependiendo de si el nodo es un descendiente de su padre
6. **Parse**, convierte una representación de texto canónica en un HiearchyID
7. **Read**, interpreta una representación binaria de un HiearchyID
8. **Reparent**, permite asignar un nuevo padre a un nodo hijo
9. **ToString**, devuelve una cadena de texto que representa un HierarchyID
10. **Write**, escribe un HiearchyId en un binario

Basado en estos métodos realicemos consultas sobre nuestra estructura creada.

```
--- BUSQUEDA DE RELACION JERARQUICA

DECLARE @Empleado hierarchyID

SELECT @Empleado = Nodo
FROM Organigrama o WHERE o.Empleado = 12
```

```
SELECT * FROM Organigrama o WHERE @Empleado.IsDescendantOf(Nodo) = 1
--- BUSQUEDA DE HIJOS
DECLARE @Empleado hierarchyID

SELECT @Empleado = Nodo
FROM Organigrama o WHERE o.Empleado = 16

SELECT Nodo.ToString(), *   FROM Organigrama o
WHERE Nodo.GetAncestor(1) = @Empleado
```

EJMEPLO 385

Crear un tipo de datos que esté compuesto por las siguientes columnas : Sexo, apellido, nombre y edad

```
CREATE TYPE tdatos AS TABLE(
Sexo        char(1)  CHECK(SEXO IN ("Masculino","Femenino")),
Apellido    Varchar(40),
Nombre      Varchar(40),
Edad        Integer);
```

Otra de la manera para crear un tipo de datos desde el SQL MANAGEMENT STUDIO, desde el explorador de l Base de datos encontraremos las opciones: Diagramas, tablas, vistas, sinónimos, Programación, luego Tipo y dentro de ella Tipos definidos por el usuario y luego Nuevo Tipo de datos definido por el usuario

EJMEPLO 386

Consideremos que tenemos una tabla CLIENTE cuya columna tiene la propiedad IDENTITY, queremos insertar registros

```
USE MASTER
GO
SET IDENTITY_INSERT Cliente ON
INSERT INTO Master.dbo.cliente (Codigo,ubicacion)
        VALUES(100,"Almacen01");
```

IDENT_INCR

Se emplea para conocer el increment del valor IDENTITY

IDENT_SEED

Se emplea para conocer el valor inicial fijado por IDENTITY

EJMEPLO 387

Haciendo uso de variables, diga usted como obtener la tabla Clientes, en el siguiente resultado

```
DECLARE @Monto      float
DECLARE@Tasa        float
DECLARE@x           char(1)
DECLARE@Totalx      int
SELECT @Totalx = COUNT(*) FROM Clientes;
SELECT @Tasa =0.20;
SELECT @Monto = (Pagomes+0.20*Interes*@Tasa) FROM Clientes;
SELECT @x =Sexo
SELECT  "TOTAL DE CLIENTES  = " @totalx;
IF (@x) ="F" PRINT "Sexo Femenino"
ELSE PRINT "Sexo Masculino"
```

NOTA

Si bien sabemos que desde SQL SERVER 2005, DEFAULT ya no tiene validez y no debe ser usado en sus nuevos desarrollos; ya que es preferible definir valores predeterminados en el momento de crear la tabla o al efectuar una modificación mediante ALTER TABLE.

De acuerdo a la tabla de conversiones de Fecha en el capitulo V observamos el uso de la expresión CONVERT , puesto que la manera mas sencilla y evitar errores de conversión es emplear el formato siguiente:
126 – ISO8601 – aaaa –mm – jj Thh;m:ss:mmm (24 horas)

```
DECLARE @Fecha    datetime2;
          SET @Fecha ="2010 – 08 – 27 T 09:00: 20";
SELECT @Fecha
SELECT CONVERT(char,@Fecha,103)
```

EJMEPLO 388

Diga usted como listar todos los nombres de las tablas de un usario identificado para su numero

EXEC Nombre_tabla 1

EJMEPLO 389

Haciendo uso de variables, diga usted como obtener la tabla Clientes, en el siguiente resultado

DECLARE @Monto float

Tipos de JOIN en T-SQL ANEXO REPASO

INNER JOIN

Permite combinar 2 o más tablas a través de al menos un campo en común. Es la unión natural entre las tablas. Los resultados son los datos que tienen un común ambas tablas.

SINTAXIS

SELECT * FROM TABLA1 T1 INNER JOIN TABLA2 T2 ON T1.CampoA = T2.CampoA

LEFT OUTER JOIN

Permite hacer una mezcla y conservar todos los valores de la tabla izquierda (la primera tabla que se menciona en la consulta) sin importar que no tengan equivalente con la de la derecha. Los resultados serán siempre todos los registros de la tabla izquierda TABLA1 sin que exista coincidencia en la otra tabla TABLA2.

SINTAXIS

SELECT * FROM TABLA1 T1 LEFT OUTER JOIN TABLA2 T2 ON T1.CampoA = T2.CampoA

RIGHT OUTER JOIN

Permite hacer una mezcla y conservar todos los valores de la tabla derecha (la segunda tabla que se menciona en la consulta) sin importar que no tengan equivalente con la primera. Los resultados serán siempre todos los registros de la tabla derecha TABLA2 sin que exista coincidencia en la otra tabla TABLA1.

SINTAXIS

SELECT * FROM TABLA1 T1 RIGHT OUTER JOIN TABLA2 T2 ON T1.CampoA = T2.CampoA

CROSS JOIN

Nos permite hacer un producto cartesiano entre las tablas que estamos comparando. Es una multiplicación de ambas tablas. Se puede realizar de manera normal o bien de manera implícita.

---CROSS JOIN NORMAL

SELECT * FROM Tabla1 CROSS JOIN Tabla2

---CROSS JOIN IMPLICITO

SELECT * FROM Tabla1 ,Tabla2

FULL OUTER JOIN

Es la combinación completa. Nos permitirá hacer una mezcla total y conservar todos los valores de ambas tablas, los valores que no tengan equivalencia aparecerán acompañados de un NULL y se mostraran todos los registros.

SELECT * FROM Tabla1 FULL OUTER JOIN Tabla2 ON Tabla1.CampoA = Tabla2.CampoA

Movimiento **de Filas en una tabla**

La instrucción DELETE permite eliminar una o mas filas de una tabla o vista

SINTAXIS:

DELETE FROM Nombre_objeto_o_tabla_o_vista WHERE CONDICION

EJEMPLO 390

Eliminar todas las ventas del operador cuyo código es 2010 y la tabla de ventas es MOVIMIENTO y la de clientes CONTACTOS

DELETE FROM B FROM CONTACTO C, MOVIMIENTO B WHERE B.Codoperador=C.Codigo AND B.Codoperador=2010;

EJEMPLO 391

Consideremos que tenemos una tabla llamada CLIENTES y es necesario listar todas las columnas que estén concatenadas y pueda mostrar los dos primeros códigos.

```
SELECT RTRIM(Nombre)+" "+RTRIM(Apellidos) AS "NOMBRES COMPLETOS ", SUBSTRING(CONVERT(CHAR(8),Codigocli),1,2) CODIGO FROM CLIENTES
```

EJEMPLO 392

Consideremos que tenemos la tabla de CLIENTES, similares al ejemplo anterior pero es necesario listar todos los clientes que tengan mas de 06 registros

```
SELECT SUBSTRING(CONVERT(8),Codigocli),1,2), Cantidad = COUNT(*) FROM CLIENTES GROUP BY SUBSTRING(CONVERT(CHAR(8),Codigocli),1,2) HAVING COUNT(*)>6
```

EJEMPLO 393

Consideremos que tenemos tres tablas relacionadas entre si, las cuales son MOVIM(Movimiento de ventas), ITEMMOVIM (Item de los movimientos de ventas) y la tabla ARTICULOS (Stock de bienes del almacen)

```
SELECT ARTICULOS.Codigobien,ARTICULOS.Detalle_bien,MOVIM.Numdocum, day(MOVIM.Fecha),ARTICULOS.Stock,ARTICULOS.Undmed FROM MOVIM INNER JOIN ITEMMOVIM ON MOVIM.Numdocum =ITEMMOVIM.Numdocum INNER JOIN ARTICULOS ON ITEMMOVIM.Codigobien = ARTICULOS.Codigobien ORDER BY ARTICULOS.Codigobien;
```

EJEMPLO 394

Empleando UPDATE ejecute insertar valores en una tabla

```
UPDATE tCoches
SET  marca = '1'
WHERE marca = 'FORD';
UPDATE tCoches
SET  marca = '2'
WHERE marca = 'RENAULT';
UPDATE tCoches
SET  marca = '3'
WHERE marca = 'SEAT';
```

Utilizando sentencias UPDATE combinadas con subconsultas:

```
UPDATE tCoches
SET  marca = (SELECT CODIGO FROM tMarcas
        WHERE tMarcas.Marca = tCoches.Marca )
WHERE marca IN ('FORD','RENAULT','SEAT');
```

Por cada registro de la tabla tCoches se ejecutará la subconsulta, actualizando el campo marca a el valor del código de la marca en la tabla tMarcas.

El uso de subconsultas para actualizar datos tiene algunas limitaciones:

- La subconsulta sólo puede devover un único campo.
- La subconsulta sólo puede devolver un sólo registro.
- El tipo de datos devuelto por la subconsulta debe ser del mismo tipo que el campo al que estamos asignando el valor.
- No todos los sistemas de bases de datos permiten usar subconsultas para actualizar datos (Access) aunque si una buena parte de ellos (ORACLE, SQL Server, Sybase ...)

Pero en nuestro ejemplo el campo codigo de la tabla tMarcas es numérico y el campo marca de la tabla tCoches es texto. ¿Por qué funciona? Muy facil, el motor de la base de datos es capaz de convertir el valor numérico a un valor texto de forma automática, si bien esta es una excepción.

Ahora que ya tenemos modificado el valor de la marca de los registros, es conveniente modificar su tipo de datos y crear una foreign key contra la tabla tMarcas. Para ello ejecutaremos las siguientes sentencias.

PROCEDIMIENTOS ALMACENADOS

Cuando nos referimos a los procedimientos almacenados que tiene SQL SERVER, nos referimos al uso de los ASPX con el objetivo de poder mejorar el estilo de programación mediante el uso de Store Procedures ó Procedimientos Almacenados..

Que son procedimientos Almacenados: Es un conjunto de codigos del lenguaje que utilizamos dentro de las bases de datos como por ejemplo: create table, insert into, Select from. Que se almacena fisicamente en la base de datos y de la misma forma se exportan cuando creamos una copia de seguridad de nuestra base de datos.

Como funcionan: Los procedimientos almacenados se basan en una estructura definida por SQL SERVER
Iniciamos con:

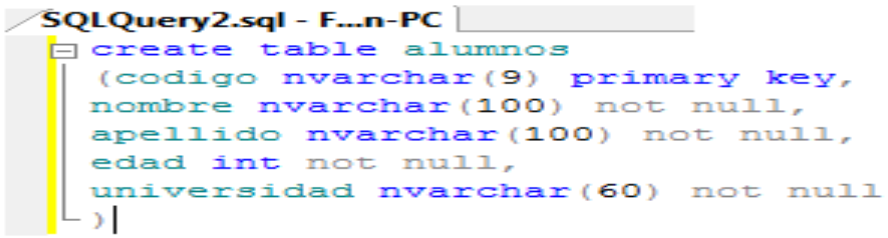
- CREATE PROCEDURE nombre_del_procedimiento @variables tipo(longitud) }
- AS
- BEGIN
- Todo el codigo puro de base de datos
- END

Como probamos: Esto es lo más fácil de realizar solo necesitamos lo siguiente: Escribimos EXEC nombre_del_procedimiento ' variables',12,' variables'

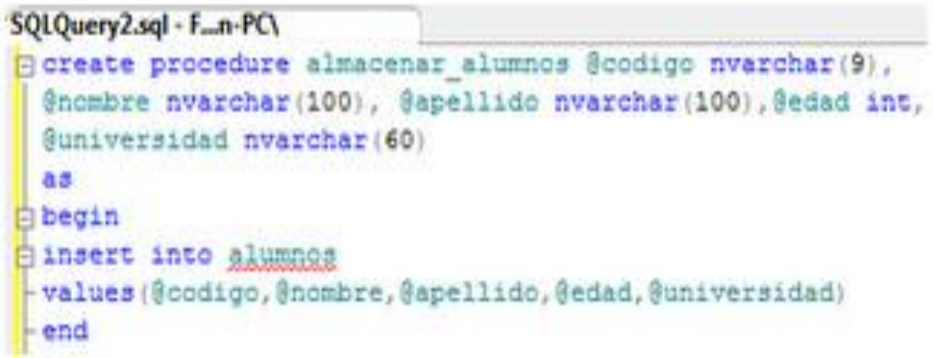
Nota: Es necesario resaltar que no necesariamente se deben definir variables en el caso de un Select * From solo pondremos EXEC y nombre del procedimiento

Vamos a realizar un ejemplo donde aplicaremos lo antes aprendido

- Elaboremos una base de datos
- Luego la siguiente tabla



Ahora definimos el procedimiento almacenado para insertar un nuevo registro(Recordemos que todo lo que esta dentro del Begin puede ser cualquier sentencia de código puro de SQL SERVER: Insert into, Delete From, Update set, etc) Ejecutamos el cogido y ahora probamos con los siguientes datos:



Codigo: 100
Nombre: Cesar
Apellido: Pereda Torres
Edad: 42
Universidad: Inca Garcilaso de la Vega

Ejecutamos el procedimiento almacenado con los datos anteriores tomando en consideración que: los campos tipo int se envian sin comilla el resto dentro de comilla.

Para ejecutar el procedimiento almacenado escribimos EXEC + el nombre del procedimiento + las variables si fuera el caso y clic en ejecutar

Ahora cuenaod revisemos la tabla observaremos el registro ingresado, pues funciona sin ningun problema, para esto es necesario tomar en cuenta que todas las sentencias de SQL SERVER se pueden enviar dentro de un procedimiento almacenado, inclusive inner joins, busquedas &like entre otras.

Ejecutar un Procedimiento Almacenado

Consideremos que desde una sentencia SQL (Select, Insert, Update, Delete, Etc) se puede lograr parametrizar la consulta y posteriormente elaborar su procedimiento almacenado.

```
Vamos a considerar que tenemos una tabla llamada PERSONAL
USE personal;
GO

INSERT INTO dbo.CONTROL_PERSONAL (nombre, sueldo, Afinidad, fecha)
VALUES('Lucky Pereda', 275.50,'Hermano menor',GETDATE());
GO

SELECT TOP 1 * FROM dbo.CONTROL_PERSONAL ORDER BY id DESC;
GO
```

Asegurar que la sentencia SQL que ejecuten se realice correctamente.

Luego : asignarles valores mediante INSERT y sustituirlos

```
USE PERSONAL;
GO

DECLARE @Afinidad varchar(100),
      @sueldo smallmoney,
      @descripcion varchar(750);

SET @nombre = 'Ana Torres';
SET @sueldo = 14550;
SET @Afinidad = 'Tia';

INSERT INTO dbo.CONTROL_PERSONAL (nombre, sueldo, Afinidad, fecha)
VALUES(@nombre, @sueldo,@Afinidad,GETDATE());
GO

SELECT TOP 2 * FROM dbo.CONTROL_PERSONAL ORDER BY id DESC;
GO
```

Ya que tenemos la sentencia parametrizada, el paso final es convertirla en procedimiento almacenado, para esto tenemos que agregar la estructura de un “STORE PROCEDURE – SP” de SQL Server 2008 como lo muestra el siguiente ejemplo:

```
CREATE PROCEDURE <Procedure_Name, sysname, Procedure_Name>
   <@param1, sysname, @p1> <datatype_for_param1, , int> =
<default_value_for_param1, , 0>,
   <@param2, sysname, @p2> <datatype_for_param2, , int> =
<default_value_for_param2, , 0>
AS
   <Instrucciones T-SQL (Insert, Select, Update, Delete, ETC)>
GO
```

Primero es asignarle un nombre al SP luego es crear los parámetros que recibirá el SP. Y asimismo crear el Batch de T-SQL que se ejecutara al llamar a nuestro SP.

El código:

```
CREATE PROCEDURE InsertaAfinidadSP
   @nombre varchar(100),
   @sueldo smallmoney,
```

```
    @Afinidad varchar(750)
AS
    INSERT INTO dbo.CONTROL_PERSONAL (nombre, sueldo, Afinidad, fecha)
    VALUES(@nombre, @sueldo,@Afinidad,GETDATE());
GO
```

La ejecución:

```
USE PERSONAL;
GO

EXEC InsertaAfinidadSP 'Gunacho', 16667.80, 'Titular';
GO

SELECT TOP 3 * FROM dbo.CONTROL_PERSONAL ORDER BY id DESC;
GO
```

RECOMENDACIONES SOBRE EL USO DE TABLAS TEMPORALES

Las tablas temporales se crean en tempdb, y al crearlas se producen varios bloqueos sobre esta base de datos como por ejemplo en las tablas sysobjects y sysindex. Los bloqueos sobre tempdb afectan a todo el servidor.

Al crearlas es necesario que se realicen accesos de escritura al disco (no siempre si las tablas son pequeñas)

Al introducir datos en las tablas temporales de nuevo se produce actividad en el disco, y ya sabemos que el acceso a disco suele ser el "cuello de botella" de nuestro sistema

Al leer datos de la tabla temporal hay que recurrir de nuevo al disco. Además estos datos leídos de la tabla suelen combinarse con otros.

Al borrar la tabla de nuevo hay que adquirir bloqueos sobre la base de datos tempdb y realizar operaciones en disco.

Al usar tablas temporales dentro de un procedimiento almacenado perdemos la ventaja de tener compilado el plan de ejecución de dicho procedimiento almacenado y se producirán recompilaciones más a menudo. Lo mismo pasará cuando el SQL Server intenta reutilizar el plan de ejecución de una consulta parametrizada. Si en la consulta tenemos una tabla temporal difícilmente se reutilizará dicho plan de ejecución.

En vez de tablas temporales podemos mejorar nuestro código para que no sean necesarias, podemos usar subconsultas (normalmente usar una subconsulta mejora drásticamente el rendimiento respecto a usar tablas temporales), usar tablas permanentes, usar tablas derivadas.

Hay que recordar siempre que cualquier alternativa es buena si evitamos usar tablas temporales (¡cursores excluidos por supuesto!)

De todos modos si alguna vez tenemos que usarlas es mejor conocerlas bien, así que vamos a ello.

Tipos de tablas temporales

Las tablas temporales son de dos tipos en cuanto al alcance la tabla. Tenemos tablas temporales locales y tablas temporales globales.

#locales: Las tablas temporales locales tienen una # como primer carácter en su nombre y sólo se pueden utilizar en la conexión en la que el usuario las crea. Cuando la conexión termina la tabla temporal desaparece.

##globales Las tablas temporales globales comienzan con ## y son visibles por cualquier usuario conectado al SQL Server. Y una cosa más, estás tablas desaparecen cuando ningún usuario está haciendo referencias a ellas, no cuado se desconecta el usuario que la creo.

Temp Realmente hay un tipo más de tablas temporales. Si creamos una tabla dentro de la base de datos temp es una tabla real en cuanto a que podemos utilizarla como cualquier otra tabla en cualquier base de datos, y es temporal en cuanto a que desaparece en cuanto apagamos el servidor.

SQL Dinamico en TSQL

Transact SQL permite dos formas de ejecutar SQL dinamico(construir sentencias SQL dinamicamente para ejecutarlas en la base de datos):

- La instrucción **EXECUTE -** o simplemente **EXEC**
- El procedimiento almacenado **sp_executesql**

Desde aquí recomendamos la utilización de sp_**executesql** si bien vamos a mostrar la forma de trabajar con ambos métodos.

La instrucción EXECUTE

La instrucción EXECUTE - o simplemente EXEC - permite ejecutar una cadena de caracteres que representa una sentencia SQL. La cadena de caracteres debe ser de tipo **nvarchar** .

El siguiente ejemplo muestra como ejecutar una cadena de caracteres con la instrucción **EXEC**.

```
DECLARE @sql nvarchar(1000)

 SET @sql = 'SELECT

                    COD_PAIS,

                    NOMBRE_PAIS,

                    ACTIVO,

                    FX_ALTA

            FROM

                    PAISES'

EXEC (@sql)
```

También con SQL dinamico podemos ejecutar sentencias de tipo DDL (Data Definition Languaje), como CREATE TABLE.

```
DECLARE @sql nvarchar(1000)

SET @sql='CREATE TABLE TEMPORAL

            ( ID int IDENTITY, DATO varchar(100))'

EXEC (@sql)
```

```
SET @sql = 'SELECT * FROM TEMPORAL'

EXEC (@sql)
```

El uso de la instrucción EXEC es menos eficiente, en terminos de rendimiento, que sp_executesql.

Para solventar el problema debemos trabajar siempre con sq_executesql, que permite el uso de parametros y con el que obtendremos un mejor rendimiento de nuestras consultas.

El procedimiento almacenado sp_executesql

Para ejecutar sql dinamico, se recomienda utilizar el procedimiento almacenado **sp_executesql**, en lugar de una instrucción **EXECUTE**.

- **sp_executesql** admite la sustitución de parámetros
- **sp_executesql** es más seguro y versátil que **EXECUTE**

- **sp_executesql** genera planes de ejecución con más probabilidades de que SQL Server los vuelva a utilizar, es más eficaz que **EXECUTE**.

El siguiente ejemplo muestra el uso (muy simple) de **sp_executesql**.

```
DECLARE @sql nvarchar(1000)

SET @sql = 'SELECT
                COD_PAIS,
                NOMBRE_PAIS,
                ACTIVO,
                FX_ALTA
            FROM
                PAISES'

EXEC sp_executesql @sql
```

sp_executesql admite la sustitución de valores de parámetros para cualquier parámetro especificado en la cadena Transact-SQL a ejecutar.

El siguiente ejemplo muestra el uso de sp_executesql con parámetros:

```
DECLARE @sql nvarchar(1000),
        @paramDefinition nvarchar(255),
        @paramValue char(3)

SET @paramDefinition = '@codPais char(3)'
SET @paramValue = 'ESP'
SET @sql = 'SELECT
                COD_PAIS,
                NOMBRE_PAIS,
                ACTIVO,
                FX_ALTA
            FROM
                PAISES
            WHERE COD_PAIS = @codPais'
```

```
EXEC sp_executesql @sql, @paramDefinition, @paramValue
```

Insertar valores en una table temporal con todas las tablas y los indices existentes

```
INSERT INTO ##objetos (dbid, database_name, objectid, object_name, indexid,
index_name, xtype)

SELECT dbid, db_name, so.id, so.name, si.indid, si.name, so.xtype FROM

<base_datos>..sysobjects so join

<base_datos>..sysindexes si on so.id = si.id WHERE so.id > 0

-- obtener todos los dbid, objectid

SET NOCOUNT ON

GO

IF NOT OBJECT_ID ('tempdb.dbo.##objetos') IS NULL

DROP TABLE ##objetos

CREATE TABLE ##objetos (dbid INT, database_name SYSNAME, objectid INT,
object_name SYSNAME, indexid INT, index_name SYSNAME NULL, xtype char(2))

DECLARE c1 CURSOR READ_ONLY

FOR SELECT dbid, name FROM master..sysdatabases where dbid >= 5

DECLARE @db_id SMALLINT

, @db_name SYSNAME

OPEN c1

FETCH NEXT FROM c1 INTO @db_id, @db_name

WHILE (@@fetch_status <> -1)

BEGIN

IF (@@fetch_status <> -2)

BEGIN

DECLARE @sql NVARCHAR(4000)

SET @sql = 'INSERT INTO ##objetos (dbid, database_name, objectid, object_name,
indexid, index_name, xtype) '

SET @sql = @sql + 'SELECT ' + CAST(@db_id AS SYSNAME) + ', ''' +
CAST(@db_name AS SYSNAME)

+ ''', so.id, so.name, si.indid, si.name, so.xtype FROM ' +

@db_name + '..sysobjects so join ' +

@db_name + '..sysindexes si on so.id = si.id WHERE so.id > 0'

EXEC (@sql)

END
```

```
FETCH NEXT FROM c1 INTO @db_id, @db_name

END

CLOSE c1

DEALLOCATE c1

 SELECT * FROM ##objetos
```

EJEMPLO_395
Modificar los datos nombres,Apellidos y sexo de la tabla CLIENTES cuyo código es CPT001

```
UPDATE Gune.dbo.CLIENTES SET Nombres =”Mariluisa”,
Apellido=”Pascal”,Sexo=”F”,Direccion=NULL WHERE Codigocli=”CPT001”;
GO
```

MEMORIA AYUDA

Ahora el R2 de SQL SERVER 2008 trae Potentes herramientas de BI en expansión para todos los usuarios con SQL Server PowerPivot para Excel y la potenciación de una nueva clase de los usuarios de negocios para generar y compartir potentes soluciones de BI con poco o ningún soporte de TI, mientras todavía permite TI supervisar y gestionar soluciones de BI generados por el usuario.
Herramientas de análisis autoservicio permiten a los usuarios finales crear rápidamente soluciones dentro de una familiar interfaz de usuario de Microsoft Office Excel de orígenes de datos distintas. Mediante la publicación de estas soluciones en SharePoint Server, los usuarios pueden fácilmente compartirlos con otros. TI obtiene una administración y supervisión por lo que pueden ayudar a garantizar la fiabilidad, rendimiento y seguridad de los activos controladas por datos en toda la empresa, al tiempo que obtiene también una mayor visibilidad de la manera en que las personas utilizan sus datos.

Improve Data Quality

Garantiza la coherencia de presentación de informes a través de datos y sistemas por lo que es más rápido, puede ofrecer resultados más precisos en toda la empresa con herramientas que permiten adoptar un enfoque centralizado para definir, implementar y administrar datos maestros. El concentrador de datos maestros en SQL Server 2008 R2 proporciona a las organizaciones una forma coherente para realizar el seguimiento de las versiones de datos maestros y responder a las preguntas sobre datos maestros en puntos específicos en el tiempo. Datos maestros consistentes mejoran la calidad de los sistemas de datos para toda la empresa y ayuda a mantener los requisitos operacionales y de negocios de inteligencia.

Crear, editar y actualizar los datos maestros eficientemente a través de un portal central. Los portales de la administración de datos maestros proporciona control centralizado de datos maestros, incluidos los miembros y las jerarquías y permite a los administradores de modelo de datos garantizar la calidad de los datos por desarrollar, revisión y gestión de modelos de datos y su aplicación coherente en todos los dominios.

Deliver BI Data Secure

Extender el control efectivo de TI a creado para el uso de soluciones analíticas a través de una consola de administración centralizada utilizando Microsoft SQL Server 2008 R2 y el ***PowerPivot add-in*** para SharePoint Server 2010. En SQL Server 2008 R2, los usuarios finales pueden publicar sus soluciones basadas en datos a través de SharePoint, lo que permite el Departamento de TI administrar de manera efectiva el control de versiones, el acceso de usuario y la utilización de recursos. Esto permite que el Departamento de TI para controlar quién tiene acceso a datos confidenciales y para ayudar a asegurar la disponibilidad coherente de los datos corporativos. También les da mayor conocimiento sobre lo que datos y soluciones de gente utiliza al mismo tiempo, ayudar a prevenir la proliferación de aplicaciones no administradas "sombra".

Enable Self-Service Analysis and Easier Collaboration

Ampliar el alcance de herramientas de inteligencia de negocio a un público más amplio y fomentar el análisis ad-hoc para capacitar a los usuarios crear sus propias aplicaciones analíticas a través de la **PowerPivot** addin de Excel (antes conocido como "Géminis") y SQL Server 2008 R2. Con **PowerPivot**, los usuarios están facultados para crear soluciones mediante datos de fuentes tanto administrado por TI y externas, llevar a cabo avanzados análisis ad-hoc y modelado, extraer el valor de datos más fácilmente y publicar y compartir informes para sus colegas.

Permiten análisis autoservicio manteniendo TI directrices y el control proporcionando plataformas apoyados por TI como SharePoint para publicar sus soluciones

garantizando accesibilidad y TI gobernanza.

Producir nuevos conocimientos y experiencias más ricos mediante la integración de datos geoespaciales junto con otras fuentes de datos corporativos y tipos. SQL Server 2008 R2 admite la asignación, enrutamiento, formas personalizadas y otros datos espaciales y facilita la tarea de combinar con otros datos y mapas de Bing para empresas, crear una nueva clase de soluciones basadas en datos ricas.

Simplificar la recopilación de datos que requieren mucho tiempo y las tareas de consolidación con herramientas que facilitan el crear "datos mash ups", que combinan datos de diferentes fuentes. Datos mash ups proporcionan nuevas maneras de

combinar los datos existentes, desde las bases de datos corporativas o fuentes externas y analizan mediante herramientas familiares en Office Excel.

Build Rich Applications with all Data

Producir nuevos conocimientos y experiencias más ricos mediante la integración de datos geoespaciales junto con otras fuentes de datos corporativos y tipos. SQL Server 2008 R2 admite la asignación, enrutamiento, formas personalizadas y otros datos espaciales y facilita la tarea de combinar con otros datos y mapas de Bing para empresas, crear una nueva clase de soluciones basadas en datos ricas.

Simplificar la recopilación de datos que requieren mucho tiempo y las tareas de consolidación con herramientas que facilitan el crear "datos mash ups", que combinan datos de diferentes fuentes. Datos mash ups proporcionan nuevas maneras de combinar los datos existentes, desde las bases de datos corporativas o fuentes externas y analizan mediante herramientas familiares en Office Excel.

VISTAS – MANTENIMIENTO DE DATOS

En el modelo de datos relacional la forma de guardar la información no es la mejor para ver los datos

VISTAS EN SQL

Una vista es una consulta, que refleja el contenido de una o más tablas, desde la que se puede acceder a los datos como si fuera una tabla.

Dos son las principales razones por las que podemos crear vistas.

- Seguridad, nos pueden interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla.
- Comodidad, como hemos dicho el modelo relacional no es el más comodo para visualizar los datos, lo que nos puede llevar a tener que escribir complejas sentencias SQL, tener una vista nos simplifica esta tarea.

Las vistas no tienen una copia física de los datos, son consultas a los datos que hay en las tablas, por lo que si actualizamos los datos de una vista, estamos actualizando realmente la tabla, y si actualizamos la tabla estos cambios serán visibles desde la vista.

Nota: No siempre podremos actualizar los datos de una vista, dependerá de la complejidad de la misma (dependerá de si el cojunto de resultados tiene acceso a la clave principal de la tabla o no), y del gestor de base de datos. No todos los gestores de bases de datos permiten actualizar vistas, ORACLE, por ejemplo, no lo permite, mientrar que SQL Server si.

CREACION DE VISTAS

Para crear una vista debemos utilizar la sentencia **CREATE VIEW**,
debiendo proporcionar un nombre a la vista y una sentencia SQL **SELECT** válida.

```
CREATE VIEW <nombre_vista>
AS
(<sentencia_select>);
```

Ejemplo: Crear una vista sobre nuestra tabla alquileres, en la que se nos muestre el nombre y apellidos del cliente en lugar de su código.

```
CREATE VIEW vAlquileres
AS
(
SELECT nombre,
       apellidos,
       matricula
FROM tAlquileres,
     tClientes
WHERE ( tAlquileres.codigo_cliente = tClientes.codigo )
)
```

Si queremos, modificar la definición de nuestra vista podemos utilizar la sentencia ALTER VIEW, de forma muy parecida a como lo haciamos con las tablas. En este caso queremos añadir los campos fx_alquiler y fx_devolucion a la vista.

```
ALTER VIEW vAlquileres
AS
(
SELECT nombre,
       apellidos,
       matricula,
       fx_alquiler,
       fx_devolucion
FROM tAlquileres,
     tClientes
WHERE ( tAlquileres.codigo_cliente = tClientes.codigo )
)
```

Por último podemos eliminar la vista a través de la sentencia **DROP VIEW**. Para eliminar la vista que hemos creado anteriormente se uitlizaría:

```
DROP VIEW vAlquileres;
```

Recordemos que una vista se consulta como si fuese una tabla.

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. Sin embargo, a menos que esté indexada, una vista no existe como conjunto de valores de datos almacenados en una base de datos. Las filas y las columnas de datos proceden de tablas a las que se hace referencia en la consulta que define la vista y se producen de forma dinámica cuando se hace referencia a la vista.

Una vista actúa como filtro de las tablas subyacentes a las que se hace referencia en ella.

La consulta que define la vista puede provenir de una o de varias tablas, o bien de otras vistas de la base de datos actual u otras bases de datos. Asimismo, es posible utilizar las consultas distribuidas para definir vistas que utilicen datos de orígenes heterogéneos. Esto puede resultar de utilidad, por ejemplo, si desea combinar datos de estructura similar que proceden de distintos servidores, cada uno de los cuales almacena los datos para una región distinta de la organización.

No existe ninguna restricción a la hora de consultar vistas y muy pocas restricciones a la hora de modificar los datos de éstas.
En esta ilustración se muestra una vista basada en dos tablas.

Las principales razones por las que podemos crear vistas son:

Seguridad, nos pueden interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla.

Comodidad, como hemos dicho el modelo relacional no es el más cómodo para visualizar los datos, lo que nos puede llevar a tener que escribir complejas sentencias SQL, tener una vista nos simplifica esta tarea.

Las vistas no tienen una copia física de los datos, son consultas a los datos que hay en las tablas,
por lo que si actualizamos los datos de una vista, estamos actualizando realmente la tabla, y si actualizamos la tabla estos cambios serán visibles desde la vista.

Nota: No siempre podremos actualizar los datos de una vista, dependerá de la complejidad de la misma (dependerá de si el conjunto de resultados tiene acceso a la clave principal de la tabla o no).

Antes de crear una vista, considere las siguientes indicaciones:
Sólo puede crear vistas en la base de datos actual. Sin embargo, las tablas y las vistas a las que se haga referencia desde la nueva vista pueden encontrarse en otras bases de datos e, incluso, en otros servidores, si la vista se define mediante consultas distribuidas.

Los nombres de las vistas deben seguir las reglas que se aplican a los identificadores y ser únicos para cada esquema. Además, el nombre debe ser distinto del de las tablas incluidas en ese esquema.

Es posible generar vistas dentro de otras vistas. Microsoft SQL Server permite anidar vistas.El anidamiento no debe superar los 32 niveles. Es posible que el límite real del anidamiento de vistas sea inferior en función de la complejidad de la vista y de la memoria disponible.

No puede asociar con las vistas reglas ni definiciones DEFAULT.
Los desencadenadores AFTER no se pueden asociar con las vistas; sólo se pueden asociar los desencadenadores INSTEAD OF.

La consulta que define la vista no puede incluir las cláusulas COMPUTE ni COMPUTE BY, y tampoco puede incluir la palabra clave INTO.
La consulta que define la vista no puede incluir la cláusula ORDER BY, a menos que también haya una cláusula TOP en la lista de selección de la instrucción SELECT.

La consulta que define la vista no puede incluir la cláusula OPTION que especifica una sugerencia de consulta.

La consulta que define la vista no puede incluir la cláusula TABLESAMPLE.
No se pueden definir definiciones de índice de texto completo en las vistas.
No se pueden crear vistas temporales, ni vistas dentro de tablas temporales.
Las vistas, las tablas o las funciones que participan en una vista creada con la cláusula

SCHEMABINDING no se pueden quitar, a menos que se quite o cambie esa vista de forma que deje de tener un enlace de esquema. Además, las instrucciones ALTER TABLE sobre tablas que participan en vistas que tienen enlaces de esquemas provocarán un error si estas instrucciones afectan a la definición de la vista.

Si una vista no se crea con la cláusula SCHEMABINDING, debe ejecutarse sp_refreshview
cuando se realicen cambios en los objetos subyacentes de la vista que afecten a la definición de ésta. De lo contrario, la vista puede generar resultados inesperados cuando se realiza una consulta.

No puede emitir consultas de texto completo en una vista, aunque una definición de vista puede incluir una consulta de texto completo si ésta hace referencia a una tabla configurada para la indización de texto completo.

Debe especificar el nombre de todas las columnas de la vista en el caso de que:

I. Alguna de las columnas de la vista derive de una expresión aritmética, una función integrada o una constante.
II. Dos o más columnas de la vista tuviesen, en caso contrario, el mismo nombre (normalmente, debido a que la definición de la vista incluye una combinación y las columnas de dos o más tablas diferentes tienen el mismo nombre).
III. Desee darle a una columna de la vista un nombre distinto del de la columna de la que deriva. (También puede cambiar el nombre de las columnas en la vista). Una columna de una vista hereda los tipos de datos de la columna de la que deriva, aunque no cambie su nombre.

EJEMPLO 396
Crear una vista sobre la tabla alquileres, en la que se nos muestre el nombre y apellidos del cliente en lugar de su código.

```
CREATE VIEW vAlquileres
AS
(
SELECT nombre,
apellidos,
```

matricula
FROM tAlquileres,
tClientes
WHERE (tAlquileres.codigo_cliente = tClientes.codigo)
)

Si queremos, modificar la definición de nuestra vista podemos utilizar la sentencia ALTER VIEW, de forma muy parecida de cómo se realiza con las tablas. En este caso queremos añadir los campos fx_alquiler y fx_devolucion a la vista.

ALTER VIEW vAlquileres

AS
(
SELECT nombre,
apellidos,
matricula,
fx_alquiler,
fx_devolucion
FROM tAlquileres,
tClientes
WHERE (tAlquileres.codigo_cliente = tClientes.codigo)
)

Por último podemos eliminar la vista a través de la sentencia DROP VIEW. Para eliminar la vista
que hemos creado anteriormente se utilizaría:

DROP VIEW vAlquileres;

VISTAS INDEXADAS

Una vista indexada (o indizada) es una vista que ha "materializado" un conjunto de valores únicos en forma de índice agrupado. Otro nombre que toman estas vistas viene de ahí precisamente: vistas materializadas, el cual es el nombre que toman en SGBDs como Oracle. Su ventaja es que proporcionan una búsqueda rápida para colocar información junto a una vista. Tras el primer índice, el cual ha de ser agrupado de un conjunto único de valores, podemos crear índices adicionales sobre la vista usando la clave agrupada del primer índice como punto de referencia. De todas formas, este tipo de vistas tiene una serie de restricciones sobre cuándo podemos y no podemos crear índices sobre las vistas.

EJEMPLO 397

Crear una tabla llamada Alumnado, asimismo crear una vista e ingresar los índices, cuya base de datos se llame Pruebas

USE Pruebas
GO
CREATE TABLE [dbo].[Alumnado](
[Id_Alumnos] [int] IDENTITY(1,1) NOT NULL,
[Nombre] [varchar](100) NOT NULL,
[Descripcion] [varchar](8000) NOT NULL,
[Id_Pais] [int] NULL)

CREATE TABLE [dbo].[Paises](
[Id_Pais] [int] IDENTITY(1,1) NOT NULL,
[Pais] [varchar](50) NOT NULL)

SET STATISTICS IO ON Hace que SQL Server muestre información relacionada con la cantidad de actividad de disco generada por las instrucciones Transact-SQL.

SET STATISTICS IO ON

Insertemos registros en la tabla Alumnado
INSERT INTO DBO.Alumnado (Nombre,Descripcion,Id_Pais) VALUES
('Almendra','Alumna peruana Alquimia',1)
INSERT INTO DBO.Alumnado (Nombre,Descripcion,Id_Pais) VALUES
('Arabela','Alumna peruana Arabela',1)
INSERT INTO DBO.Alumnado (Nombre,Descripcion,Id_Pais) VALUES
('MaHar','Alumnosa española ',2)

– Insertamos registros en Paises
INSERT INTO DBO.Paises (Pais) VALUES ('Perú')
INSERT INTO DBO.Paises (Pais) VALUES ('España')

```
SELECT dbo.Paises.Id_Pais, dbo.Paises.Pais, COUNT_BIG(*) AS NumeroAlumnado
FROM dbo.Alumnado INNER JOIN
 dbo.Paises ON dbo.Alumnado.Id_Pais = dbo.Paises.Id_Pais
GROUP BY dbo.Paises.Id_Pais, dbo.Paises.Pais
```

Ahora crearemos una vista indexada apartir de la tabla Alumnado

```
CREATE VIEW DBO.NumerodeAlumnado
WITH SCHEMABINDING
AS
SELECT dbo.Paises.Id_Pais, dbo.Paises.Pais, COUNT_BIG(*) AS NumeroAlumnado
FROM dbo.Alumnado INNER JOIN
    dbo.Paises ON dbo.Alumnado.Id_Pais = dbo.Paises.Id_Pais
GROUP BY dbo.Paises.Id_Pais, dbo.Paises.Pais
GO
```

Creamos el indice sobre la vista

```
CREATE UNIQUE CLUSTERED INDEX Id_Pais ON
DBO.NumerodeAlumnado(Id_Pais)
```

Mostramos la vista

```
SELECT * FROM DBO.NumerodeAlumnado
```

Eliminamos las tablas y la vista creada

```
DROP TABLE DBO.Alumnado
DROP TABLE DBO.Paises
DROP VIEW DBO.NumerodeAlumnado
```

Consultas de tablas y de Vistas en SQL

En Sql Server tenemos una sencilla consulta que nos devolverá las tablas y vistas de una base de datos:

SELECT * from Information_Schema.Tables

De aquí nos interesa el campo table_name (nombre de la tabla) y table_type (nos dice si es una tabla o una vista). Por tanto, filtrar por tablas o vistas es bastante sencillo (con el campo table_type).

Para saber si existe una tabla en la base de datos (también nos sirve para las vistas) podemos utilizar la siguiente consulta (vamos a consultar la tabla 'Clientes'):

SELECT * from Information_Schema.Tables where table_name='Clientes';

Si esta consulta nos devuelve registros es que existe la tabla (o vista) y si nos devuelve vacío, es que no existe.

EJEMPLO 398

CREAR LA VISTA PROVEEDORES

```
CREATE VIEW VIEW_CODPROVEEDOR_PRODUCTOS
AS
SELECT S.CODPROVEEDOR,S.NOMBRE,S.DIRECCION
,P.CODPRODUCTO,P.NOMBREPROD P.PUNITARIO
FROM PROVEEDORES  AS S INNER JOIN PRODUCTOS AS P
ON S.CODPROVEEDOR=P.COPROVEEDOR
GO
```

CREACION DE LA VISTA CON INSTRUCCION GROUP

```
CREATE VIEW VIEW_SUBTOTALES(CODIGO_ORDEN,SUB_TOTAL)
AS
SELECT OD.ORDERID,SUM(CONVERT(MONEY,(OD.UNITPRICE* QUANTITY*(1-
DISCOUNT)/100))*100)
FROM [Detalles de Pedido] OD
GROUP BY OD.ORDERID
GO
```

LLAMAR A LA CONSULTA ANTERIOR

```
SELECT * FROM VIEW_SUBTOTALES
```

Ahora procedamos a visualizar las vistas

SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE='VIEW'

/*
SOLO LOS DEL ROL DEL SISTEMA SYSADMIN, O DE EL ROL DE BASE DE DATOS DB_OWNER,
DB_DLADMIN HACEN CONSULTAS
O BIEN AL TENER PERMINOS CREATE VIEW Y PERMISOS DE SELECT EN LAS TABLAS A UTILIZAR
-NO SE PUEDE USAR COMPUTE Ó COMPUTE BY
-NO SE PUEDE USAR ORDER BY SO SI SE USA TOP
-NO SE PUEDE HACER REFERENCIA A TABLAS TEMPORALES
-NO SE PUEDE HACER REFERENCIA A MAS DE 1024 COLUMNAS
-NO SE PUEDE COMBINAR CREATE VIEW EN OTRO LOTE
*/

BORRAR UNA VISTA

DROP VIEW VIEW_SUBTOTALES

ALTERAR UNA VISTA

ALTER VIEW VIEW_SUPPLIER_PRODUCS
AS
SELECT **S.**CODPROVEEDOR,**S.**NOMBRE,**S.**DIRECCION
,P.CODPRODUCTO,**P.**NOMBREPROD **P.**PUNITARIO
FROM **PROVEEDORES AS S** INNER JOIN **PRODUCTOS AS P**
ON
S.CODPROVEEDOR=**P.**COPROVEEDOR
GO

ESCONDER EL CODIGO DE LA VISTA NO ELIMINE LAS ENTRADAS DE syscomments
ALTER VIEW VIEW_SUPPLIER_PRODUCS
WITH ENCRYPTION
AS
SELECT **S.**CODPROVEEDOR,**S.**NOMBRE,**S.**DIRECCION
,P.CODPRODUCTO,**P.**NOMBREPROD **P.**PUNITARIO
FROM **PROVEEDORES AS S** INNER JOIN **PRODUCTOS AS P**
ON
S.CODPROVEEDOR=**P.**COPROVEEDOR
GO

Ahora deseamos consultar toda la información de la vista mediante el uso de **INFORMATION_SCHEMA**
SI ESTA PUESTO WITH ENRCRYPTION ESTA INFORMACION NO ESTA VISIBLE

SELECT * FROM INFORMATION_SCHEMA.TABLES --o sysobjects --Nombres de vistas y tablas
SELECT * FROM INFORMATION_SCHEMA.VIEW_TABLE_USAGE --o select * from sysdepends
--Nombres de las tablas o vistas base.
SELECT * FROM INFORMATION_SCHEMA.VIEWS -- o select * from syscomments --Definición de como se hicieron las vistas.
SELECT * FROM INFORMATION_SCHEMA.VIEW_COLUMN_USAGE -- o syscolumns
--tablas y vistas base y Columnas definidas en una vista.

PARA VER EL SCRIPT DE UNA CONSULTA O INFORMATION_SCHEMA.VIEWS O
SP_HELPTEXT
SP_HELPTEXT [ORDERS QRY]
SP_DEPENDS [ORDERS QRY] --NOMBRES DE TABLA Y SUS DEPENDENCIAS

PUEDE HACER INSERCIONES EN UNA CONSULTA PERO CONSIDERE QUE WITH CHECK OPTION OBLIGA A QUE LAS INSTRUCCIONES DE MODIFICACION SE COMPRUEBEN CONTRA EL WHERE

CREAR UNA VISTA CON WITH CHECK OPTION PARA COMPROBAR EL EJERCICIO
CREATE VIEW CLIENTESARGENTINA
AS
SELECT * FROM CUSTOMERS WHERE COUNTRY='ARGENTINA'
WITH CHECK OPTION

ESTA INSERCION DEBE PRODUCIR SOBRE LA CONSULTA ANTERIOR DEBE PRODUCIR UN ERROR.

INSERT CLIENTESARGENTINA
(CUSTOMERID,COMPANYNAME,CONTACTNAME,CONTACTTITLE,CITY,COUNTRY)
VALUES
('VHCVV','UNIVERSIDAD GALILEO','VICTOR HUGO CARDENAS','VENTAS AGENT','GUATEMALA','GUATEMALA')

CREACION DE UNA TABLA PARA COMPROBAR LA INSERCCION A MULTIPLES TABLAS
CREATE VIEW PROVEEDORES_PRODUCTOS
AS

SELECT **S.**CODPROVEEDOR,**S.**NOMBRE,**S.**DIRECCION
,P.CODPRODUCTO,**P.**NOMBREPROD **P.**PUNITARIO
FROM **PROVEEDORES AS S** INNER JOIN **PRODUCTOS AS P**
ON **S.**CODPROVEEDOR=**P.**COPROVEEDOR

CREACION DEL INDICE, EL PRIMER INDICE DEBE SER UN INDICE UNICO AGRUPADO

CREATE UNIQUE CLUSTERED INDEX CL_SUPPLIERID
ON DBO.PROVEEDORES_PRODUCTOS (SUPPLIERID,PRODUCTID)

CREACION DE OTROS INDICES NO AGRUPADOS

CREATE NONCLUSTERED INDEX VIEW_PRODUCTNAME
ON PROVEEDORES_PRODUCTOS(PRODUCTNAME)

PARA REVISAR SI ES INDEXABLE UNA CONSULTA

SELECT OBJECTPROPERTY
(object_id('DBO.PROVEEDORES_PRODUCTOS'),'IsIndexable')

WITH CHECK OPTION

Exige que todas las instrucciones de modificación de datos ejecutadas contra la vista se adhieran a los criterios establecidos en select_statement. Cuando una fila se modifica mediante una vista, WITH CHECK OPTION garantiza que los datos permanecerán visibles en toda la vista después de confirmar la modificación.

WITH ENCRYPTION

Indica que SQL Server cifra las columnas de la tabla del sistema que contienen el texto de la instrucción CREATE VIEW. Utilizar WITH ENCRYPTION evita que la vista se publique como parte de la duplicación de SQL Server.

SCHEMABINDING

Enlaza la vista al esquema. Cuando se especifica SCHEMABINDING, select_statement debe incluir los nombres con dos partes (propietario.objeto) de las tablas, vistas o funciones definidas por el usuario a las que se hace referencia.

Las vistas o las tablas que participan en una vista creada con la cláusula de enlace de esquema no se pueden quitar ni alterar, de forma que deja de tener un enlace de esquema. De lo contrario, SQL Server genera un error. Además, las instrucciones ALTER TABLE sobre tablas que participan en vistas que tienen enlaces de esquemas provocarán un error si estas instrucciones afectan a la definición de la vista.

VIEW_METADATA

Especifica que SQL Server devolverá a las API de DBLIB, ODBC y OLE DB la información de metadatos sobre la vista, en vez de las tablas o tabla base, cuando se soliciten los metadatos del modo de exploración para una consulta que hace referencia a la vista. Los metadatos del modo de exploración son metadatos adicionales devueltos por SQL Server a las API DB-LIB, ODBC y OLE DB del cliente, que permiten a las API del cliente implementar cursores actualizables en el cliente. Los metadatos del modo de exploración incluyen información sobre la tabla base a la que pertenecen las columnas del conjunto de resultados.

Para las vistas creadas con la opción VIEW_METADATA, los metadatos del modo de exploración devuelven el nombre de vista en vez de los nombres de la tabla base cuando se describen las columnas de la vista en el conjunto de resultados.

Cuando se crea una vista WITH VIEW_METADATA, todas sus columnas (excepto timestamp) son actualizables si la vista tiene los desencadenadores INSERT o UPDATE INSTEAD OF. Consulte Vistas actualizables, más adelante en este capítulo.

EJEMPLO 399

Crear una Vista en relación a la tabla ALUMNOS

```
CREATE VIEW VALUM AS
        SELECT          Tipo=          CONVERT(Char(5),cód_alumno)+"-"+
        Apellido_paterno,Apellido_materno,Edad FROM ALUMNOS
        GO
        SELECT * FROM VALUM;
```

Tipo	Edad
1001 - PEREDA PASCAL	16
1002 - PASCAL AMPUDIA	38
1003 - PEREDA TORRES	42
1004 - PEREDA LUCKY	6

Recordemos que para crear vistas también se pueden emplear mediante SQL SERVER MANAGEMENT STUDIO dentro del explorador de objetos , para luego seleccionar las opciones de vistas, (Nueva Vista)

NOTA

Debemos de dsaber que cuando aplicamos los operadores tales como cadenas, como por ejemplo SUBSTRING , lo podemos efectuar y ejecutar en la misma consulta, de esta manera:

TITULO = SUBSTRING(COLUMNA,1,10)

Uso de SUBSTRING

Esta función devuelve una parte de una cadena binaria o de caracteres, o una cadena de texto, y toma los parámetros siguientes:

- Una cadena de caracteres o binaria, un nombre de columna o una expresión que da como resultado una cadena e incluye un nombre de columna.
- La posición en la que debe empezar la subcadena.
- La longitud, en número de caracteres o en número de bytes para binary,, de la cadena que se va a devolver.

En el ejemplo siguiente se muestra la primera inicial y el apellido de cada empleado.

```
USE Padrones;
GO
SELECT SUBSTRING(FirstName, 1, 1), LastName
FROM Person.Person;
GO
```

En el ejemplo siguiente se muestra el segundo, tercer y cuarto carácter de la constante de cadena abcdef:

```
SELECT x = SUBSTRING('abcdef', 2, 3);
```

La función SUBSTRING () se utiliza para extraer una cadena de caracteres de una posición dada de partida para una longitud dada.

```
SELECT TOP 100
 SUBSTRING (course_designater, 6,3) como 'Número del curso'
Desde cursos
DONDE course_designater LIKE '% Excel'
```

Formatear una columna mediante substring () y LOWER () y UPPER () funciones

Es posible utilizar el INFERIOR SQL Server () y alta () en conjunción con la función SUBSTRING () para llevar a cabo diferentes tipos de formato.

```
SELECT TOP 10
  SUPERIOR (SUBSTRING (apellido, 1,1)) + Baja (SUBSTRING (apellido, 2,29))
  AS 'Apellido'
DESDE estudiantes
```

EJEMPLOS 400 –REPASO

En una tabla llamada VENTAS, que se encuentra en un base de datos llamada ALMACEN, efectuar un incremento del Precio de venta en 20%, con la condición que dicho precio sea mayor o igual que 20

UPDATE Almacen.dbo.Ventas SET P_Venta=P_venta*1.02 WHERE P_costo>=20;

Actualizar el campo UBICACION de la tabla Personal por Region Norte , si y solo si el campo llamado "Desempeño", sea Jefatura

USE Alamacen

UPDATE Almacen.dbo.Personal SET ubicación =”Region Norte” WHERE Desempeño=”Jefatura”;

EJEMPLO 401 - REPASO

Tenemos una tabla llamada "Resultado de Ventas", nos piden actualizar las cantidades de los pedidos de los productos vendidios cuyo código del articulo sea 2020A de la Factura Numero 1000

UPDATE [Resultado de Ventas] SET Cantidad = Cantidad*100 WHERE Factura=1000 and Codigo_articulo =”2020A” ;

EJEMPLO 402 - REPASO

Actualizar el campo CIUDAD con LIMA a todos los registros de loc clientes cuya Provincia sea "PV_LIMA"

UPDATE Almacen.dbo.personal SET Ciudad =”Lima” WHERE Provincia=”PV_LIMA”;

EJEMPLO 403 - REPASO

Dentro de una tabla STOCKS, actualizar la cantidad de stocks del producto 110011, que se encuentre ubicado en el almacén 07, respecto a todas las líneas de pedido que conciernen a ese articulo.

UPDATE Stocks SET Cant_stock =Cant-Stock – (**SELECT SUM(Cant-stock) FROM Lineas _pedido1 WHERE 1.Articulo = stocks.referencia_art)** WHERE Stocks.referencia_art =”110011” and Stocks.Almacen =07;

NOTAS: DELETE y TRUNCATE

Cuando manejamos una base de datos SQL, además de manejar creaciones de tablas (CREATE TABLE), inserciones (INSERT), consultas (SELECT) y actualizaciones (UPDATE); dentro de las operaciones básicas también tenemos las que implican borrado. Borrado de diferentes tipos: de filas que cumplan una serie de condiciones, de todos los datos de una tabla o de la tabla con su estructura. Veamos cada una de ellas, con su sintaxis y un ejemplo.

Manejamos para el ejemplo una tabla entradas, que trata sobre la entradas de un blog y que almacena básicamente la siguiente información: identificador, título, cuerpo y tiempo de salida.

DELETE

Borra una serie de filas de la tabla. Podemos usar una claúsula WHERE para limitar las filas a borrar, a las que cumplan una condición. La sintaxis sería:

DELETE FROM nombre_tabla WHERE condicion

Para nuestro caso:

DELETE FROM entradas WHERE id = 2;

TRUNCATE

A diferencia de DELETE, TRUNCATE elimina todas las filas de la tabla sin borrar la tabla. También resetea los contadores de auto incremento a 0. No borra la tabla como tal, la llamada estructura, por lo que luego puede comenzar a hacer inserciones. La sintaxis es:

TRUNCATE TABLE nombre_tabla;

Y para nuestro caso:

TRUNCATE TABLE entradas;

La instrucción TRUNCATE TABLE es un método rápido y eficiente para eliminar todas las filas de una tabla. TRUNCATE TABLE es equivalente a la instrucción DELETE sin una cláusula WHERE. Sin embargo, TRUNCATE TABLE es más rápida y utiliza menos recursos de registro de sistema y de transacciones.

En comparación con la instrucción DELETE, TRUNCATE TABLE ofrece las siguientes ventajas:

- Utiliza menos espacio de registro de transacciones.

  La instrucción DELETE quita una a una las filas y graba una entrada en el registro de transacciones por cada fila eliminada. TRUNCATE TABLE quita los datos al cancelar la asignación de las páginas de datos utilizadas para almacenar los datos de la tabla y sólo registra la página de asignaciones anuladas en el registro de transacciones.

- Suele utilizar menos bloqueos.

  Cuando la instrucción DELETE se ejecuta mediante un bloqueo de fila, cada fila de la tabla se bloquea para su eliminación. TRUNCATE TABLE siempre bloquea la tabla y la página pero no cada fila.

- Sin excepción, las páginas vacías permanecen en la tabla.

  Después de ejecutar una instrucción DELETE, la tabla todavía puede contener páginas vacías. Por ejemplo, las páginas vacías de un montón no se pueden desasignar sin por lo menos un bloqueo de tabla exclusivo (LCK_M_X). Si la operación de eliminación no utiliza un bloqueo de tabla, la tabla (montón) contendrá muchas páginas vacías. Para los índices, la operación de eliminación puede dejar páginas vacías, aunque éstas se desasignarán rápidamente por medio de un proceso de limpieza en segundo plano.

Como en el caso de DELETE, la definición de una tabla vaciada con TRUNCATE TABLE permanece en la base de datos, junto con sus índices y sus objetos asociados. Si la tabla contiene una columna de identidad, el contador para dicha columna se restablece al valor de inicialización definido para ella. Si no se define ningún valor de inicialización, se utiliza el valor predeterminado 1. Para conservar el contador de identidad, utilice DELETE.

DROP

Finalmente llegamos a DROP. A diferencia de la anterior, DROP no sólo elimina los datos, sino que también eliminar la estructura de la tabla.

DROP TABLE nombre_tabla;

Y para nuestro caso:

DROP TABLE entradas;

DROP DATABASE

A modo de bonus os traigo una instrucción más. Es idéntica a la anterior pero en lugar de borrar una tabla, borra una base de datos al completo. Podemos incluir en la sentencia IF EXISTS de forma que evitemos el error en caso de que no exista la base de datos (muy útil a la hora de hacer copias de seguridad de las bases de datos).

DROP DATABASE [IF EXISTS] nombre_base_datos;

Y para nuestro caso:

DROP DATABASE blog;

Señalar que en MySQL no eliminar la estructura de permisos asociada a la base de datos. Para ello usaremos GRANT, pero de eso ya hablaremos en otra entrada en la que comentaremos los permisos con MySQL.

EJEMPLO 404

Tomar como ejemplo una operación TRUNCATE, para esto consideramos los datos: CLIENTE es una base de datos y CONTROL es una tabla

```
USE CLIENTE
GO
TRUNCATE TABLE CONTROL;
```

EJEMPLO 405

Ahora vamos a desarrollar un ejemplo donde vamos a deshacer una operación

```
USE PADRON
GO
BEGIN TRAN
-- Obtener el numero de filas
        SELECT COUNT(*) FROM PERSONAL
        TRUNCATE TABLE PERSONAL
          -- Ahora vamos a deshacer la operación TRUNCATE
        ROLLBACK TRAN
        -- Ahora podemos observar que al deshacer la operación las filas continúan
                SELECT COUNT(*) FROM PERSONAL;
```

En el capitulo anterior hablamos sobre consultas recursivas, ademas tocamos el temas de estos tipos de complejidades y que pueden ser empledas mediante el uso de SELECT asi como INSERT, UPDATE y DELETE

LAS TABLAS CTE

Las tablas CTE en SQL2008 son muy fáciles. Asi como la utilización de un solo consultas CTE sólo hay un CON y cada definición de CTE está separado de cada coma otro utilizando ","

Después de la sección de definición de los ejemplos de SQL CCMA, se trata de la declaración principal de selección o instrucción SQL de actualización, etc En esta parte del CTE las tablas se pueden utilizar como cualquier tabla de otra base de datos SQL. Usted puede unirse a las tablas de SQL CTE con combinación interna, los tipos de combinación externa, etc

Múltiple Sintaxis CTE

Aquí está la base de sintaxis SQL para permitir CTE cómo los desarrolladores de SQL Server puede utilizar múltiples CTE 's en una sola instrucción SQL SELECT.

```
WITH CTE1 AS (
SELECT TOP 2 Name FROM Ventas.Store
),
CTE2 AS (
SELECT TOP 2 ProductNumber, Name FROM Trabajo.Product
),
CTE3 AS (
SELECT TOP 2 Name FROM Person.ContactType
)
SELECT * FROM CTE1,CTE2,CTE3
-- Or use INNER JOIN, LEFT JOIN instead of Cartesian Joins
```

A pesar de lo anterior CTE múltiple utilizando T-SQL consulta de selección no produce un resultado visible, en el MS SQL Server 2005 o un entorno SQL Server 2008 es probable que encontrar soluciones para sus necesidades de manipulación de datos con varias consultas del CTE.

Aquí es un simple sql múltiples CTE que se compone de dos consultas de CTE en una instrucción Select TSQL.

Si usted descargar e instalar MS SQL Server 2008 bases de datos de ejemplo, puede ejecutar la siguiente instrucción Select múltiples CTE directamente sobre la base de datos de ejemplo AdventureWorks2008.

```
WITH CTE1 AS (
select
ProductID, SUM(OrderQty) as TotalOrderQty
from Ventas.VentasOrderDetail
group by ProductID
),
CTE2 AS (
select
p.ProductID, pc.ProductCategoryID, pc.Name
from Trabajo.Product p
inner join Trabajo.ProductSubcategory psc
on psc.ProductSubcategoryID = p.ProductSubcategoryID
inner join Trabajo.ProductCategory pc
on pc.ProductCategoryID = psc.ProductCategoryID
)
SELECT * FROM (
SELECT
CTE2.ProductCategoryID,
CTE2.Name,
CTE1.ProductID,
CTE1.TotalOrderQty,
rn = ROW_NUMBER() OVER (PARTITION BY CTE2.ProductCategoryID
ORDER BY CTE1.TotalOrderQty DESC)
FROM CTE1
INNER JOIN CTE2 on CTE1.ProductID = CTE2.ProductID
) CTE
WHERE rn <= 3
```

Y la salida de la anterior consulta SQL múltiple CTE es como sigue

ProductCategoryID	Name	ProductID	TotalOrderQty	rn
1	Bikes	782	2977	1
1	Bikes	783	2664	2
1	Bikes	779	2394	3
2	Components	738	1581	1
2	Components	809	1465	2
2	Components	835	1435	3
3	Clothing	712	8311	1
3	Clothing	715	6592	2
3	Clothing	864	4247	3
4	Accessories	870	6815	1
4	Accessories	711	6743	2
4	Accessories	708	6532	3

NOTA

Recordemos que cuando creamos las tablas dentro de la Base de datos estas vana necesitar efectuar intersecciones, actualizaciones y eliminaciones de los datos contenidos en las tablas., pues para esto emplearemos tres instrucciones conocidas como INSERT, UPDATE, Y DELETE.

Pues las modificaciones de los valores de cada columna de las filas existentes se efectúan mediante la instrucción update; esta instrucción puede actualizar varias columnas de varias filas de una tabla. Las instrucción UPDATE puede cambiar los valores de las filas individuales, grupos de filas o de todas las filas de una tabla o vista.

SET	**Especifica la lista de los nombres de columnas o variablesque se van a actualizar**
DEFAULT	**Especifica que el valor predeterminado definido para la columna debe sustituir al valor existente en la columna; también se puede utilizar para cambiar la columna a NULL si no tiene valor predeterminado y en su definición acepta valores NULL.**

EJEMPLO 406

AHORA vamos a generar otro ejemplo, pero para esto vamos a generar una tabla CTE llamada PEDIDO de la tabla MOVIMIENTOS

```
WITH PEDIDO AS (
        SELECT Num_pedido,Nombres, Apellidos FROM MOVIMIENTOS WHERE
        Num_factura BETWEEN 10 AND 30)
        SELECT * FROM PEDIDO ;
```

Recordemos que una vez creada la tabla CTE, es necesario activar y utilizar la tabla, caso contrario generaríamos un error; asimismo que de acuerdo al uso de las tablas CTE pueden ser consideradas como una tabla temporal local (#tabal_temporal).

EJEMPLO 407

Crear una tabla CTE donde esté contemplada la tabla PRODUCTO y cuya condición del filtro sea que la descripción del articulo contenga la cadena NEGR antes del ultimo carácter.

```
WITH ART01 AS(
        SELECT Nombre_art,marca,Precio FROM PRODUCTO WHERE Nombre_art
        LIKE “%NEGR_”)
        SELECT * FROM ART01;
```

Lo que debemos saber es que las tablas CTE son mas fáciles de crear que las tablas temporales y además no se debe utilizar en ellas las instrucciones COMPUTE, ORDER BY, INTO, FOR y XML ni mucho menos FOR BROWSE.

MEMORIA AYUDA

SEGURIDAD EN SQL 2008.

La seguridad sigue siendo un punto muy importante dentro de la tecnología y ha crecido aun mas su importancia ahora que las redes se han interconectado. Toda la información debe ser protegida, particularmente las bases de datos cuya información es de vital importancia para las empresas.

SQL Server ahora puede usar claves de cifrado almacenadas en un módulo externo de seguridad de hardware de terceros. Asimismo los datos almacenados en SQL Server se pueden cifrar en un método que es transparente para las aplicaciones que se conectan a la base de datos.

Esto significa que los administradores de bases de datos pueden cifrar fácilmente todos los datos almacenados en una base de datos entera sin tener que modificar el código de aplicación existente. La primera mejora la hace posible la nueva característica Administración extensible de claves (EKM, Extensible Key Management), que está disponible en las ediciones Enterprise, Developer y Evaluation de SQL Server 2008. EKM posibilita que los proveedores de terceros de soluciones de administración de claves empresariales y módulos de seguridad de hardware (HSM, Hardware Security Module) registren sus dispositivos en SQL Server. Una vez que estos dispositivos están registrados, los usuarios pueden usar las claves de cifrado almacenadas en estos módulos. Estos proveedores pueden incluso exponer características de cifrado avanzadas (como el vencimiento y la rotación de las claves) en estos módulos. En algunas configuraciones, esto permite la protección de datos por parte de administradores de bases de datos que no son miembros del grupo de administradores del sistema. Las instrucciones de cifrado de T-SQL pueden entonces cifrar y descifrar los datos usando las claves almacenadas en el dispositivo EKM externo. Otra característica nueva, el cifrado de datos transparente, le permite cifrar archivos de base de datos sin tener que alterar ninguna de sus aplicaciones. Realiza cifrado y descifrado de E/S en tiempo real de los archivos de datos y de registro. El cifrado usa una clave de cifrado de base de datos (DEK, Database Encryption Key) que se almacena en el registro de arranque de base de datos para adquirir disponibilidad durante la recuperación. La DEK está protegida mediante un certificado almacenado en la base de datos maestra del servidor.

Además de la seguridad que podemos poner a través de políticas de grupo y la del propio sistema operativo, SQL Server 2008 cuenta con un esquema de seguridad muy interesante tanto para la validación del usuario como la seguridad de los datos.
Aunque la autenticación todavía implica el determinar y verificar quien es el usuario que intenta acceder a la base de datos, es necesario mejorar los métodos de autenticación existentes. SQL Server 2008 proporciona características de autenticación más robustas que proveen un mejor soporte al servidor y los datos.
SQL Server Authentication proporciona la autentificación para los clientes no-Windows o para las aplicaciones usando una secuencia simple de conexión que contiene las identificaciones del usuario y contraseñas. Este tipo de logon es fácil de usar y popular entre los desarrolladores de aplicación, sin embargo no es tan seguro como la autenticación de Windows y no es recomendado como un mecanismo de autenticación.
SQL Server 2008 tiene varias mejoras en la autenticación con SQL. Primero, Soporta la encripción del canal por default a través del uso de certificados generados por SQL. Los administradores no tienen que adquirir e instalar un certificado SSL válido para asegurar que el canal por el que viajan las credenciales del usuario de SQL Sea seguro. Con SQL Server 2008 automáticamente generas estos certificados, el cual encripta el canal de manera automática y por default cuando se transmiten los paquetes de validación. Esto ocurre con los clientes de SQL Server 2005 o superior. El certificado nativo generado por SQL Server protege contra el ataque pasivo del hacker cuando este esta monitoreando la red para tratar de sacar provecho y obtener acceso a la información
Los permisos en SQL Server son ahora granulares, de tal forma que ahora puedes asignar permisos específico en lugar de otorgarlos en base a un role que probablemente le de mas permisos de los que ocupa. ahora tienes por mucho más entidades seguras, a las cuales puedes asignar los permisos que son más granulares. Además de la protección reforzada para los datos de usuario, la información estructural y los meta datos sobre una entidad particular están disponibles ahora solamente para los usuarios principales que tienen permiso para tener acceso a dicha entidad.

Mejoras de la autenticación Como probablemente sepa, Kerberos es un protocolo de autenticación de redes que proporciona un medio sumamente seguro para autenticar mutuamente las entidades cliente y servidor (o las entidades principales de seguridad) en una red. Kerberos ayuda a los usuarios a mitigar vulnerabilidades de seguridad como los ataques por seducción o de tipo "man-in-the-middle". En relación con la autenticación NTLM de Windows®, Kerberos es más seguro, más robusto y ofrece un mejor rendimiento. Para autenticar una conexión mutuamente con Kerberos, los nombres principales de servicio (SPN, Service Principal Nombress) de una instancia de SQL Server deben registrarse en Active Directory®, y un controlador de cliente debe proporcionar un SPN registrado durante la conexión. En SQL Server 2008, la autenticación Kerberos se ha ampliado a todos los protocolos de red, incluidos TCP, Canalización con nombre, Memoria compartida y Adaptador de interfaz virtual (VIA). De forma predeterminada, el controlador de cliente infiere automáticamente un SPN correcto para una instancia de SQL Server a la que se conecta. También puede especificar explícitamente un SPN en el parámetro de cadena de conexión para mejorar la seguridad, el control y la solución de problemas. Internet Information Services (IIS) ya no proporciona acceso a ASP.NET, Administrador de informes o al servicio web del servidor de informes. En SQL Server 2008, Reporting Services administra todas las solicitudes de autenticación a través de un nuevo subsistema de autenticación que es compatible con la autenticación basada en el Windows y personalizada. Ahora, Reporting Services hospeda las tecnologías de Microsoft® .NET Framework y ASP.NET incorporadas en el Common Language Runtime (CLR) de SQL Server y usa las capacidades del sistema operativo. El servidor de informes incluye un escucha HTTP que acepta las solicitudes y las dirige a la dirección URL y al puerto definidos durante la configuración del servidor. Ahora las reservas y el registro de direcciones URL se administran directamente por el servidor de informes a través de HTTP.SYS.

Con SQL Server 2005 tu puedes encriptar la base de datos escribiendo un código personalizado en el Transact SQL que usa las capacidades criptograficas del motor de bases de datos. SQL Server 2008 tiene mejoras en base a esta situación y es el uso de la encriptación transparente de datos. La encriptación transparente de datos realiza todas las operaciones criptográficas a nivel base de datos, lo cual elimina cualquier necesidad de que un desarrollador tenga que crear un código personalizado de cifrado y descifrado de los datos. Los datos son encriptados el momento en que están siendo grabados en el disco y los desencripta en el momento en que son leídos. Usando el administrador de SQL para el encriptado y desencriptado de datos transparente puedes asegurar los datos del negocio en la base de datos sin tener que solicitar cambios a las aplicaciones existentes

Auditorías de la seguridad

SQL Server Audit es una característica nueva que le permitirá crear auditorías personalizadas de eventos de motor de base de datos. Esta característica usa eventos aumentados para registrar información para las auditorías y proporciona las herramientas y procesos necesarios para habilitar, almacenar y visualizar las auditorías en varios objetos de servidor y base de datos. Además, SQL Server Audit es más rápido que SQL Server Trace y SQL Server Management Studio facilita la creación y supervisión de los registros de auditoría. Ahora puede auditar a un nivel más granular, capturando instrucciones SELECT, INSERT, UPDATE, DELETE, REFERENCES y

EXECUTE para usuarios individuales. Además, SQL Server Audit se puede incluir totalmente en scripts con las instrucciones de T-SQL CREATE SERVER AUDIT y CREATE SERVER AUDIT SPECIFICATION y sus instrucciones relacionadas ALTER y DROP. Para establecer la auditoría, debe crear una auditoría y especificar la ubicación en la que se registrarán los eventos auditados . Las auditorías se pueden guardar en el registro de seguridad de Windows, el registro de aplicaciones de Windows o en un archivo situado en la ubicación que especifique. A continuación, le da un nombre a la auditoría y configura sus características, como la ruta de acceso al archivo de auditoría y su tamaño máximo. También puede seleccionar que se cierre SQL Server si la auditoría genera errores. Y si necesita registrar los eventos auditados en más de una ubicación, sólo tiene que crear más de una auditoría. El siguiente paso consiste en crear una o más especificaciones de auditoría. Una especificación de auditoría de servidor recopila información acerca de la instancia de SQL Server e incluye objetos de ámbito de servidor, como los inicios de sesión y la pertenencia a funciones de servidor. Incluye también información de base de datos que se administra en la base de datos maestra, como el derecho de acceso a una base de datos. Al definir una especificación de auditoría, también indica qué auditoría recibirá los eventos supervisados. Puede definir varias auditorías de servidor y múltiples especificaciones de auditoría de servidor, pero cada auditoría de servidor puede incluir sólo una especificación de auditoría de servidor habilitada a la vez. También puede crear especificaciones de auditoría de base de datos que supervisen los eventos de una sola base de datos. Puede agregar varias especificaciones de auditoría de base de datos a una auditoría, pero una auditoría de servidor sólo puede habilitar una especificación de auditoría de base de datos por cada base de datos a la vez. Los eventos de acción de auditoría de SQL Server que se usan para las especificaciones de auditoría de servidor se agrupan en colecciones de eventos de acción de auditoría relacionados. Éstos se exponen como grupos de acción de auditoría. Al agregar un grupo a la especificación de auditoría, podrá supervisar todos los eventos incluidos en ese grupo. Por ejemplo, hay un grupo de acción de auditoría llamado DBCC_GROUP que expone los comandos DBCC. Sin embargo, los comandos DBCC no están disponibles para auditorías individuales. Hay 35 grupos de acción de auditoría disponibles para el servidor, algunos de los cuales están estrechamente relacionados entre sí. Por ejemplo, hay un SUCCESSFUL_LOGIN_GROUP, un FAILED_LOGIN_GROUP y un LOGOUT_GROUP. Hay también un tipo de acción de auditoría AUDIT_ CHANGE_GROUP que puede usar para auditar el proceso de auditoría. Las especificaciones de auditoría de base de datos también pueden especificar grupos de eventos de acción de auditoría reunidos en grupos de acción de auditoría a nivel de base de datos. Además de los grupos de acción de auditoría, la especificación de auditoría de base de datos puede incluir eventos de acción de auditoría individuales para auditar instrucciones de lenguaje de manipulación de datos. Estos eventos se pueden configurar para supervisar toda la base de datos o algunos objetos específicos de base de datos. La acción de auditoría SELECT, por ejemplo, se puede usar para auditar consultas SELECT para una sola tabla o un esquema entero. Estos eventos también se pueden configurar para supervisar las acciones por funciones o usuarios específicos, como todo db_writers. Por ejemplo, puede usar la acción de auditoría SELECT para auditar las consultas SELECT de una sola tabla por el nombre de usuario María y la función de base de datos FINANCE_DEPT o base de datos pública. Claramente, esto ofrece mucho control y flexibilidad para crear las auditorías que necesita.

AUDITORIA DE SEGURIDAD

En versiones anteriores de SQL Server, dispone una variedad de herramientas para auditar acciones realizadas contra el servidor de SQL Server.

En la siguiente tabla se le muestra las distintas técnicas utilizadas hasta ahora con sus ventajas e inconvenientes:

Técnica	Ventajas	Desventajas
Auditoría C2	Captura casi todos los eventos a disco	Disminuye el rendimiento del servidor
DML Triggers	Controla INSERT DELETE UPDATE	Habría que crearlo en cada tabla Penaliza el rendimiento
DDL Triggers	Controlan cambios en la definición de objetos	Se generan por código. No todos los eventos están disponibles
Trazas del Profiler	Captura muchos tipos de eventos	Limitaciones con los filtros Si se detiene el servicio, se detiene la traza

Algunos ejemplos de que técnica se usaría para cada caso, son los siguientes:

- Como controlo quien cambia la definición de una tabla: **DDL Trigger**
- Como se quien realiza cada INSERT en una tabla: **DML Trigger**
- Como puedo saber las consultas que realiza un usuario en una base de datos: **SQL Profiler con filtros**

Como puede deducir, al final debería usar una combinación de distintas técnicas para tener bien auditado su servidor, lo cual eleva la carga administrativa sustancialmente.
En SQL Server 2008, la auditoría va más allá. Gráficamente, podrá recoger cualquier evento que ocurra en el servidor o en una base de datos en concreto agregando los filtros que necesite con una única herramienta: **La auditoría**

Implica el seguimiento y registro de eventos que ocurren en el sistema. Basándose en la información acumulada que sería capaz de rastrear los cambios en la base de datos, el acceso a la base de datos, etc Una auditoría es la combinación de varios elementos en un solo paquete para un grupo específico de acciones de servidor o base de datos de acciones. Los componentes de SQL Server Audit se combinan para producir una salida que se llama una auditoría, así como una definición de informe junto con los gráficos y elementos de datos produce un informe. Auditoría de SQL Server utiliza extendido eventos para ayudar a crear una auditoría.

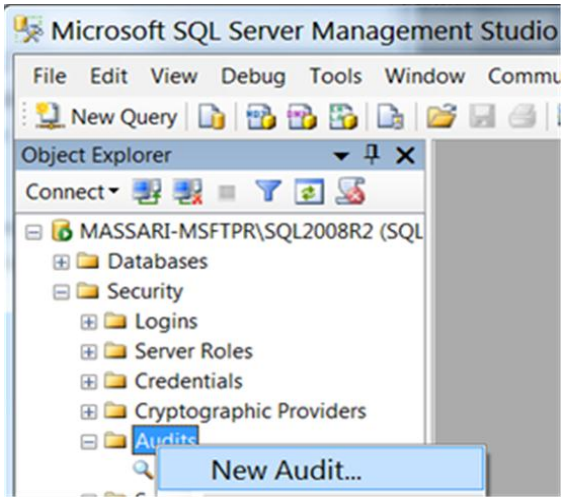
Principios Basicos Porque auditamos
Tener políticas de seguridad es un factor crítico para mantener los datos seguros. La auditoría también nos ayuda a identificar lo que accedió un intruso en el evento de que un ataque sea exitoso.
En SQL 2005 el enfoque estaba orientado a asegurarse de que el usuario no tuviera privilegio mayores a los que necesitaba y que los cambios fueran realizados por el personal autorizado. La herramienta que se promovía para auditar los cambios a la base de datos era el SQL Profiler ya que podía auditar cambios al esquema, operaciones de insertar, actualizar o eliminar registros; y eventos relacionados con cambios de permisos o la creación de nuevos Logins.

Una de las mejoras que trajo SQL Server 2008 Enterprise es un incremento en la capacidad de auditoría a través del uso del SQL Server Audit. A través de esta funcionalidad se puede rastrear y registrar de forma automática los eventos que ocurren a nivel del servidor o a nivel de la base de datos. Esto es posible a través del uso del objeto Audit. Veamos entonces como crear una auditoría, como crear y habilitar una especificación de auditoría a nivel de Servidor o a nivel de base de datos y como visualizar los registros de auditoría.

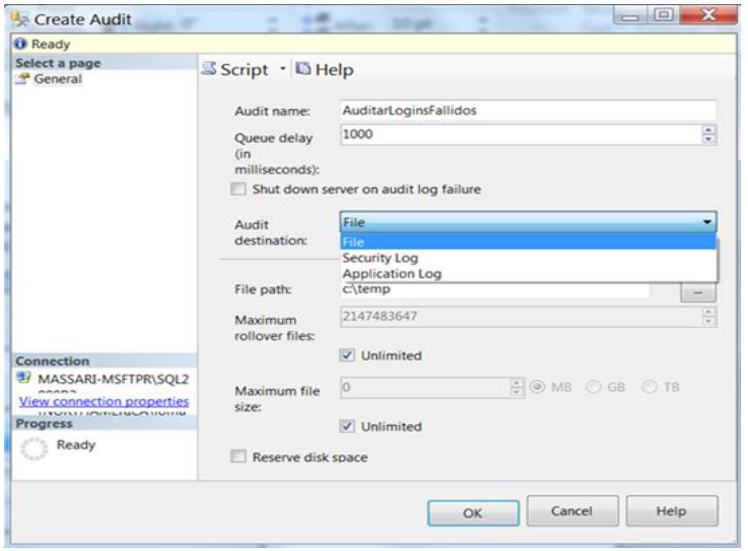
Si bien estamos trabajando con SQL Server 2008 de auditoría que tenemos que tener presentes cuatro cosas en mente:
1.SQL Server Audit
2.Especificación de auditoría de servidor (Eventos para capturar en el nivel de instancia de servidor)
3.Base de datos de auditoría pliego de condiciones (Eventos para capturar en una base de datos específica)
4.Target (sería el caso de los eventos se registra)

Crear una Auditoría
Un objeto de auditoría es una colección de una o más acciones individuales o un grupo de acciones que podrán ser rastreadas. Por ejemplo, se puede configurar un objeto de auditoría para identificar todos los logins fallidos. Los eventos se escriben en la localización que se especifique. Se pueden almacenar en un archivo, la bitácora de eventos de aplicaciones o la bitácora de eventos de seguridad.

El objeto de auditoría se puede crear a través del Management Studio (SSMS) o utilizando T-SQL. Desde SSMS se debe presionar el botón de la derecha del mouse sobre la opción New Audit localizada en la carpeta de auditoría bajo el árbol de Seguridad, como se muestra a continuación:



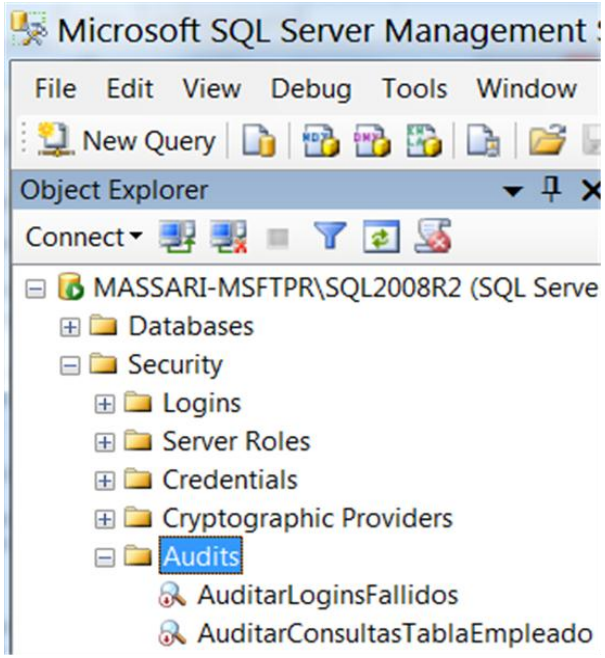
En la pantalla Create Audit se debe ingresar el nombre del objeto de auditoría, y se debe especificar el destino. Si se indica que el destino será un archivo, hay que especificar la ruta donde será almacenado. Finalmente se debe presionar el botón OK para crear el objeto de auditoría.



Para propósitos de este ejemplo cree un segundo objeto de auditoría e ingrese el nombre “AuditarConsultasTablaEmpleado”. Seleccione como destino un archivo e indique la ruta donde desee almacenarlo. Ambos objetos estarán localizados bajo la carpeta Audits, como se muestra a continuación. Este objeto de auditoría se utilizará para rastrear las transacciones SELECT realizadas contra la tabla HumanResources.Employee de la base de datos Padrones.

SQL Server Audit

El objeto de SQL Server Audit recoge una sola instancia de servidor o base de datos de las acciones a escala y los grupos de acciones para el seguimiento. La auditoría es a nivel de la instancia de SQL Server. Usted puede tener múltiples auditorías por cada instancia de SQL Server. Cuando se define una auditoría, se especifica la ubicación de la salida de los resultados. Este es el destino de auditoría. La auditoría se crea en un estado de movilidad reducida, y no de forma automática todas las acciones de auditoría. Después de la auditoría está habilitada, el destino de la auditoría recibe los datos de la auditoría.

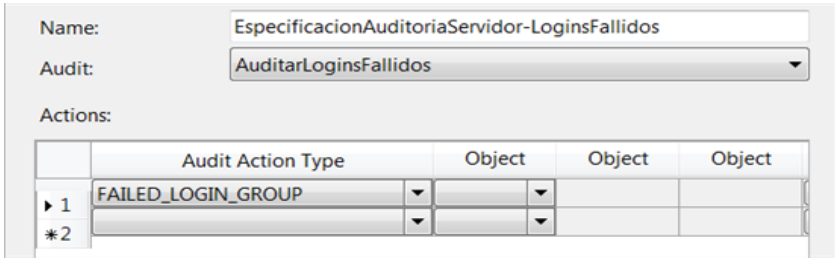


Si desea crear un objeto de auditoría a través de T-SQL lo puede hacer utilizando el comando CREATE SERVER AUDIT. La siguiente consulta crea el objeto "AuditarConsultasTablaEmpleado". Esta operación fue realizada anteriormente a través de SSMS.

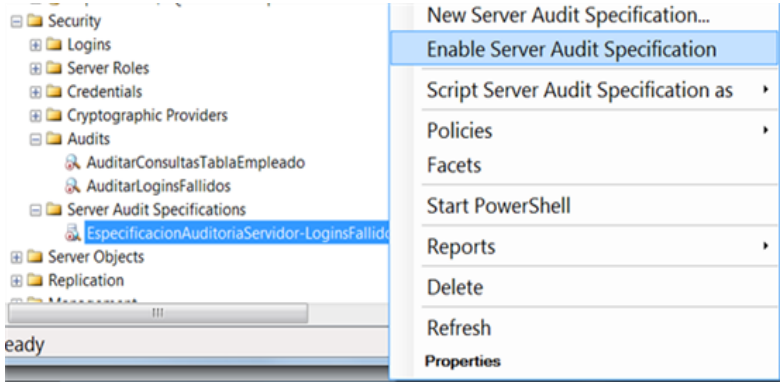
```
USE master
CREATE SERVER AUDIT [AuditarConsultasTablaEmpleado]
TO FILE  (FILEPATH = N'C:\TEMP');
```

Crear y habilitar una especificación de auditoría a nivel de Servidor

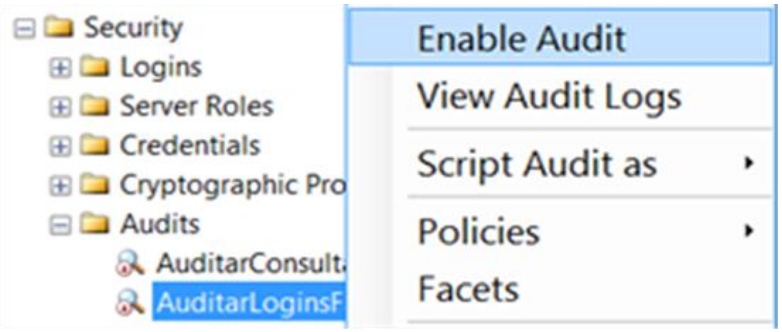
Una vez se ha creado los objetos de auditoría, el siguiente paso es crear las especificaciones apropiadas de auditoría. Las especificaciones de auditoría le indican al objeto de auditoría lo que debe rastrear. En el caso del objeto de auditoría llamado "AuditarLoginsFallidos", debemos crear una especificación que busque los logins que no son exitosos. Para lograrlo debemos seleccionar el botón de la derecha del mouse sobre la carpeta "Server Audit Specifications" ubicada bajo el árbol de Seguridad.
Asigne a la especificación de auditoría el nombre: "EspecificacionAuditoriaServidor-LoginsFallidos". Bajo audit seleccione la opción "AuditarLoginsFallidos". Esto asignará la especificación de auditoría "EspecificacionAuditoriaServidor-LoginsFallidos" al objeto de auditoría "AuditarLoginsFallidos". Seleccione el tipo de acción para auditar: "FAILED_LOGIN_GROUP" y presione OK para crear y asignar el objeto de auditoría.



Presione el botón de la derecha del mouse sobre "EspecificacionAuditoriaServidor-LoginsFallidos" para habilitar la especificación a través de la opción: "Enable Server Audit Specification"



Finalmente debe habilitar el objeto de auditoría presionando el botón de la derecha del mouse sobre "AuditarLoginsFallidos" para habilitar la auditoria a través de la opción Enable Audit como se muestra en la siguiente figura.



Si desea crear una especificación de auditoría a nivel de servidor, a través de T-SQL, lo puede hacer utilizando el comando CREATE SERVER AUDIT SPECITICATION. La siguiente consulta crea la especificación de auditoría "EspecificacionAuditoriaServidor-LoginsFallidos". Esta operación fue realizada anteriormente a través de SSMS.

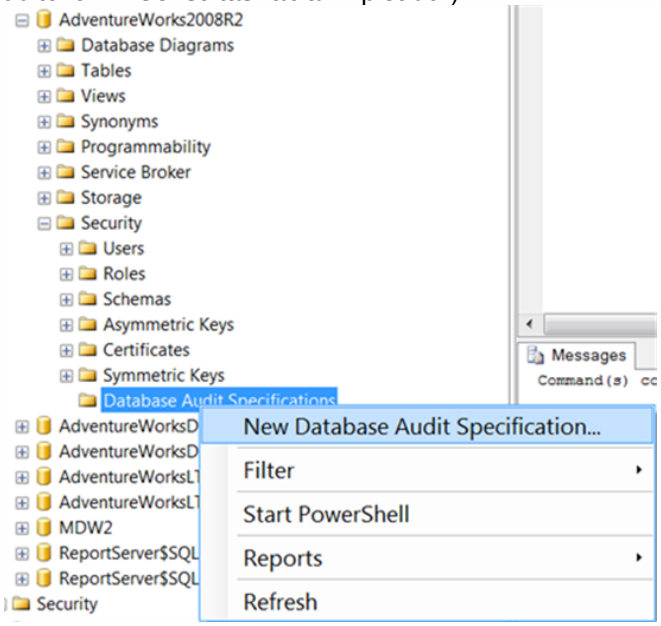
```
USE master
CREATE SERVER AUDIT SPECIFICATION [EspecificacionAuditoriaServidor-LoginsFallidos]
FOR SERVER AUDIT [AuditarLoginsFallidos]
ADD (FAILED_LOGIN_GROUP) WITH (STATE = ON)
GO
```

Especificación de auditoría de servidor

El objeto de auditoría de servidor Especificación pertenece a una auditoría. Puede crear una especificación de auditoría de servidor por la auditoría, ya que ambos se crean en el alcance de la instancia de SQL Server. La especificación de auditoría de servidor recoge servidor de muchos grupos de acción a nivel nacional planteadas por el extendido Eventos característica. Usted puede incluir a los grupos de acción de auditoría en una especificación de auditoría de servidor. Los grupos de acción de auditoría son grupos predefinidos de acciones, que son los eventos atómicos expuestos por el motor de base de datos. Estas acciones son enviados a la auditoría, que registra en el objetivo.
Servidor de auditoría de los grupos de nivel de acción se describe en el tema de SQL Server Grupos de auditoría de acciones y acciones.

Ejecutar y Crear una auditoría a nivel de base de datos

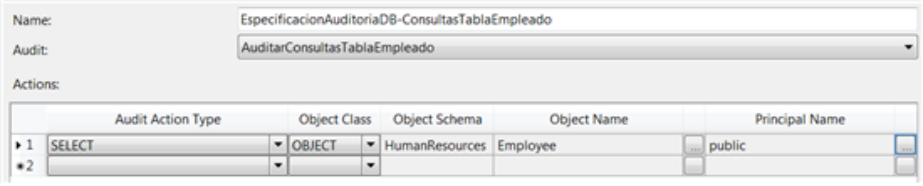
Para crear y habilitar una especificación de auditoría a nivel de base de datos debe expandir la base de datos, en este ejemplo utilice Padrones, y seleccione con el botón de la derecha del mouse la opción "Audit Specifications" bajo el árbol de seguridad de la base de datos. Seleccione la opción "New Database Audit Specification" y asigne un nombre (para propósito de este ejemplo asignaremos el nombre: "EspecificacionAuditoriaDB-ConsultasTablaEmpleado")



Como se muestra en la siguiente figura, seleccione la opción "AuditarConsultasTablaEmpleado" Bajo audit. Esto asignará la especificación de auditoría "EspecificacionAuditoriaDB-ConsultasTablaEmpleado" al objeto de auditoría "AuditarConsultasTablaEmpleado".

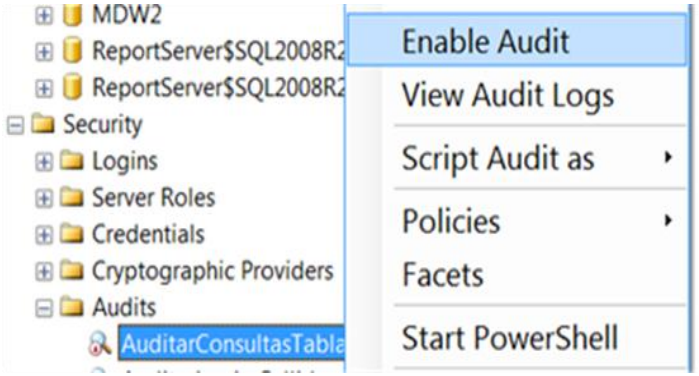
Nota: **La clase de objeto se utiliza para indicar lo que se quiere auditar. Las opciones son: Objeto, Base de datos o Esquema. Seleccione objeto para auditar Tablas, Funciones, Procedimientos Almacenados o Vistas. El nombre del Principal son entidades que pueden solicitar recursos de SQL Server.**

Para propósitos de este ejemplo, seleccione “SELECT” en el tipo de acción para auditar; en la clase del objeto seleccione “OBJECT”; en el nombre del objeto ingrese [HumanResources].[Employee]; en el nombre del Principal ingrese [public] y presione OK para crear y asignar el objeto de auditoría.
Esto permite rastrear las consultas con SELECT que realizan todos los usuarios a la tabla [HumanResources].[Employee].



Presione el botón de la derecha del mouse sobre “EspecificacionAuditoriaDB-ConsultasTablaEmpleado” para habilitar la especificación a través de la opción: “Enable Database Audit Specification”

Finalmente debe habilitar el objeto de auditoría presionando el botón de la derecha del mouse sobre “AuditarConsultasTablaEmpleado” para habilitar la auditoria a través de la opción Enable Audit como se muestra en la siguiente figura.

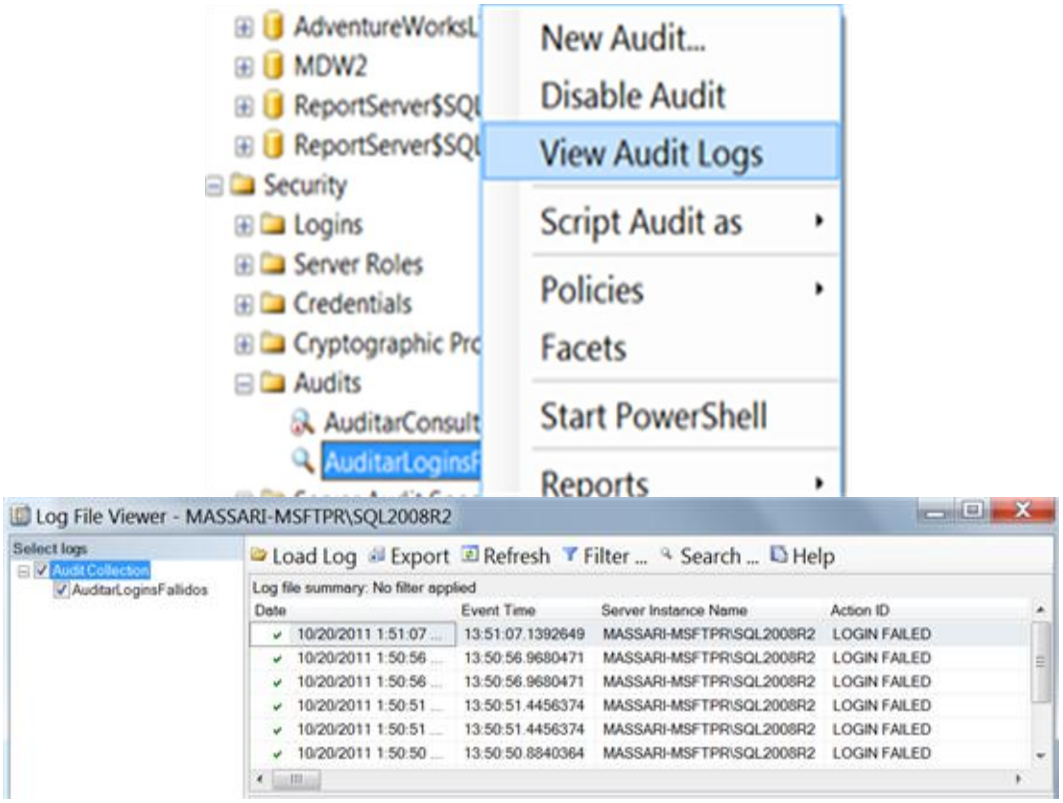


Si desea crear una especificación de auditoría a nivel de base de datos, a través de T-SQL, lo puede hacer utilizando el comando CREATE DATABASE AUDIT SPECITICATION. La siguiente consulta crea la especificación de auditoría “EspecificacionAuditoriaDB-ConsultasTablaEmpleado”. Esta operación fue realizada anteriormente a través de SSMS.

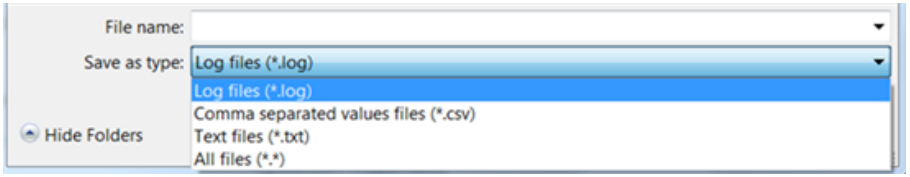
```
USE master
CREATE DATABASE AUDIT SPECIFICATION [DatabaseAuditSpec-EmployeesTable]
FOR SERVER AUDIT [Audit-EmployeeQueries]
ADD (SELECT ON OBJECT::[HumanResources].[Employee] BY [public])
WITH (STATE = ON)
GO
```

Visualizar los registros de auditoría

Los registros de auditoría pueden ser accedidos a través la opción “View Audit Logs” o a través de la bitácora de eventos de aplicaciones o de seguridad, dependiendo de dónde se especificó que se almacenaría la auditoría.



Los resultados pueden ser filtrados o inclusive hasta exportados en los siguientes formatos: log, csv y txt



La auditoría en SQL 2008 Enterprise es muy poderosa y flexible ya que permite crear auditorías a nivel de servidor o a nivel de base de datos. La configuración es sencilla ya que solo se requiere especificar donde se almacenará la auditoría (en un archivo, la bitácora de eventos de aplicaciones o la bitácora de eventos de seguridad), que objeto se desea auditar (Base de datos, Esquema, Tablas, Funciones, Procedimientos Almacenados o Vistas) y para cual Principal (entidades que pueden solicitar recursos de SQL Server).

Especificación de base de datos de auditoría

El objeto de auditoría de base de datos especificación también pertenece a una Auditoría de SQL Server. Puede crear una especificación de auditoría de base de datos por base de datos de SQL Server para la auditoría. La especificación de auditoría de base de datos de base de datos recoge las acciones de auditoría de alto nivel planteada por el extendido Eventos característica. Puede agregar cualquiera de los grupos de auditoría de la acción o los eventos de auditoría a una especificación de auditoría de base de datos. Los sucesos de auditoría son las acciones atómicas que pueden ser auditados por el motor de SQL Server. Los grupos de acción de auditoría son grupos predefinidos de acciones. Ambos se encuentran en el alcance de la base de datos de SQL Server. Estas acciones son enviados a la auditoría, que registra en el objetivo. Base de datos de auditoría de los grupos de nivel de acción y las acciones de auditoría se describe en el tema de SQL Server Grupos de auditoría de acciones y acciones

Los resultados de una auditoría son enviados a un objetivo, que puede ser un archivo, el registro de sucesos de seguridad de Windows, o el registro de sucesos de aplicación para Windows. (Escribir en el registro de seguridad no está disponible en Windows XP). Registros deben ser revisados y archivados periódicamente para asegurarse de que el destino tiene espacio suficiente para escribir registros adicionales. Escribiendo en el registro de seguridad de Windows requiere la cuenta de servicio de SQL Server que se añade a la Generar auditorías de seguridad. Por defecto, el sistema local, Servicio Local y Servicio de red forman parte de esta política. Esta configuración se puede configurar utilizando el complemento de la política de seguridad-en (secpol.msc). Además, el acceso a objetos de auditoría de políticas de seguridad debe estar habilitado para el éxito y el fracaso. Esta configuración se puede configurar utilizando el complemento de la política de seguridad-en (secpol.msc). En Windows Vista o Windows Server 2008, puede configurar la aplicación más granular de políticas generadas desde la línea de comandos mediante el programa de la política de auditoría (Auditpol.exe). Para obtener más información acerca de los pasos para habilitar la escritura en el registro de seguridad de Windows, vea Cómo: Escribir servidor de eventos de auditoría en el registro de seguridad. Para obtener más información sobre el programa de Auditpol.exe, véase el artículo de Knowledge Base 921469, Cómo utilizar Directiva de grupo para configurar la auditoría de seguridad detalladas. Los registros de sucesos de Windows

son globales para el sistema operativo Windows. Para obtener más información acerca de los registros de sucesos de Windows, consulte el Visor de sucesos general. Si usted necesita permisos más precisa sobre la auditoría, el uso del archivo de destino binario. Para obtener más información acerca de la auditoría de los registros por escrito a la meta, vea SQL Server Audit Records.

NOTA: Cualquier usuario autenticado puede leer y escribir en el registro de sucesos de la aplicación de Windows. El registro de sucesos de la aplicación necesita un nivel de permisos que el registro de sucesos de seguridad de Windows y es menos seguro que el registro de sucesos de seguridad de Windows.

Tipos de Datos FileStream SQL 2008

Tipo de Dato FileStream

Para almacenar datos no estructurados como ser mapas de bits (bitmaps), archivos de texto, archivos de audio, etc. Algunas de ellas han sido, por ejemplo guardar en la base de datos la ruta donde el archivo se alojaba en el sistema de archivos (file system). Claro que esto trae problemas de seguridad, o de pérdidas de referencias por cambios en la ruta, o la necesidad de coordinar la transacción entre los dos recursos con DTC.

El tipo de dato FILESTREAM nos trae solución a aquellos problemas, permitiéndole a las aplicaciones utilizar las API de streaming con el mismo rendimiento que el sistema de archivos, y a su vez mantener una consistencia transaccional el mismo esquema de seguridad entre la información estructurada y la no estructurada.

La seguridad para los objetos almacenados en un campo del tipo FILESTREAM goza de los mismos privilegios que cualquier tipo de dato ya existente. Solo es necesario establecer los permisos a la tabla o la columna que lo contenga. Es decir, si el usuario tiene permisos de acceso a la columna correspondiente, podrá abrir el archivo asociado a ella.

El tipo de dato FILESTREAM es implementado como una columna varbinary(max), y totalmente integrado en el motor. Lo cual permite utilizar todas las técnicas de copias de respaldo y restauración para resguardar los datos.

Antes que nada es necesario habilitar el uso de FILESTREAM en el motor:

```
EXEC sp_filestream_configure
    @enable_level = 3,
    @share_name = "RecursoCompartidoSQL";
```

Donde @enable_level = 3, habilita el uso para T_SQL, acceso a sistema de archivos local y remoto; y @share_name = "RecursoCompartidoSQL" es el nombre del recurso compartido en el sistema de archivos.

EJEMPLO

Crear una tabla con un campo FILESTEAM y su sintaxis para insertar datos:

Crea un tabla que contiene infromación adicional como FILESTREAM

```
CREATE TABLE dbo.Persona
(
      ID int,
      Nombre varchar(100),
      InfoAdicional varbinary(max) FILESTREAM
);
GO
--Agrego el registro en la tabla
Insert into dbo.Persona
Values(1,'Juanita Torres',Cast ('Aqui información adicional' As varbinary(max)))
Go
```

Es aconsejable utilizar el tipo de dato FILESTREAM cuando los objetos a almacenar son en promedio mayor a 1 Mb, y su acceso rápido de lectura es un punto a considerar. Para objetos de tamaño menor, es recomendable seguir utilizando el tipo de dato varbinary(max), el cual provee un rendimiento menor en estos casos.

Una pequeña mejora introduce SQL Server 2008 al lenguaje T-SQL respecto de la inicialización y asignación de variables. Cuántas veces hemos deseado escribir código de la siguiente manera:

```
declare
@fecha datetime = getdate(),
@edad int = 36,
@nombre varchar(100) = 'carlos';

-- Muestro los valores...
select @fecha, @edad, @nombre
```

En versiones anteriores hubiésemos obtenido el siguiente error de sintaxis:

Msg 139, Level 15, State 1, Line 0Cannot assign a default value to a local variable.
Ahora, en SQL Server 2008 es completamente válido.

Veamos esta sintaxis:

```
-- Incremento i, agrego algo al nombre....
  select @edad += 1, @nombre += ' walzer';
-- Muestro los valores...
select @edad, @nombre;
```

Estos operadores funcionan incluso con sentencias DML y columnas. Veamos la forma de incrementar en 100 la columna de una tabla:

```
update tabla set comlumna += 100;
```

o realizar la operación entre dos columnas:

```
update tabla set columna1 += columna2;
```

MERGE

Este comando se lo conoce informalmente como UPSERT ya que permite hacer inserciones o actualizaciones (UPDATE INSERT) de acuerdo a la existencia del registro. Este ejemplo insertará un registro en la tabla Destino si no tiene contraparte en la tabla Fuente, de existir se actualizará el correspondiente:

```
MERGE INTO Destino D
USING Fuente F ON F.Id = D.Id
WHEN MATCHED THEN UPDATE
        SET D.Cantidad = F.cantidad
WHEN NOT MATCHED THEN INSERT (Id, Cantidad)
VALUES (F.Id, F.Cantidad)
```

Esta instrucción es posible definir qué acción se debe llevar a cabo sobre los registros de una tabla cuando éstos se encuentran (*match*) o no sobre los registros de una tabla origen distinto.
En el siguiente ejemplo añadiremos paises a nuestra tabla a partir de una tabla origen (*person.CountryRegion* de la base de datos *AdventureWorks*), siempre y cuando no existan previamente. Si se produce este último caso, entonces unicamente actualizaremos su fecha de modificacion.

```
MERGE dbo.Paises AS target
USING
 (
  SELECT CountryRegionCode,
   Name
  FROM   Person.CountryRegion
 ) AS source (codigo,nombre)
ON  target.IDPais=source.codigo
WHEN MATCHED THEN UPDATE SET FechaModif=SYSDATETIME()
WHEN NOT MATCHED THEN INSERT VALUES (codigo,nombre,SYSDATETIME());
```

Es importante observar la sintaxis de la instrucciones *update* e *insert*, ya que en ninguna de ambas se especifica la tabla donde se realiza la operación (se supone que es la tabla de destino o *target*, que en nuestro caso es la tabla *dbo.Paises*).

Esta instrucción también se puede usar conjuntamente con la instrucción *output*, de forma que podamos capturar la información de los registros afectados por la operación.

```
MERGE dbo.Paises AS target
USING
 (
  SELECT CountryRegionCode,
   Name
  FROM   Person.CountryRegion
 ) AS source (codigo,nombre)
ON  target.IDPais=source.codigo
WHEN MATCHED THEN UPDATE SET FechaModif=SYSDATETIME()
WHEN NOT MATCHED THEN INSERT VALUES (codigo,nombre,SYSDATETIME())
OUTPUT $action, Inserted.IDPais, Inserted.Descripcion, Inserted.FechaModif,
Deleted.FechaModif;
```

En este caso, tenemos acceso a una columna especial llamada *$action*, que nos permite saber qué operación se ha realizado (*insert, update* o *delete*). En la tabla *inserted* accedermos a los registros añadidos o actualizados (valor nuevo) a la tabla destino, mientras que en la tabla *Deleted* accederemos a los registros eliminados o actualizados (valor antiguo).

Esta cláusula nos va a permitir definir lógica de combinación para operaciones atómicas de inserción, borradas y actualización de datos.
La idea es comparar 2 conjuntos de datos y detectar las diferencias de datos entre las 2 tablas y en función de eso, ejecutar alguna operación de actualización sobre la tabla.

La clausula MERGE nos sirve básicamente para 2 cosas:
Sincronizar los datos de 2 tablas. Supongamos que tenemos 2 bases distintas (Producción y Desarrollo por ejemplo) y queremos sincronizar los datos de una tabla para que queden exactamente iguales. Lo que antes hubiese implicado algunas sentencias mezcladas con INNER JOIN y NOT EXISTS, ahora es posible resumirlo en una operación atómica mucho más sencilla y eficiente.

La otra razón por la cual podríamos usar MERGE, es cuando tenemos nuevos datos que queremos almacenar en una tabla y no sabemos si la primary key de la tabla ya existe o no, por lo tanto, no sabemos si hacer un UPDATE o un INSERT en la tabla. El famoso IF EXISTS…

```
MERGE dbo.Tabla1 AS Target
USING (SELECT ID,Campo1,Campo2,Campo3 FROM dbo.Tabla2) AS Source
ON (Target.ID = Source.ID)
WHEN MATCHED THEN
UPDATE
SET Target.Campo1 = Source.Campo1, Target.Campo2 = Source.Campo2
WHEN NOT MATCHED BY TARGET THEN
INSERT (ID,Campo1,Campo2,Campo3)
VALUES (Source.ID,Source.Campo1,Source.Campo2, Source.Campo3)
WHEN NOT MATCHED BY SOURCE THEN
DELETE;
```

Esto significa que si una operación de delete/insert/update falla, se hace un rollback de todo el conjunto de datos y más importante aún, se garantiza en escenarios de alta concurrencia, donde puede haber otros procesos escribiendo en la misma tabla en el mismo instante, no existan incoherencias.

Capturar salida:
Algo fantástico que existe a partir de SQL Server 2005, es la clausula OUTPUT que permite capturar todo lo que sucedió dentro de una operación INSERT/DELETE/UPDATE. En SQL Server 2008 el uso conjunto de MERGE + OUTPUT nos sirve saber que registros fueron modificados y que acción se hizo sobre

ese registro (INSERT, UPDATE o DELETE).

La nueva función $action indica que operación se realizó, mientras que los atributos deleted e inserted guardan la información sobre el registro afectado (de la misma manera que funcionan con los triggers).

Ejemplo:

```
MERGE dbo.Tabla1 AS Target
USING (SELECT ID,Campo1,Campo2,Campo3 FROM dbo.Tabla2) AS Source
ON (Target.ID = Source.ID)
WHEN MATCHED THEN
UPDATE SET Target.Campo1 = Source.Campo1, Target.Campo2 = Source.Campo2
WHEN NOT MATCHED BY TARGET THEN
INSERT (ID,Campo1,Campo2,Campo3)
VALUES (Source.ID,Source.Campo1,Source.Campo2, Source.Campo3)
WHEN NOT MATCHED BY SOURCE THEN
DELETE
OUTPUT $action, deleted.*, inserted.*;
```

Resultado:

	$action	ID	Campo1	Campo2	Campo3	ID	Campo1	Campo2	Campo3
1	UPDATE	0	1	1	0	0	1	1	1
2	DELETE	1	2	2	2	NULL	NULL	NULL	NULL
3	INSERT	NULL	NULL	NULL	NULL	2	2	2	2
4	DELETE	3	3	3	3	NULL	NULL	NULL	NULL
5	INSERT	NULL	NULL	NULL	NULL	4	4	4	4
6	DELETE	5	5	5	5	NULL	NULL	NULL	NULL
7	INSERT	NULL	NULL	NULL	NULL	6	6	6	9
8	DELETE	8	8	8	8	NULL	NULL	NULL	NULL
9	DELETE	9	9	9	9	NULL	NULL	NULL	NULL
10	INSERT	NULL	NULL	NULL	NULL	10	10	10	10
11	INSERT	NULL	NULL	NULL	NULL	12	12	12	12
12	UPDATE	13	13	13	13	13	13	13	13

DELETED INSERTED

En primer lugar, aparece un nuevo operador lógico en el plan, que se llama *"Clustered Index Merge"* y es nuevo de SQL Server 2008. Este operador en función de un conjunto de datos, realiza un delete, insert o update a una tabla usando su respectivo índice clustered. Si la tabla no tuviese un índice clustered, entonces se usaría el también nuevo operador *"Table Merge"*. Esta es una de las razones por la cual MERGE no es una simple encapsulación que simplifica cosas que ya podíamos hacer antes, sino que al ser una operación interna del motor de SQL Server, es más eficiente que cualquier otra alternativa existente en las versiones previas de SQL Server.

Otro punto interesante es que podemos ver es que cuando existen índices y las 2 tablas tienen similares volúmenes de datos, el operador MERGE JOIN resulta el método más eficiente en estas operaciones, ya ambas tablas son escaneadas una sola vez y no hay necesidad de ordenar los datos. Es posible cambiar estableciendo otro join hint, pero en la mayoría de los casos, el MERGE JOIN es lo más óptimo.

Algunas recomendaciones para obtener mejor performance con esta sentencia:

- Crear un índice clustered sobre las columnas relacionadas en el JOIN, en la tabla destino.
- Crear un índice único sobre las columnas relacionadas en el JOIN, en la tabla de origen (en caso que hubiese).

Esto garantiza al motor que no tiene que ejecutar ninguna validación adicional ni efectuar ningún sort extra.

Servicios de Entidades de Datos

La Inclusión de los Servicios de Entidades de Datos (Entity Data Services) permiten ahora , con SQL Server 2008 y ADO.NET, poder crear objetos de negocio de alto nivel, por ejemplo Clientes o Facturas. Estas entidades se pueden utilizar y reemplazan el método estándar de devolver filas y tablas. Si estas utilizando un modelado basado en relación entre entidades, los objetos en SQL Server ahora acompañan tu modelo. Hay nuevas herramientas de ADO.NET que pueden tener acceso a estas entidades como ser el Lenguaje de Consultas Integrado LINQ to SQL.

Programar en un alto nivel de abstracción es altamente productivo y nos permite sacar buen provecho del modelo de entidad-relación. ADO.NET nos permite entonces programar y percibir a la información relacional como entidades de negocio.

LINQ

El Lenguaje de Consultas Integrado LINQ, nos permite realizar consultas a una fuente de datos utilizando lenguajes de programación manejados tales como Visual Basic.NET o C#, en lugar de usar sentencias SQL. Utilizaremos una sintaxis consistente para obtener datos de diversas fuentes de datos, incluyendo datos relacionales, entidades, XML, DataSets de ADO.NET, y colecciones de objetos en memoria. Esta nueva sintaxis embebida en un lenguaje de propósito general adquiere todos sus beneficios: validación de tipos, revisión de errores al momento de la codificación, etc.

Ejemplo de ordenamiento de un vector por el tamaño de sus ítems:

```
public void OrdernarPalabras()
{
   string[] palabras = { "naranja", "manzana", "uva" };
   var palabrasOrdenadas =
      from p in palabras
      orderby p.Length
      select p;

      Console.WriteLine("La lista ordenada por tamaño:");
      foreach (var p in palabrasOrdenadas) {
            Console.WriteLine(p);
      }
}
```

Ejemplo de agrupamiento de palabras por sus iniciales:

```
public void AgrupamientoPalabras()
{
   string[] palabras = { "naranja", "manzana", "níspero", "melón", "mandarina", "sandía" };
   var palabrasAgrupadas =
      from p in palabras
      group p by p[0] into g
      select new { FirstLetter = g.Key, Palabras = g };

    foreach (var g in palabrasAgrupadas) {
      Console.WriteLine("Palabras que empiezan con la letra '{0}':", g.FirstLetter);
      foreach (var p in g.Palabras) {
         Console.WriteLine(p);
      }
   }
}
```

LINQ to SQL

LINQ to SQL es la implementación de un mapeador de objetos a modelo relacional ORM (object relational mapping), que está integrado al .NET Framework 3.5. Visual Studio 2008 provee un diseñador gráfico que permite modelar y visualizar una base de datos como un modelo de objetos LINQ to SQL.

Cuando grabemos un diseño realizado en esta herramienta, Visual Studio persistirá las clases que representas las entidades de negocio y la base de datos relacional que hayamos modelado. Además veremos generado una clase que representa el contexto (DataContext). Esta clase es el nexo entre los objetos entidades del negocio y la base de datos, defiendo propiedades para las tablas de la base de datos y métodos para los procedimientos almacenados.

Veamos algunos ejemplos de consumo de datos usando LINQ:

```
Dim db As New NorthwindDataContext
Dim productos  = From p in db.Products _
   Where p.Category.CategoryName = "Beverages" _
   Select p
```

Este código muestra cómo recuperar un producto de la base de datos, modificarlo y grabarlo.

```
Dim db As New NorthwindDataContext

Dim producto = (From p in db.Products _
            Where p.ProductName = "Toy 1" _
            Select p).Single

producto.UnitsOnOrder = 23
producto.UnitsInStock = 20
```

db.SubmitChanges()

Tamaño de los Tipos de Datos Definidos por el Usuario basados en CLR

Se ha aumentado la restricción de límite de 8000 bytes a 2 Gb para los tipos de datos definidos por el usuario y agregados basados en CLR (CLR UDT y CLR UDA).

UTLIZAR DATABASE MIRROING SQL 2008

Database Mirroring es una solución de Alta Disponibilidad en SQL Server, disponible desde SQL Server 2005 y sensiblemente mejorada en SQL Server 2008, mostrándose como una alternativa a los sistemas de Alta Disponibilidad basados en Microsoft Cluster y/o Replicación de Almacenamiento Datos, siendo también una alternativa interesante a otras tecnologías como Log Shipping o a la Replicación de SQL Server.

Database Mirroring, al igual que Log Shipping, sólo protege a nivel de base de datos (es decir, sólo las bases de datos de usuario) y no a nivel de Instancia, para lo cual sería necesario implementar Server Clustering (y así proteger también las bases de datos del sistema y demás elementos que forman una instancia de SQL Server).

Database Mirroring es una tecnología de Alta Disponibilidad basada en un modo de funcionamiento Activo / Pasivo. Es decir, mientras una Instancia realiza un papel de Servidor Principal (Activo) para una base de datos en particular, la otra instancia realiza el papel de Servidor Espejo o Secundario (Pasivo) para dicha base de datos. En consecuencia, no será posible el acceso a la copia de la base de datos del Servidor Espejo.

Database Mirroring requiere que la base de datos que se desee proteger, esté configurada con el Modo de Recuperación Completo (Full Recover Model), algo bastante evidente, al tratarse de una tecnología que basa su funcionamiento en el envío de transacciones de una base de datos principal a una base de datos espejo o secundaria.

Es posible montar Database Mirroring sobre SQL Server 2005 y SQL Server 2008. El hecho de poder montar el Servidor Principal sobre SQL Server 2005 y el Servidor Espejo sobre SQL Server 2008, permite plantearse soluciones de Database Mirroring interesantes para Migraciones de SQL Server 2005 a SQL Server 2008 con a penas corte de servicio y luego romper el Database Mirroring, si no lo queremos mantener.

Es importante tener en cuenta que NO es posible hacer funcionar Database Mirroring con el Servidor Principal en SQL Server 2008 y el Servidor Espejo en SQL Server 2005

Los posibles papeles o roles que puede desempeñar una instancia de SQL Server en una solución de Database Mirroring son:

• Servidor Principal. Mantiene la copia activa de la base de datos (base de datos principal), a través de la cual, se ofrece el servicio a los usuarios. Todas las transacciones son enviadas al Servidor Espejo antes de aplicarlas en la base de datos principal.

• Servidor Espejo (Mirror). Mantiene una copia de la base de datos principal (base de datos espejo o mirror database), y aplica todas las transacciones enviadas por el Servidor Principal, manteniendo sincronizada la base de datos espejo.

• Servidor Testigo (Witness). Se trata de un elemento opcional. No es obligatorio o necesario implementar un Servidor Testigo (Witness) en una solución de Database Mirroring. Sin embargo, si deseamos que nuestra solución de Database Mirroring ofrezca recuperación automática ante fallos (automatic failover), entonces sí será necesario implementar un Servidor Testigo (Witness Server), pues éste es quién monitorizará los Servidores Principal y Espejo partícipes de una Sesión de Espejo (Mirror Session) con el objetivo de asignar el papel de Principal al servidor Espejo en caso de una caída de servicio o pérdida del primero (es decir, en caso de caída del Servidor Principal, se asignará el papel de Principal al Servidor Espejo, manteniéndose así el servicio). El trabajo realizado por el Servidor Testigo (Witness) no es muy intenso, por lo cual, no requiere de grandes recursos, y además, un mismo servidor puede actuar como Servidor Testigo (Witness) para múltiples sesiones de espejo, sin pérdida de rendimiento.

Database Mirroring ofrece tres modos de funcionamiento, como antes adelantamos:

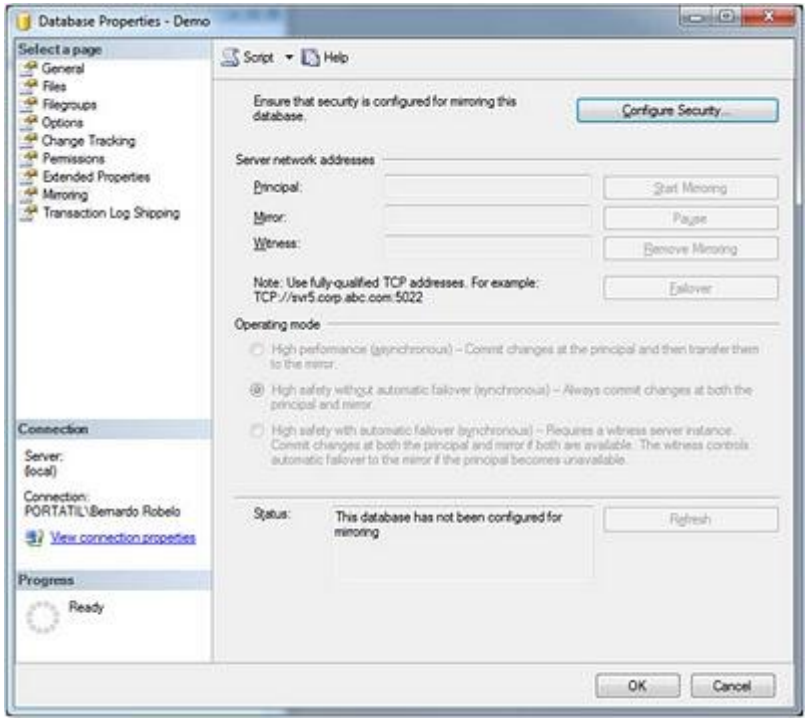
• Modo de Alta Disponibilidad (síncrono y con testigo). Las transacciones son aplicadas de forma síncrona a las base de datos principal y espejo. Requiere de un Servidor Testigo (Witness) ubicado sobre una tercera máquina (que no sea ni el Servidor Principal ni el Servidor Espejo), gracias al cual es posible la recuperación automática ante fallos (automatic failover) o conmutación automática de roles. En caso de fallo del Servidor Principal durante el envío de transacciones, el Servidor Espejo tiene que terminar las transacciones encoladas antes de poder levantarse como Servidor Principal. Por supuesto también es posible la recuperación manual ante fallos (manual failover) o conmutación manual de roles. En caso de una caída o pérdida del Servidor Espejo, la base de datos principal se mantendrá activa.

• Modo de Alta Protección (síncrono y sin testigo). Las transacciones son aplicadas de forma síncrona a las base de datos principal y espejo. Sin embargo, no utiliza un Servidor Testigo (Witness). En este modo de funcionamiento, no es posible la existencia de pérdida de datos, pero la recuperación ante fallos se realiza de forma manual (manual failover). En caso de una caída o pérdida del Servidor Espejo, la base de datos principal dejará de estar activa, al haber perdido el Quorum.

• Modo de Alto Rendimiento (asíncrono y sin testigo). Las transacciones son aplicadas de forma asíncrona a la base de datos espejo, ofreciendo mejor rendimiento que los anteriores modos de funcionamiento, pero pagando como precio la existencia de posibles pérdidas de transacciones (y en consecuencia, potenciales pérdidas de datos). Evidentemente, la recuperación ante fallos se realiza de forma manual (manual failover), hablando de conmutación forzada (es decir, cambio de roles sin comprobación de datos escritos en el servidor espejo). En caso de una caída o pérdida del Servidor Espejo, el Servidor Principal no se verá afectado.

1. Primeramente preparamos nuestra base de datos espejo en nuestro server o instancia que fungirá como tal, aquí dos puntos importantes: Que la base datos que restauremos sea el ultimo backup realizado desde la principal. A la hora de restaurarla tenemos que marcar la opción de NON RECOVERY.

2. En el Management Studio, Explorador de Objetos, Seleccionamos una base de datos, hacemos click derecho sobre ella en la opción, Task, Mirror.



3. El primer paso sería configurar la seguridad, para lo cual vamos a seguir un asistente.

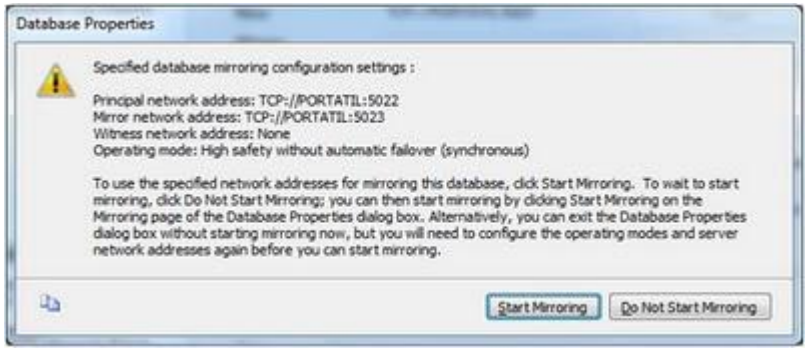
4. El asistente nos preguntara si queremos tener una instancia de testigo, para este primer ejempo le diremos que No.

5. Luego definiremos el servidor principal

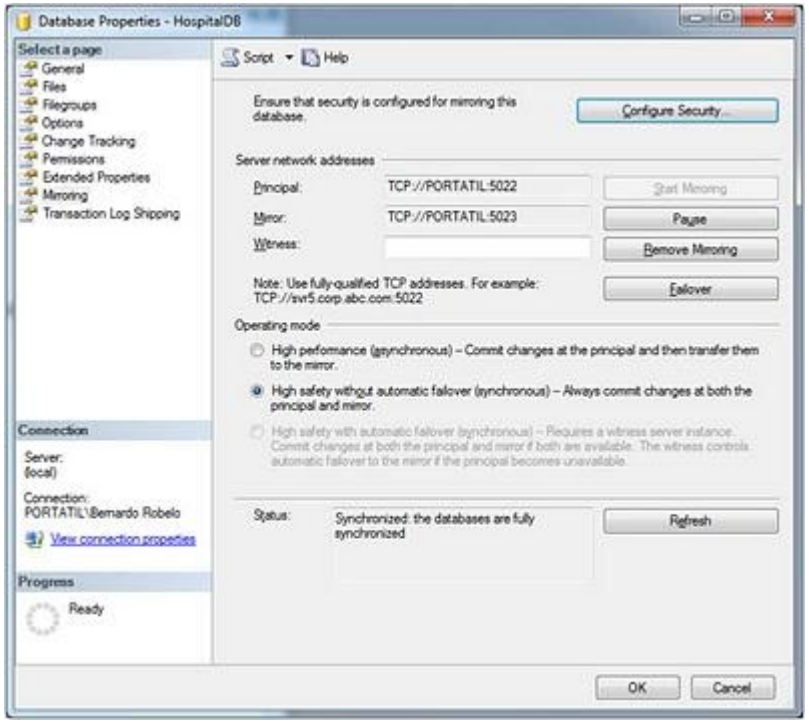
5. Ahora definiremos nuestra instancia o servidor espejo

6. En este paso se definen las cuentas de usuario que utilizaran tanto el servidor principal como el espejo que estén en un dominio.

7. Finalmente terminanos de configurar el asistente de seguridad.
8. Una vez finalizado nos pedirá si deseamos iniciar el mirroring, le diremos iniciar.



Ya tendremos configurado nuestro mirroring como se muestra en la pantalla siguiente, desde aquí podemos iniciar el mirroring, y podemos configurar el tipo de operación que deseamos, tal y cual se planteo al inicio del articulo. Hacemos click en OK.



La Redirección Automática del cliente en una infraestructura de Database Mirroring, es una funcionalidad muy apreciada, y en este caso, es tan fácil como utilizar una sintaxis determinada en la cadena de conexión a SQL Server, como se muestra en el siguiente:

"Data Source=PORTATIL;Failover Partner=PORTATIL\MIRROR;Initial Catalog=Demo;Integrated Security=True;"

Database Mirroring también está disponible para conexiones ODBC; recuerda que para que la aplicación cliente sepa a qué servidor intentar "re-conectar", en algún sitio en su configuración debe conocer los servidores miembros.

Puedes cambiar la cadena de conexión de tu aplicación usando la siguiente sintaxis:

- Server=instancia_servidor_principal;
- Failover Partner=instancia_servidor_mirror;

Cambiar la cadena de conexión, es posible que no requiera recompilar la aplicación si lo gestionas a través de ficheros INI, pero en las "viejas" aplicaciones VB6, no se utilizaba esta técnica con demasiada frecuencia por lo que con bastante probabilidad requeriría recompilación de la aplicación.

Como debes saber, las APIs de acceso a datos también implementan Mirroring para ODBC, por lo que si tienes DSNs de usuario también es muy fácil de cambiar; tan sólo tendrás que cambiar el Driver al nuevo Driver SQL Native Cliente (fíjate en esta primera pantalla el elemento seleccionado):

Y luego, en la configuración de la base de datos a la que deseas conectar, tienes la posibilidad de establecer cual es servidor mirror (en el caso de ejemplo sería DELL-ERH\INSTANCE2):

Otras consideraciones interesantes que comenté en la presentación de ayer de nuestro SQL Summit:

- Recuerda que cuando tu aplicación conecta al servidor principal, el servidor principal envía a la aplicación cliente quien es el servidor mirror: en este caso la información que tienes en la cadena de conexión, las APIs de acceso a datos no la necesitan porque tiene de primera mano (el servidor principal) quien es el servidor mirror.
- La información que tienes en la cadena de conexión del servidor mirror, solamente será utilizada cuando el primer intento de conexión no se puede satisfacer contra el servidor principal. SOLO en este caso, tirará de la información que tiene de la cadena de conexión.

En SQL Server 2008, (según comentan los whitepapers iniciales), no será necesario codificar quien el servidor mirror porque "milagrosamente" la aplicación cliente tendrá cacheada esa información. Digo "milagrosamente", porque no conozco todavía el mecanismo que utilizarán para cachear esa información. Recuerdo que en las primeras

fases betas de SQL Server 2005, implementaban un mecanismo similar, pero con el paso del tiempo la implementación se quitó del producto por razones que desconozco.

SERVICE BROKER

SQL Server 2008 R2 introduce la opción de activar y desactivar el control de mensajes dudosos en una cola. Una cola donde se haya establecido desactivar los mensajes dudosos no se deshabilitará tras las reversiones consecutivas de transacciones consecutivas. Con esta característica, si lo desea, una aplicación puede definir una estrategia personalizada de control de mensajes dudosos.

El SQL Service Service Broker incluye la infraestructura necesaria para la programación asincrónica y se puede utilizar para la creación de aplicaciones distribuidas a través de múltiples bases de datos.

Las instrucciones Transact-SQLCREATE QUEUE y ALTER QUEUE se han actualizado y la propiedad [IsPoisonMessageHandlingEnabled] se ha agregado a la API [Microsoft.SqlServer.Management.Smo.Broker.ServiceQueue

Una aplicación utiliza Service Broker ejecutando las instrucciones Transact-SQL que funcionan en objetos de Service Broker definidos en una base de datos. Esta sección describe las consideraciones generales a la hora de crear los objetos de Service Broker para una aplicación.

El Microsoft SQL Server 2008 R2 Service Broker activador externo ha sido desarrollado para ser una extensión de la función de activación interna de SQL Server 2008 R2 que permite mover la lógica de la recepción y procesamiento de mensajes de Service Broker en el servicio de motor de base de datos a un ejecutable de la aplicación que se ejecuta fuera de SQL Server. De esta manera, las tareas de uso intensivo de CPU o de larga duración, pueden ser descargados de SQL Server para un ejecutable de la aplicación, posiblemente en otro equipo. El ejecutable de la aplicación también se puede ejecutar bajo una cuenta de Windows diferente del proceso del motor de base de datos. Esto proporciona a los administradores mayor control sobre los recursos que la aplicación puede acceder. Ejecute el paquete de descarga autoextraíble para crear una carpeta de instalación. Lea la documentación adjunta, o los libros en pantalla para obtener más información. El programa de instalación único instalará el servicio en x86, x64 e IA64. Lea la documentación para más información.

A continuacion se detalla la sintaxis basica para utilizar el SQL Server Service Broker.

1. HABILITAR LA BASE DE DATOS
ALTER DATABASE Demo SET ENABLE_BROKER

2. MESSAGE TYPE
Define el nombre del mensaje y el tipo de informacion que el mensaje contiene. Estos mensajes deben ser creados en ambas partes de la conversacion

CREATE MESSAGE TYPE message_type_name
[AUTHORIZATION owner_name]
[VALIDATION =
{NONE EMPTY WELL_FORMED_XML
VALID_XML WITH SCHEMA COLLECTION schema_collection_name}]

Los posibles valores para VALIDATION son:
NONE: no se realiza ninguna validación
EMPTY: El cuerpo del mensaje debe tener valor NULO
WELL_FORMED_XML: Debe contener un XML bien formado
VALID_XML WITH SCHEMA COLLECTION: El contenido del XML debe cumplir con el XML SCHEMA señalado

Para modificarlo:
ALTER MESSAGE TYPE message_type_name
VALIDATION =
{NONE EMPTY WELL_FORMED_XML
VALID_XML WITH SCHEMA COLLECTION schema_collection_name}]

Y para eliminar:
DROP MESSAGE TYPE message_type_name

3. CONTRACT
Define los tipos de mensajes que un servicio puede utilizar en una conversación y la dirección en que los mensajes pueden ser enviados

CREATE CONTRACT contract_name

[AUTHORIZATION owner_name]
({ message_type_name SENT BY { INITIATOR TARGET ANY }
[DEFAULT] } [,...n])

Los posibles valores para SENT BY son:
INITIATOR: indica que solo el iniciador puede enviar dicho tipo de mensaje
TARGET: Indica que solo el destino puede enviar dicho tipo de mensaje
ANY: Indica que tanto el iniciador y el destino pueden enviar dicho tipo de mensaje

Para eliminar un contrato la sintaxis es la siguiente:

DROP CONTRACT contract_name

4. QUEUE
Define la ubicación donde se almacenaran los mensajes hasta que un servicio este disponible para atender los mensajes

CREATE QUEUE [database_name.[schema_name].schema_name.] queue_name
[WITH
[STATUS = { ON OFF } [,]]
[RETENTION = { ON OFF } [,]]
[ACTIVATION (
[STATUS = { ON OFF } ,]
PROCEDURE_NAME = stored_procedure_name,
MAX_QUEUE_READERS = max_readers ,
EXECUTE AS { SELF 'user_name' OWNER }
)]
]
[ON { filegroup [DEFAULT] }]

STATUS: Especifica si la cola esta habilitada. Cuando esta en OFF ningún servicio podría retirar mensajes de la cola.

RETENTION:Indica si la cola debe mantener todos los mensajes hasta que la conversación finalice

ACTIVATION STATUS: Indica si se debe activar el procedimiento almacenado cuando llegue un mensaje a la cola

MAX_QUEUE_READERS: Indica la cantidad máxima de servicio que correrán simultáneamente

EXECUTE AS: especifica la cuenta de usuario con que correrá el servicio

Para modificar esta es la sintaxis

ALTER QUEUE [database_name.[schema_name].schema_name.] queue_name
[WITH
[STATUS = { ON OFF } [,]]
[RETENTION = { ON OFF } [,]]
[ACTIVATION (
[STATUS = { ON OFF } ,]
PROCEDURE_NAME = stored_procedure_name,
MAX_QUEUE_READERS = max_readers ,
EXECUTE AS { SELF 'user_name' OWNER }
] DROP })
]

Para eliminar una cola la sintaxis es:

DROP QUEUE [database_name.[schema_name].schema_name.] queue_name

5. SERVICE
Relaciona las colas con los contractos

CREATE SERVICE service_name
[AUTHORIZATION owner_name]
ON QUEUE [schema_name.]queue_name
[(contract_name [DEFAULT] [,...n])]

Para modificar el servicio utilice la siguiente sintaxis:

ALTER SERVICE service_name
[ON QUEUE [schema_name].queue_name]
[(ADD CONTRACT contract_nameDROP CONTRACT contract_name)]

ON QUEUE: Especifica la nueva cola para el servicio y mueve todos los mensajes de la

cola vieja a la nueva

ADD CONTRACT: Añade un contrato a la colección de contratos asociados a este servicio

DROP CONTRACT: Especifica los contratos que se eliminaran del servicio. En caso de que alguno se este ejecutando mostrara un mensaje de error.

Para comenzar a usar los servicios se deben establecer una conversación. Estos son los pasos para establecer una conversación.

Crear la variable que identificara de manera única la conversación.

DECLARE @dialog_handle uniqueidentifier

Iniciar la conversación

BEGIN DIALOG [CONVERSATION] @dialog_handle
FROM SERVICE initiator_service_name
TO SERVICE 'target_service_name'
[, { 'service_broker_guid' 'CURRENT DATABASE' }]
[ON CONTRACT contract_name]
[WITH
[{ RELATED_CONVERSATION = related_conversation_handle
RELATED_CONVERSATION_GROUP = related_conversation_group_id }]
[[,] LIFETIME = dialog_lifetime]
[[,] ENCRYPTION = { ON OFF }]]

RELATED_CONVERSATION o RELATED_CONVERSATION_GROUP relaciona un nuevo dialogo con una conversacion existente

LIFETIME= tiempo en segundo en la que será valido el dialogo

Enviar mensaje

SEND
ON CONVERSATION conversation_handle
[MESSAGE TYPE message_type_name]
[(message_body_expression)]

El destino recibe el mensaje

[WAITFOR ( ]
RECEIVE [TOP (n)]
[,...n]
FROM
[INTO table_variable]
[WHERE { conversation_handle = conversation_handle
conversation_group_id = conversation_group_id }]
[ )] [, TIMEOUT timeout]

WAITFOR: especifica que la clausula RECEIVE espera un mensaje

RECEIVE: lee los mensajes de la cola y los elimina en caso de que la opción RETENTION de la cola este desactivada.

TOP: Indica cuantos mensajes se van a leer de la cola, sino se especifica se leerán todos los mensajes

INTO: Ingresa todos los mensajes en una tabla para ser tratados después

WHERE: especifica la conversación o grupo de conversaciones para los mensajes leídos

TIMEOUT: Especifica el tiempo en milisegundos en que la instrucción espera un mensaje.

Para terminar la conversación

END CONVERSATION conversation_handle
[[WITH ERROR = failure_code DESCRIPTION = 'failure_text']
[WITH CLEANUP]

EJEMPLO

Diseñar y crear una base de datos, empleando SERVICE BROKER, y luego habilitar el servicio

```
CREATE DATABASE ServiceBrokerTest
GO
USE ServiceBrokerTest
GO
-- Habilitar Service Broker
ALTER DATABASE ServiceBrokerTest SET ENABLE_BROKER
GO
```

Son 4 Elementos Basicos

- Mensaje
- Contrato
- Cola
-Cola de Envio
-Cola de Recepcion
- Servicio
-Servicio de Envio
-Servicio de Recepcion

```
-- Crear Mensaje
CREATE MESSAGE TYPE SBMessage
VALIDATION = NONE
GO
-- Crear Contrato
CREATE CONTRACT SBContract
(SBMessage SENT BY INITIATOR)
GO
-- Crear Cola de Envio
CREATE QUEUE SBSendQueue
GO
-- Crear Cola de Recepcion
CREATE QUEUE SBReceiveQueue
GO

-- Crear Servicio de Envio para Cola de Envio
CREATE SERVICE SBSendService
ON QUEUE SBSendQueue (SBContract)
GO
-- Creando Servicio de Recepcion para Cola de Recepcion
CREATE SERVICE SBReceiveService
ON QUEUE SBReceiveQueue (SBContract)
GO

-- Iniciar Dialogo usando el servicio en el contrato
DECLARE @SBDialog uniqueidentifier
DECLARE @Message nvarchar(128)
BEGIN DIALOG CONVERSATION @SBDialog
FROM SERVICE SBSendService
TO SERVICE 'SBReceiveService'
ON CONTRACT SBContract
WITH ENCRYPTION = OFF
-- Enviando Mensajes al Dialogo
SET @Message = N'Primer Mensaje';
SEND ON CONVERSATION @SBDialog
MESSAGE TYPE SBMessage (@Message)
SET @Message = N'Segundo Mensaje';
SEND ON CONVERSATION @SBDialog
MESSAGE TYPE SBMessage (@Message)
SET @Message = N'Tercer Mensaje';
SEND ON CONVERSATION @SBDialog
MESSAGE TYPE SBMessage (@Message)
GO

-- Viendo Mensajes de la cola de Recepcion
SELECT CONVERT(NVARCHAR(max), message_body) AS Message
FROM SBReceiveQueue
GO

-- Recibiendo Mensajes de la Cola de Recepcion
RECEIVE TOP(1) CONVERT(NVARCHAR(max), message_body) AS Message
FROM SBReceiveQueue
GO

-- Recibiendo Mensajes de la Cola de Recepcion
```

```
RECEIVE CONVERT(NVARCHAR(max), message_body) AS Message
FROM SBReceiveQueue
GO
```

AVANZADO IDENTITY EN SQL SERVER 2008

1) Modificar una columna para que sea identity:

No se puede. SQL no permite modificar la propiedad Identity de una columna, ya sea para habilitar o deshabilitar esa propiedad. No existe algo asi como un ALTER COLUMN Nombre_Campo SET IDENTITY ON (Como dato curioso, Sybase si tiene esta posibilidad). Pero muchos diran, ¿Como no puede ser posible, si yo lo hago desde el enterprise manager?. Bueno, lo que hace ese programa cuando se habilita (o deshabilita) la propiedad identity a una columna existente, es reconstruir toda la tabla. Esto significa borrar todas las dependencias de una tabla, crear una tabla auxiliar similar a la tabla original pero con la propiedad identity cambiada, copiar todo el contenido de la tabla original a la auxiliar, borrar la tabla original, renombrar la tabla auxiliar por el nombre que tenia la tabla original y reconstruir todas las dependencias. En una tabla vacia, el costo de hacer esto es casi nulo, en una tabla con 20 millones de registros y 30 tablas asociadas a esta via llaves foraneas es casi suicida. Por tal motivo, y con muy buen criterio, a partir de SQL Server 2008, el Enterprise Manager deshabilita por defecto todas las operaciones sobre edicion de tablas y columnas, que signifiquen una reconstruccion de la tabla. Esto sirve para evitar que algun despistado sin conocimiento sobre lo que hace, reconstruya una tabla en un servidor de produccion. De todas maneras, si necesitamos hacer esto independientemente del costo que nos representa, este este es un script de ejemplo:

```
/*Paso 1: Borro todos los objetos dependedientes de la tabla a modificar*/
ALTER TABLE TablaAsociada DROP CONSTRAINT TablaAsociada_Original_FK

/*Paso 2: Creo una tabla nueva exactamente similar a la original, pero agregando la propiedad Identity*/
CREATE TABLE TablaOriginalAuxiliar
(
ID int IDENTITY(1,1),
Campo1 varchar(200)
)

/*Paso 3: Habilitamos el insertado explicito de valores en la columna de tipo identity para mantener los valores antiguos del campo ID*/

SET IDENTITY_INSERT TablaOriginalAuxiliar ON

/*Paso 4: Copio el contenido de una tabla a la otra*/
INSERT INTO TablaOriginalAuxiliar (ID, Campo1) SELECT ID, Campo1 FROM TablaOriginal

/*Paso5: Deshabilitamos el insertado explicito de valores en la columna de tipo identity*/
SET IDENTITY_INSERT TablaOriginalAuxiliar OFF

/*Paso 6: Borro la tabla original*/
DROP TABLE TablaOriginal
/*Paso 7: Renombro la tabla auxiliar por la tabla original*/
sp_rename 'TablaOriginalAuxiliar','TablaOriginal'
/*Paso 8: Creo todos los objetos dependedientes de la tabla a modificada que habia borrado en el paso 1*/

ALTER TABLE TablaAsociada ADD CONSTRAINT TablaAsociada_Original_FK
```

Todo esto tambien aplica en el caso inverso, cuando tenemos una columna identity y queremos deshabilitarla.

2) Determinar el estado del IDENTITY_INSERT en una tabla:

En algunas ocasiones (no muchas) puede ser necesario verificar si una tabla que tiene una columna identity, tiene en estado ON o OFF la opcion IDENTITY_INSERT. Nuevamente, no es posible de manera directa hacer esto. No hay ninguna vista de sistema que nos indique el estado de la tabla con respecto a esta situación.
Sin embargo, existen métodos alternativos para hacer esto.
El método mas común es insertar un registro en tabla que queremos seteandole un valor explicito que sabemos que no existe en la columna (ejemplo: un 0, que no suele ser un valor habitual para un ID) en la columna identity. Si tira error, es porque la opción

esta deshabilitada. Toda esta operación debe estar encapsulada dentro de una transacción y al finalizar, deberemos hacer un rollback.

Ejemplo:

```
BEGIN TRANSACTION
DECLARE @err int
INSERT INTO tabla1 (id) values (0)
SET @err=@@error
ROLLBACK TRANSACTION
IF @err =0
print 'identity_insert = on'
else
print 'identity_insert = off'
```

3) Consultar si una columna es identity

Para esto, debemos ver las vistas de sistema de SQL Server.

La forma mas práctica, es consultar la vista sys.column y verificar el valor del campo is_identity.

Ejemplo:

```
select name, is_identity from sys.columns where OBJECTNAME(object_id) =
'NombreDeLaTabla' and name = 'NombreColumna'
```

Existe tambien una vista llamada sys.identity_columns que nos devuelve todas las columnas identity de todas las tablas de la base de datos.

Recordemos que estos ejemplos solo aplican a SQL Server 2005/2008 en adelante. No funciona en SQL Server 2000.

4) Modificar la propiedad NOT FOR REPLICATION de una columna identity sin reconstruir la tabla:

Por alguna razón desconocida, si deseamos modificar la propiedad NOT FOR REPLICATION de una columna identity desde el enterprise manager de SQL Server, este reconstruye la tabla (hace exactamente lo mismo que vimos en el punto 1 al principio del post).

Sin embargo, existe una manera de hacer esto muchísimo mas eficiente y es llamar al store procedure de sistema sys.sp_identitycolumnforreplication.

Internamente este store procedure llama a un proceso interno de SQL Server que modifica la tabla, sin necesidad de reconstruirla.

Ejemplo para habilitar la propiedad NOT FOR REPLICATION:

```
EXEC sys.sp_identitycolumnforreplication OBJECT_ID("NombreDeLaTabla"), 1
```

Ejemplo para deshabilitar la propiedad NOT FOR REPLICATION:

```
EXEC sys.sp_identitycolumnforreplication OBJECT_ID("NombreDeLaTabla"), 0
```

5) Buscar huecos dentro de una columna

Esta es una pregunta habitual, pero la realidad es que no se me ocurre un escenario real donde un hueco entre los valores de una columna identity pueda ser relevante para nosotros. Generalmente quienes consideran relevante esto, es porque estan haciendo un mal uso conceptual de los identities. Recordemos que un identity funciona perfecto como valor interno para una clave primaria. Pero le queremos dar otros usos (como por ejemplo, que sea el codigo numerico de una factura) estamos cometiendo un error suicida. Los identities no son transaccionales y no se puede confiar en una tabla no tenga huecos si se efectuan operaciones de DELETE sobre la misma.

De todos modos, quienes esten buscando gaps, les recomiendo entrar a este post del excelente blog de Pinal Dave, donde no solamente está disponible un script para realizar dicha consulta (para nada algo trivial), sino que ademas hay una muy buena polémica al respecto del uso de los identities.

XML SQL SERVER 2008

El tipo de datos **xml** permite almacenar documentos y fragmentos XML en una base de datos de SQL Server. Un fragmento XML es una instancia XML en la que falta un solo elemento de nivel superior. Puede crear columnas y variables de tipo **xml** y almacenar instancias XML en las mismas.

También puede asociar una colección de esquemas XML con una columna, un parámetro o una variable del tipo de datos **xml**. Los esquemas de la colección se utilizan para validar y asignar un tipo a las instancias XML. En este caso, se dice que el XML tiene un tipo.

Limitaciones del tipo de datos xml

Tenga en cuenta que el tipo de datos **xml** tiene las limitaciones siguientes:

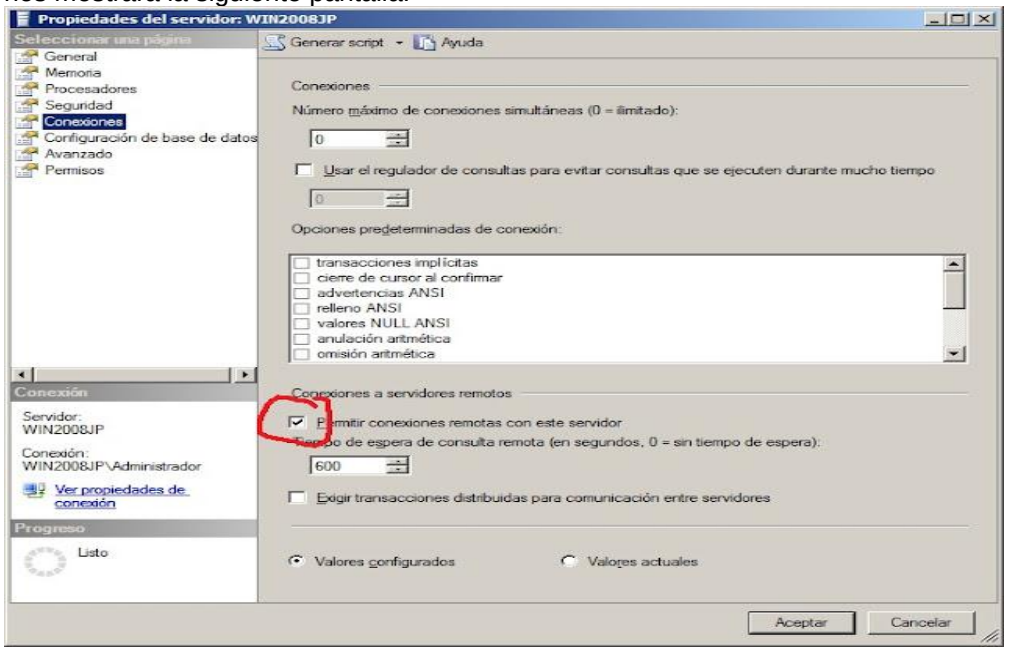
- La representación almacenada de las instancias del tipo de datos **xml** no puede superar los 2 GB.
- No puede utilizarse como un subtipo de una instancia de **sql_variant**.
- No admite la conversión a **text** ni a **ntext**. Use **varchar(max)** o **nvarchar(max)** en su lugar.
- No puede compararse ni ordenarse. Esto significa que un tipo de datos **xml** no puede utilizarse en una instrucción GROUP BY.
- No puede utilizarse como parámetro de ninguna función integrada escalar que no sea ISNULL, COALESCE o DATALENGTH.
- No puede utilizarse como columna de clave de un índice. Sin embargo, puede incluirse en forma de datos en un índice agrupado o puede agregarse explícitamente a un índice no agrupado mediante el uso de la palabra clave INCLUDE al crear el índice no agrupado.

PASOS PARA CONFIGURAR SQL SERVER 2008 PARA ADMITIR CONEXIONES REMOTAS

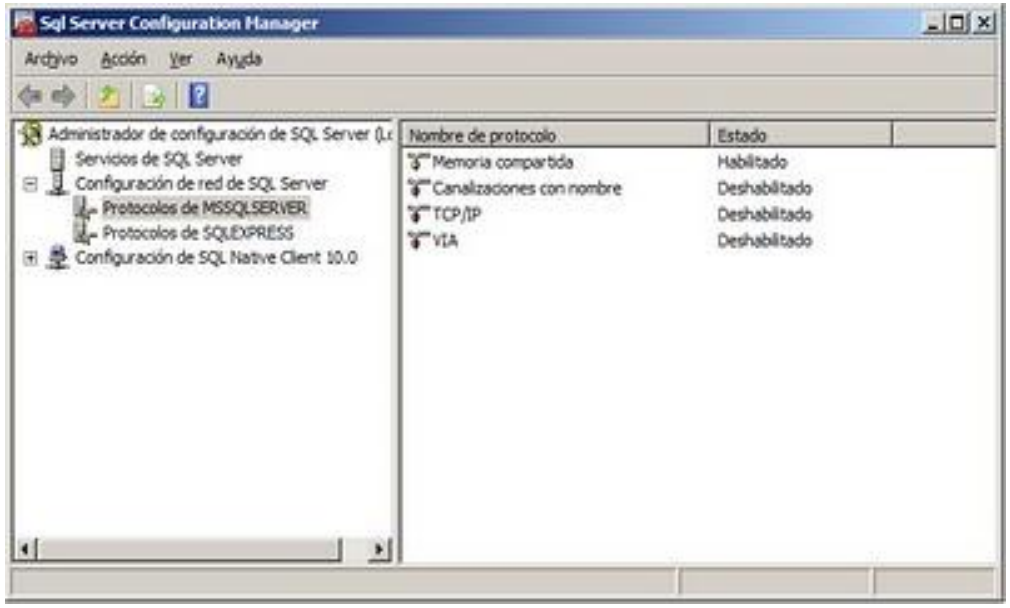
Las nuevas políticas de Seguridad de Microsoft incorporan restricciones a la hora de comenzar a usar SQL Server 2008. Los Servicios que antes estaban habilitados por defecto, ahora no lo están, y es función del Administrador ir habilitándolos según las necesidades de uso del mismo.

Una de las cuestiones más importantes es la de Admitir Conexiones Remotas en nuestro Servidor. A fin de habilitarlas y asegurarnos que se pueden conectar desde otros ordenadores debemos seguir unos sencillos pasos:

1. Abriremos SQL Server Management Studio, nos situamos encima de la instancia de nuestro Servidor y pulsamos botón derecho, Propiedades, seleccionamos Conexiones, nos mostrara la siguiente pantalla:



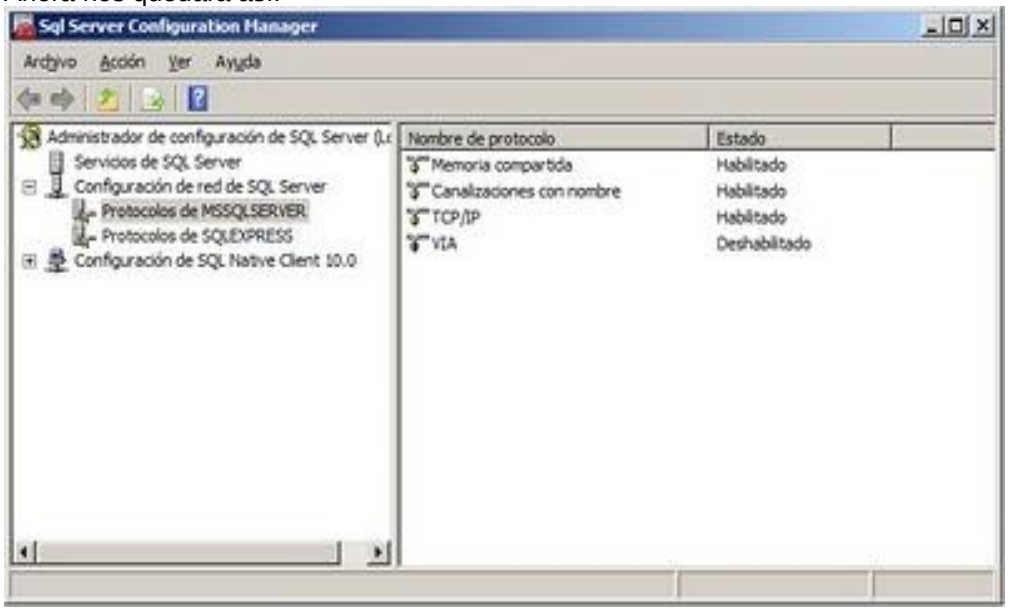
Ahora marcamos el checkbox: “Permitir conexiones remotas con este servidor” u pulsamos aceptar.
2. Vamos al Menú de Inicio > Programas > Microsoft SQL Server 2008 > Herramientas de Configuración > Administrador de Configuración de SQL Server, aparece la siguiente ventana:



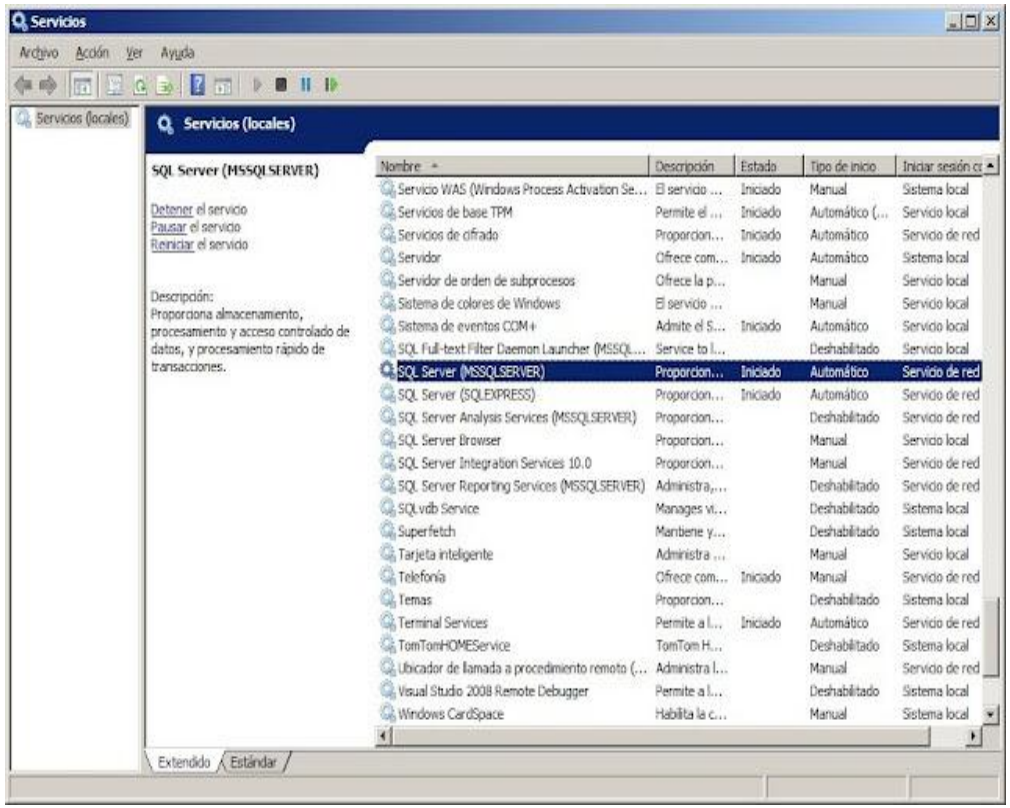
Seleccionamos la Configuración de red de SQL Server y luego Protocolos de MSSQLSERVER. Como podemos observar, por defecto solo tiene habilitado el protocolo de Memoria compartida, el resto estan deshabilitados. Básicamente es el tipo de protocolo que se usa, cuando nos conectamos a SQL Server desde el mismo Servidor.

Lo que debemos hacer es habilitar los protocolos: “Canalizaciones con nombre” y “TCP/IP”. Para lo cual, pulsamos con el botón derecho del ratón encima de los mismos y pulsamos Habilitar. En las dos ocasiones nos mostrara un mensaje informándonos, que para que la nueva configuración surta efecto abra que reiniciar el Servicio de SQL Server.

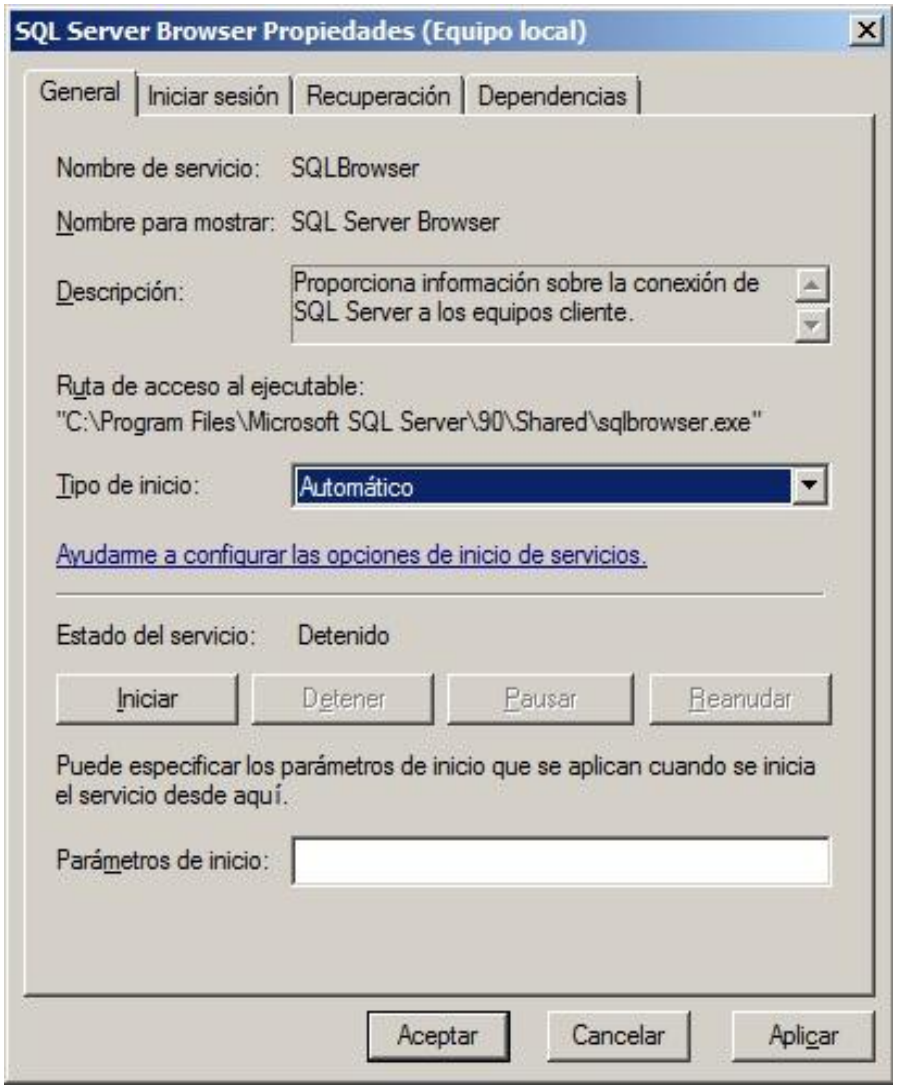
Ahora nos quedará así:



3. Vamos al Menú de Inicio > Ejecutar y escribimos services.msc y le damos aceptar. De esta forma nos abrirá la Consola de Administración de Servicios. Nos desplazamos hasta el Servicio con nombre “SQL Server (MSSQLSERVER)”, nos situamos encima y pulsamos el botón derecho del ratón, seleccionando; reiniciar. Con esto aplicaremos los cambios efectuados en el paso anterior.



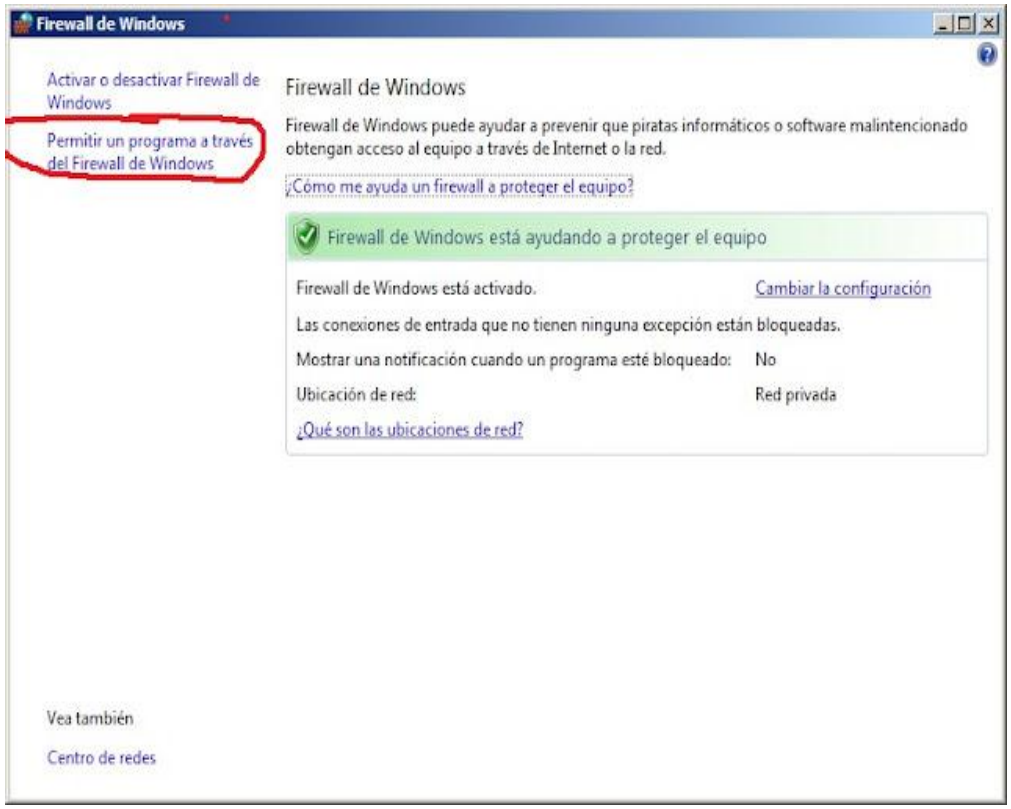
4. SI estamos utilizando SQL Server con nombre de instancia y sin emplear un número concreto de puerto TCP/IP, debemos habilitar el servicio SQL Server Browser, que se encuentra en la misma ventana de Servicios con el nombre de “SQL Server Browser”. Nos situamos encima y con el botón derecho del ratón pulsamos en Propiedades, o bien podemos hacer doble click, es lo mismo.



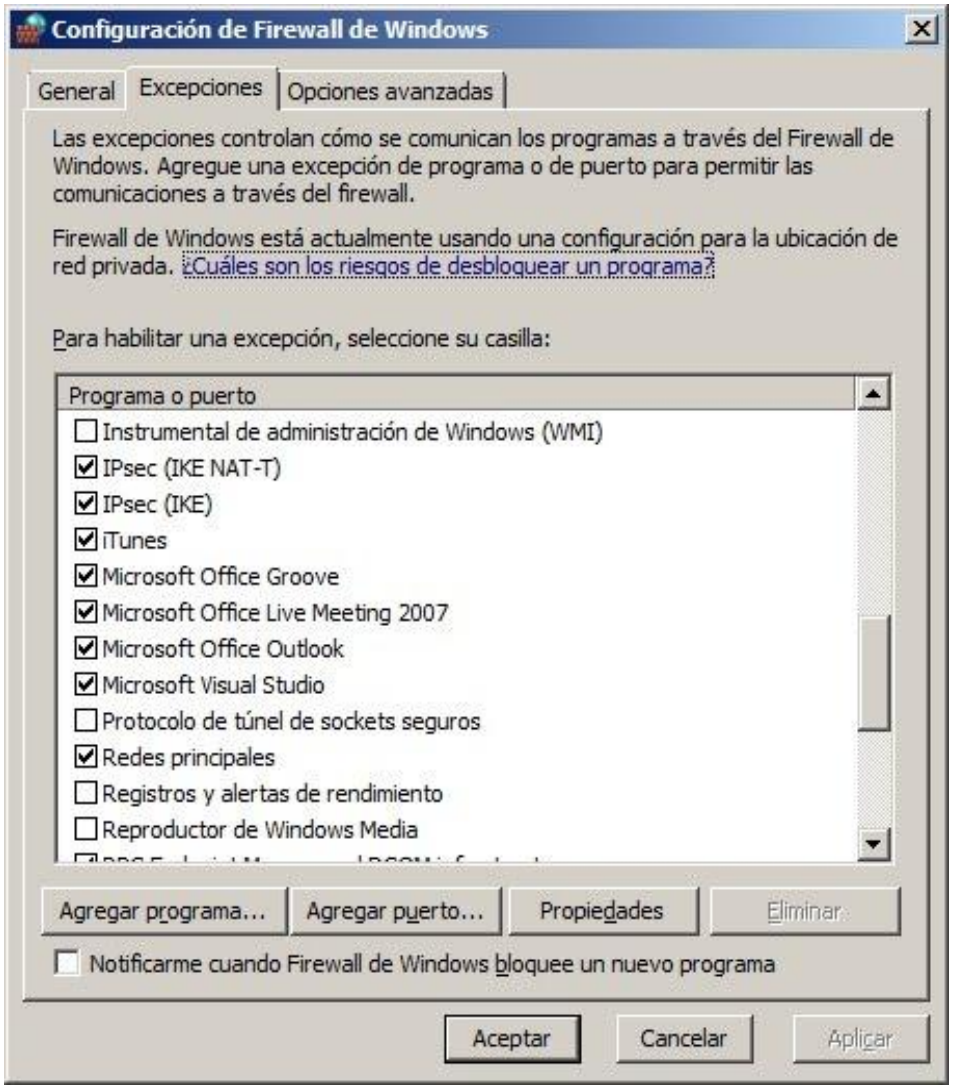
En el Tipo de Inicio, seleccionamos Automático y pulsamos Iniciar para que el Servicio arranque. Aceptar para cerrar la pantalla.

Este Servicio comporta ciertos riesgos de seguridad que deben ser considerados, pues existen otras alternativas a utilizar como configurar el Cliente de SQL Servidor con el Alias del Servidor, o utilizar la conexión incorporando el puerto de TCP/IP a usar, por defecto en SQL Server es el 1433.

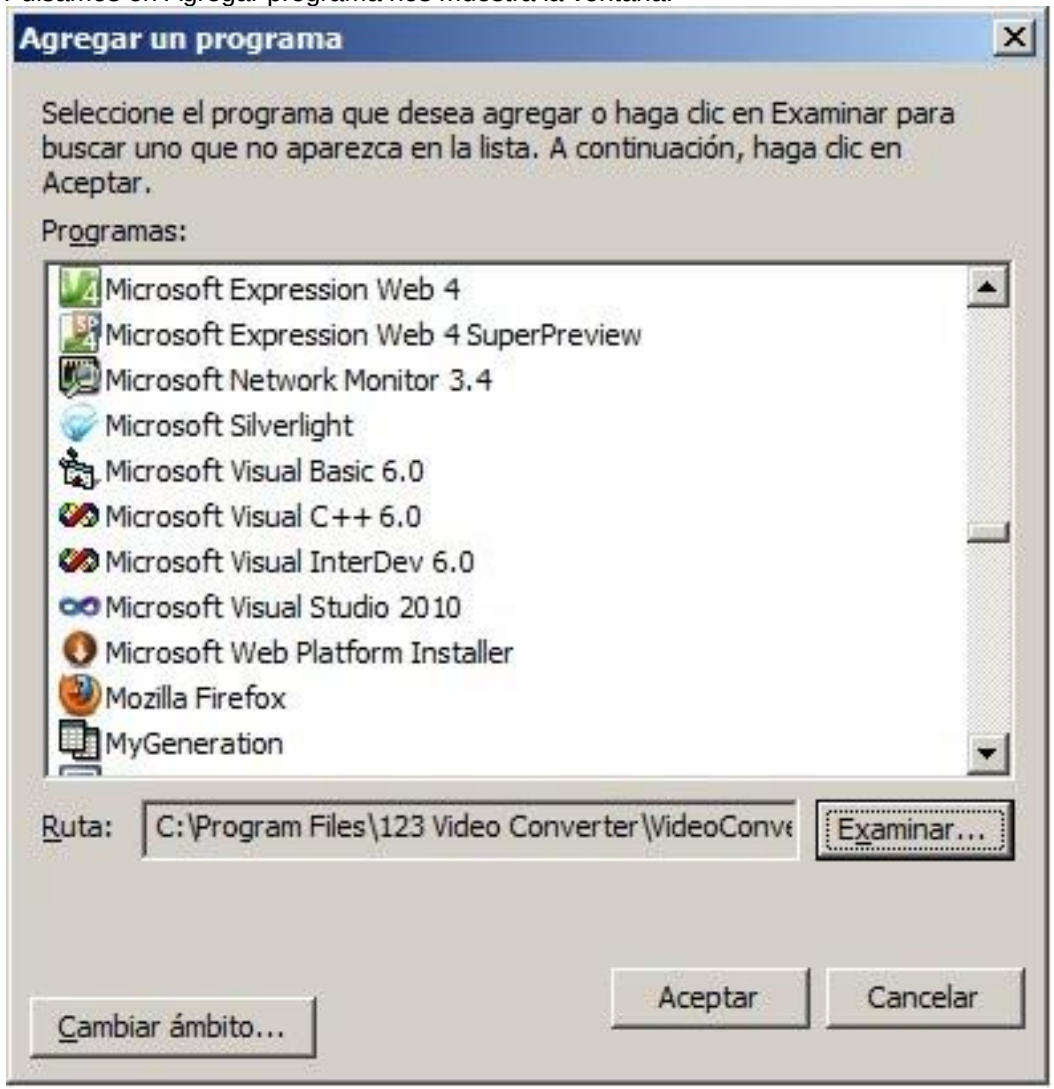
5. Y por último, en caso de tener habilitado el Firewall de Windows, cosa que deberíamos tener por Seguridad, deberemos configurarlo para que los Servicios de SQL Server y SQL Browser puedan comunicarse con el exterior. Vamos a Menú de Inicio , hacemos clic en Ejecutar , escribimos firewall.cpl y pulsamos Aceptar. Nos muestra esta ventana:



Pulsamos en “Permitir un programa a través del Firewall de Windows” nos muestra:



Pulsamos en Agregar programa nos muestra la ventana:



Pulsamos en Examinar e introducimos la carpeta donde se encuentra el Servicio de SQL Server:
"C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Binn" en la misma seleccionamos el programa: "sqlservr.exe" y pulsamos Aceptar. Repetimos la operación para añadir el SQL Server Browser que se encuentra en la carpeta: "C:\Program Files\Microsoft SQL Server\90\Shared". Seleccionamos el programa: "sqlbrowser.exe" y pulsamos Aceptar. Veremos que ambas excepciones nos aparecen en la pantalla de Configuración del Firewall.

Y con esto ya tenemos configurado nuestro Servidor SQL 2008 para permitir Conexiones desde cualquier ordenador de nuestra red.

.

ANEXOS

ANEXO 01

FUNCIONES SQL

Funciones de valores simples:

ABS(n)= Devuelve el valor absoluto de (n).
CEIL(n)=Obtiene el valor entero inmediatamente superior o igual a "n".
FLOOT(n) = Devuelve el valor entero inmediatamente inferior o igual a "n".
MOD (m, n)= Devuelve el resto resultante de dividir "m" entre "n".
NVL (valor, expresión)= Sustituye un valor nulo por otro valor.
POWER (m, exponente)= Calcula la potencia de un numero.
ROUND (numero [, m])= Redondea números con el numero de dígitos de precisión indicados.
SIGN (valor)= Indica el signo del "valor".
SQRT(n)= Devuelve la raíz cuadrada de "n".
TRUNC (numero, [m])= Trunca números para que tengan una cierta cantidad de dígitos de precisión.
VAIRANCE (valor)= Devuelve la varianza de un conjunto de valores.

Funciones de grupos de valores:

AVG(n)= Calcula el valor medio de "n" ignorando los valores nulos.
COUNT (* | Expresión)= Cuenta el numero de veces que la expresión evalúa algún dato con valor no nulo. La opción "*" cuenta todas las filas seleccionadas.
MAX (expresión)= Calcula el máximo.
MIN (expresión)= Calcula el mínimo.
SUM (expresión)= Obtiene la suma de los valores de la expresión.
GREATEST (valor1, valor2…)= Obtiene el mayor valor de la lista.
LEAST (valor1, valor2…)= Obtiene el menor valor de la lista.

Funciones que devuelven valores de caracteres:

CHR(n) = Devuelve el carácter cuyo valor en binario es equivalente a "n".
CONCAT (cad1, cad2)= Devuelve "cad1" concatenada con "cad2".
LOWER (cad)= Devuelve la cadena "cad" en minúsculas.

UPPER (cad)= Devuelve la cadena "cad" en mayúsculas.
INITCAP (cad)= Convierte la cadena "cad" a tipo titulo.
LPAD (cad1, n[,cad2])= Añade caracteres a la izquierda de la cadena hasta que tiene una cierta longitud.
RPAD (cad1, n[,cad2])= Añade caracteres a la derecha de la cadena hasta que tiene una cierta longitud.
LTRIM (cad [,set])= Suprime un conjunto de caracteres a la izquierda de la cadena.
RTRIM (cad [,set])= Suprime un conjunto de caracteres a la derecha de la cadena.
REPLACE (cad, cadena_busqueda [, cadena_sustitucion])= Sustituye un carácter o caracteres de una cadena con 0 o mas caracteres.
SUBSTR (cad, m [,n])= Obtiene parte de una cadena.
TRANSLATE (cad1, cad2, cad3)= Convierte caracteres de una cadena en caracteres diferentes, según un plan de sustitución marcado por el usuario.

Funciones que devuelven valores numéricos:

ASCII(cad)= Devuelve el valor ASCII de la primera letra de la cadena "cad".
INSTR (cad1, cad2 [, comienzo [,m]])= Permite una búsqueda de un conjunto de caracteres en una cadena pero no suprime ningún carácter después.
LENGTH (cad)= Devuelve el numero de caracteres de cad.

Funciones para el manejo de fechas:

SYSDATE= Devuelve la fecha del sistema.
ADD_MONTHS (fecha, n)= Devuelve la fecha "fecha" incrementada en "n" meses.
LASTDAY (fecha)= Devuelve la fecha del último día del mes que contiene "fecha".
MONTHS_BETWEEN (fecha1, fecha2)= Devuelve la diferencia en meses entre las fechas "fecha1" y "fecha2".
NEXT_DAY (fecha, cad)= Devuelve la fecha del primer día de la semana indicado por "cad" después de la fecha indicada por "fecha".

Funciones de conversión:

TO_CHAR= Transforma un tipo DATE ó NUMBER en una cadena de caracteres.
TO_DATE= Transforma un tipo NUMBER ó CHAR en DATE.
TO_NUMBER= Transforma una cadena de caracteres en NUMBER.

ANEXO 02

CAST y CONVERT

Debido a que SQL Server proporciona dos funciones, puede haber cierta confusión acerca de cuál es la mejor manera de utilizar y bajo qué circunstancias. **CONVERT** es específica de SQL Server, y permite una mayor amplitud de flexibilidad de la conversión entre valores de fecha y hora, los números fraccionarios, y significantes monetarias.

CAST es el más estándar ANSI de las dos funciones, es decir, que si bien es más portátil (es decir, una función que utiliza **REPARTO** puede ser utilizado en aplicaciones de base de datos más o menos como está), también es menos potente. **REPARTO** también se requiere al realizar la conversión entre los valores **decimales** y **numéricos** para conservar el número de decimales en la expresión original. Por estas razones, es mejor utilizar **elenco de** primera, a menos que haya algo específico que sólo puede proporcionar **CONVERT** en el trabajo que estás haciendo.

CAST y **CONVERT** también puede ser utilizado en conjunción con otros para conseguir ciertos efectos. Por ejemplo, una forma típica para producir una variable **char** con la fecha actual sería utilizar:

```
SELECT CONVERT (CHAR (10), CURRENT_TIMESTAMP,
```

Convierte una expresión de un tipo de datos a otro.
CAST y CONVERT tiene una funcionalidad similar.

CAST y CONVERT de SQL Sintaxis

Utilizar CAST:
CAST (expresión que data_type)

Uso de CONVERT:

CONVERT (data_type [(length)], expresión [, estilo])

SQL interpretar y convertir - Cadena
SELECT SUBSTRING ('CAST y CONVERT', 1, 3)
Valor devuelto = **CAS** (se obtiene de índice de 1 a 3)

SELECT CAST (CAST y CONVERT 'como char (3))
Valor devuelto = **CAS** (conseguirla sólo 3 caracteres)

SQL interpretar y convertir - Fecha Hora

-La conversión de fecha y hora a los datos de caracteres (vachar)
-Los valores predeterminados (style 0 ó 100, 9 o 109, 13 o 113, 20 o 120, y 21 o 121) siempre devuelven el año sin el siglo (aa).
-Añadir 100 a un valor de estilo para obtener un año de cuatro plazas que incluye el año del siglo (aaaa).
-A continuación se muestra ejemplo para la conversión de formato 1 de fecha y hora a formato diferente de fecha y hora, de modo que puede ser aplicada en varias condiciones.

Valor de la hora actual del Horario Fecha GETDATE ()
SELECT (GETDATE ()) = 06/06/2007 23:41:10.153

SELECT CONVERT (varchar, GETDATE (), 0)
Valor devuelto = **06 de junio 2007 23:07**
SELECT CONVERT (varchar, GETDATE (), 100)
Valor devuelto = **06 de junio 2007 23:07**

SELECT CONVERT (varchar, GETDATE (), 1)
Valor devuelto = **06/06/07**
SELECT CONVERT (varchar, GETDATE (), 101)
Valor devuelto = **06/06/2007**

SELECT CONVERT (varchar, GETDATE (), 2)
Valor devuelto = **6.7.06**

SELECT CONVERT (varchar, GETDATE (), 102)
Valor devuelto = **06/06/2007**
SELECT CONVERT (varchar, GETDATE (), 3)
Valor devuelto = **06/06/07**
SELECT CONVERT (varchar, GETDATE (), 103)
Valor devuelto = **06/06/2007**

SELECT CONVERT (varchar, GETDATE (), 4)
Valor devuelto = **6.6.07**
SELECT CONVERT (varchar, GETDATE (), 104)
Valor devuelto = **06/06/2007**

SELECT CONVERT (varchar, GETDATE (), 5)
Valor devuelto = **6.6.07**
SELECT CONVERT (varchar, GETDATE (), 105)
Valor devuelto = **06/06/2007**

SELECT CONVERT (varchar, GETDATE (), 6)
Valor devuelto = **06 de junio 07**
SELECT CONVERT (varchar, GETDATE (), 106)
Valor devuelto = **06 de junio 2007**

SELECT CONVERT (varchar, GETDATE (), 7)
Valor devuelto = **06 de junio 07**
SELECT CONVERT (varchar, GETDATE (), 107)
Valor devuelto = **06 de junio 2007**

SELECT CONVERT (varchar, GETDATE (), 8)
Valor devuelto = **23:38:49**
SELECT CONVERT (varchar, GETDATE (), 108)
Valor devuelto = **23:38:49**

SELECT CONVERT (varchar, GETDATE (), 9)
Valor devuelto = **06 de junio 2007 11:39:17:060 AM**
SELECT CONVERT (varchar, GETDATE (), 109)
Valor devuelto = **06 de junio 2007 11:39:17:060 AM**

SELECT CONVERT (varchar, GETDATE (), 10)
Valor devuelto = **6.6.07**
SELECT CONVERT (varchar, GETDATE (), 110)
Valor devuelto = **06/06/2007**

SELECT CONVERT (varchar, GETDATE (), 11)
Valor devuelto = **07/06/06**
SELECT CONVERT (varchar, GETDATE (), 111)
Valor devuelto = **06/06/2007**

SELECT CONVERT (varchar, GETDATE (), 12)
Valor devuelto = **070606**
SELECT CONVERT (varchar, GETDATE (), 112)
Valor devuelto = **20070606**

SELECT CONVERT (varchar, GETDATE (), 13)
Valor devuelto = **06 de junio 2007 23:40:14:577**
SELECT CONVERT (varchar, GETDATE (), 113)
Valor devuelto = **06 de junio 2007 23:40:14:577**

SELECT CONVERT (varchar, GETDATE (), 14)
Valor devuelto = **23:40:29:717**
SELECT CONVERT (varchar, GETDATE (), 114)
Valor devuelto = **23:40:29:717**

SELECT CONVERT (varchar, GETDATE (), 20)
Valor devuelto = **06/06/2007 23:40:51**
SELECT CONVERT (varchar, GETDATE (), 120)
Valor devuelto = **06/06/2007 23:40:51**

SELECT CONVERT (varchar, GETDATE (), 21)
Valor devuelto = **06/06/2007 23:41:10.153**
SELECT CONVERT (varchar, GETDATE (), 121)
Valor devuelto = **06/06/2007 23:41:10.153**

SELECT CONVERT (varchar, GETDATE (), 126)
Valor devuelto = **2007-06-06T23: 41:10.153**

SELECT CONVERT (varchar, GETDATE (), 131)
Valor devuelto = **21/05/1428 11:41:10:153 AM**

ANEXO 03

FILAS ESTADISTICAS

COMPUTE

Sabemos que esto me va permitir generar totales que aparecen como columnas de resumen adicionales al final del conjunto de resultados. Cuando se utiliza con BY, la cláusula COMPUTE genera interrupciones de control y subtotales en el conjunto de resultados. Puede especificar COMPUTE BY y COMPUTE en la misma consulta.

Una cláusula COMPUTE BY le permite ver tanto el detalle como las filas de resumen con una instrucción SELECT. Puede calcular valores de resumen para los subgrupos, o un valor de resumen para el conjunto de resultados completo.

Debemos recordar como conclusión que esta clausula crea nuevas filas que contienen resultados estadísticos a partir de funciones de agregación, asimismo dentro d las funciones Estadísticas tenemos COUNT, SUM, AVG y MIN

EJEMPLO

SELECT NombrE_Articulo, Und, Pventa,Almacen,Cant_stock FROM BIENES ORDER BY Nombre_Articulo COMPUTE SUM(Cant_Stock) BY Nombre_Articulo COMPUTE SUM(Cant_stock)

Nombre_Articulo	Und	Pventa	Almacen	Cant_stock
Ampicilina 500 Mg	Amp	0.3	Almacen 01	100
Megacilina 100 Mg	Und	0.6	Almacen 02	200
Aspirina 100 Mg	Caps	0.1	Almacen 01	500
Riboxon 125 Mg	Amp	1.5	Almacen 01	400
Calcioferol	Und	3.2	Almacen 02	600

WITH ROLLUP

EJEMPLO

Consideremos que tenemos 03 tablas relacionadas entre sí, de la cuales se detalla lo siguiente: siendo esta un movimiento de Universidad

- La tabla Matriculados es donde se almacena todos los registros de los alumnos matriculados
- La tabla Alumnos es donde se encuentra rehistrado, el padorn general de los alumnos como datos generales
- La tabla semestre es donde se encuentra especificado los ciclos academicos para la tabla Alumnos
- Además la tabla Matriculados esta relacionada con la tabla Alumnos por el Código de alumno y la tabla Alumnos esta relacionada con la tabla semestre por el semestre académico.

SELECT Codigo_alumno,Nombre_alumno,Curso_acad,Facultad_acad,COUNT (Cant_creditos) CREDITOS FROM Matriculados MAT INNER JOIN Alumnos ALU ON MAT.Codigo_alumno = ALU.Codigo_alumno INNER JOIN Semestre SEM ON ALU.Codsemestre = SEM.Codsemestre WHERE ALU.Codigo_alumno LIKE “[A1121]%” AND Facultad_acad =”Ingenieria” GROUP BY Facultad_acad,ALU.Codsemestre;

Codigo_alumno	Nombre_alumno	curso_acad	Facultad_acad	Creditos
A112120	Mariluisa Pascal	Ingles I	Ingenieria de Sistemas	02
A112110	Mariluisa Pereda	Analisis sistemas	Ingenieria de Sistemas	10
A112130	César Pereda	Matematica I	ingenieria Industrial	04
A112140	Harumi Pereda	Matematica IV	Ingeniria Industrial	04
A112160	Mariluisa Harumi	Ingles VI	Ingenieria Civil	02
A112100	Juana Torres	Estadistica I	Ingenieria Industrial	02
A112106	Pascal Ampudia	Analisis II	Ingenieria Civil	02

EJEMPLO

Según el ejemplo anterior vamos ahora aplicar WITH ROLLUP con el mismo ejemplo, y podremos analiar la diferencia entre cada una de ellas

SELECT Codigo_alumno,Nombre_alumno,Curso_acad,Facultad_acad,COUNT (Cant_creditos) CREDITOS FROM Matriculados MAT INNER JOIN Alumnos ALU ON MAT.Codigo_alumno = ALU.Codigo_alumno INNER JOIN Semestre SEM ON ALU.Codsemestre = SEM.Codsemestre WHERE ALU.Codigo_alumno LIKE "[A1121]%" AND Facultad_acad ="Ingenieria" GROUP BY Facultad_acad,ALU.Codsemestre WITH ROLLUP;

odigo_alumno	Nombre_alumno	curso_acad	Facultad_acad	Creditos
A112120	Mariluisa Pascal	Ingles I	Ingenieria de Sistemas	02
A112120	Mariluisa Pascal	Matematica IV	Ingenieria de Sistemas	03
A112120	Mariluisa Pascal	Estadistica I	Ingenieria de Sistemas	06
A112110	Mariluisa Pereda	Analisis sistemas	Ingenieria de Sistemas	10
A112130	César Pereda	Matematica I	ingenieria Industrial	04
A112140	Harumi Pereda	Matematica IV	Ingenieria Industrial	04
A112140	Harumi Pereda	Fisica II	Ingenieria Industrial	04
A112140	Harumi Pereda	Matematica IV	Ingenieria Industrial	04
A112140	Harumi Pereda	Planos y Diseños	Ingenieria Civil	04
A112160	Mariluisa Harumi	Ingles VI	Ingenieria Civil	02
A112100	Juana Torres	Estadistica I	Ingenieria Industrial	02
A112106	Pascal Ampudia	Analisis II	Ingenieria Civil	02

EJEMPLO

Según el ejemplo anterior 01 vamos ahora aplicar la misma sentencia, pero añadiremos **WITH CUBE**

SELECT Codigo_alumno,Nombre_alumno,Curso_acad,Facultad_acad,COUNT (Cant_creditos) CREDITOS FROM Matriculados MAT INNER JOIN Alumnos ALU ON MAT.Codigo_alumno = ALU.Codigo_alumno INNER JOIN Semestre SEM ON ALU.Codsemestre = SEM.Codsemestre WHERE ALU.Codigo_alumno LIKE "[A1121]%" AND Facultad_acad ="Ingenieria" GROUP BY Facultad_acad,ALU.Codsemestre WITH CUBE;

Codigo_alumno	Nombre_alumno	curso_acad	Facultad_acad	Creditos
A112120	Mariluisa Pascal	Ingles I	Ingenieria de Sistemas	02
A112120	Mariluisa Pascal	Matematica IV	Ingenieria de Sistemas	03
A112120	Mariluisa Pascal	Estadistica I	Ingenieria de Sistemas	06
A112110	Mariluisa Pereda	Analisis sistemas	Ingenieria de Sistemas	10
A112110	NULL	NULL	Ingenieria de Sistemas	04
A112130	César Pereda	Matematica I	ingenieria Industrial	04
A112130	NULL	NULL	ingenieria Industrial	04
A112140	Harumi Pereda	Matematica IV	Ingenieria Industrial	04
A112140	Harumi Pereda	Fisica II	Ingenieria Industrial	04
A112140	Harumi Pereda	Matematica IV	Ingenieria Industrial	04
A112140	Harumi Pereda	Planos y Diseños	Ingenieria Civil	04
A112160	Mariluisa Harumi	Ingles VI	Ingenieria Civil	02
A112160	NULL	NULL	Ingenieria Civil	02
A112100	Juana Torres	Estadistica I	Ingenieria Industrial	02
A112106	Pascal Ampudia	Analisis II	Ingenieria Civil	02

OVER

Esta Clausula permite dividir y/o clasificar los datos:

EJEMPLO

Tenemos dos tablas relacionadas entre si, las cuales son los alumnos y los tipos de carreras, donde nos piden que ordenemos los campos aplicando la clausula OVER.

SELECT Nombres,Alu.codcarrera,Alu.Codalumno,Tipo.detallecarrera, Alu.Semestre, SUM(Cant_cred) OVER (PARTITION BY Codalumno,Alu.Codcarrera) AS TOTAL FROM Matriculados

EJEMPLO

Vamos a sostener que tenemos las tablas Clientes, donde se registran todos los datos de los clientes, el cual simplificaremos una columna al máximo, pero para esto vamos a aplicar ejemplos practicos como ejecutar un reporte en forma consecutiva, pero para esto debemos aplicar la funcion ROW_NUMBER( ).

SELECT ROW_NUMBER() OVER (ORDER BY Nombre_Cliente), * FROM dbo.clientes;

Para esto obtendremos el siguiente Reporte consecutivo

	Nombre_cliente
1	Mariluisa Pascal
2	Harumi Pereda
3	Cesar Pereda
4	Mariluisa Pereda

EJEMPLO

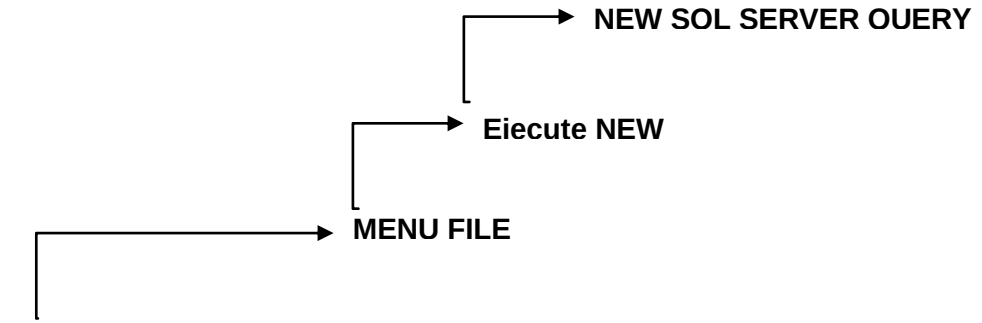
Asimismo se puede añadir calculos al resutlado de la funcion, por ejemplo:
SELECT (SELECT COUNT(*) FROM dbo.otratabla) + ROW_NUMBER() OVER (ORDER BY Nombre_cliente), * FROM bo.clientes

ANEXO 04

PROPIEDADES Y RESTRICCIONES
CODE EDITOR

Es un editor que se emplea para crear y editar scripts, dentro de las cuales consideremos 04 tipos del SQL SERVER MANAGEMENT STUDIO:

1. SQL QUERY EDITOR
2. MAX QUERY EDITOR
3. XML QUERY EDITOR
4. SQL MOBILE QUERY



Asimismo para ejecutar el script pulsareamos **F5.**

En el dialogo CONNECT TO SQL SERVER, selecciónе el servidor que desee conectarse y luego añada una ficha con extensión SQL, claro esta que para guardar el script debemos dar click en el botón o icono SAVE de la barra de herramientas o caso contrario Eejcute clic en el Menú FILE la Opción SAVE.

Para ver las Opciones de configuración de una Base de datos a olo lectura en el CODE EDITOR

Exec SP_dbOption Nombre_base_datos

Para cambiar el estado de una opción de base de datos a solo lectura

EXEC Sp_dboption NOMBRE_Basedatos, "READ ONLY", TRUE
EXEC Sp_dboption NOMBRE_Basedatos

Para cambiar el estado de una Opción de Base de datos a lectura y Escritura, aplicaremos:

EXEC SP_dboption NOMBRE_BASEDATOS, "READ WRITE", TRUE
EXEC Sp_dboption NOMBRE_Basedatos
GO

ALTER DATABASE – MODIFY FILE

Recordemos que en una base de datos el incremento que se utiliza con FILEGROWTH y un tamaño máximo en el momento de la creación del archivo con ALTER DATABASE Bdatos

EJEMPLO
Aumentar el tamaño de un archivo existente empleando ALTER DATABASE

```
ALTER DATABASE Base_datos
MODIFY FILE (
Name = listados_data,
SIZE = 20 Mb);
```

Recordemos que el comando ALTER DATABASE permite una acción mucho más importante sobre la base de datos tanto para modificar el tamaño de los archivos, como cambiar el nombre de la base de datos.

SP_dboption

Este procedimiento permite emplear en versiones anteriores no siendo esta recomendada debido a que inicialmente permite revisar y cambiar las opciones de configuración de una base de datos, siendo mas viusla y correcto emplear la opción ALTER DATABASE

EXEC sp_dboption
Go

Recordemos que esta Opcióm se utiliza para definir las opciones de configuración de la base de datos; Esta opción todavía existe en SQL 2008, pero solamente para conservar la compatibilidad de los scrips existentes; asimismo resaltar que ALTER DATABASE efectua las mismas funciones.

IDENTITY

Esta Propiedad debe ser asignada a una columna numérica entera durante la creación o la modificación de la tabla y debe ser definida al mismo tiempo que la columna a la que esta vinculada; el cual esta puede realizarce en un comando CREATE TABLE o también en un comando ALTER TABLE

Se puede utilizar las siguientes funciones para obtener más información sobre los tipos de identidad:

_IDENT_INCR	Para concoer el incremento del valor IDENTITY
_IDENT_SEED	Para conocer el valor inicial fijado en la creación del tipo IDENTITY.

Empleandose asi:
SET IDENT_INCR NOMBRE_TABLA
SET IDENT_SEED NOMBRE_TABLA

EJEMPLO
Crear una tabla Aplicando las propiedades IDENTITY

```
CREATE TABLE MOVIMIENTOS(
Numdoc              INT     IDENTITY(100,1),
FechaReg            DATETIME,
Monto               SMALLMONEY,
Tipofactura         CHAR(2) COLLATE Actua_vencida AS NULL
CONSTRAINT PK_Movimk    PRIMARY KEY NONCLUSTERED (Numdoc);
```

CONSTRAINT FK_MovimF FOREIGN KEY(Numdoc) REFERENCES VENTA(Numero))

Adicionar una columna con la Propiedad Identity; denominada Mumero_hoja que inicie en el numero 20 incrementadose de uno en uno

ALTER TABLE FOLIOS
ADD Numero_hoja INT IDENTITY (20,1) CONSTRAINT Idem_hoja PRIMARY KEY (Numero_hoja)

SET IDENTITY_INSERT TABLE ON

Permite la inserción de datos sin usar la propiedad IDENTITY y la numeración automática.
Ejemplo

USE Padron
GO
SET IDENTITY_INSERT Movimientos ON
INSERT INTO Padron.dbo.Movimientos (FechaReg,Monto)
VALUES("08-27-2010",100.00);
SET IDENTITY_INSERT Movimientos OFF

PRIMARY KEY

Esta Restricción automáticamente crea un índice único agrupado, por defecto con el nombre de la restricción; de ahí las opciones NONCLUSTERED y FILLFACTOR.
Una clave principal puede constar hasta 16 columnas, no puede haber más de una clave principal en una tabla ya que las columnas deben ser NOT NULL.

EJEMPLO
CREATE TABLE PERSONAL(
Codigo INT NOT NULL,
Nombres NVARCHAR(60) NULL,
CONSTRAINT PK_Codigo PROMARY KEY (Codigo));
-- Adicionemos una clave principal en una tabla VENTAS
ALTER TABLE VENTAS
ADD CONSTRAINTPK_MOV PRIMARY KEY NONCLUSTERED(Numdoc,Num_serie);

UNIQUE

Esta Restricción permite también traducir la regla de unidad para las otras claves únicas de la tabla o identificadores y claves secundarias; teniendo por conocimiento que pueden haber varias restricciones UNIQUE por cada tabla; asimismo las columnas utilizadas pueden ser NULL (No recomendado).

REFERENCIAS (REFERENCES)

Esta Restricción traduce la integridad referencial entre una clave externa de una tabla y una clave primaria o secundaria de otra tabla; ya que al definir una restricción a través de las instrucciones CREATE TABLE o ALTER TABLE es posible especificar las clausulas ON DELETE y ON UPDATE.

SET NULL

Cuandos e elimina la fila que corresponde a la clave primaria en la tabla relacionada, la clave externa toma el valor NULL.

EJEMPLO
Crear una restricción Foranea, en la tabla Producto que esté relacionado con la tabla Categoria.
ALTER TABLE Producto
ADD CONSTRAINT FK_Producto_Categoria FOREIGN KEY(Codigo) REFERENCES Categoria(Codigo);

DEFAULT

Se puede definir un valor predeterminado para todas las columnas con excepción de las columnas de tipo TIMESTAMP o las que poseen un tipo IDENTITY.
Recordemos que el valor puede ser una constante, una función como por ejemplo (USE, CURRENT,…..) o valores NULOS.

EJEMPLO
Adicionar una restricción DEFAULT para la columna SEXO de la tabla PERSONAL
ALTER TABLE PERSONAL
ADD CONSTRAINT DF_SEXO DEFAULT “Varon” FOR SEXO;

Ahora dentro de las Propiedades del SQL MANAGEMENT STUDIO, observaremos el valor por defecto dentro de las propiedades (Valor o enlace predeterminado).

CHECK

Esta restricción se asocia automáticamente a la columna especificada en la expresión de la condición.

USE EJEMPLO
GO
ADD CONSTRAINT CHK_Modelo CHECK (MODELO LIKE “B%”);

DEFAULT

Se puede definir un valor predeterminado para todas las columnas con excepción de las columnas de tipo TIMESTAMP o las que poseen un tipo IDENTITY.
Recordemos

ANEXO 05

ORDENES DE SQL

PIVOT

Esta instrucción es muy potente y fácil de utilizar permitiendo generar un resultado en forma de filas distintas previendo definiar los datos y cual es la columna PIVOT.

EJEMPLO
Tenemos una tabla llamada MATRICULADOS en la cual están inscritos los alumnos de una universidad, siendo la siguiente estructura:

Codigo_alumno	VARCHAR(10)
Semestre	VARCHAR(15)
Creditos	INTEGER
Curso_cod	VARCHAR(10)

Ahora vamos a listar la consulta

USE GESTOR
GO
SELECT * FROM Matriculados;

Empleando PIVOT vamos a listar la tabla
SELECT Codigo_alumno,[I] AS “I CICLO” ,[II] AS ”II CICLO”, [III] AS “III CICLO”, [IV] AS “IV CICLO”, [V] AS “V CICLO” FROM MATRICULADOS PIVOT ((SUM(Creditos) FOR Semestre IN ([I], [II], [III], [IV], [V])) AS PVT;

EJEMPLO
Ahora aplicando con dos tablas, realizaremos la misma instrucción PIVOT considerando que existe una tabla llamada ALUMNOS donde se encuentran registrados el Padron general de los alumnos

SELECT ALU.Codigo_alumno, [I] AS “I CICLO”, [II] AS “II CICLO”, [III] AS “III CICLO”, [IV] AS “IV CICLO”, [V] AS “V CICLO”,ALU.Nombre_alumno FROM MATRICULADOS

PIVOT (SUM(Creditos) FOR SEMESTRE IN ([I], [II], [III], [IV], [V]) AS PVT INNER JOIN ALUMNOS ALU ON ALU.Codigo_alumno = PVT.Codigo_alumno;

Codigo_alumno	Semestre	Creditos	Curso_cod
101000	I	10	A100
101000	I	2	A200
101000	I	4	A001
101000	II	8	A008
104000	I	12	A102
104000	II	6	A804
102000	I	8	A120

UNPIVOT

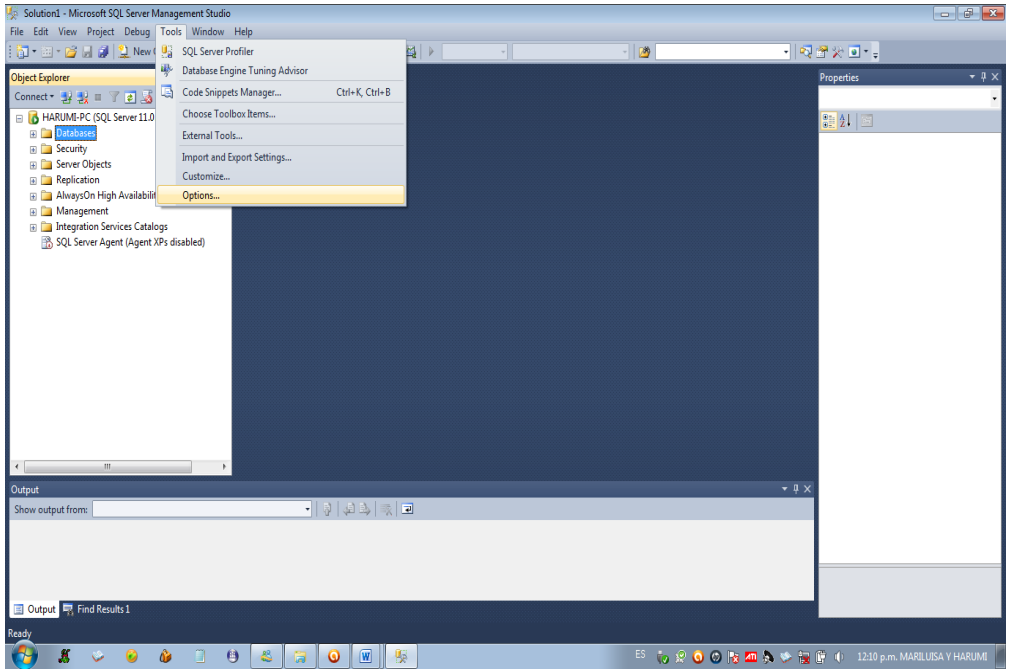
Esta instrucción realiza la inversa de lo que efectua la instrucción PIVOT, pero se emplea dentro de ella la misma función de las tablas CTE.

USE GESTOR
GO
WITH CONTROL(Codigo_alumno,Semestre I,Semestre II,Semestre III, Semestre IV, Semestre V, Nombre_alumno AS (SELECT ALU.Nombre_alumno, [I] AS “I CICLO”, [II] AS “II CICLO”, [III] AS “III CICLO”, [IV] AS “IV CICLO”, [V] AS “V CICLO”, ALU.codigo_alumno FROM MATRICULADOS PIVOT (SUM(Creditos) IN ([I], [II], [III], [IV], [V]) AS PVT INNER JOIN ALUMNOS ALU ON ALU.Codigoalumno= PVT.Codigoalumno) SELECT Codigoalumno,Nombre_alumno,Creditos FROM CONTROL UNPIVOT (Creditos FOR CONTROL IN ([V]) AS UNPIVOT;

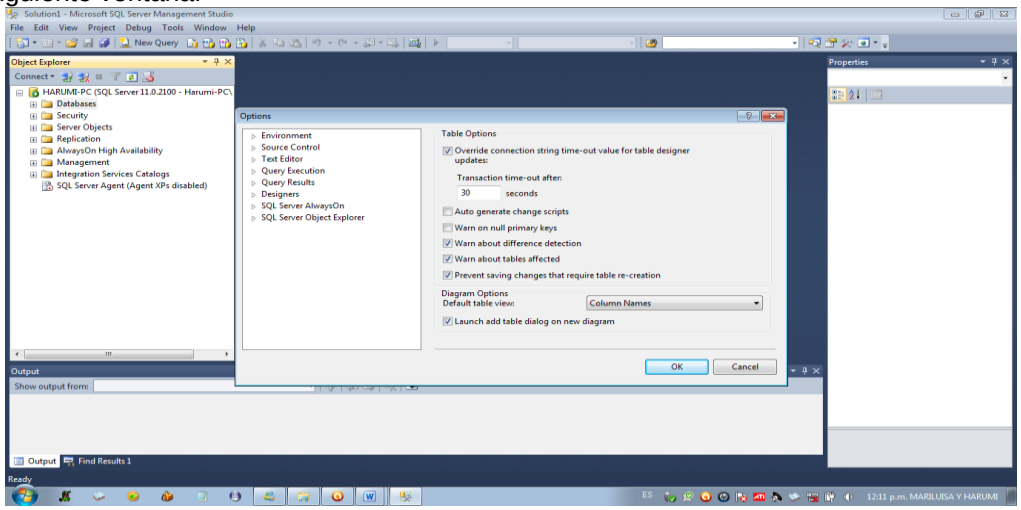
ANEXO 06

PROBLEMAS PERSISTENTES AL QUERER MODIFICAR O REALIZAR CAMBIOS EN UNA TABLA

Supongamos que tenemos creado nuestras tablas y posteriormente deseamos modifica o realizar cambios en su estructura, muchas veces nos encontraremos que inicialmente muestra un Mensaje de Error “Error Where encontred the sabe process some database saved…..”.
Pues para esto el operador después de crear las tablas, debera dar clic en la Opción HERRAMIENTAS del Menú principal.



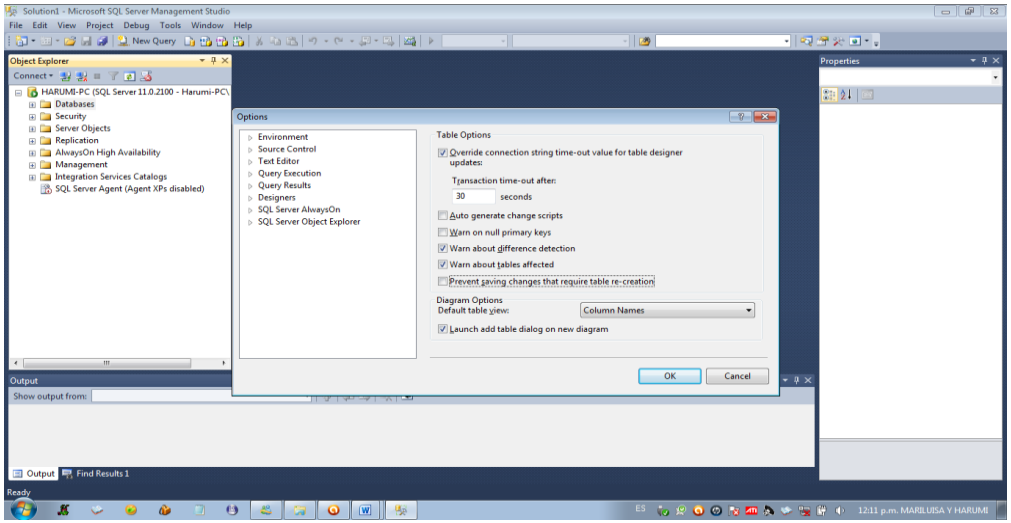
Luego de seleccionar dicha opción en el el menú barra observaremos, que muestra la siguiente ventana:



Donde el operador deberá seleccionar la opción desplegable de DESIGNERS, encontrando dentro de ella dos opciones tales como:

- Table and Database Designers
- Analysis Services Designers

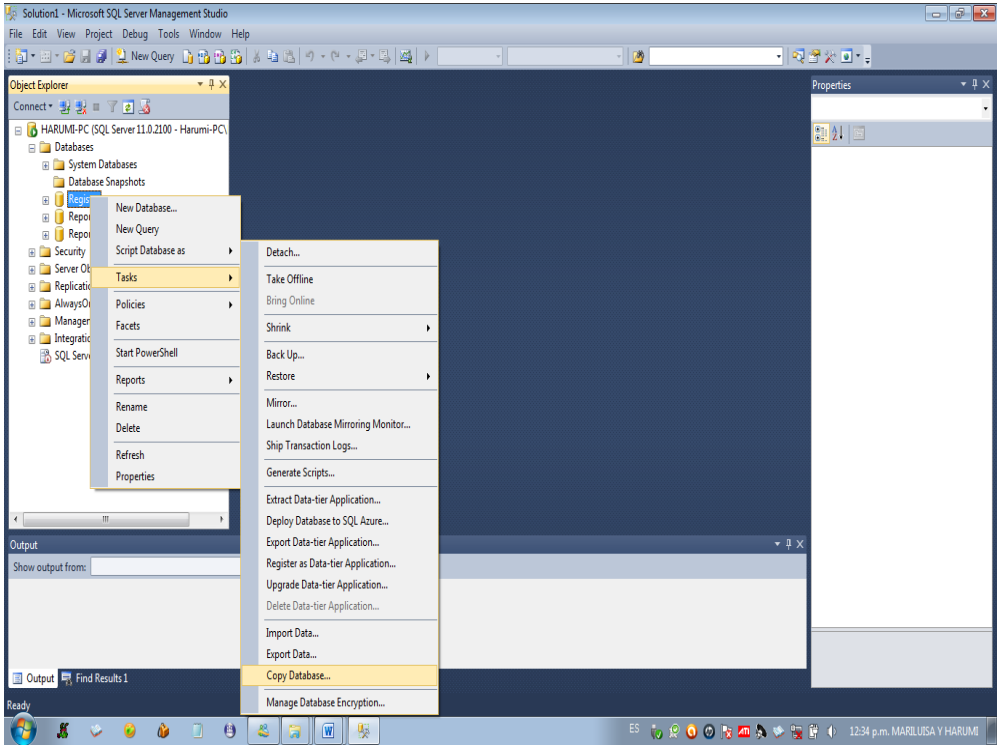
Para esto desactivaremos la Casilla que dice “Impedir Guardar Cambios que requieran re-creación”

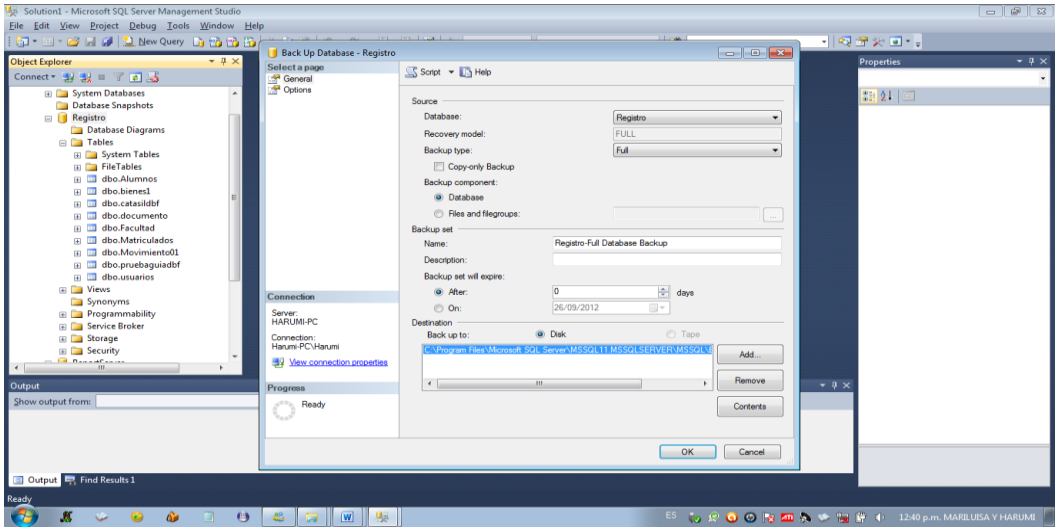


ANEXO 07

COPIA DE SEGURIDAD DE LA BASE DE DATOS

Esto te mostrará un cuadro de diálogo como el mostrado, puedes pulsar en el botón **Aceptar** para hacer la copia, pero si quieres elegir la ruta en la que se hará la copia, tendrás que pulsar en el botón **Agregar...** con idea de que puedas elegir donde quieres guardarlo





Seleccionamos la opción del botón Agregar, donde mostrará un nuevo cuadro de diálogo, para poder colocar la ruta de la nueva carpeta donde se grabara la base de datos copia backup, por lo que es necesario escribir el nombre del fichero de copia de seguridad; en nuestro caso colocaremos el nombre Backup bd Registro en la Carpeta Alumno del Disco.

INDICE

INTRODUCCION
Que es SQL
Caracteristicas del SQL

Capitulo I
Instalación de SQL
Requisitos de instalación del SQL
Notas de seguridad
Compatibilidad de Sistemas operativos
Implementación en Windows XP
Actualizar el SQL a nuevas versiones
Compatibilidad de idiomas
Restuar Base de datos antiguas
Generar copias de seguridad
Propiedades de las base de datos
Compatibilidad de la Base de datos

Capitulo II
Base de datos
Estructuración de una base de datos
Archivos de una base de datos
Tamaño de base de datos
Tipos de Archivos
Crear una base de datos usando SQL Server Management Studio.
Sintaxis de Base de datos
Notas de seguridad
Base de datos y grupo de archivos
Modelo de base de datos
Información de la Base de datos
Ejemplos practicos del 01 al 022
Ejercicios para realizar

Capitulo III
Tablas

Estructuras de Tablas
Creación de tablas
Nomenclatura de tablas
Modificación de tablas
Conceptos de:
Indices
Restricciones
Vistas
Estructuración de una base de datos
T-SQL
Procedimientos Almacenados
Propiedades
Normalizaciones
Integridad y tipos
Integridad de dominio
Integridad Referencial
Integridad fijada por el operador
Formas de Normalizacion
Definición de claves primarias
Identidades
Restricciones de identidades
Primary Key
Foreign Key
Unique
Default
Check
Verificación de la definición de la estructura de una table
Tabla del Sistema SYSOBJECTS
Modificar una tabla
Verificar los cambios de una tabla
Ejemplos de TSQL

Capitulo IV

Integridad de datos
Tipos de Restricciones
Clausula CONSTRAINT
Restriccion FOREIGN KEY
Diccionario de datos para restricciones
Uso de DROP TABLE
Uso de ALTER TABLE
Integridad de Identidad
Uso de Claves primarias
Tipos de Claves compuestas
INSERT INTO
Uso de claves foráneas y relacionas entre tablas
Formas de deshabilitar restricciones
Restricciones UNIQUE
Integridad de Dominio
CREATE DEFAULT
Uso de SP_HELPCONSTRAINT
Propiedades de identity
Integridades de Dominio
Ejercicios practicos de Restricciones
Uso del Administrador Corporativo
Tipos de diagramas
Integridad Referencial en cascada

Capitulo V

T-SQL
Uso de la función REPLACE
Script y lotes
Tipos de datos
Uso de variables
Uso de funciones de SQL
Tipos de datos de fecha y hora
Funciones de seguridad de SQL
Funciones de tipo carácter
Transact TSQL
Eejemplos con tipos de variables
Variables locales y globales
Uso de operadores
Ejemplos con precedencia de Operadores
Funciones matemáticas
Funciones de Metadatos
Store Procedure
Fucniones de Fecha y hora
Funciones del sistema
Restaurar un backup de una base de datos

Capitulo VI

Consultas Basicas
Uso y definición de la clausula WHERE
Tipos de operadores
Ejemplos practicos de consultas básicas
Uso de la clausula DISTINCT
Uso de la clausula INTO
Uso de la función CONVERT
Uso de la función LIKE
Uso de la clausula ORDER BY
Uso de GROUP BY
Uso de HAVING
Insertar registros empleando identity
Truncate

Capitulo VII

Agrupamiento de datos
GROUP BY con el operador ROLLUP
COMPUTE BY
Consultas y agrupamientos de datos
Definición de las caracteristicas de grouping sets
GROUPING SETS en SQL
Uso de la función UNION ALL
Auditoria y permisos en SQL
Uso de combinaciones
Combinaciones Internas
Combinaciones externas
Combinaciones Cruzadas
Autocombinación
Combinaciones y funciones de agrupamiento
Combinaciones de dos a más tablas
Combinaciones externas haciendo uso de funciones de agrupamiento
Uso de WITH ROLLUP
Sub consultas de multiples tablas
Sub consultas dentro de una clausula FORM
Sul consultas escalares
Sub consultas correlacionadas
Sub consultas anidadas
Funciones de Sub consultas
Uso del operador EXISTS
Funciones agregadas
Consultas con parámetros
Consultas de unidades externas
Agrupamiento de Registros
SQL DINAMICO TSQL
TRIGGERS SQL
Ejecutar un procedimiento almacenado
Uso de Reglas en SQL
Eeliminar valores predeterminados
Consultas Recursivas
ORDPATH

Capitulo IIX

Sentencias Anidadas
Datos definidos por el usuario
Indices de Gestion
TRANSACT TSQL
Importancia de los índices CLUSTERED
Tipos de índices
Autocombinaciones
Importancia de los índices
Crear reglas
Uso de EXCEPT
Ejemplos con INTER SECT
Uso de ejemplos con UNION, INTERCEP, EXCEPT
Diferencias simétricas
Indexar tablas de desarrollo
Uso de CREATE INDEX
Uso de variables de TSQL
TSQL en tablas persistentes
TSQL en fecha y hora
HEARCHYID
Procedimientos almacenados
Recomendaciones sobre el uso de tablas temporales

Capitulo IX

Improve Data Quality
Vistas y mantenimientos de datos
Creación de Vistas
Vistas Indexadas
Borrar una vista
Uso de SUBSTRING
Uso de DELETE y TRUNCATE
Tablas CTE

Capitulo X

Auditoria de Seguridad
Crear y habilitar la Auditoria de seguridad
Tipos de datos FileStream
Empleo de MERGE
LINQ to SQL
Utilizar base de datos espejos
Servicio BROKER
Uso de propiedad IDENTITY avanzada
Configurar conexiones remotas en SQL