

SQL SERVER – DESCRIPCIÓN DEL ENTORNO Y CREACIÓN DE BASES DE DATOS.....	5
Introducción a SQL Server 2000	5
Ediciones e Instalación de SQL Server	9
Ediciones SQL Server 2000.....	9
Instalación de SQL Server	11
Alguna edición de Windows 2000 Server	11
Server y client tools.....	12
Pasos para la instalación sobre Windows NT Server.....	12
Verificar la instalación de SQL Server.....	22
Modos de autenticar las cuentas de los usuarios.....	26
INICIO DE SESIÓN.....	26
Para modificar la autenticación realice los siguientes pasos:	26
Usuarios de Base de Datos	30
Roles por Servidor	32
Dbcreator	32
Roles por Base de Datos	33
BASES DE DATOS DE SQL SERVER	34
Objetos de una Base de Datos.....	34
Creación de Base de Datos.....	37
Páginas y extensiones	38
Archivos y grupos de archivos físicos de la base de datos	40
Archivos de Registro (LOG de Transacciones)	41
Creación de Base de Datos	42
Desde el Asistente	42
Desde el Administrador Empresarial	48
Desde el Analizador de Consultas	51
¿Quiénes pueden crear bases de datos?	54
Ejemplos de creación de base de datos empleando el Analizador de Consultas	54
Creando Múltiples Archivos	57
Renombrando Base de Datos	63
CREACIÓN DE TABLAS.....	65
Temas:	65
Tipos de Datos de SQL Server 2000	66
Utilizar datos binarios	66
Tipos de datos definidos por el usuario	70
Empleo de Comandos DDLL (Data Definition Language).....	73
Tablas del Sistema	73
Tablas del Usuario	73
Permanentes	74
Temporales.....	74

Creación de tablas.....	74
Consideraciones al crear tablas.....	74
Modificación de la estructura de las tablas	79
Valores autogenerados para las columnas.....	82
Propiedad Identity	82
Función NEWID y Datos de tipo UNIQUEIDENTIFIER.....	83
Eliminación de tablas	83
Implementar Restricciones	85
Definir restricción PRIMARY KEY	87
Definir FOREIGN KEY Constraint	89
Definir CHECK CONSTRAINT.....	90
Implementar DEFAULT CONSTRAINTS.....	91
DIAGRAMA DE BASE DE DATOS.....	94
RECUPERAR INFORMACIÓN.....	97
Objetivos:.....	97
Temas:.....	97
Select	97
Insert.....	105
Update	105
Delete	106
Recuperar información de dos o más tablas (Joins)	107
Desencadenadores.....	110
Asignar Roles y/o Permisos – Comandos Dcl (Data Control Language)	112
EJERCICIOS PROPUESTOS.....	118
IMPLEMENTAR VISTAS Y PROCEDIMIENTOS ALMACENADOS.....	128
¿Qué es una vista?.....	128
Agregar, Modificar y Eliminar una Vista.....	130
Crear Vistas.....	130
Modificar Vistas.....	134
Eliminar Vistas.....	136
Procedimientos Almacenados.....	138
Crear, Modificar y Eliminar un Procedimiento Almacenado	139
Crear Procedimientos Almacenados.....	139
Modificar Procedimientos Almacenados	145
Eliminar Procedimientos Almacenados.....	146
Funciones en SQL Server 2000 (1/2)	147
Tipos de funciones	147
Funciones Escalares	147
Funciones de tabla en línea	148
Las funciones de tabla de multi sentencias.....	148
Llamando Funciones	149
Limitaciones.....	149
Columnas computadas	149

GLOSARIO 151**REFERENCIA DEL TRANSACT-SQL 153****TIPOS DE VALOR 153**

Utilizar datos char y varchar.....	153
Utilizar datos de fecha y hora.....	154
Formato alfabético de las fechas	155
Formato numérico de fecha	156
Formato de cadena sin separar.....	157
Formatos de hora.....	157
Formato datetime de ODBC	158
Utilizar datos enteros	159
Utilizar datos decimal, float y real.....	159
Utilizar datos float y real	160
Utilizar datos text e image	160
Utilizar Constantes.....	161
Utilizar constantes en Transact-SQL.....	162

Funciones 162

Utilizar funciones del sistema.....	164
Utilizar funciones de cadena.....	165
Utilizar SUBSTRING	166
Comparación de CHARINDEX y PATINDEX	166
Utilizar STR	167
Utilizar STUFF	168
Comparación de SOUNDEX y DIFFERENCE	168
Utilizar las funciones text, ntext e image	169
Utilizar funciones matemáticas	170
Utilizar funciones trigonométricas	171
ACOS y COS	172
ASIN y SIN	172
ATAN , ATN2, TAN y COT	172
DEGREES.....	172
RADIANS	173
Comparación de CEILING y FLOOR.....	173
Comparación de LOG y LOG10.....	173
Utilizar las funciones exponenciales POWER y EXP	173
Utilizar RAND	173
Funciones de fecha.....	174
Utilizar GETDATE	174
Comparación de DATEPART y DATENAME	175
Comparación de DATEADD y DATEDIFF	175
Funciones que devuelven identificadores y nombres de usuarios	176
Obtener identificadores o cuentas de inicio de sesión	177
Obtener nombres de usuario de base de datos o identificadores de usuario.....	179
Funciones de conversión.....	180
El parámetro <i>estilo</i>	183

Expresiones..... 183

Utilizar operadores en expresiones.....	184
Operadores aritméticos	185
Operadores binarios	186
Operadores de comparación.....	186
Operador de concatenación de cadenas	188
Valores NULL.....	189

Miscelaneo..... 190

Utilizar comentarios	190
----------------------------	-----

UPDATE Products.....	192
Utilizar palabras clave reservadas	192
Sinónimos.....	192

SQL Server – Descripción del Entorno y Creación de Bases de Datos

Objetivos:

- Entender que es SQL Server
- Requisitos de Hardware y Software
- Seguridad
- Crear bases de datos

Temas:

- Introducción a SQL Server
- Ediciones de SQL Server
- Instalación de SQL Server
- Modos de autenticar las cuentas de los usuarios
- Bases de Datos de SQL Server
- Creación de Bases de Datos

Introducción a SQL Server 2000

SQL Server 2000 es un sistema de gestión de bases de datos relacionales (SGDBR o RDBMS: Relational Database Management System) diseñado para trabajar con grandes cantidades de información y la capacidad de cumplir con los requerimientos de proceso de información para aplicaciones comerciales y sitios Web.

SQL Server 2000 ofrece el soporte de información para las tradicionales aplicaciones Cliente/Servidor, las cuales están conformadas por una interfaz a través de la cual los clientes acceden a los datos por medio de una LAN.

La hoy emergente plataforma .NET exige un gran porcentaje de distribución de recursos, desconexión a los servidores de datos y un entorno descentralizado, para ello sus clientes deben ser livianos, tales como los navegadores de Internet los cuales accederán a los datos por medio de servicios como el Internet Information Services(IIS).

SQL Server 2000 está diseñado para trabajar con dos tipos de bases de datos :

- **OLTP (OnLine Transaction Processing)** Son bases de datos caracterizadas por mantener una gran cantidad de usuarios conectados concurrentemente realizando ingreso y/o modificación de datos. Por ejemplo : entrada de pedidos en línea, inventario, contabilidad o facturación.
- **OLAP (OnLine Analytical Processing)** Son bases de datos que almacenan grandes cantidades de datos que sirven para la toma de decisiones, como por ejemplo las aplicaciones de análisis de ventas.

SQL Server puede ejecutarse sobre redes basadas en Windows Server así como sistema de base de datos de escritorio en máquinas Windows NT Workstation, Windows Millenium y Windows 98.

Los entornos Cliente/Servidor, están implementados de tal forma que la información se guarde de forma centralizada en un computador central (**servidor**), siendo el servidor responsable del mantenimiento de la relación entre los datos, asegurarse del correcto almacenamiento de los datos, establecer restricciones que controlen la integridad de datos, etc.

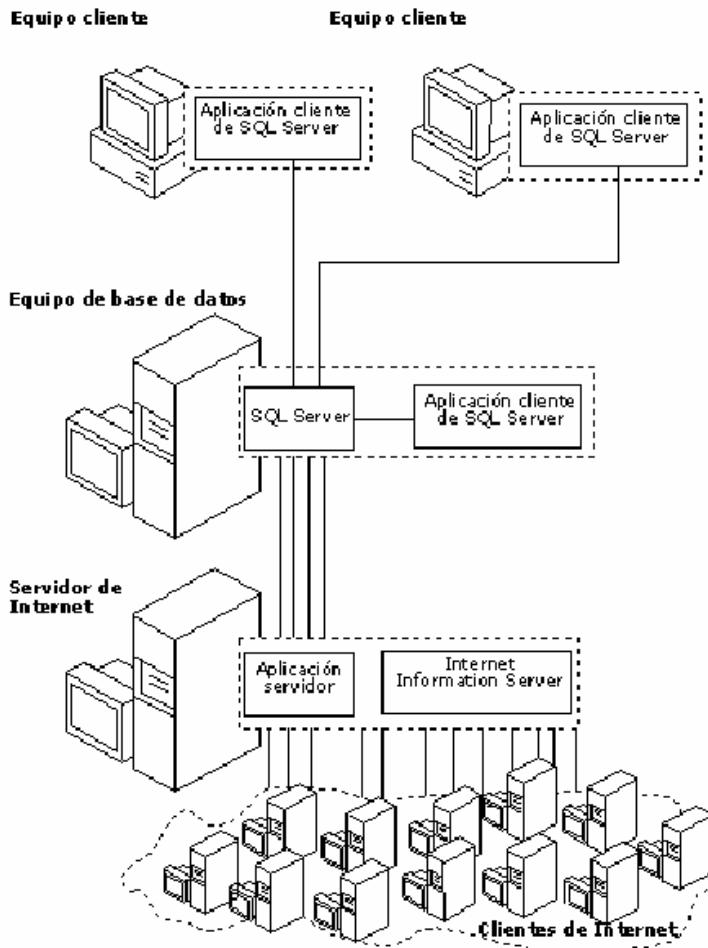
Del lado cliente, este corre típicamente en distintas computadoras las cuales acceden al servidor a través de una aplicación, para realizar la solicitud de datos los clientes emplean el **Structured Query Language (SQL)**, este lenguaje tiene un conjunto de comandos que permiten especificar la información que se desea recuperar o modificar.

Existen muchas formas de organizar la información pero una de las formas más efectivas de hacerlo está representada por las **bases de datos relacionales**, las cuales están basadas en la aplicación de la teoría matemática de los conjuntos al problema de la organización de los datos. En una base de datos relacional, los datos están organizados en tablas (llamadas relaciones en la teoría relacional).

Una tabla representa una clase de objeto que tiene importancia para una organización. Por ejemplo, se puede tener una base de datos con una tabla para empleados, otra para clientes y otra para productos del almacén. Las tablas están compuestas de columnas y filas (atributos y tuplas en la teoría relacional).

La tabla **Empleados** tendría columnas para el nombre, el apellido, código del empleado, departamento, categoría laboral y cargo. Cada fila representa una instancia del objeto representado por la tabla. Por ejemplo, una fila de la tabla **Empleados** representa el empleado cuyo Id. de empleado es 12345.

Al organizar los datos en tablas, se pueden encontrar varias formas de definirlas. La teoría de las bases de datos relacionales define un proceso, la normalización, que asegura que el conjunto de tablas definido organizará los datos de manera eficaz.



SQL Server incluye un conjunto de herramientas que facilitan la instalación y administración del servidor así como un conjunto de herramientas que facilitan el diseño e implementación de base de datos, entre ellos podemos mencionar:

- **SQL Server 2000 Database Engine**, diseñado para almacenar detalladamente los registros de las operaciones transaccionales (OLTP), este motor es responsable de mantener la seguridad de los datos, proveer un adecuado nivel de tolerancia a fallos, optimizar las consultas, emplear adecuadamente los bloqueos de recursos para optimizar la concurrencia, etc.
- **SQL Server 2000 Analysis Services**, provee herramientas para consultar información almacenada en *data warehouses* y *data marts*, como por ejemplo cuando se desea obtener información totalizada acerca de los niveles de ventas mensuales por regiones de ventas, etc.

Soporte para aplicaciones, SQL Server brinda a las aplicaciones clientes la posibilidad de acceder a los datos a través de un lenguaje denominado Transact-SQL, asimismo es importante mencionar que ahora existe un soporte para devolver la información en formato XML.

Como soporte para las aplicaciones clientes tenemos:

1. *SQL Distributed Management Objects (SQL-DMO)* API que brinda un conjunto de objetos COM que encapsulan toda la funcionalidad administrativa del motor de datos.
2. *Decision Support Objects (DSO)* API que brinda un conjunto de objetos COM que encapsulan las funcionalidades de los SQL Server 2000 Analysis Services.
3. *Windows Management Instrumentation (WMI)*, es una API orientada a objetos que permite administrar aplicaciones *scripts* para monitorear, configurar y controlar los servicios, recursos y aplicaciones de Windows. SQL Server ofrece una API que devuelve la información del motor de datos y de todas sus instancias, esta API se denomina SQL Server 2000 WMI.

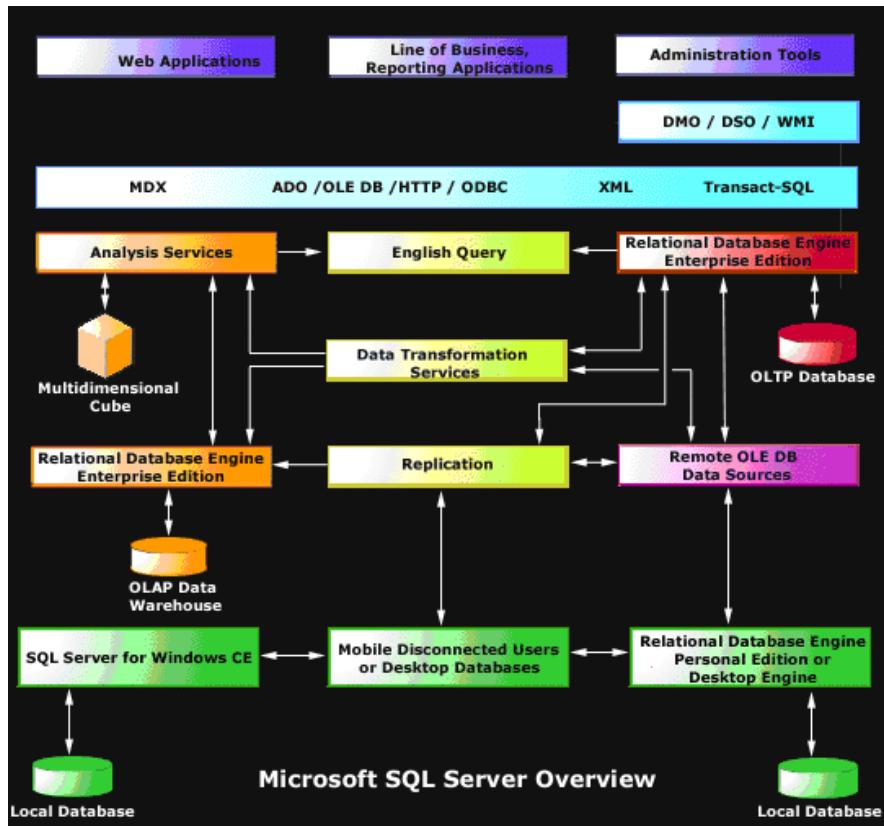
Entre los componentes adicionales de SQL Server 2000, podemos mencionar:

- *Data Transformation Services*, permite recuperar información de un origen de datos, realizar transformaciones sencillas o complejas (como totalización de datos) y almacenarlos en otro origen de datos, como una base de datos SQL o un cubo multidimensional.
- Replicación, se puede distribuir la información a través de un mecanismo de replicación con la finalidad de optimizar el rendimiento o de mantener autonomía, mientras una copia de la información almacenada en diferentes computadoras mantengan sincronización.
- *English Query*, provee de un sistema que permite a los usuarios plantear una pregunta en lenguaje natural en lugar de emplear un formato Transact-SQL. Por ejemplo: “List all customers”, “How many blue dress were sold in 2001?”, etc.

Meta Data Services, son un conjunto de servicios que permiten almacenar información acerca de las bases de datos y aplicaciones clientes que las emplean, esta información es aprovechada cuando se requiere intercambiar con otras aplicaciones. Los Meta Data Services proveen tres estándares: *Meta Data Coalition Open Information Model (MDC OIM)*, Interfaces COM y *XML Encoding*.

Además de ello cuenta con la documentación apropiada para poder obtener información detallada de cada uno de los tópicos de SQL Server.

Componentes de SQL Server 2000



Ediciones e Instalación de SQL Server

Ediciones SQL Server 2000

SQL Server 2000 está disponible en seis diferentes versiones además en cualquier edición se incluye el SQL Server 2000 *Desktop Engine*:

- **Enterprise**, soporta todas las características de SQL Server 2000. Esta edición es para empresas que implementan medianas y grandes bases de datos, las cuales brindan recursos a soluciones web, organizaciones con un alto índice de trabajo transaccional, y soporte para *data warehouse*.
- **Estándar**, ideal para aplicaciones que necesiten brindar información a grupos de trabajos o departamentos dentro de una organización.
Entre las características más saltantes que no se encuentran disponibles para el motor relacional, podemos mencionar:

Clustering
Log Shipping
Vistas indexadas

Entre las características más destacadas que no se encuentran disponibles para los servicios de análisis:

Definición de cubos particionados
Cubos OLAP enlazados
Soporte para dimensiones ROLAP
Celdas calculadas

Personal, soporta todas las características del SQL Server 2000 *Standard Edition*, excepto la replicación transaccional, para lo cual sólo puede ser definido como un suscriptor, además de esto tampoco se encuentra disponible el full text search cuando se instala sobre Windows Me y Windows 98.

Esta edición puede ser empleada para aplicaciones *standalone* y usuarios móviles que requieran un almacenamiento local de información.

- **Windows CE Edition**, es empleado para almacenar información en dispositivos Windows CE. SQL Server 2000 CE es implementado como un conjunto de librerías (DLLs) que operan como un OLE DB CE Provider. Esta implementación permite que SQL Server 2000 CE soportar *ActiveX Data Objects for Windows CE (ADOCE)* y OLE DB CE APIs en Windows CE versiones disponibles para Visual Basic y Visual C++. Además también es posible que múltiples aplicaciones puedan compartir al mismo tiempo un conjunto de DLLs.

Los dispositivos Windows CE pueden conectarse a la red empleando Remote Data Access (RDA) característica de SQL Server CE para:

Conectarse a instancias de SQL Server de diferentes plataformas
Ejecutar sentencias SQL y colocarlas en un *recordset*
Modificar la información de un *recordset* y enviarlas a una instancia de SQL Server inclusive de diferentes plataformas.
Ser suscriptor en una replicación de tipo *merge*.

- **Developer Edition**, soporta todas las características de SQL Server 2000, además de un conjunto de herramientas gráficas para la configuración de idiomas, esta es una edición sólo para desarrolladores que emplean SQL Server como su origen de datos. Esta edición sólo está licenciada para desarrollo y prueba de los sistemas.
- **Enterprise Evaluation Edition**, soporta todas las características de SQL Server 2000, a excepción de las herramientas gráficas para configuración del lenguaje. Esta edición es libre y se puede descargar desde el Web aunque sólo podrá ejecutarla por 120 días.
- **SQL Server 2000 Desktop Engine**, es una versión distribuible del motor de base de datos relacional de SQL Server 2000. Esta edición es empleada para aquellas aplicaciones que no requieran la implementación de tareas administrativas para el cliente. Debe recordar que las bases de datos no deben exceder los 2 Gb. de tamaño.

Instalación de SQL Server

Antes de instalar SQL Server 2000 es necesario conocer cuales son los requisitos mínimos para instalar este producto, el siguiente cuadro muestra los requerimientos para instalar SQL Server de acuerdo a la edición que Ud. emplee:

Recurso	Requerimiento
Computador	Intel o compatible
Procesador	Pentium 166
Monitor	800*600
Dispositivo puntero	Mouse
Tarjeta de red	Opcional (requerido para acceso a los recursos de la red)
CD-ROM	Requerido para la instalación

Para determinar correctamente el requerimiento de memoria, emplear la siguiente tabla:

	Enterprise	Estándar	Evaluation	Developer	Personal y Desktop Engine
Alguna edición de Windows 2000 Server	256 MB (128 MB soportado)	256 MB (128 MB soportado)	256 MB (128 MB soportado)	256 MB (128 MB soportado)	256 MB (128 MB soportado)
Alguna edición de Windows NT 4.0 Server con SP5 o posterior	128 MB (64 MB soportado)	64 MB	128 MB recomendado (64 MB soportado)	64 MB	32 MB
Windows 2000 Professional	N/A	N/A	128 MB recomendado (64 MB soportado)	64 MB	64 MB
Windows NT 4.0 Workstation, con SP5 o posterior	N/A	N/A	128 MB recomendado (64 MB soportado)	64 MB	32 MB
Windows ME	N/A	N/A	N/A	N/A	32 MB
Windows 98	N/A	N/A	N/A	N/A	32 MB

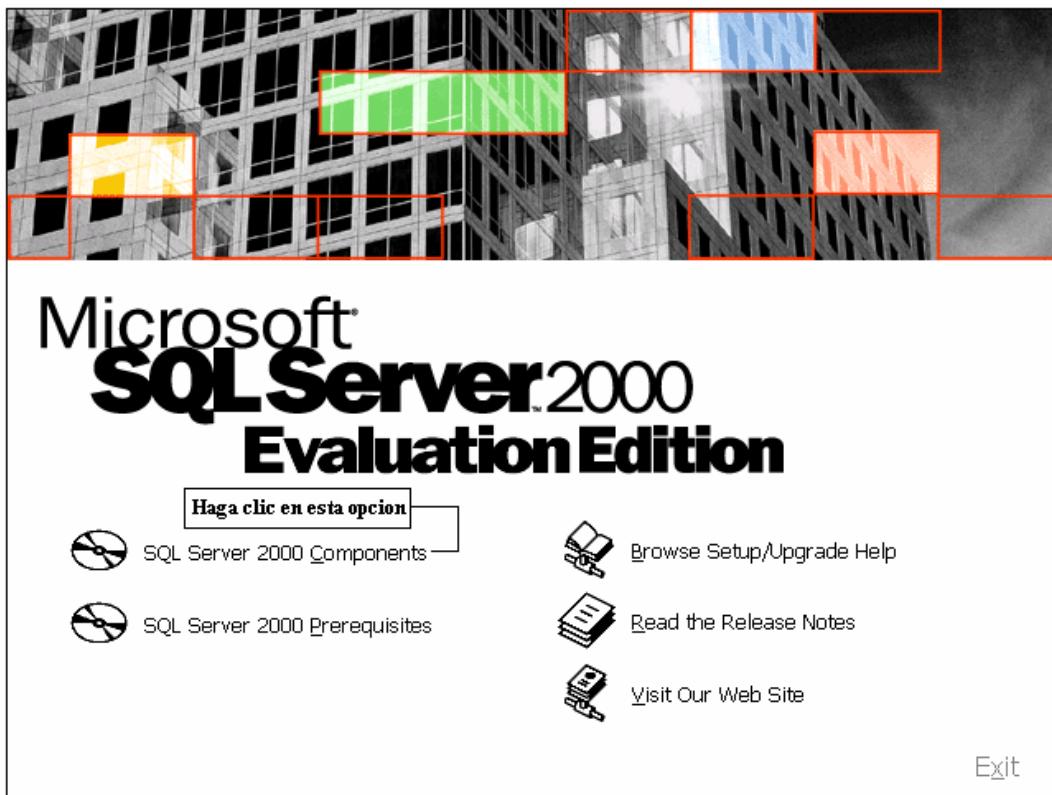
Como software tener en cuenta que para instalar SQL Server 2000 se requiere de Internet Explorer 5.0 o posterior, si desea instalar SQL Server 2000 sobre Windows NT en cualquiera de sus ediciones debe instalar previamente el Service Pack 5.0 o posterior.

Asimismo tenga en cuenta la siguiente tabla, para poder determinar el espacio en disco requerido para su instalación:

Opción de Instalación seleccionada	Espacio en disco requerido
Server y client tools	95-270 MB dependiendo de las opciones seleccionadas
Instalación Typical	250 MB (178 MB para el sistema, más 72 MB para programas y archivos de datos)
Instalación mínima	110 MB (73 MB para el sistema, más 37 MB para programas y archivos de datos)
Herramientas administrativas	113 MB (sistema solamente)
BoAceptars Online	30 MB (sistema solamente)
Analysis Services	47 MB mínimo 120 MB typical
English Query	80 MB
Sólo Desktop Engine	44 MB

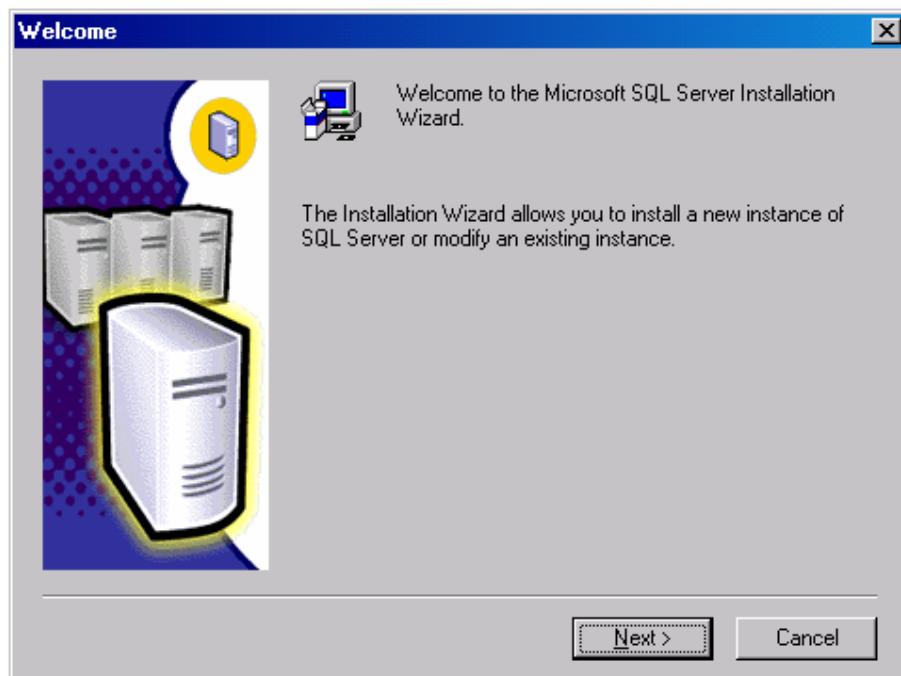
Pasos para la instalación sobre Windows NT Server

Coloque el CD de instalación
Aparecerá la siguiente pantalla

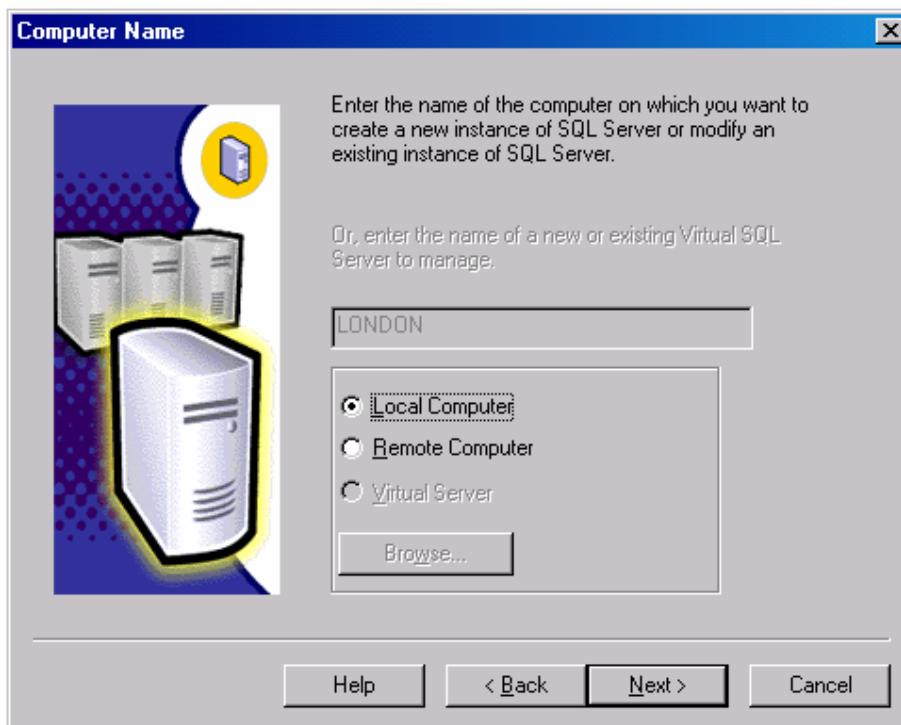


Después se presentará la siguiente pantalla:

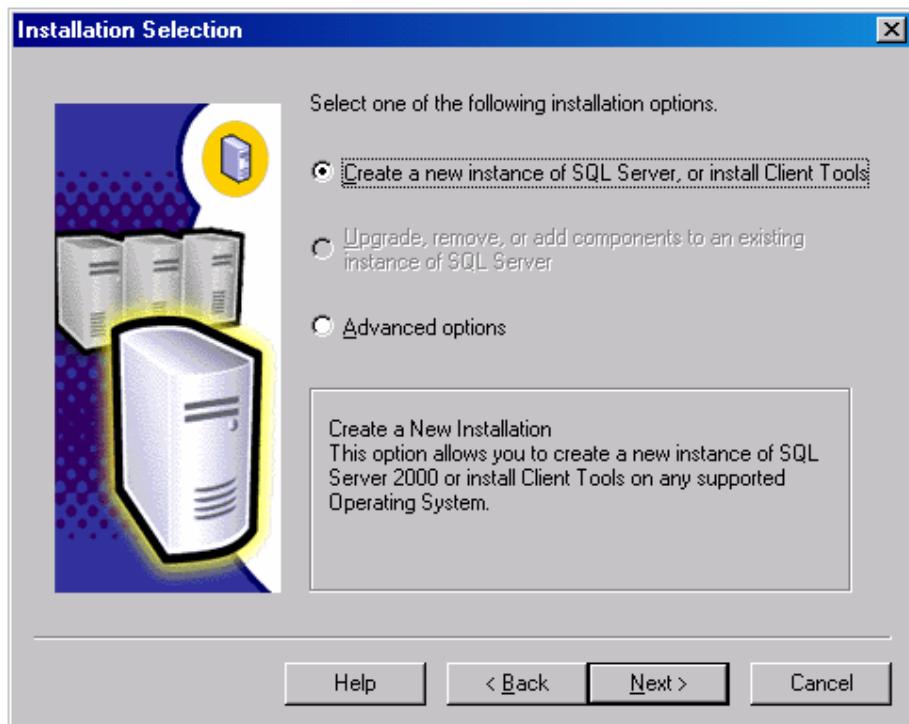
A continuación aparecerá una ventana que da la bienvenida al proceso de instalación, pulse Siguiente (*Siguiente*) en la siguiente pantalla:



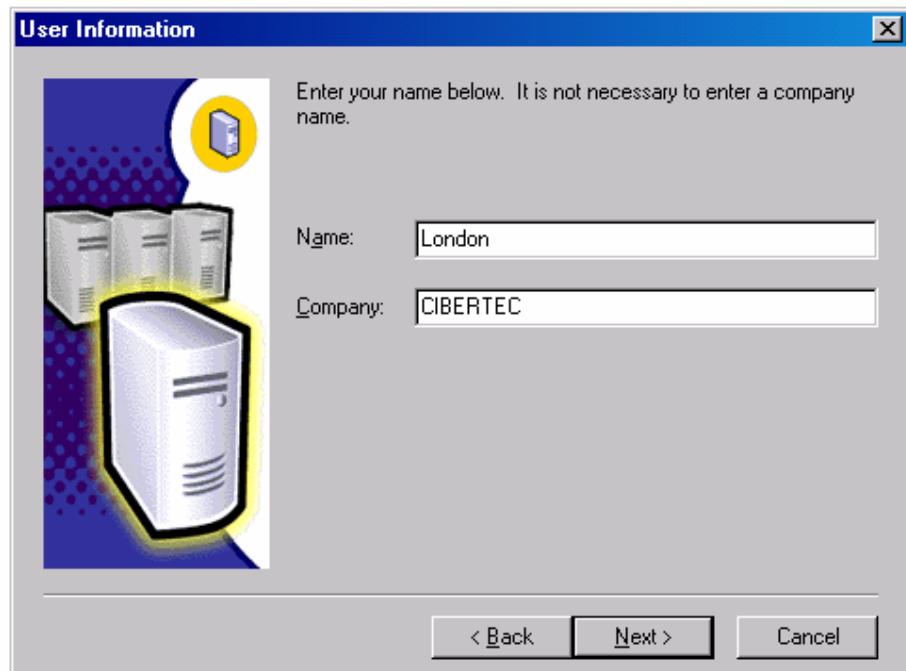
1. A continuación aparece una pantalla que le solicitará elegir entre una instalación local o una instalación remota, pulse Siguiente (*Siguiente*) en la siguiente pantalla:



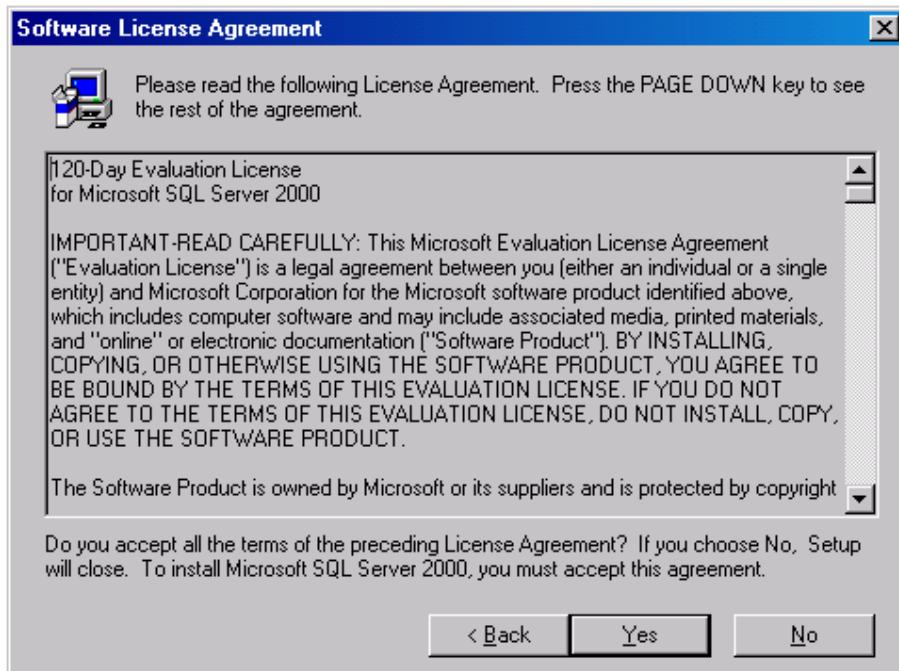
Si es la primera vez que instala SQL Server 2000 aparecerá la siguiente pantalla:



2. Ingrese la información del usuario y pulse Siguiente (*Siguiente*).



Acepte las condiciones del licenciamiento:

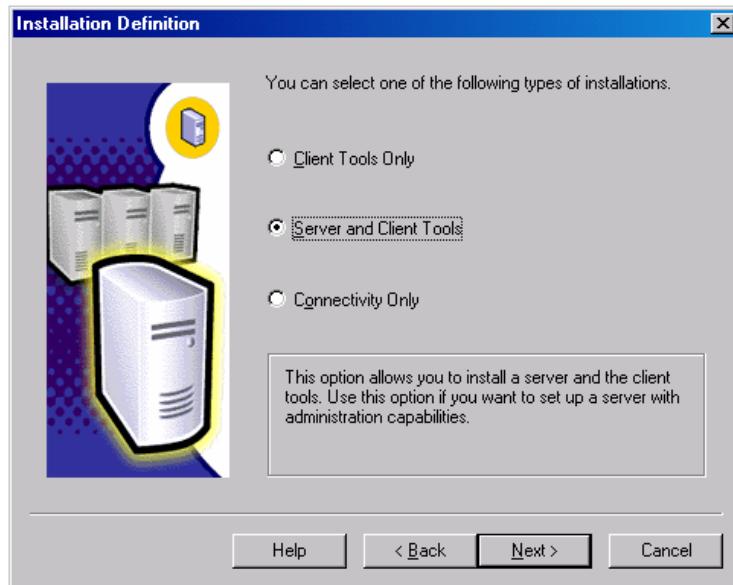


Luego de aceptar las condiciones del licenciamiento aparecerá una caja de diálogo solicitándole que seleccione uno de los tipos de instalación, para ello tendrá las siguientes opciones:

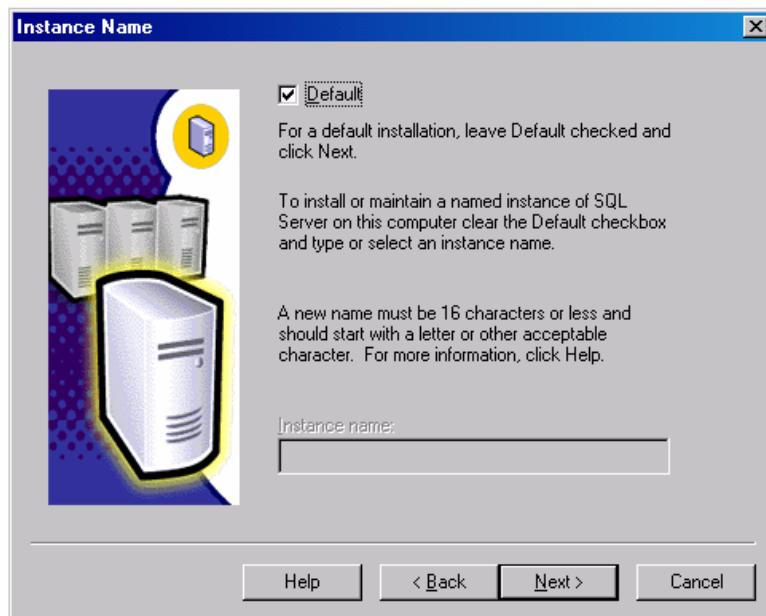
Sólo Herramientas Cliente (*Client Tools only*), cuando requiera instalar herramientas clientes para administrar un servidor SQL Server existente, así como también los componentes de conectividad los libros en línea y opcionalmente los ejemplos.
Servidor y Herramientas Cliente (*Server and Client Tools*), selecciona esta opción cuando requieras instalar un servidor SQL Server 2000, el cual deba contar con todas las herramientas.

Sólo Conectividad (*Connectivity Only*), seleccione esta opción para instalar las librerías de conectividad para los clientes.

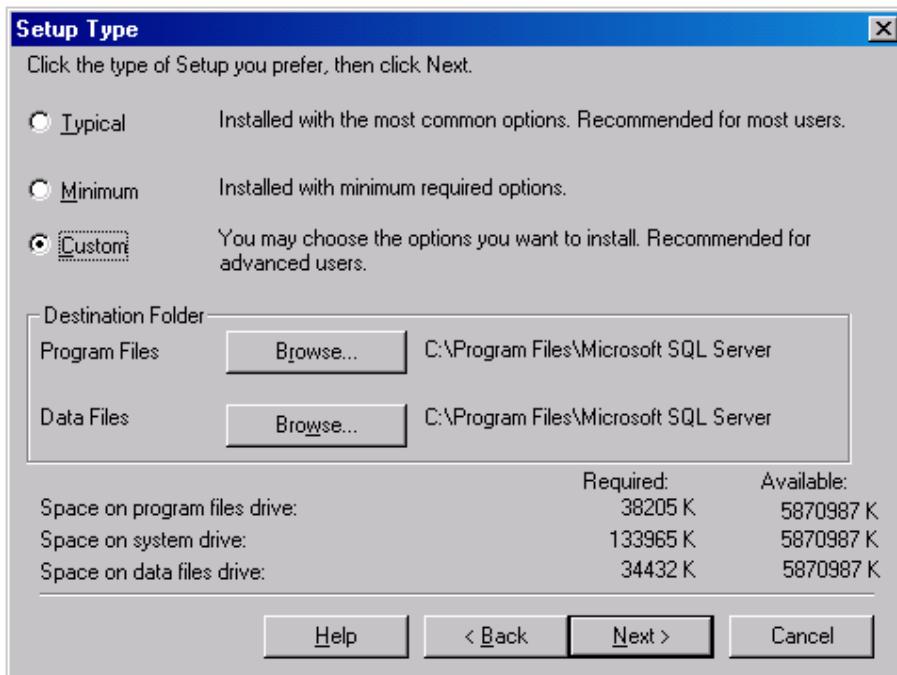
Para cualquiera de las tres opciones se instalará previamente MDAC 2.6, para la instalación que estamos realizando seleccione Servidor y Herramientas Cliente (*Server and Client Tools*) y luego pulse Siguiente (*Siguiente*):



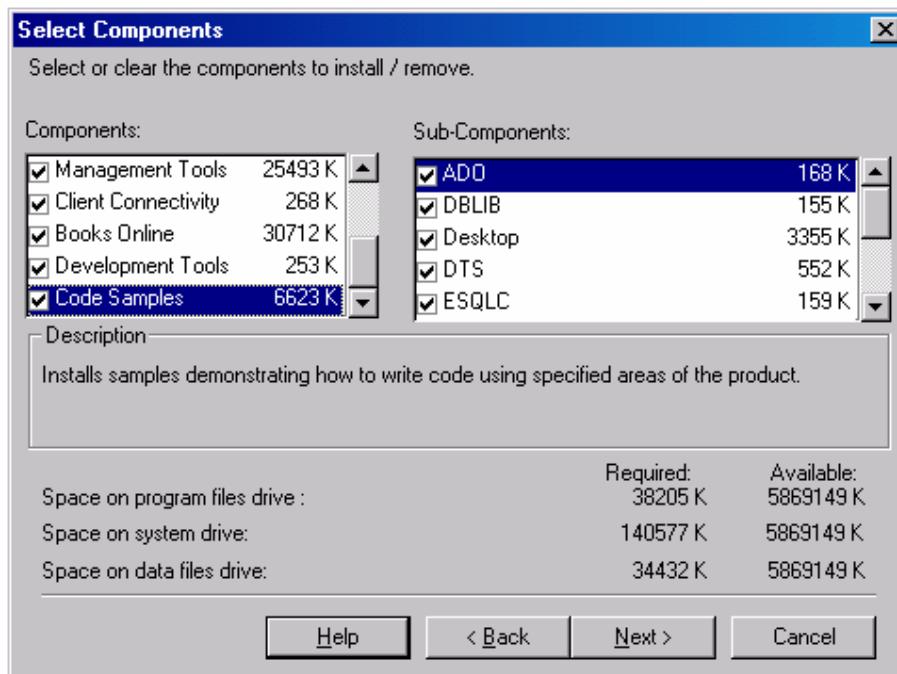
A continuación aparecerá una caja de diálogo donde especificará el nombre de la instancia que está instalando, si es la primera vez En forma predeterminada tomará el nombre del equipo donde se encuentra instalando:



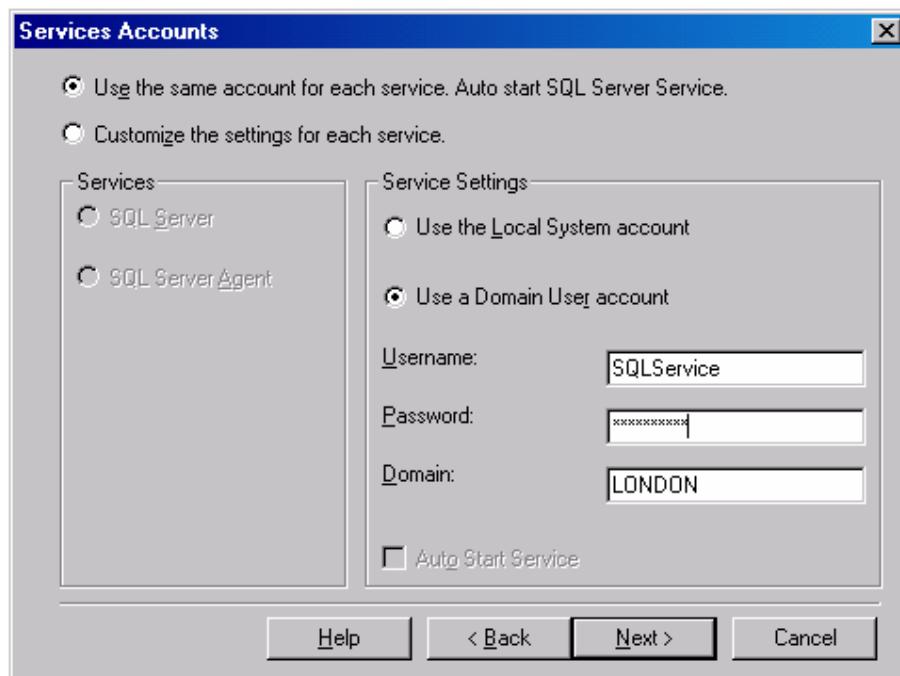
3. Luego de pulsar Siguiente (*Siguiente*), tendrá la posibilidad de seleccionar el tipo de instalación a ejecutar, seleccione Personalizada (*Custom*) para que pueda observar las diferentes opciones que configura el instalador, en esta primera pantalla se muestran los espacios requeridos así como también las carpetas donde se almacenaran las diferentes librerías de SQL Server:



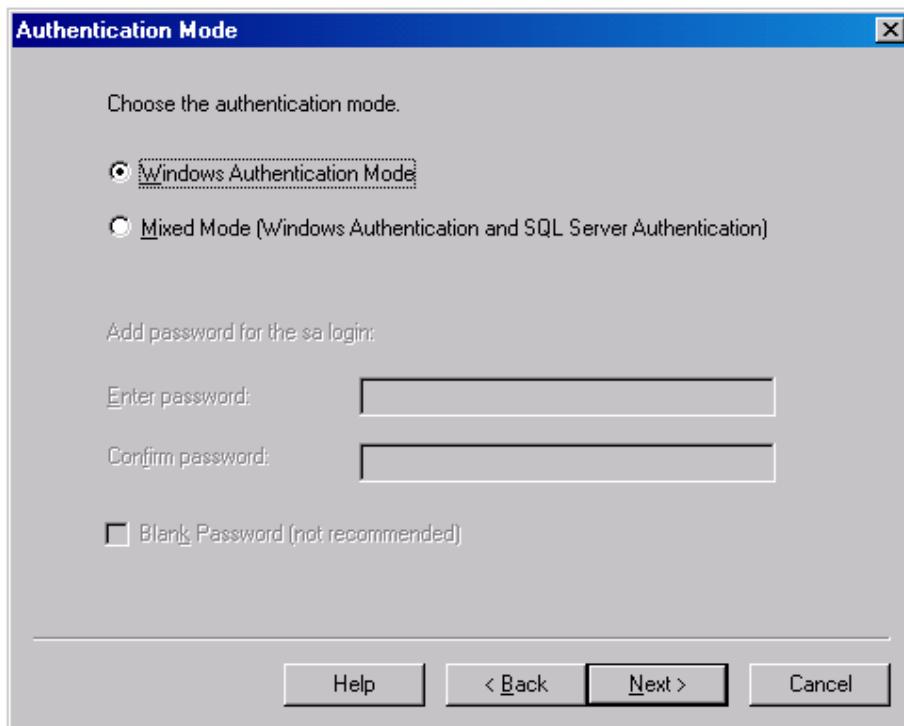
4. Luego de pulsar Siguiente (*Siguiente*) aparecerá una lista que le permitirá seleccionar los componentes a instalar, desplazar la lista Componentes (*Components*) y activar las casillas Ejemplos de Código (*Code Samples*):



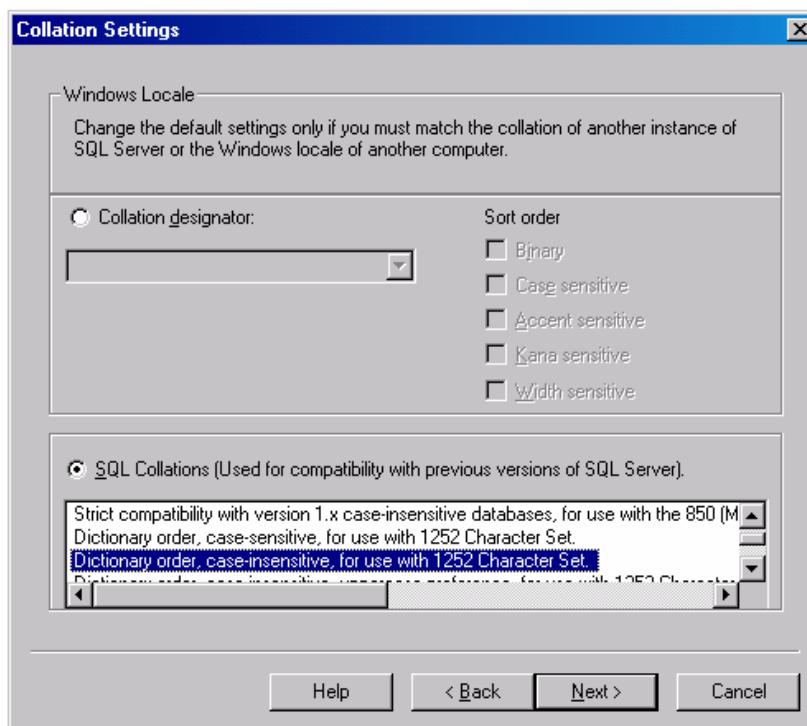
5. Inmediatamente se le solicitará una cuenta para los servicios, si se encuentra trabajando en un entorno de red, asigne una cuenta de un usuario que pertenezca al grupo Administradores (*Administrators*) del Dominio:



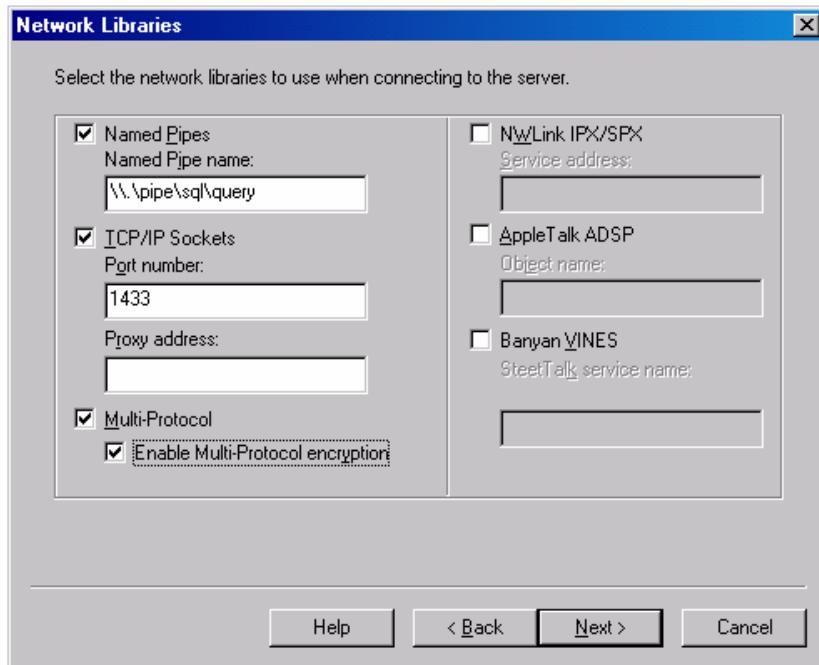
Seleccione el modo de autentificación para acceder a SQL Server 2000:



A continuación podrá determinar el conjunto de caracteres con los cuales trabajará, asimismo podrá determinar si las consultas distinguirán o no las mayúsculas de las minúsculas



Activar las librerías de red de acuerdo a los usuarios que tendrá su origen de datos:



6. Luego de pulsar Siguiente (*Siguiente*) aparecerá una pantalla indicándole que se ha completado el trabajo de recolección de información, pulse Siguiente (*Siguiente*) para iniciar el copiado de archivos:



7. Al completar la instalación se muestra la siguiente pantalla, pulse Finalizar (*Finish*) para finalizar:

Setup Complete



Setup has finished installing an instance of Microsoft SQL Server 2000 on your computer.

Click Finish to complete Setup.

Finish

Verificar la instalación de SQL Server

Una vez finalizada la instalación debe revisar la instalación para cerciorarse que el producto se ha instalado correctamente para ello puede mostrar el Administrador de Servicios (*Service Manager*) que le permitirá mostrar el estado de los servicios, este utilitario tiene el siguiente aspecto:



Seguidamente observará una caja de diálogo con el siguiente aspecto:



Otra de las formas de verificar el estado de la instalación es haciendo pruebas con las sentencias a nivel del símbolo del sistema que ofrece SQL Server como es el caso del utilitario OSQL, para comprobar su funcionamiento abra una ventana del sistema y digite el siguiente comando:

```
C:\>osql -S<Susuariovisor> -U<usuario> -P<contraseña> -q "Select  
CategoryName From Northwind..Categories" <pulse Enter>  
Reemplace el texto en negrita por los valores adecuados.
```

El resultado será:

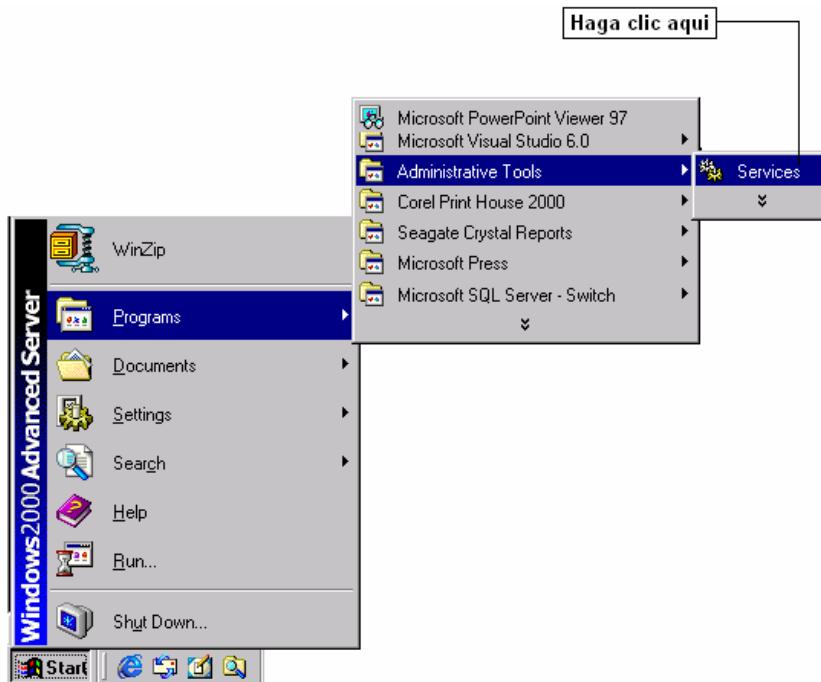
CategoryName

Beverages
Condiments
Confections
Dairy Products
Grains/Cereals
Meat/Poultry
Produce
Seafood

(8 rows affected)

1> quit <**pulse Enter**>

Otra manera de poder verificar la instalación de SQL Server es revisar los servicios que se cargan, para ello presione el botón del menú Inicio (Start), seleccione Programas (Programs), Herramientas Administrativas (Administrative Tools) y haga clic en Servicios (Services):



Compruebe que los siguientes servicios se encuentren iniciados:

MSSQL Server Este servicio es el motor de base de datos, este es el componente que procesa todas las sentencias del Transact-SQL y administra todos los archivos que comprometen las bases de datos del servidor, entre sus principales funciones podemos mencionar:

- La asignación de recursos del servidor entre múltiples usuarios concurrentes.
- Previene los problemas lógicos, como por ejemplo prevenir que los usuarios modifiquen la misma información al mismo tiempo.
- Asegura la consistencia e integridad de datos.

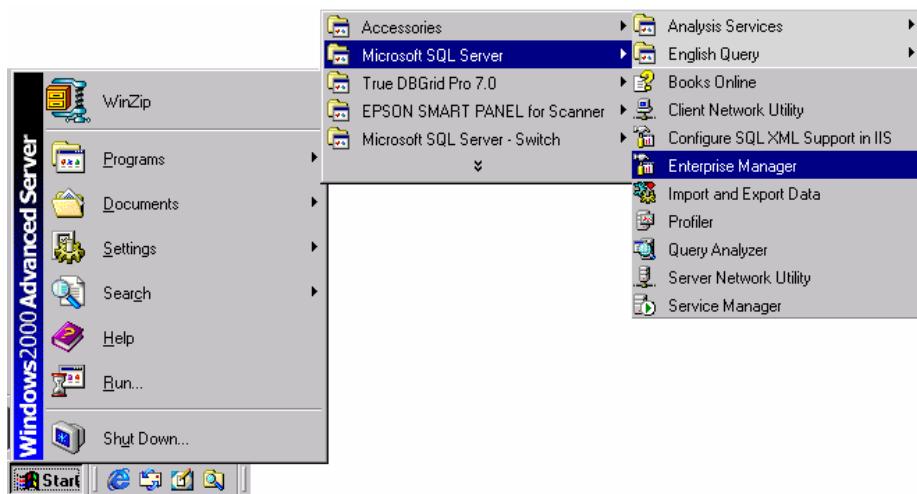
SQL Server Agent Este servicio trabaja junto al MSSQL Server para crear y administrar Alertas, Tareas (locales o multiserver) y Operadores. Entre sus principales funciones podemos mencionar:

- Las alertas proveen información acerca del estado de un proceso, como por ejemplo indicar cuando finalizo una tarea con éxito o fracaso.
- Este servicio incluye un motor que permite crear tareas y programarlos para que se ejecuten automáticamente.
- Puede enviar correos electrónicos, puede indicar la ejecución de una tarea cuando una alerta ocurre.

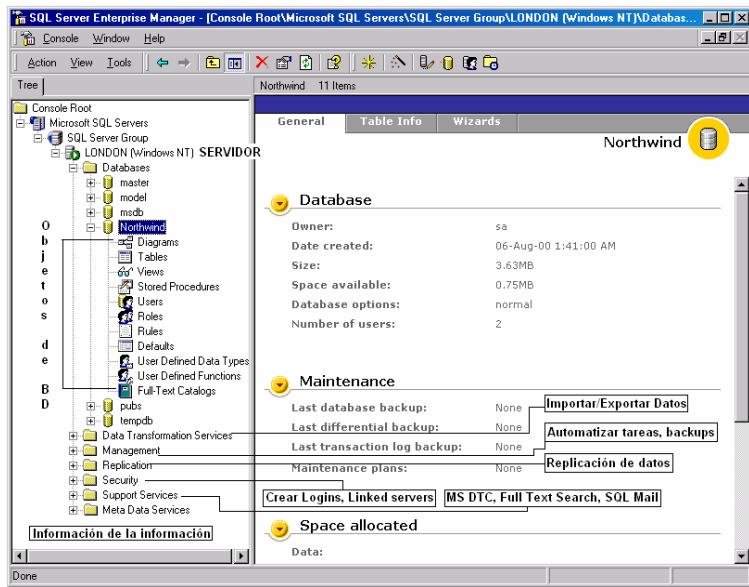
MS DTC Permite incluir múltiples orígenes de datos en una transacción, se encarga de coordinar y asegurar que las actualizaciones sobre todos los servidores sean permanentes, y si en caso estos cambios causaran un error deshacer todos.

Microsoft Search Este es un servicio opcional y se encarga de realizar búsquedas sobre información tipo carácter creando índices para facilitar estas consultas.

Además de ello podrá ingresar a la consola de administración de SQL Server denominada Administrador Corporativo (Administrador Empresarial), para ello siga la siguiente secuencia:



A continuación tendrá la interfaz del Administrador Corporativo (Administrador Empresarial), tal como lo muestra la siguiente representación:



Modos de autenticar las cuentas de los usuarios

SQL Server valida a los usuarios en dos niveles de seguridad: una a través de un Inicio de sesión que establece el hecho de realizar la conexión a SQL Server y otro a partir de la validación de los permisos que tienen los usuarios sobre una base de datos.

INICIO DE SESIÓN

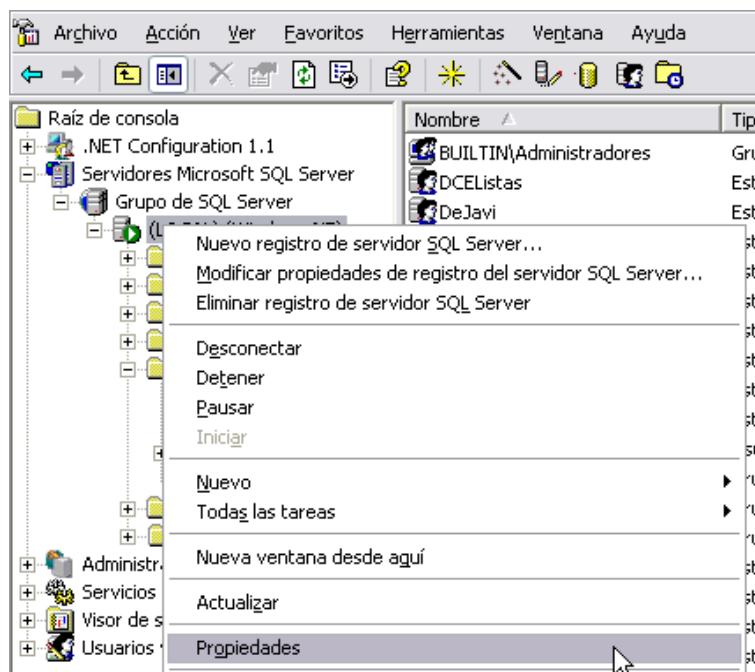
Todos los usuarios deben tener un Inicio de sesión para poder conectarse a SQL Server, para esto SQL Server reconoce 2 mecanismos de autenticación:

SQL Server es cuando el usuario debe proveer de un usuario y una contraseña que serán validados por el propio SQL Server cuando el cliente intente conectarse.

Windows NT es cuando una cuenta o grupo de Windows NT controla el acceso a SQL Server, el cliente no provee usuario y contraseña, ya que se empleará la cuenta con la que se ingresa al sistema operativo.

Para modificar la autenticación realice los siguientes pasos:

- 1 Haga clic derecho sobre el servidor, en el menú contextual haga clic sobre la opción Properties.



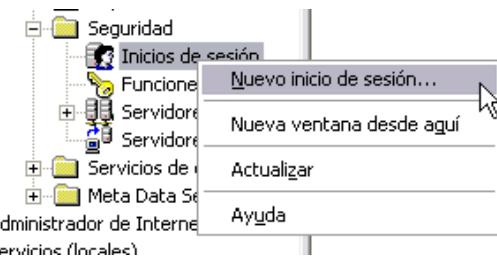
- 2 En la caja de diálogo haga clic sobre la ficha **Seguridad**, se presentará la siguiente pantalla:



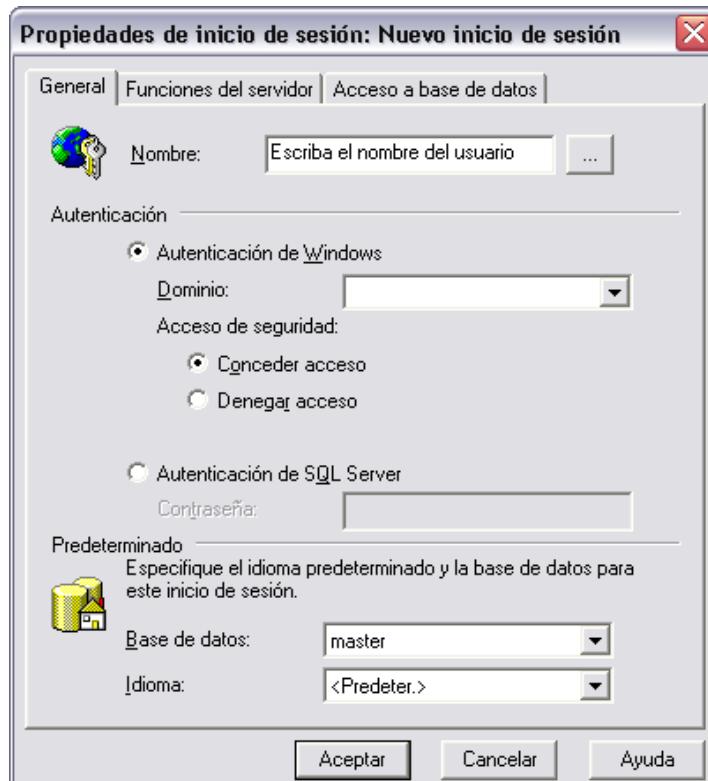
Seleccione la opción “SQL Server y Windows” cuando desee brindar servicios de información a terceros por ejemplo a usuarios de internet. Seleccione “Sólo Windows” cuando los datos estarán disponibles sólo a los empleados de la organización. En cualquiera de los dos casos debe pulsar Aceptar, espere por un instante mientras SQL Server 2000 detiene los servicios y los vuelve a iniciar para hacer efectivos los cambios.

Hecho esto Ud. podrá definir sus Inicios de sesión de acceso a SQL Server, para ello realice la siguiente secuencia desde el Administrador Empresarial:

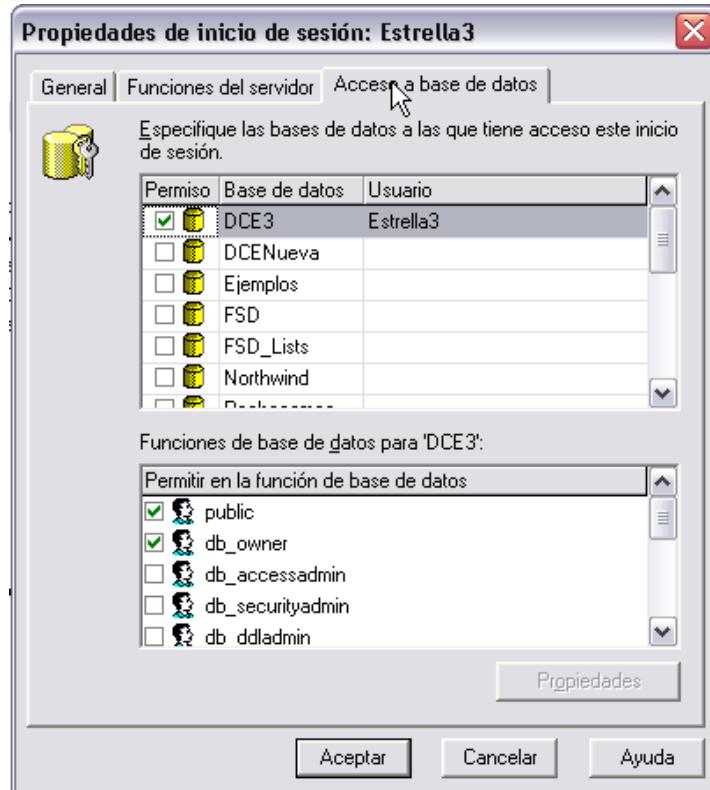
Expanda la carpeta Seguridad del Administrador Empresarial y haga clic derecho sobre Inicios de sesión



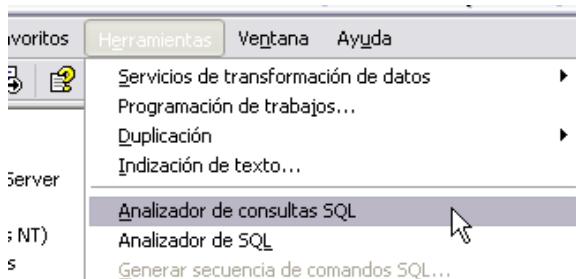
Aparecerá la siguiente caja de diálogo:



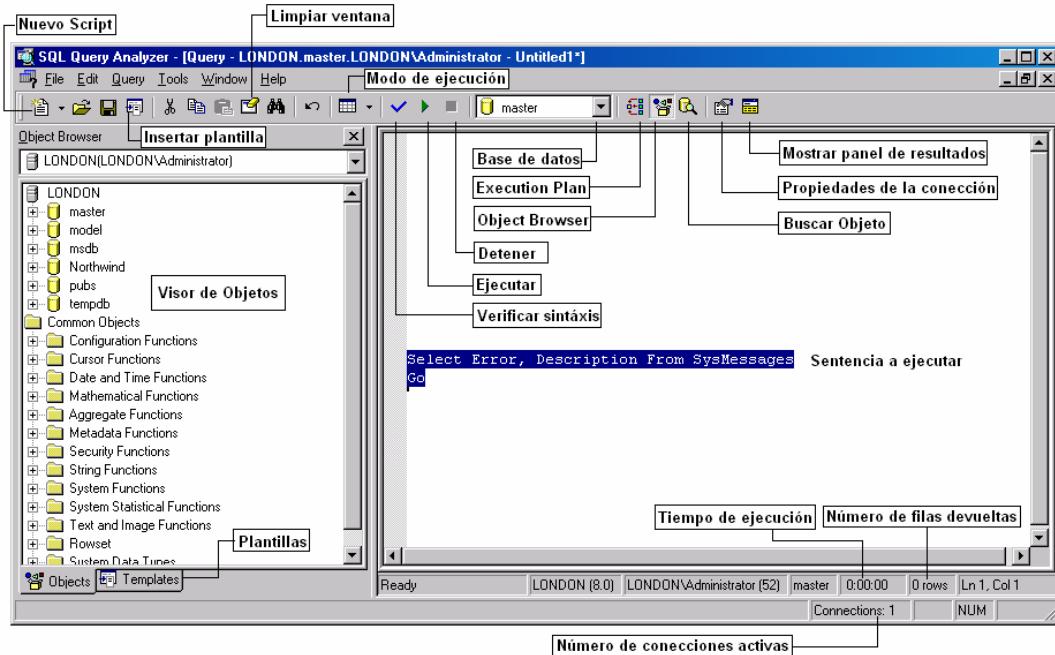
En la ficha Acceso a base de datos podrá especificar que el Inicio de sesión se definirá como usuario de alguna de las bases de datos existentes. Pulse Aceptar al finalizar.



La creación de Inicios de sesión también es posible desde el Analizador de Consultas, que es una herramienta a la cual accedemos a partir de la siguiente secuencia:



Observará el siguiente entorno:



Ahora que conocemos el entorno podemos digitar las siguientes sentencias para poder crear un nuevo Inicio de sesión:

```
/* Activar Base de datos */
Use Master
GO
/* Crear nuevos login */
Sp_Addlogin 'mhidalgo', 'mhidalgo'
GO
Sp_Addlogin 'Usuario01', 'contraseña'
GO
/* Comprobar la creación del nuevo login */
Select Name From Syslogins
GO
```

NOTA: Se pueden colocar comentarios empleando (--) al final de una sentencia, pero si desea tener un grupo de filas comentadas emplee los delimitadores (*.... *)

Usuarios de Base de Datos

Una de las tareas comunes al administrar SQL Server es permitir el acceso a bases de datos y la asignación de permisos o restricciones sobre los objetos que conforman una base de datos.

SQL Server 2000 permite trabajar a nivel de **Roles** y **Usuarios**.

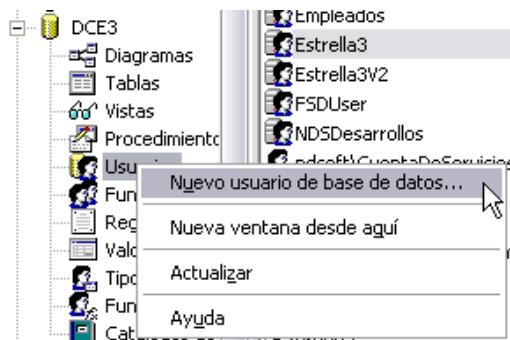
Un **rol** es un conjunto de derechos asignados, los cuales se convierten en una gran alternativa para agrupar un conjunto de permisos, de tal forma que cuando se incorpore un nuevo usuario a la base de datos, ya no se le tiene que dar permiso por permiso por cada uno de los objetos que requiera emplear, sino mas bien su cuenta de usuario es agregada al rol, y si al rol tiene que asignársele acceso sobre un nuevo elemento automáticamente el permiso o la restricción afectará a los usuarios que pertenezcan a un rol.

Los **usuarios** representan los usuarios que tienen acceso a la base de datos y están mapeados a un Inicio de sesión, aunque pueden tener diferente identificador, por ejemplo el Inicio de sesión puede tener como nombre Jcabrera pero al definir un Usuario podemos usar Jorge.

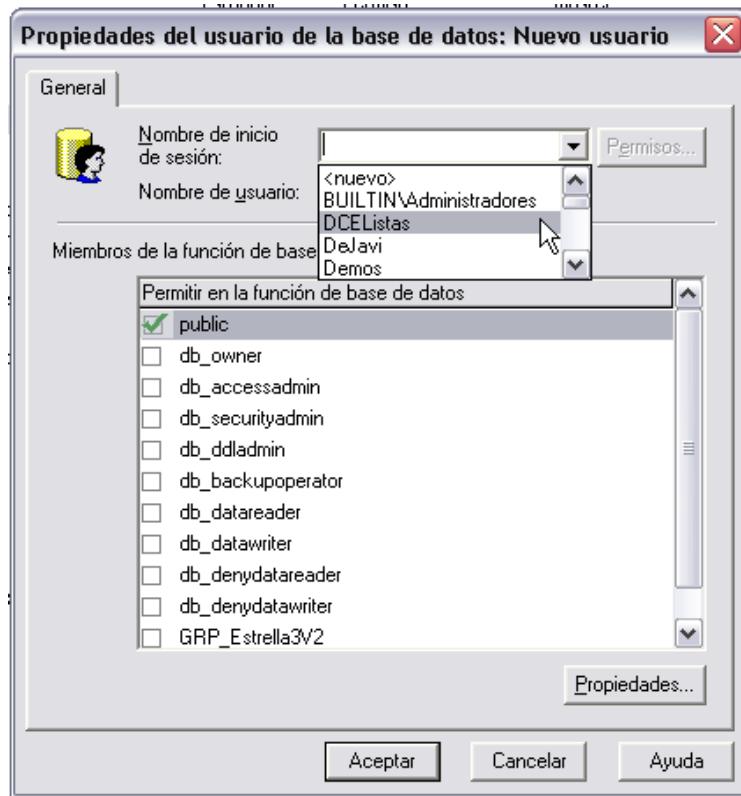
Después de que se crearon los Inicios de sesión para conectarse a SQL Server, se deben definir los accesos a las bases de datos requeridas, para ello es necesario definir Usuarios en cada BD, estos usuarios permitirán controlar el acceso a los distintos objetos incluyendo los datos que estos contienen.

Para ello realice el siguiente proceso:

Expanda la base de datos donde desea definir al nuevo usuario y haga clic derecho sobre la carpeta Usuarios



Seleccione un Inicio de sesión de la lista y pulse Aceptar.



También es posible realizar esta tarea desde el Analizador de Consultas para ello emplee la siguiente secuencia de instrucciones:

```
Use Northwind
GO
Sp_GrantDBAccess 'Usuario01'
GO
```

Además de los Inicios de sesión y usuarios SQL Server brinda un conjunto de roles por servidor y por base de datos que son derechos predefinidos que podrán especificarse por cada usuario de ser necesario. También es posible crear roles personalizados.

Los roles son los siguientes:

Roles por Servidor	
Rol	Descripción
Dbo	Crea y modifica bases de datos.
Diskadmin	Administra los archivos de datos.
Processadmin	Administra los procesos de SQL Server.
SecurityAdmin	Administra los Inicios de sesión.
Serveradmin	Opciones de configuración del servidor.
Setupadmin	Instala la replicación.
Sysadmin	Realiza cualquier actividad.

Roles por Base de Datos	
Rol	Descripción
public	Mantiene los permisos En forma predeterminada para todos los usuarios.
db_owner	Realiza cualquier actividad en la BD
db_accessadmin	Agrega o retira usuarios y/o roles
db_ddladmin	Agrega, modifica o elimina objetos
db_SecurityAdmin	Asigna permisos sobre objetos o sobre sentencias
db_backupoperator	Backup y Restore de la base de datos
db_datareader	Lee información desde cualquier tabla
db_datawriter	Agrega, modifica o elimina datos
db_denydatareader	No puede leer la información
db_denydatawriter	No puede modificar la información

Bases de Datos de SQL Server

SQL Server soporta bases de datos del sistema y bases de datos del usuario.

Las bases de datos del sistema, almacenan información que permite operar y administrar el sistema, mientras que las de usuario almacenan los datos requeridos por las operaciones del cliente.

Las bases de datos del sistema son:

- **master**

La base de datos **master** se compone de las tablas de sistema que realizan el seguimiento de la instalación del servidor y de todas las bases de datos que se creen posteriormente. Asimismo controla las asignaciones de archivos, los parámetros de configuración que afectan al sistema, las cuentas de inicio de sesión. Esta base de datos es crítica para el sistema, así que es bueno tener siempre una copia de seguridad actualizada.

- **tempdb**

Es una base de datos temporal, fundamentalmente un espacio de trabajo, es diferente a las demás bases de datos, puesto que se regenera cada vez que arranca SQL Server. Se emplea para las tablas temporales creadas explícitamente por los usuarios, para las tablas de trabajo intermedias de SQL Server durante el procesamiento y la ordenación de las consultas.

- **model**

Se utiliza como plantilla para todas las bases de datos creadas en un sistema. Cuando se emite una instrucción CREATE DATABASE, la primera parte de la base de datos se crea copiando el contenido de la base de datos **model**, el resto de la nueva base de datos se llena con páginas vacías.

- **msdb**

Es empleada por el servicio SQL Server Agent para guardar información con respecto a tareas de automatización como por ejemplo copias de seguridad y tareas de duplicación, asimismo solución a problemas.

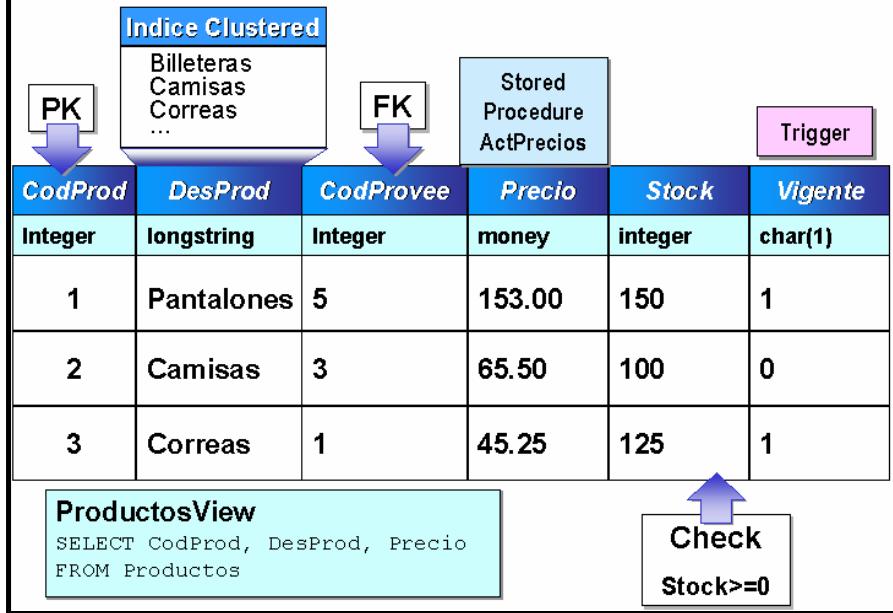
La información contenida en las tablas que contiene esta base de datos, es fácilmente accedida desde el Administrador Empresarial, así que se debe tener cuidado de modificar esta información directamente a menos que se conozca muy bien lo que se está haciendo.

- **Distribution**

Almacena toda la información referente a la distribución de datos basada en un proceso de replicación.

Objetos de una Base de Datos

Objetos de una base de datos



Las **Tablas** son objetos de la base de datos que contienen la información de los usuarios, estos datos están organizados en filas y columnas, similar al de una hoja de cálculo. Cada columna representa un dato aislado y en bruto que por sí solo no brinda información, por lo tanto estas columnas se deben agrupar y formar una fila para obtener conocimiento acerca del objeto tratado en la tabla. Por ejemplo, puede definir una tabla que contenga los datos de los productos ofertados por una tienda, cada producto estaría representado por una fila mientras que las columnas podrían identificar los detalles como el código del producto, la descripción, el precio, las unidades en stock, etc.

Una **Vista** es un objeto definido por una consulta. Similar a tabla, la vista muestra un conjunto de columnas y filas de datos con un nombre, sin embargo, en la vista no existen datos, estos son obtenidos desde las tablas subyacentes a la consulta. De esta forma si la información cambia en las tablas, estos cambios también serán observados desde la vista. Fundamental emplean para mostrar la información relevante para el usuario y ocultar la complejidad de las consultas.

Los **tipos de datos** especifican que tipo de valores son permitidos en cada una de las columnas que conforman la estructura de la fila. Por ejemplo, si desea almacenar precios de productos en una columna debería especificar que el tipo de datos sea **money**, si desea almacenar nombres debe escoger un tipo de dato que permita almacenar información de tipo carácter.

SQL Server nos ofrece un conjunto de tipos de datos predefinidos, pero también existe la posibilidad de definir **tipos de datos de usuario**.

Un **Procedimiento Almacenado** es una serie de instrucciones SQL precompiladas las cuales organizadas lógicamente permiten llevar a cabo una operación transaccional o de

control. Un Procedimiento almacenado siempre se ejecuta en el lado del Servidor y no en la máquina Cliente desde la cual se hace el requerimiento. Para ejecutarlos deben ser invocados explícitamente por los usuarios.

Un **Desencadenador** es un Procedimiento Almacenado especial el cual se invoca automáticamente ante una operación de Insert, Update o Delete sobre una tabla. Un Desencadenador puede consultar otras tablas y puede incluir complejas instrucciones SQL, se emplean para mantener la integridad referencial, preservando las relaciones definidas entre las tablas cuando se ingresa o borra registros de aquellas tablas.

Los **Valores Predeterminados** especifican el valor que SQL Server insertará en una columna cuando el usuario no ingresa un dato específico. Por ejemplo, si se desconoce el apellido materno de un empleado SQL Server podría incluir automáticamente la cadena NN para identificar este campo.

Las **Reglas** son objetos que especifican los valores aceptables que pueden ser ingresados dentro de una columna particular. Las Reglas son asociadas a una columna o a un tipo de dato definido por el usuario. Una columna o un Tipo de dato puede tener solamente una Regla asociada con el.

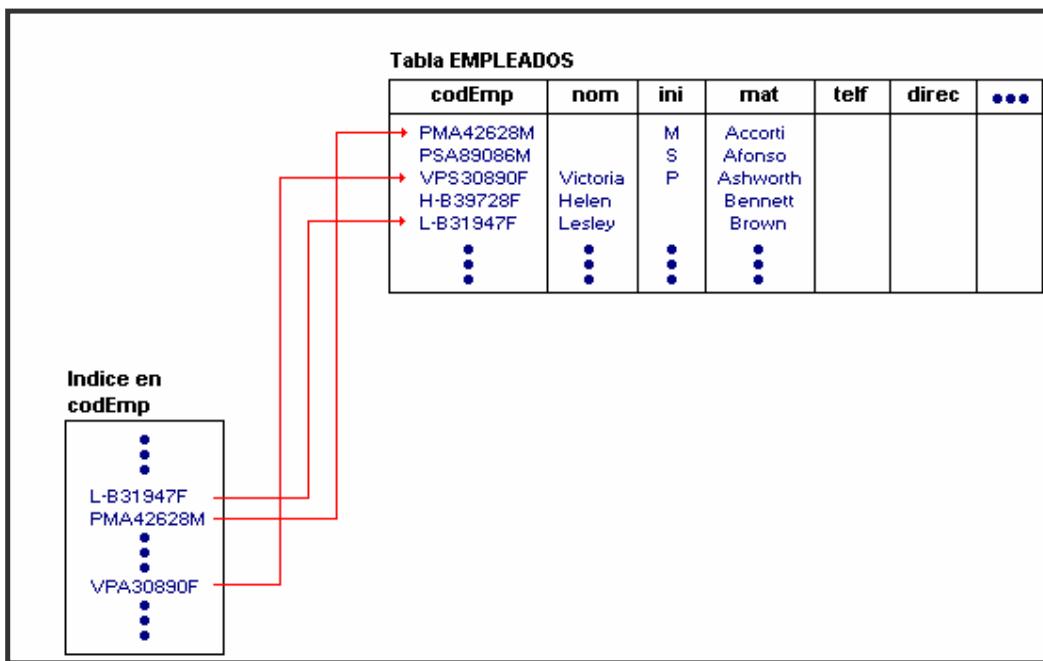
Las **Restricciones** son restricciones que se asignan a las columnas de una tabla y son controladas automáticamente por SQL Server.

Esto nos provee las siguientes ventajas:

- Se puede asociar múltiples constraints a una columna, y también se puede asociar un constraints a múltiples columnas.
- Se pueden crear los Restricciones al momento de crear la tabla CREATE TABLE. Los Restricciones conforman el standars ANSI para la creación y alteración de tablas, estos no son extensiones del Transact SQL.

Se puede usar un constraints para forzar la integridad referencial, el cual es el proceso de mantener relaciones definidas entre tablas cuando se ingresa o elimina registros en aquellas tablas.

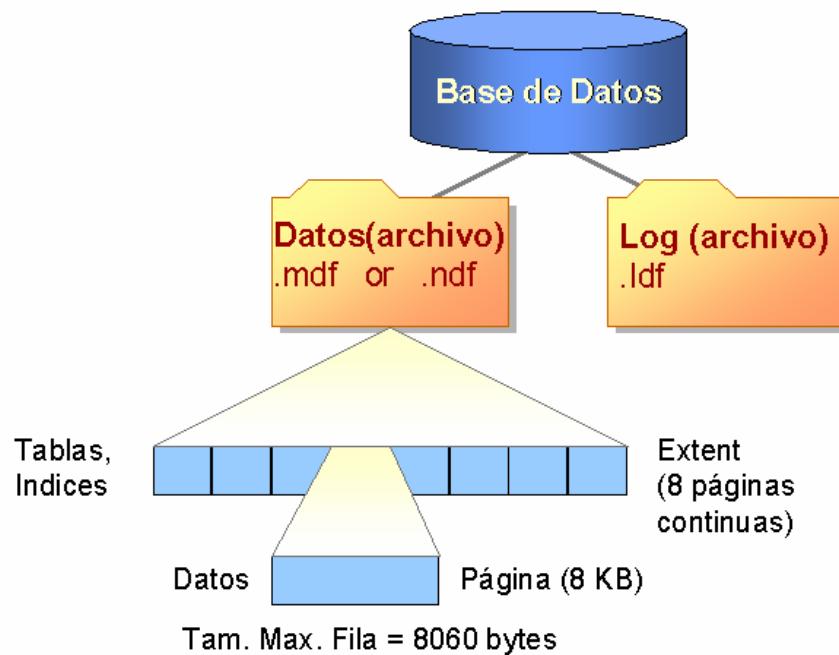
Los **índices** de SQL Server son similares a los índices de un libro que nos permiten llegar rápidamente a las páginas deseadas sin necesidad de pasar hoja por hoja, de forma similar los índices de una tabla nos permitirán buscar información rápidamente sin necesidad de recorrer registro por registro por toda la tabla. Un índice contiene valores y punteros a las filas donde estos valores se encuentran.



Creación de Base de Datos

En términos sencillos una base de datos de SQL Server es una colección de objetos que contiene y administra datos. Antes de crear una base de datos es importante entender como es que SQL Server almacena la información.

Como se almacena la data



Páginas y extensiones

Antes de crear una base de datos con SQL Server 2000, debemos tomar en cuenta que la unidad básica de almacenamiento en SQL Server es la página(data page), el tamaño de cada pade es de 8 KB, lo cual representa un total de 128 páginas por cada megabyte.

El comienzo de cada página es una cabecera de **96 bytes** que se utiliza para almacenar información de cabecera tal como el tipo de página, la cantidad de espacio libre de la página y el Id. del objeto propietario de la página.

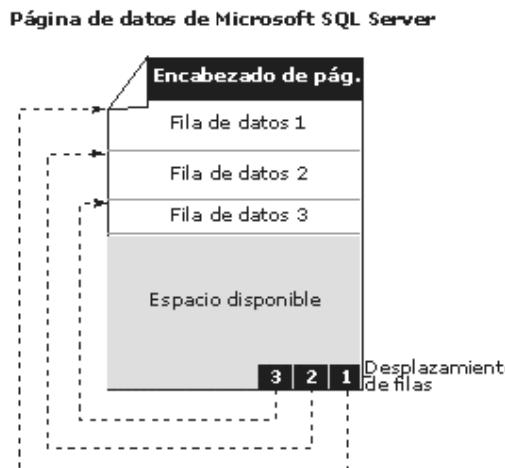
Existen ocho tipos de páginas en los archivos de datos de una base de datos SQL Server 2000.

Tipo de página	Contenido
Datos	Filas con todos los datos excepto los de tipo text , ntext e image .
Índice	Entradas de índices
Texto o imagen	Datos de tipo text , ntext e image .
Mapa de asignación global/ Mapa de asignación global Secundario	Información acerca de las extensiones asignadas.
Espacio libre en la página	Información acerca del espacio libre disponible en las páginas.
Mapa de asignación de índices.	Información acerca de las extensiones utilizadas por una tabla o un índice.

Bulk Changed Map	Información de los extends modificados por operación bulk desde el último backup del log.
Differential Changed Map	Información de los extends modificados desde el último full database backup.

Los archivos de registro (LOG) no contienen páginas, contienen series de registros.

Las páginas de datos contienen todos los datos de las filas de datos excepto los datos **text, ntext e image**, que están almacenados en páginas separadas. Las filas de datos se colocan en las páginas una a continuación de otra, empezando inmediatamente después de la cabecera, al final de cada página se encuentra una tabla de posiciones de filas que contiene una entrada por cada fila de la página y cada entrada registra la posición, desde el principio de la página, del primer byte de la fila. Las entradas de la tabla de posiciones de filas están en orden inverso a la secuencia de las filas de la página.

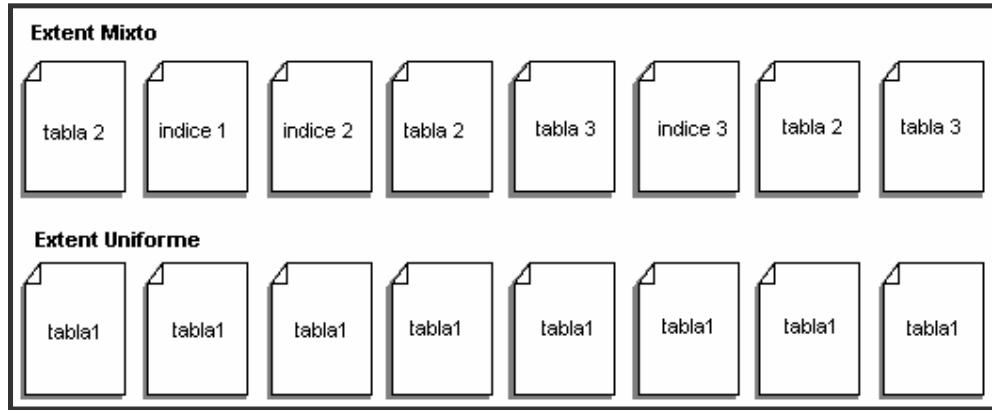


En SQL Server, las filas no pueden continuar en otras páginas.

Las extensiones son la unidad básica de asignación de espacio a las tablas e índices. Consta de 8 páginas contiguas, es decir 64 KB. Lo cual representa 16 extensiones por MB.

Para hacer que la asignación de espacio sea eficiente, SQL Server 2000 no asigna extensiones enteras a tablas con poca cantidad de datos. SQL Server 2000 tiene dos tipos de extensiones:

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



Las tablas o índices nuevos son asignados a páginas de extensiones mixtas. Cuando la tabla o el índice crecen hasta el punto de ocupar ocho páginas, se pasan a extensiones uniformes.

Archivos y grupos de archivos físicos de la base de datos

Un archivo de base de datos no es más que un archivo del sistema operativo. Una base de datos se distribuye en por lo menos dos archivos, aunque es muy probable que sean varios los archivos de base de datos que se especifican al crear o al modificar una base de datos.

Principalmente SQL Server divide su trabajo en un archivo para datos y otro para el registro de las transacciones (log).

SQL Server 2000 permite los tres siguientes tipos de archivos:

- **Archivos de datos primarios**

Toda base de datos tiene un archivo de datos primario que realiza el seguimiento de todos los demás archivos, además de almacenar datos. Por convenio este archivo tiene la extensión MDF.

- **Archivos de datos secundarios**

Una base de datos puede tener cero o varios archivos de datos secundarios. Por convenio la extensión recomendada para los archivos de datos secundarios es NDF.

- **Archivos de registro (LOG)**

Todas las bases de datos por lo menos tendrán un archivo de registro que contiene la información necesaria para recuperar todas las transacciones que suceden sobre la misma. Por convenio la extensión de este archivo es LDF.

Por lo tanto al crear una base de datos, debemos considerar los siguientes premisas y reglas para el almacenamiento de los datos:

1. Todas las Bases de Datos tienen un archivo de base de datos primario (.mdf) y uno para el Log de Transacciones (.ldf). Además puede tener archivos de datos secundarios (.ndf).

2. Cuando se crea una Base de Datos, una copia de la Base de Datos Model, la cual incluye tablas del sistema, es copiada en la Nueva Base de Datos.
3. La Data es almacenada en bloques de 8-kilobytes (KB) de espacio de disco contiguo llamado páginas.
4. Las filas o registros no pueden atravesar páginas. Esto, es, que la máxima cantidad de datos en una fila de datos simple es de 8060 bytes.
5. Las tablas y los índices son almacenados en Extents. Un Extents consta de ocho páginas contiguas, o sea 64 KB.
6. El Log de Transacciones lleva toda la información necesaria para la recuperación de la Base de Datos en una eventual caída del sistema. Por default, el tamaño del Log de Transacciones es del 25% del tamaño de los archivos de datos. Use esta configuración como punto de partida y ajuste de acuerdo a las necesidades de su aplicación.

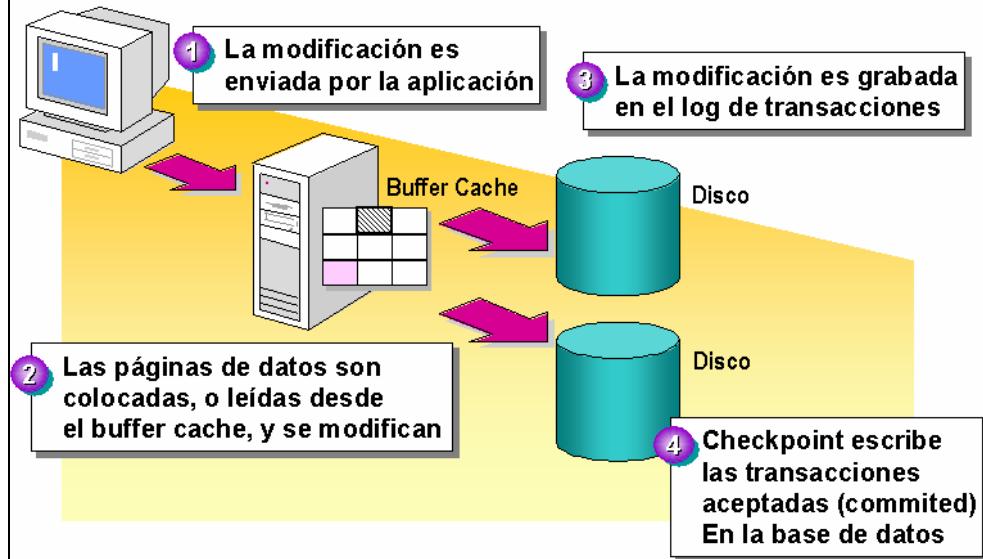
Archivos de Registro (LOG de Transacciones)

El LOG de transacciones archiva todas las modificaciones de los datos tal cual son ejecutados. El proceso es como sigue:

1. Una modificación de datos es enviada por la aplicación cliente.
2. Cuando una modificación es ejecutada, las páginas afectadas son leídas del disco a memoria (Buffer Cache), provista de las páginas que no están todavía en la Data Cache del query previo.
3. Cada comando de modificación de datos es archivado en el LOG. El cambio siempre es archivado en el LOG y es escrito en el disco antes que el cambio sea hecho en la Base de Datos. Este tipo de LOG es llamado LOG de tipo write-ahead.
4. Una vez que las páginas de datos residen en el Buffer Cache, y las páginas de LOG son archivadas sobre el disco en el archivo del LOG, el proceso de CHECKPOINT, escribe todas las transacciones completas a la Base de Datos en el disco.

Si el sistema falla, automáticamente el proceso de recuperación usa el LOG de Transacciones para llevar hacia delante todas las transacciones comprometidas (COMMIT) y llevar hacia atrás alguna transacción incompleta (ROLLBACK).

Como trabaja el Log de transacciones



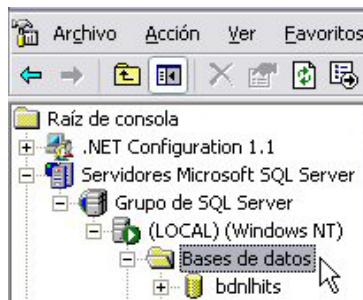
Los marcadores de transacción en el LOG son usados durante la recuperación automática para determinar los puntos de inicio y el fin de una transacción. Una transacción es considerada completa cuando el marcador BEGIN TRANSACTION tiene un marcador asociado COMMIT TRANSACTION. Las páginas de datos son escritas al disco cuando ocurre el CHECKPOINT.

Creación de Base de Datos

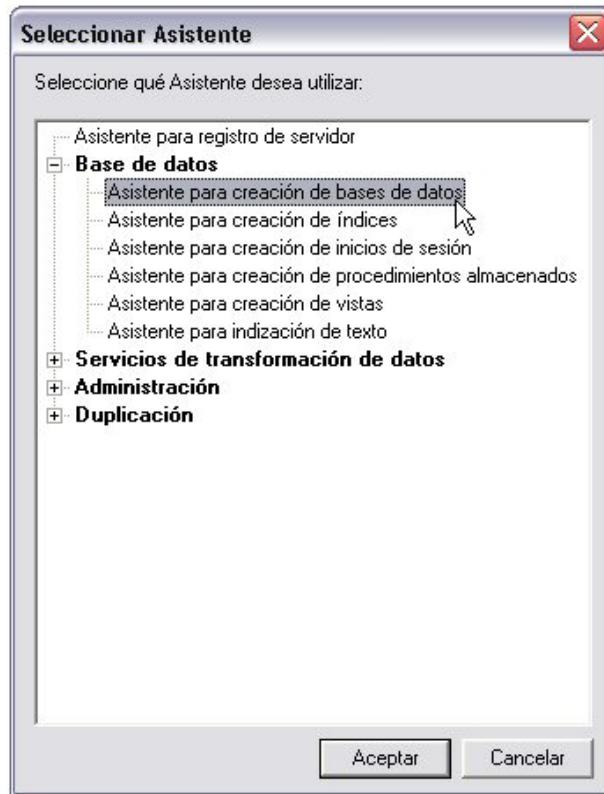
Se puede crear una base de datos de distintas maneras, utilizando el Wizard, desde el Administrador Empresarial o a través del Query Analyzer.

Desde el Asistente

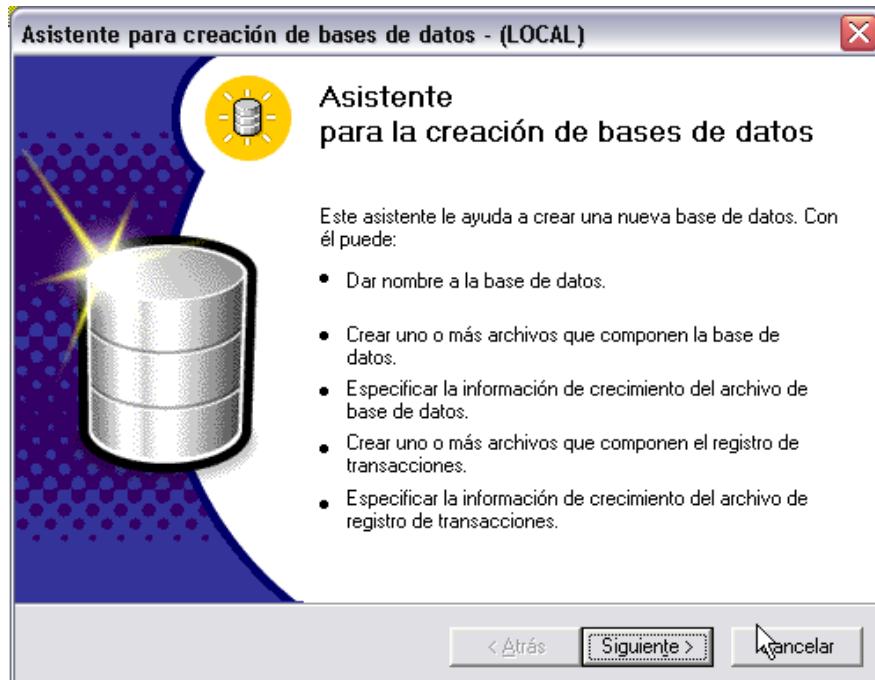
Ingrese al Administrador Empresarial y seleccione la carpeta Bases De Datos, tal como lo muestra la figura



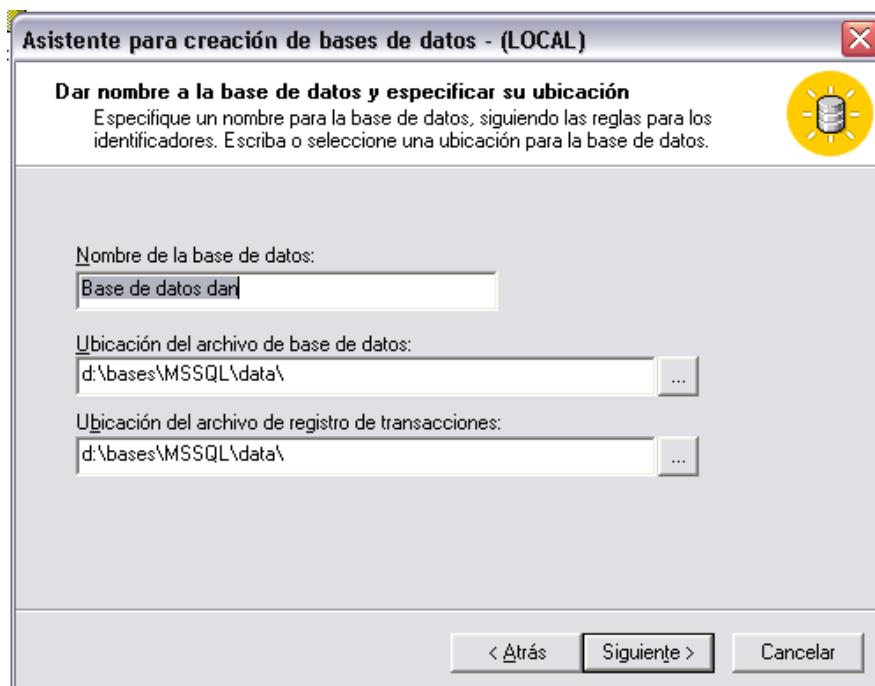
Haga clic en el menú Herramientas y seleccione la opción Asistentes, extienda la opción Base de datos y seleccione la primera opción (Asistente para creación de bases de datos), tal como lo muestra la siguiente imagen:



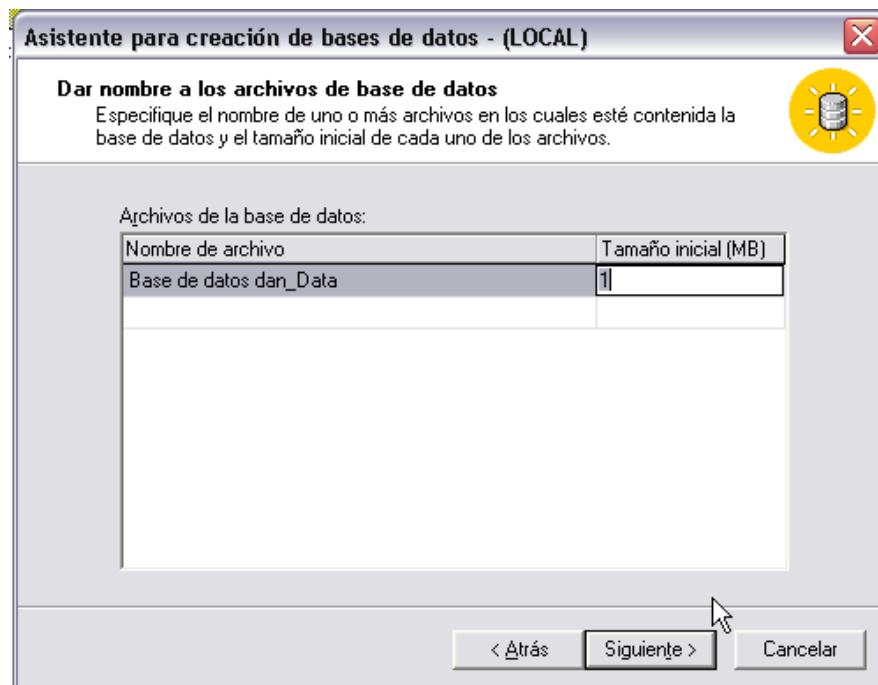
Se presentará una pantalla de bienvenida al wizard pulse Siguiente:



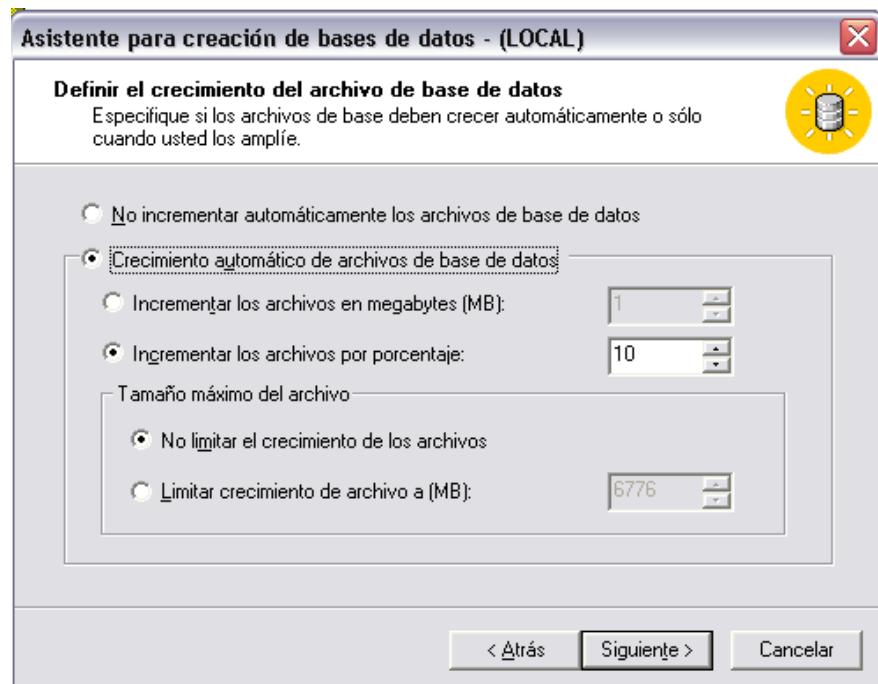
La siguiente pantalla le permitirá especificar el nombre de la base de datos y las carpetas donde se almacenaran los archivos de datos y de log.



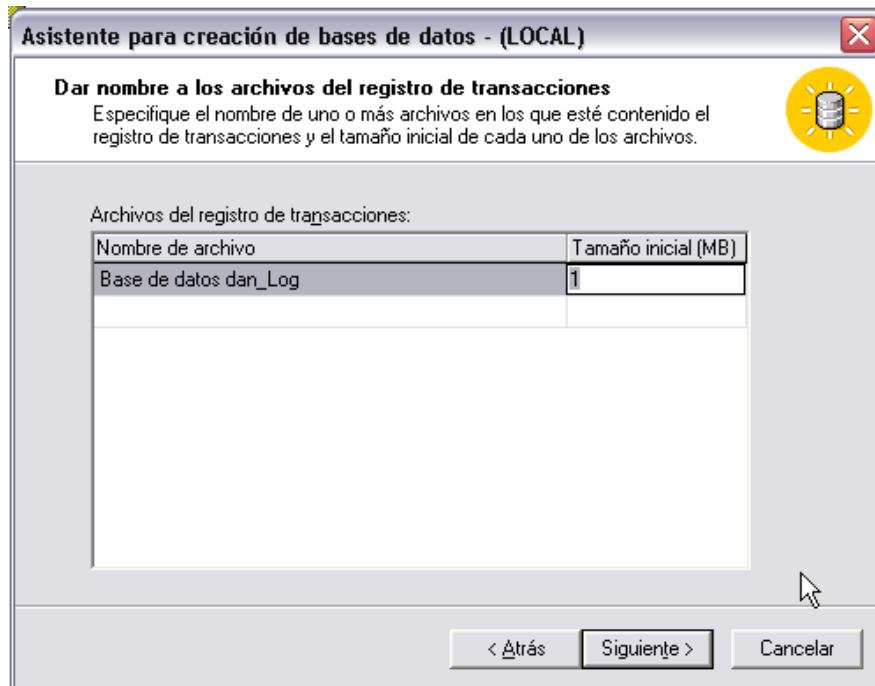
Luego de pulsar Siguiente, aparece una pantalla donde especificará si desea emplear mas de un archivo de datos así como también podrá indicar el tamaño de cada archivo:



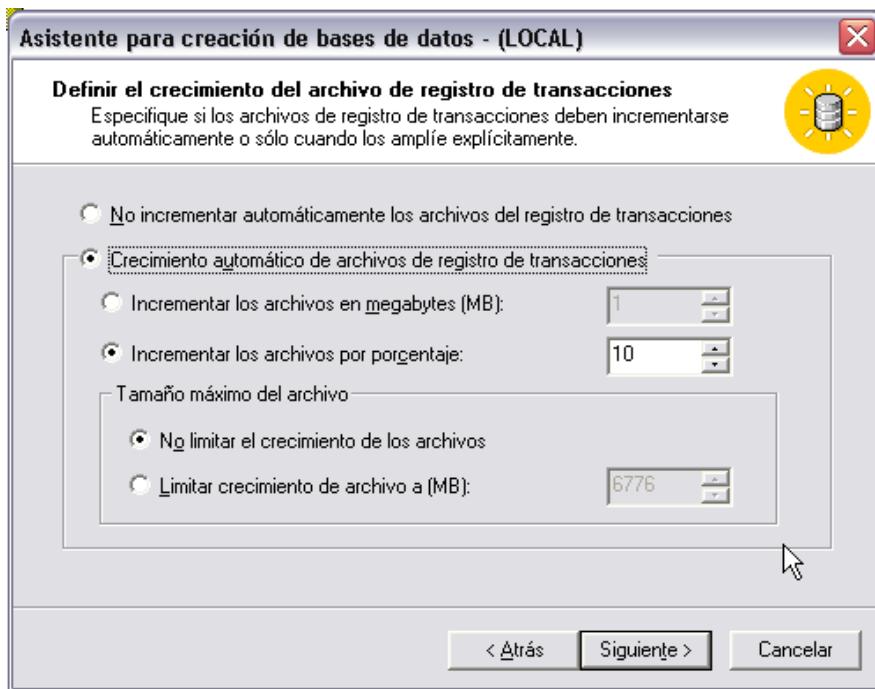
Luego de pulsar Siguiente, aparecen las opciones para personalizar el crecimiento automático del archivo de datos:



Luego de pulsar Siguiente. Especifique el nombre para el archivo de log:



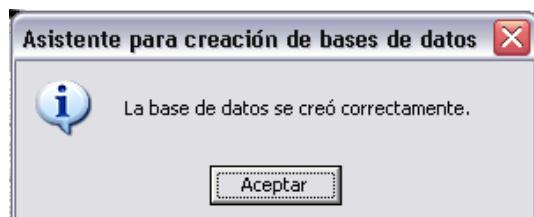
Similar al caso del archivo de datos, luego de pulsar Siguiente, también podrá establecer el crecimiento automático o no del archivo de transacciones:



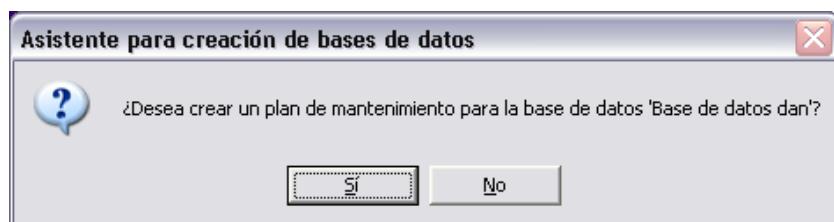
Luego de pulsar Siguiente, aparecerá la pantalla final;



Pulse Finalizar, de no haber problemas le aparecerá el siguiente mensaje:



Luego de pulsar Aceptar, aparecerá la siguiente pregunta:



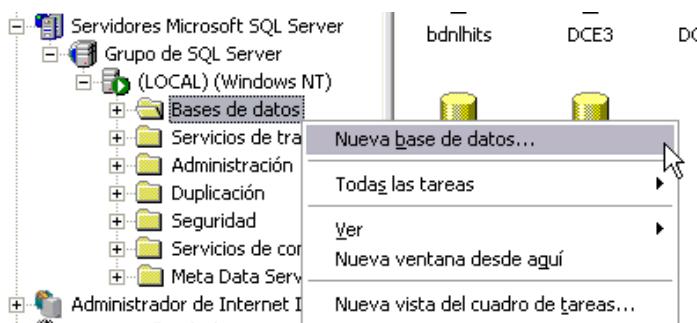
Conteste que No, luego de lo cual en el Administrador Empresarial podrá observar la nueva base de datos.



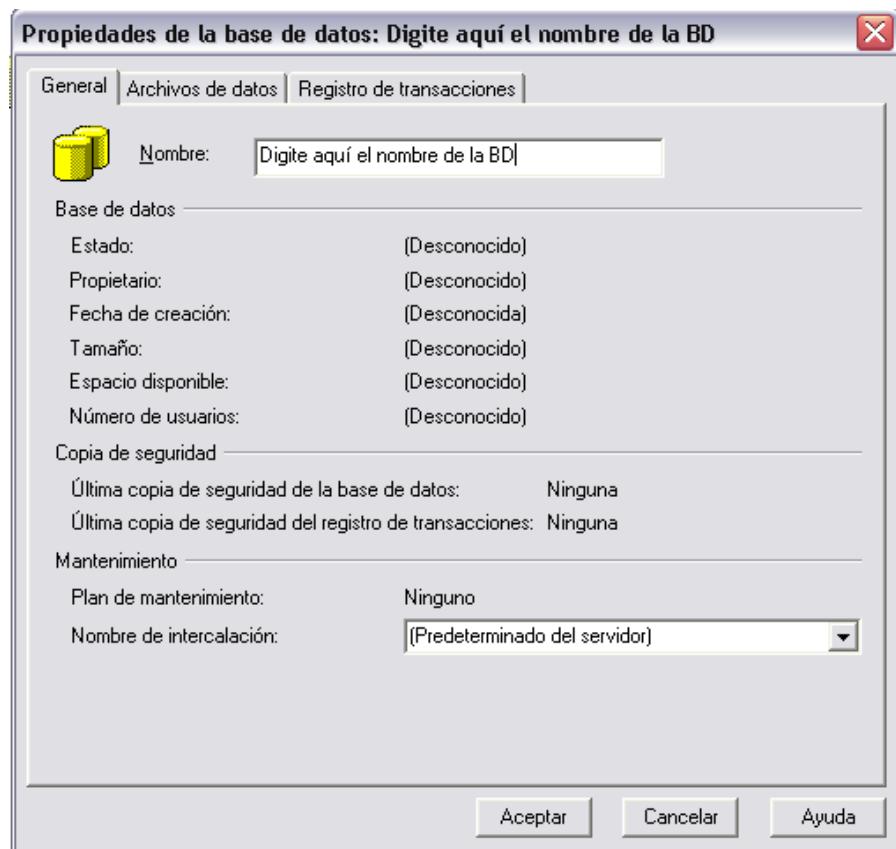
Desde el Administrador Empresarial

Otra forma de crear la base de datos es desde el Administrador Empresarial, para ello:

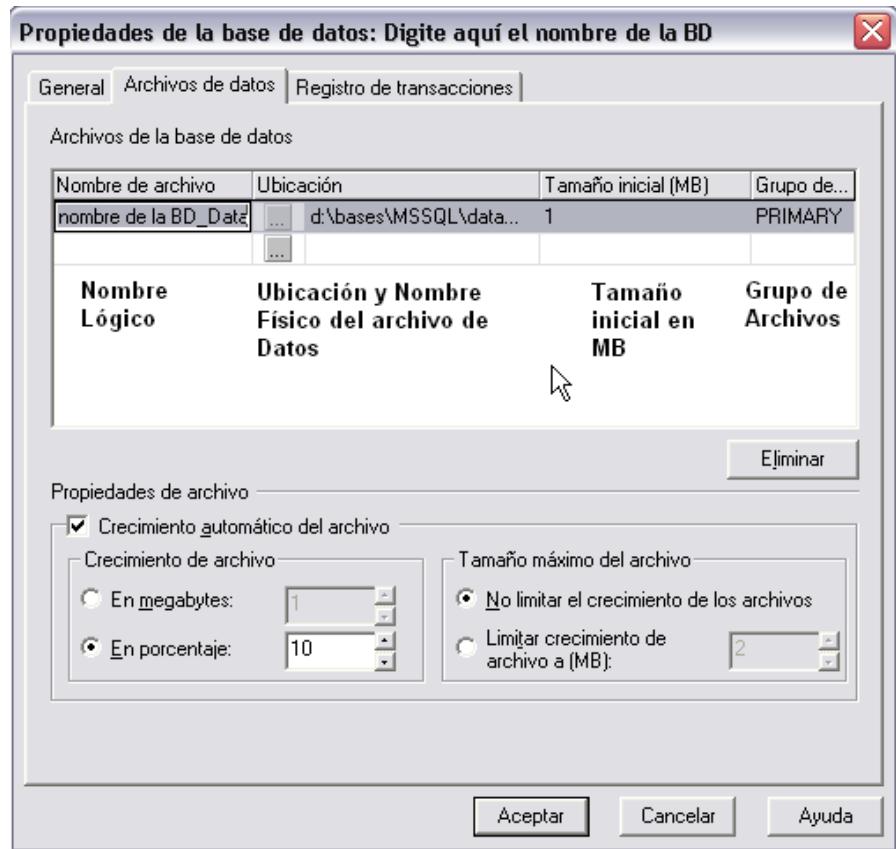
Ingresé al Administrador Empresarial, haga clic derecho sobre la carpeta Databases y seleccione la opción New Database, tal como lo muestra la figura:



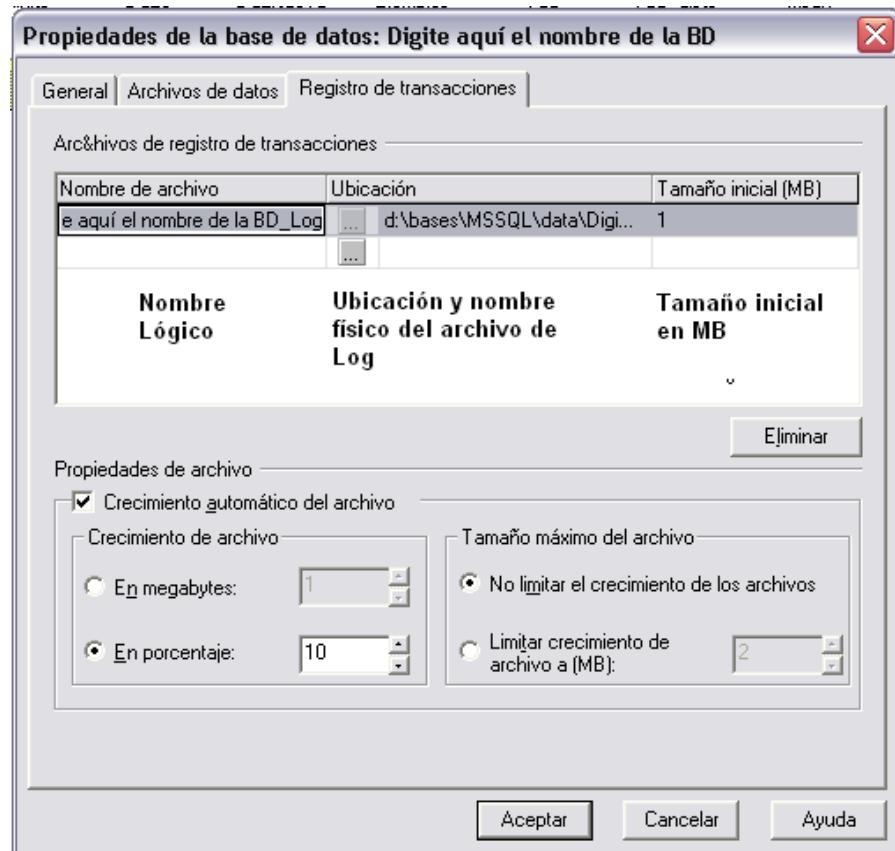
Luego aparecerá la siguiente pantalla, coloque el nombre de la base de datos y opcionalmente podrá especificar el código de página que empleará, esto lo puede seleccionar de la lista Collation Name:



Para especificar la información referente al archivo de datos, haga un clic en la ficha Data Files y complete la siguiente información:



Para poder especificar las características del archivo de log, haga clic en la ficha Transaction Log:



Una vez establecido los valores y luego de pulsar Aceptar, en el Administrador Empresarial se observara la nueva base de datos creada.

Desde el Analizador de Consultas

Otra de las formas de crear una base de datos es a través del Analizador de Consultas, donde emplearemos la sentencia CREATE DATABASE, cuya sintaxis reducida es la siguiente:

```
CREATE DATABASE NombreBaseDatos
[ ON [PRIMARY
    NAME = nombreArchivoLógico,
    FILENAME = 'nombreArchivoSO',
    SIZE = tamaño,
    MAXSIZE = { tamañoMáximo | UNLIMITED } ,
    FILEGROWTH = incrementoCrecimiento) [,..n]
]
[ LOG ON
    NAME = nombreArchivoLógico,
    FILENAME = 'nombreArchivoSO',
    SIZE = tamaño,
    MAXSIZE = { tamañoMáximo | UNLIMITED } ,
    FILEGROWTH = incrementoCrecimiento) [,..n]
```

[COLLATE nombre_collation] [FOR LOAD | FOR ATTACH]

Argumentos

nombreBaseDatos

Es el nombre de la nueva base de datos, deben ser únicos en un servidor y pueden tener hasta 128 caracteres, a menos que no se especifique ningún nombre lógico para el registro. Si no se especifica ningún nombre lógico de archivo de registro, SQL Server genera un nombre lógico al anexar un sufijo a nombreBaseDatos.

ON

Especifica que los archivos de disco utilizados para almacenar la parte de datos (archivos de datos) se han definido explícitamente. La palabra clave va seguida de una lista delimitada por comas de elementos que definen los archivos de datos del grupo de archivos principal.

PRIMARY

Especifica que la lista de archivos está asociada al grupo principal. Este grupo contiene todas las tablas del sistema de base de datos. También contiene todos los objetos no asignados a los grupos de archivos de usuario. El primer archivo especificado pasa a ser el archivo principal, el cual contiene el inicio lógico de la base de datos y de las tablas del sistema. Una base de datos sólo puede tener un archivo principal. Si no se especifica PRIMARY, el primer archivo enumerado en la instrucción CREATE DATABASE se convierte en el archivo principal.

LOG ON

Especifica que los archivos de registro de la base de datos (archivos de registro) se han definido explícitamente. La palabra clave va seguida de una lista delimitada por comas la cual define las características de los archivos de registro. Si no se especifica LOG ON, se crea automáticamente un único archivo de registro con un nombre generado por el sistema y un tamaño que es el 25% de la suma de los tamaños de todos los archivos de datos de la base de datos.

FOR LOAD

Cláusula que se mantiene por compatibilidad con versiones anteriores de SQL Server. La base de datos se crea con la opción de base de datos **dbo use only** activada y el estado se establece en "cargando". En realida esto no es necesario en SQL Server 7.0 porque la instrucción RESTORE puede volver a crear la base de datos como parte de la operación de restauración.

FOR ATTACH

Crea la base de datos desde un conjunto existente de archivos del sistema operativo. Debe existir una entrada de archivos que determine cual es el archivo principal, las otras entradas son necesarias si existen archivos creados en una ruta de acceso distinta de cuando se creó la base de datos por primera vez o se adjuntó por última vez.

Utilice el procedimiento almacenado del sistema **sp_attach_db** en lugar de emplear *CREATE DATABASE FOR ATTACH* directamente, esto deberá emplearlo si debe especificar más de 16 archivos.

COLLATE

Especifica el conjunto de caracteres que se empleará para almacenar información en la base de datos, se puede emplear un conjunto de caracteres especificado por Windows o por SQL Server. De no especificarse se empleará el conjunto de caracteres seleccionado en el momento de la instalación

NAME

Especifica el nombre lógico del archivo.

No se requiere este parámetro cuando se especifica FOR ATTACH.

Este nombre es el utilizado para referenciar al archivo en las sentencias del Transact-SQL que se ejecuten después.

FILENAME

Especifica el nombre de archivo del sistema (archivo físico).

Se debe especificar la ruta de acceso y nombre de archivo que el sistema operativo utiliza cuando crea la base de datos. La ruta de acceso debe especificar un directorio en el servidor sobre el que se instalo SQL Server.

No se puede especificar un directorio en un sistema comprimido de archivos.

SIZE

Especifica el tamaño para el archivo. De no hacerlo SQL Server utiliza el tamaño del archivo principal de la base de datos model.

Cuando este parámetro no es especificado para un archivo secundario o de registro SQL Server automáticamente le asigna 1 MB.

El valor mínimo a asignar es de 512 KB. Si no se especifica tamaño, el valor predeterminado es 1 MB. El tamaño especificado para el archivo principal debe tener al menos el tamaño del archivo principal de la base de datos model.

MAXSIZE

Especifica el tamaño máximo de crecimiento del archivo. Se pueden utilizar los sufijos KB y MB, el valor predeterminado es MB. Especifique un número entero; no incluya decimales. Si no se especifica, el archivo aumenta hasta que el disco esté lleno.

UNLIMITED

Especifica que el archivo aumenta de tamaño hasta que el disco esté lleno.

FILEGROWTH

Especifica el incremento de crecimiento del archivo, este valor no puede exceder el valor MAXSIZE. Emplee un número entero. Un valor 0 indica que no hay crecimiento. El valor se puede especificar en MB, KB o %, el valor predeterminado es MB. Cuando se especifica %, el tamaño de incremento de crecimiento es el porcentaje especificado del tamaño del archivo en el momento en que tiene lugar el incremento. De no emplear FILEGROWTH, el valor predeterminado es 10% y el valor mínimo es 64 KB. El tamaño especificado se redondea al múltiplo de 64 KB más cercano.

Observaciones

Emplee CREATE DATABASE para crear una base de datos y los archivos que almacenan ésta. SQL Server implementa CREATE DATABASE en dos pasos:

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

SQL Server rellena el resto de la base de datos con páginas vacías, excepto las páginas que tengan datos internos que registren cómo se emplea el espacio en la base de datos.

Cualquier objeto definido por el usuario en model se copiará a todas las bases de datos recién creadas.

Cada base de datos nueva hereda los valores opcionales de la base de datos model (a menos que se especifique FOR ATTACH).

En un servidor se puede especificar un máximo de 32,767 bases de datos.

Cuando especifica una instrucción CREATE DATABASE nombreBaseDatos sin parámetros adicionales, la base de datos se crea con el mismo tamaño que model.

Cada base de datos tiene un propietario con capacidad para realizar actividades especiales. El propietario es el usuario que crea la base de datos, este propietario se puede cambiar mediante **sp_changegbowner**.

Para mostrar un informe de una base de datos o de todas las bases de datos de un servidor con SQL Server, ejecute **sp_helpdb**. Para obtener un informe acerca del espacio utilizado en una base de datos, emplee **sp_spaceused**. Para obtener un informe de los grupos de archivos de una base de datos, utilice **sp_helpfilegroup**, y utilice **sp_helpfile** para obtener el informe de los archivos de la base de datos.

¿Quiénes pueden crear bases de datos?

En forma predeterminada podrán hacerlos los usuarios que pertenecen al rol **sysadmin** y **dbcreator**. Los miembros de las funciones fijas de servidor **sysadmin** y **SecurityAdmin** pueden conceder permisos CREATE DATABASE a otros inicios de sesión. Los miembros de las funciones fijas de servidor **sysadmin** y **dbcreator** pueden agregar otros inicios de sesión a la función **dbcreator**. El permiso CREATE DATABASE debe concederse explícitamente; no se concede mediante la instrucción GRANT ALL.

Estos permisos se limitan a unos cuantos inicios de sesión para mantener el control de la utilización de los discos del equipo que ejecuta SQL Server.

Ejemplos de creación de base de datos empleando el Analizador de Consultas

Primero ingrese al Analizador de Consultas para ello primero debe especificar el tipo de autentificación a realizar del sistema o estándar, vea la siguiente figura:



Ejemplo 1

Crear la base de datos Prueba1 con los parámetros En forma predeterminada.

`Use Master`

`GO`

`Create Database Prueba1`

`GO`

Verifique la creación de la base de datos y note que automáticamente SQL Server asignó tamaños y nombres lógicos para los archivos. Para ello emplee el siguiente procedimiento almacenado del sistema:

`Sp_HelpDB Prueba1`

`GO`

Debe obtener el siguiente resultado:

name	db_size	owner	dbid	created	status	Compatibility Level
Prueba1	1.12 MB	sa	8	Feb 28 2002	Status=ONLINE,	
					Updateability=READ_WRITE,	
					UsuarioAccess=MULTI_USUARIO,	
					Recovery=FULL,	
					Version=539,	
					Collation=SQL_Latin1_General_CI_AS,	
					SQLSortOrder=52,	
					IsTornPageDetectionEnabled,	
					IsAutoCreateStatistics,	
					IsAutoUpdateStatistics	8.0

Además se mostrará un informe con los archivos que se crearon automáticamente:

Columnas	Archivo de datos	Archivo de Log
name	prueba1	prueba1_log
fileid	1	2
	C:\Program Files\Microsoft SQL Server\MSSQL\data\Prueba1.mdf	C:\Program Files\Microsoft SQL Server\MSSQL\data\Prueba1_Log.ldf
filename		
filegroup	PRIMARY	NULL
size	640Kb	504Kb
maxsize	Unlimited	Unlimited
growth	10%	10%
usage	data only	log only

Ejemplo 2

Crear la base de datos Prueba2 con un archivo de datos de 10Mb, un tamaño máximo de 20Mb y un crecimiento de 1Mb., el archivo de registro debe asumir los valores por default.

```
Use Master
GO
Create Database Prueba2
On Primary
  (NAME      = 'Prueba2_Data',
   FILENAME  = 'C:\Program Files\Microsoft SQL
Server\MSSQL\data\Prueba2 _Data.Mdf',
   SIZE       = 10Mb,
   MAXSIZE    = 20Mb,
   FILEGROWTH= 1Mb)
GO
```

Verifique la creación de la base de datos anterior:

```
Sp_HelpDB Prueba2
GO
```

Puede notar como SQL Server aprovecha los valores predeterminados en la base de datos model para completar la información que corresponde al log de transacciones, la cual no se específico en la sentencia CREATE DATABASE.

Ejemplo 3

Crear la base de datos Prueba3 especificando un archivo de datos con un tamaño inicial de 15Mb, un tamaño máximo de 30Mb y un crecimiento de 5Mb., el archivo de registro debe tener un tamaño inicial de 5MB y uno máximo de 10MB, el crecimiento debe ser de 1MB.

```
Use Master
GO
Create Database Prueba3
On Primary
  (NAME      = 'Prueba3_Data',
```

```

    FILENAME  = 'C:\Program Files\Microsoft SQL
Server\MSSQL\data\Prueba3 _Data.Mdf'
        SIZE      = 15Mb,
        MAXSIZE   = 30Mb,
        FILEGROWTH= 5Mb)
Log On
    (NAME      = 'Prueba3_Log',
    FILENAME  = 'C:\Program Files\Microsoft SQL
Server\MSSQL\data\Prueba3 _Log.Ldf'
        SIZE      = 5Mb,
        MAXSIZE   = 10Mb,
        FILEGROWTH= 1Mb)
GO
-- Verifique la información con :
Sp_HelpDB Prueba3
GO

```

Otra de las formas de comprobar la creación de las bases de datos es mostrando las filas de la tabla del sistema SysDatabases.

```

Use Master
GO
Select DbID, Name From SysDatabases
GO

```

Revise los resultados.

Creando Múltiples Archivos

La ventaja de almacenar la base de datos en múltiples archivos radica en la flexibilidad de modificar en futuro la configuración del hardware sin que se vea afectada la base de datos, otro de los motivos es que si emplea una base de datos de 15GB y por algún motivo ocurre una falla y desea recuperar desde el backup, necesitaría una unidad de 15 o mas gigabytes de almacenamiento, mientras que si distribuyó la base de datos en múltiples archivos pequeños será mas probable que tenga disponibles múltiples unidades de 4 GB que unidades de 15GB.

Con las sentencias del Transact-SQL es posible modificar la lista de archivos que conforman la base de datos, agregar o quitar archivos, incluso puede definir nuevos grupos de archivos los cuales permitirán tratar múltiples archivos como si se tratará de uno solo.

Para poder realizar esta tarea emplee la sentencia ALTER DATABASE

Sintaxis

ALTER DATABASE NombreBD

```
{ ADD FILE <Especificación del archivo> [,...n] [TO FILEGROUP  
nombreGrupoArchivos]  
| ADD LOG FILE <<Especificación del archivo> [,...n]  
| REMOVE FILE nombreArchivoLógico  
| ADD FILEGROUP nombreGrupoArchivos  
| REMOVE FILEGROUP nombreGrupoArchivos  
| MODIFY FILE <<Especificación del archivo>  
| MODIFY FILEGROUP nombreGrupoArchivos propiedadGrupoArchivos  
| SET < optionspec > [ ,...n ] [ WITH < termination > ]  
| COLLATE < collation_name > }
```

Argumentos

ADD FILE

Especifica que se está agregando un archivo.

TO FILEGROUP

Especifica el grupo de archivos al que debe agregarse el archivo especificado.

ADD LOG FILE

Especifica que se agregue un archivo de registro a la base de datos indicada.

REMOVE FILE

Elimina el archivo de la base de datos y retira su referencia en las tablas del sistema además de eliminar el archivo físico. Este archivo no podrá eliminarse si no está vacío.

ADD FILEGROUP

Especifica que se va a agregar un grupo de archivos.

REMOVE FILEGROUP

Quita el grupo de archivos de la base de datos, no se puede realizar si el grupo de archivos no está vacío.

MODIFY FILE

Especifica que el archivo dado se debe modificar, incluidas las opciones *FILENAME*, *SIZE*, *FILEGROWTH* y *MAXSIZE*. Sólo se puede cambiar una de estas propiedades a la vez.

MODIFY FILEGROUP nombreGrupoArchivos propiedadGrupoArchivos

Especifica la propiedad que se aplicará a los archivos que pertenecen al grupo de archivos.

Los valores de **propiedadGrupoArchivos** son:

READONLY

Especifica que el grupo de archivos es de sólo lectura. De tal manera que no se podrán realizar modificaciones sobre los archivos pertenecientes a este grupo.

READWRITE

Invierte la propiedad READONLY. Están habilitadas las actualizaciones para los objetos del grupo de archivos.

DEFAULT

Especifica que el grupo de archivos es el predeterminado de la base de datos. Sólo un grupo puede ser el predeterminado, esta propiedad se quita del grupo de archivos que había sido anteriormente el predeterminado. CREATE DATABASE hace que el grupo de archivos principal sea el grupo predeterminado inicialmente. Si no se especifica ningún grupo de archivos en las instrucciones CREATE TABLE, ALTER TABLE o CREATE INDEX, se crean nuevas tablas e índices en el grupo predeterminado.

SET

Permite establecer valores para algunas de las características de trabajo en una base de datos, por ejemplo el tipo de recovery que se empleará para los backups.

COLLATE

Especifica el conjunto de caracteres a emplear, ya sean de Windows o de SQL Server 2000.

Observaciones

No se puede agregar o quitar un archivo mientras se está ejecutando una instrucción BACKUP.

Ejemplo 1

Modificar la base de datos Prueba2, de tal manera que le debe agregar un archivo de datos secundario de 5MB y un tamaño máximo de 10 MB. con un crecimiento de 1MB. Antes de ejecutar el siguiente comando utilice Sp_HelpDB Prueba2, para comparar luego con los resultados después de ejecutar la sentencia.

```
USE master
GO
ALTER DATABASE Prueba2
ADD FILE
(
NAME = Prueba2Sec_Data,
FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL\data\Prue2Data.ndf',
SIZE = 5MB,
MAXSIZE = 10MB,
```

```
FILEGROWTH = 1MB
)
GO
Sp_HelpDB Prueba2
GO
-- Compare los resultados con los anteriores
Si desea información de los archivos, emplee la siguiente sentencia:
```

```
Use Prueba2
GO
Sp_HelpFile
GO
```

Si desea ver las características sólo del archivo que agrego utilice la siguiente sentencia:

```
Sp_HelpFile Prueba2Sec_Data
GO
```

/* El resultado le mostrará información del archivo que acaba de agregar */

Ejemplo 2

Crear dos grupos de archivos en la base de datos Prueba2, el primer grupo se llamará CONSULTORES y el otro se llamará OPERACIONES.

```
ALTER DATABASE Prueba2
ADD FILEGROUP Consultores
GO
ALTER DATABASE Prueba2
ADD FILEGROUP Operaciones
GO
```

-- Verifique la información con las siguientes instrucciones:

```
Use Prueba2
GO
Sp_HelpFileGroup
GO
```

Se mostrará el siguiente resultado:

groupname	groupid	filecount
Consultores	2	0
Operaciones	3	0
PRIMARY	1	2

Ejemplo 3

A cada uno de los grupos creados anteriormente añadale dos archivos de datos, para ello considere lo siguiente: los archivos del grupo CONSULTORES deben tener un tamaño de 10 MB. cada uno, con un tamaño máximo de 20 MB y un crecimiento de

2 MB., mientras que los del grupo OPERACIONES tendrán un tamaño inicial de 5 MB y un máximo de 30 MB. con un crecimiento de 5 Mb.

```
Use Master
GO
ALTER DATABASE Prueba2
ADD FILE
(NAME      = 'DatCons01',
FILENAME  = 'C:\Program Files\Microsoft SQL
Server\MSSQL\data\DatCons1.ndf',
SIZE      = 10MB,
MAXSIZE   = 20MB,
FILEGROWTH = 2MB),
(NAME      = 'DatCons02',
FILENAME  = 'C:\Program Files\Microsoft SQL
Server\MSSQL\data\DatCons2.ndf',
SIZE      = 10MB,
MAXSIZE   = 20MB,
FILEGROWTH = 2MB),
TO FILEGROUP CONSULTORES
GO
ALTER DATABASE Prueba2
ADD FILE
(NAME      = 'DatOper01',
FILENAME  = 'C:\Program Files\Microsoft SQL
Server\MSSQL\data\DatOper1.ndf',
SIZE      = 5MB,
MAXSIZE   = 30MB,
FILEGROWTH = 5MB),
(NAME      = 'DatOper02',
FILENAME  = 'C:\Program Files\Microsoft SQL
Server\MSSQL\data\DatOper2.ndf',
SIZE      = 5MB,
MAXSIZE   = 30MB,
FILEGROWTH = 5MB),
TO FILEGROUP OPERACIONES
GO
```

Una vez que ejecuta estas sentencias, verifique la información con la siguiente instrucción:

```
Use Prueba2
GO
Sp_HelpFileGroup
GO
```

groupname	groupid	filecount
Consultores	2	2
Operaciones	3	2
PRIMARY	1	2

Si desea información de un grupo de archivos en particular, utilice:
Sp_HelpFileGroup Operaciones
GO

Ejemplo 4

Modificar el tamaño del DatOper01 asignándole como nuevo tamaño máximo 40 Mb.

```
Use Master
GO
ALTER DATABASE Prueba2
MODIFY FILE
( NAME      = 'DatOper01',
  MAXSIZE  = 40Mb)
GO
```

-- Para verificar el cambio emplee la siguiente instrucción:

```
Sp_HelpFile DatOper01
GO
```

Ejemplo 5

Eliminar el archivo DatOper01.

```
Use Master
GO
ALTER DATABASE Prueba2
REMOVE FILE 'DatOper01'
GO
```

Ejemplo 6

Hacer que el grupo de archivos Operaciones sea el grupo En forma predeterminada.

```
Use Master
GO
ALTER DATABASE Prueba2
  MODIFY FILEGROUP Operaciones DEFAULT
GO
```

Cuando lo que se requiere es eliminar una base de datos, debe emplear la sentencia DROP DATABASE, cuya sintaxis es la siguiente:

DROP DATABASE database_name [,....n]

Ejemplo 7

Eliminar la base de datos Prueba2.

```
Use Master
```

```
GO  
DROP DATABASE Prueba2  
GO
```

Revisar la tabla SysDatabases, verifique que se elimino la entrada de Prueba2

Ejemplo 8

Eliminar la base de datos Prueba1 y NuevoNombre

```
Use Master  
GO  
DROP DATABASE Prueba1, NuevoNombre  
GO
```

Revisar la tabla SysDatabases, verifique que se elimino la entrada de Prueba2

```
Use Master  
GO  
Select Name From SysDatabases  
GO
```

Rrenombrando Base de Datos

Para quitar una base de datos, utilice DROP DATABASE. Para cambiar el nombre de una base de datos, utilice **sp_renamedb**, pero recuerde que para esto la base de datos a renombrar tiene que tener activa la opción '**single user**', si desea comprobar el empleo de esta sentencia realice la siguiente secuencia de instrucciones desde el Analizador de Consultas, verifique la base de datos Prueba3 (creada en el Ejemplo3) no este seleccionada en el Administrador Empresarial, para asegurarse haga clic izquierdo en Databases, luego ingrese al Analizador de Consultas con una cuenta de administrador (Windows authentication) o con la cuenta sa :

```
/* Comprueba la presencia de Prueba3 */  
Use Master  
GO  
Select name From SysDatabases  
GO
```

El resultado sería:

Name
master
tempdb
model
msdb
pubs
Northwind
Prueba3
Prueba1
Prueba2

```
/* Para renombrar active la opción Single User en la Base de datos a  
renombrar */  
Sp_DBOption 'Prueba3', 'Single User', 'True'  
GO
```

El resultado sería:

```
DBCC execution completed. If DBCC printed error messages, contact your system  
administrator.  
The database is now single usuario.
```

```
/* En este instante la base de datos puede ser renombrada */  
Sp_RenameDB 'Prueba3', 'NuevoNombre'  
GO
```

El resultado sería:

```
(1 row(s) affected)  
DBCC execution completed. If DBCC printed error messages, contact your system  
administrator.  
The database is renamed and in single usuario mode.  
A member of the sysadmin rol must reset the database to multiusuario mode with  
sp_dboption.
```

```
/* Compruebe el correcto renombrado de la base de datos y luego retire la  
opción single usuario de la base de datos */  
Select Name From SysDatabases  
GO  
Sp_DBOption 'NuevoNombre', 'Single Usuario', 'True'  
GO
```

Creación de Tablas

Objetivos:

- Determinar Tipos de datos de SQL Server a utilizar en las tablas
- Implementación de tablas, establecer restricciones
- Asignar permisos

Temas:

- Tipos de datos de SQL Server – Tipos de datos de usuario
- Empleo de comando DDL (Data Definition Language)
- Implementar Restricciones
- Asignar roles y/o permisos – Comandos DCL (Data Control Language)

Tipos de Datos de SQL Server 2000

SQL Server brinda una serie de tipos de datos para almacenar la información, la correcta selección del tipo de dato es simplemente una cuestión de determinar que valores desea almacenar, como por ejemplo carácter, enteros, binario, fechas, etc. Los siguientes objetos tienen tipos de datos:

Columnas de tablas y vistas.

Parámetros de procedimientos almacenados.

VARIABLES.

Funciones de Transact-SQL que devuelven uno o más valores de datos de un tipo de datos específico.

- Procedimientos almacenados que devuelven un código, que siempre es de tipo **integer**.

Al asignar un tipo de datos a un objeto se definen cuatro atributos del objeto:

- La clase de datos que contiene el objeto, por ejemplo, carácter, entero o binario.
- La longitud del valor almacenado o su tamaño.
- La precisión del número (sólo tipos de datos numéricos).

La precisión es el número de dígitos que puede contener el número. Por ejemplo, un objeto **smallint** puede contener hasta 5 dígitos, con lo que tiene una precisión de 5.

- La escala del número (sólo tipos de datos numéricos).
- La escala es el máximo número de dígitos a la derecha del separador decimal. Por ejemplo, un objeto **int** no puede aceptar un separador decimal y tiene una escala de 0. Un objeto **money** puede tener hasta 4 dígitos a la derecha del separador decimal y tiene una escala de 4.

Si un objeto se define como **money**, puede contener hasta 19 dígitos y 4 de ellos pueden estar a la derecha del decimal. El objeto usa 8 bytes para almacenar los datos. Por tanto, el tipo de datos **money** tiene una precisión de 19, una escala de 4 y una longitud de 8.

Utilizar datos binarios

Los tipos de datos **binary** y **varbinary** almacenan cadenas de bits. Mientras que los datos de carácter se interpretan según la página de códigos de SQL Server, los datos **binary** y **varbinary** son, simplemente, un flujo de bits. Los datos **binary** y **varbinary** pueden tener una longitud de hasta 8.000 bytes.

Las constantes binarias tienen un 0x (un cero y una letra x en minúsculas) a la izquierda, seguido de la representación hexadecimal del patrón de bits. Por ejemplo, 0x2A especifica el valor hexadecimal 2A, que es equivalente al valor decimal 42 o un patrón de bits de un byte de 00101010.

La siguiente es una tabla que describe los tipos de datos provistos por SQL Server:

Categoría	Descripción	Tipo de Dato	Descripción
Binario	Almacenan cadenas de bits. La data consiste de números hexadecimales. Por ejemplo el decimal 245 es F5 en hexadecimal.	binary	La data debe tener una longitud fija (hasta 8 KB).
		varbinary	Los datos pueden variar en el número de dígitos hexadecimales (hasta 8 KB).
		image	La data puede tener una longitud variable y exceder los 8Kb.
Carácter	Consisten de una combinación de letras, símbolos y números. Por ejemplo las combinaciones "John928" y "(0*&(%B99nh jkJ".	char	Los datos deben tener una longitud fija (Hasta 8 KB).
		varchar	La data puede variar en el número de caracteres (Hasta 8 KB.)
		text	Los datos pueden ser caracteres ASCII que excedan los 8 KB.
Fecha y Hora	Consisten en combinaciones válidas de estos datos. No puede separar en tipos distintos el almacenamiento de sólo fechas o sólo horas.	Datetime	Fechas en el rango 01 Ene 1753 hasta el 31 Dic 9999 (Se requiere 8 bytes por valor).
		smalldatetime	Fechas en el rango 01 Ene 1900 hasta 06 Jun 2079 (Se requiere 4 bytes por valor).
Decimal	Consisten en información que almacena información significativa después del punto decimal.	decimal	Los datos pueden tener hasta 38 dígitos, todos los cuales podrían estar a la derecha del punto decimal. Este tipo de dato guarda un valor exacto del número y no una aproximación.
		numeric	Para SQL Server, el tipo de dato numeric es equivalente al tipo de datos decimal.
Punto Flotante	Números aproximados (Punto flotante).	float	Datos en el rango de 1.79E + 308 hasta 1.79E + 308.
		real	Datos en el rango de 3.40E + 38 hasta 3.40E + 38.
Enteros	Consiste en información numérica positiva o negativa como por ejemplo -5, 0 y 25.	bigint	Datos en el rango de 2^63 (-9223372036854775808) hasta 2^63-1 (9223372036854775807). Se requieren de 8 bytes para almacenar estos valores.
		int	Datos en el rango de -2,147,483,648 hasta 2,147,483,647. Se requieren de 4 bytes para almacenar estos valores.

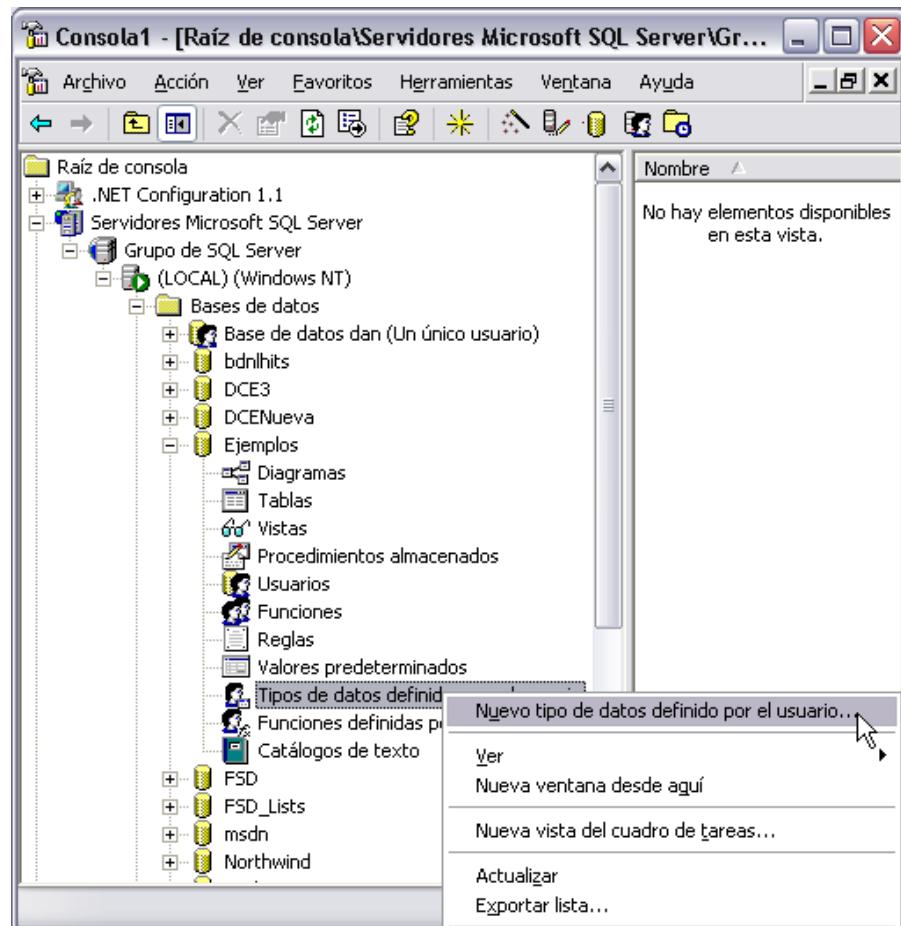
Categoría	Descripción	Tipo de Dato	Descripción
		smallint	Datos en el rango de -32,768 hasta 32,767. Se requieren 2 bytes por cada valor de este tipo.
		tinyint	Datos entre 0 y 255, se requiere de 1 byte.
Monetario	Cantidades monetarias positivas o negativas.	money	Datos monetarios entre -922,337,203,685,477.5808 y +922,337,203,685,477.5807 (Se requieren 8 bytes por valor).
		smallmoney	Datos monetarios entre -214,748.3648 y 214,748.3647 (Se requieren de 4 bytes por valor).
Especiales	Consisten en información que no recae en ninguna de las categorías anteriormente mencionadas.	bit	Datos que consisten de 1 o 0. Emplear este tipo de dato para representar TRUE o FALSE ó YES o NO.
		cursor	Este tipo de dato es empleado por variables o procedimientos almacenados que emplean parámetros OUTPUT referenciados a un cursor.
		timestamp	Este tipo de dato es empleado para indicar la actividad que ocurre sobre una fila. La secuencia de este número se incrementa en formato binario.
		uniqueidentifier	Consiste en un número hexadecimal que especifica un globally unique identifier (GUID), es útil cuando se desea asegurar la unicidad de una fila entre muchas otras.
		SQL_variant	Almacena varios tipos de datos, a excepción de text, ntext, timestamp, image y sql_variant.
		table	Almacena un resultado de una consulta para su posterior procesamiento. Se puede emplear para definir variables locales de tipo table o para retornar los valores devueltos por una función del usuario.
Unicode	Al emplear este tipo de datos se puede almacenar	nchar	Datos con longitud fija, hasta 4000 caracteres Unicode.

Categoría	Descripción	Tipo de Dato	Descripción
sobre una columna valores que incluyan este conjunto de caracteres. Hay que recordar que los datos Unicode emplean dos bytes por cada carácter a representar.		nvarchar	Datos que pueden variar, hasta 4000 caracteres Unicode.
		ntext	Datos que exceden los 4000 caracteres Unicode.

Tipos de datos definidos por el usuario

Los tipos de datos definidos por el usuario están basados en los tipos de datos disponibles a través de SQL Server 2000. Los tipos de datos definidos por el usuario se pueden emplear para asegurar que un dato tenga las mismas características sobre múltiples tablas.

Para crear un tipo de dato puede emplear el Administrador Empresarial expandiendo la base de datos donde desea crear el dato, luego deberá hacer un clic derecho sobre Tipos de datos definidos por el Usuario y seleccionar “*Nuevo tipo de datos definido por el usuario...*”, tal como lo muestra la siguiente representación:



Complete la caja de diálogo, tal como lo muestra la siguiente representación:



Desde el Analizador de Consultas puede emplear el stored procedure del sistema SP_ADDTYPE cuya sintaxis es la siguiente:

```
sp_addtype [@typename =] tipo,  
[@phystype =] tipoDatosSistema  
, [@nulltype =] 'tipoNull'
```

Argumentos

[@typename =] tipo

Es el nombre para el tipo de datos definido de usuario, deben ser únicos en cada base de datos.

[@phystype =] tipoDatosSistema

Es el tipo de datos proporcionado por SQL Server, (decimal, int, etc.) en el que se basa el tipo de datos del usuario.

[@nulltype =] 'tipoNull'

Indica la forma en que el tipo de datos del usuario trata los valores nulos. Este argumento puede tener como valor 'NULL', 'NOT NULL' o 'NONULL'. Si no se define explícitamente tipoNull, se establece de acuerdo al criterio predeterminado para valores nulos.

Ejemplo 1

En este ejemplo se creará la base de datos Ejemplo y en ella se definirá el tipo de datos RUC de tipo char(11) el cual no permitirá valores NULL.

```
USE master  
GO  
CREATE DATABASE Ejemplo
```

```

ON PRIMARY
    (NAME          = 'Ejem_Data',
     FILENAME     = 'C:\Program Files\Microsoft SQL
Server\MSSQL\data\EjemData.Mdf',
     SIZE         = 20Mb,
     MAXSIZE      = 40Mb,
     FILEGROWTH   = 2Mb)

LOG ON
    (NAME          = 'Ejem_Log',
     FILENAME     = 'C:\Program Files\Microsoft SQL
Server\MSSQL\data\EjemLog.Ldf',
     SIZE         = 5Mb,
     MAXSIZE      = 10Mb,
     FILEGROWTH   = 1Mb)

GO
Use Ejemplo
GO
Sp_Addtype RUC, 'CHAR(11)', 'NOT NULL'
GO

```

Para verificar que el tipo de dato se ha creado exitosamente, ejecute la siguiente consulta:

```

Select * From Systypes
GO

```

Ejemplo 2

En este ejemplo se creará el tipo de datos Onomástico de tipo datetime y que permitirá valores NULL.

```

EXEC sp_addtype Onomastico, datetime, 'NULL'
GO

```

Los tipos de datos que se agregan son inscritos en la tabla **systypes**, estos tipos de datos son eliminados con el procedimiento almacenado del sistema **sp_droptype**.

```

sp_droptype [@typename =] 'tipode dato'
Ejemplo:

```

```

Use Ejemplo
GO
Sp_droptype 'Onomastico'
GO
Select * From Systypes
GO
Sp_droptype 'RUC'
GO
Select * From Systypes
GO

```

Empleo de Comandos DDLL (Data Definition Language)

SQL Server 2000 emplea las tablas como objetos de almacenamiento de datos que los usuarios manipulan a través de sus aplicaciones o vía web.

Las tablas son objetos compuestos por una estructura (conjunto de columnas) que almacenan información interrelacionada (filas) acerca de algún objeto en general.

Las tablas se definen para los objetos críticos de una base de datos, por ejemplo **CLIENTES** y su estructura estaría conformada por cada uno de los atributos que se requieran de los clientes para poder obtener información de ellos, como por ejemplo: nombres, direcciones, teléfonos, celular, ruc, etc.

Cada uno de estos atributos tiene un tipo de dato definido y además la tabla debe permitir asegurar que cada código de producto es único en la misma, para asegurarse de no almacenar la información del mismo cliente dos veces.

Las tablas suelen estar relacionadas entre sí, para facilitar el hecho de consultas entre múltiples tablas.

Podemos distinguir los siguientes tipos de tablas:

Tablas del Sistema

La información usada por SQL Server y sus componentes son almacenadas en tablas especiales denominadas como **tablas del sistema**. Estas tablas no deben alterarse directamente por el usuario

Si desea obtener información almacenada en las tablas del sistema debe usar:

- Información de la vista esquema (*schema view*).
- Procedimientos Almacenados de sistema.
- Instrucciones Transact-SQL y funciones.
- SQL-DMO.
- Catálogo de funciones API.

Las tablas del sistema almacenan información, llamada Metadata, acerca del sistema y de los objetos de las bases de datos. Todas las tablas del sistema comienzan con el prefijo SYS.

Ejemplo:

`SELECT * FROM SYSUSUARIOS`

Tablas del Usuario

Permanentes

Son las tablas donde se almacena la información que los usuarios utilizan para sus operaciones. Esta información existirá hasta que se elimine explícitamente.

Temporales

Estas son tablas similares a las permanentes que se graban en tempdb, y son eliminadas automáticamente cuando ya no son usadas.

Hay dos tipos de tablas temporales, locales y globales, difieren una de la otra en sus nombres, su visibilidad y su ámbito de vida.

- **Tablas Temporales Locales.** El primer carácter del nombre de #, su visibilidad es solamente para la conexión actual del usuario y son eliminadas cuando el usuario se desconecta.
- **Tablas Temporales Globales.** Su nombre comienza con ##, su visibilidad es para cualquier usuario, y son eliminadas luego que todos los usuarios que la referencian se desconectan del SQL Server.

Creación de tablas

Cuando se crea una tabla debe asignarle un nombre a la misma, un nombre a cada columna además de un tipo de datos y de ser necesaria una longitud.

Adicional a las características antes mencionadas, SQL Server 2000 nos brinda la posibilidad de implementar columnas calculadas, definiéndolas como fórmulas.

Los nombres de las columnas deben ser únicos en la tabla

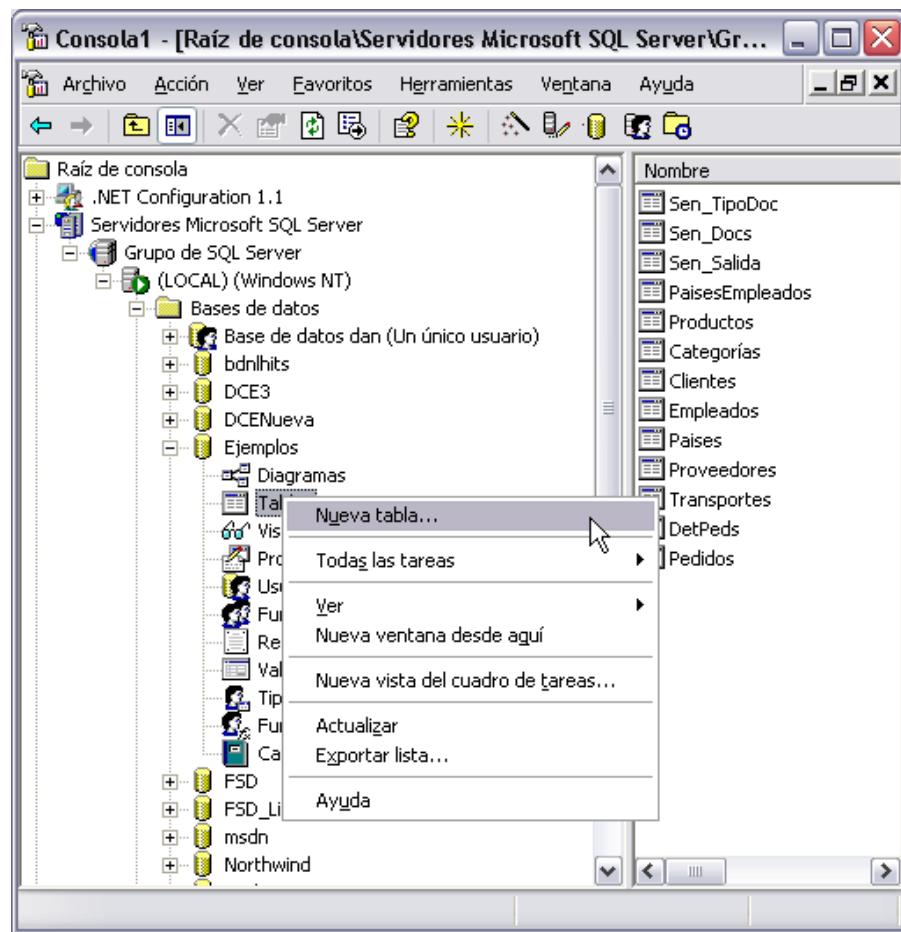
Consideraciones al crear tablas

- billones de tablas por base de datos
- 1024 columnas por tabla
- 8060 es el tamaño máximo de registro (sin considerar datos image, text y ntext)
- Al momento de definir una columna se puede especificar si la columna soporta o no valores NULL.

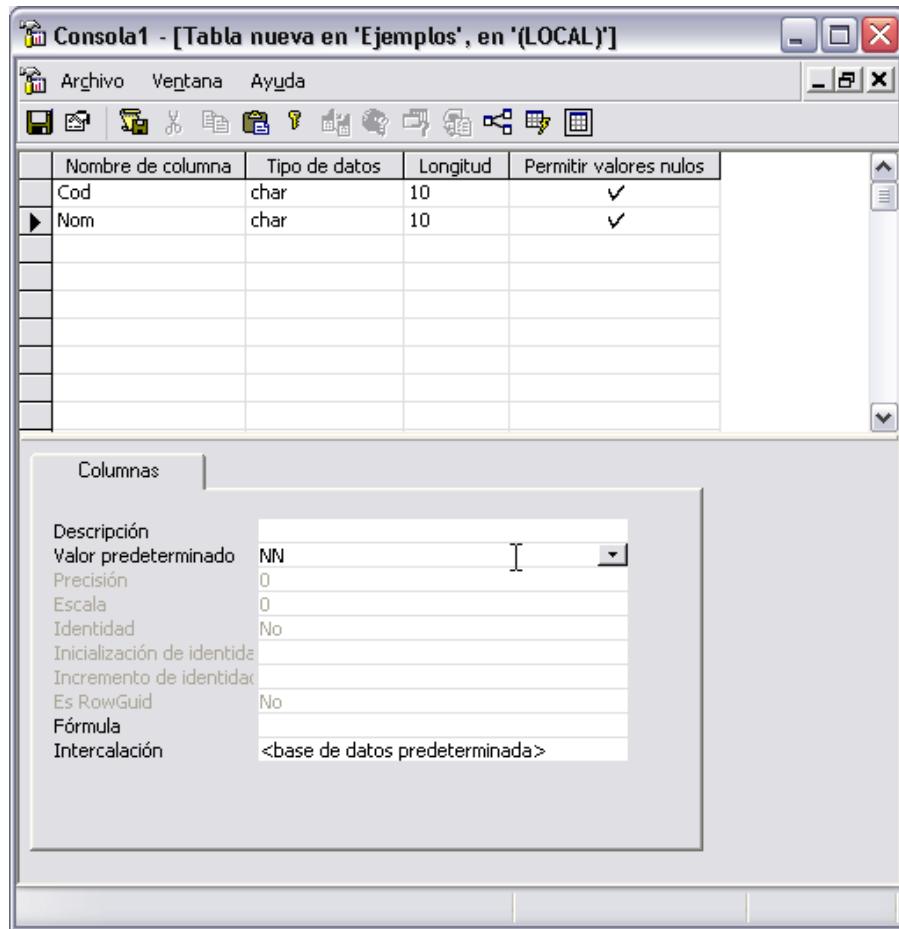
Para crear tablas debe utilizar la sentencia CREATE TABLE, cuya sintaxis es la siguiente:

```
CREATE TABLE <Nombre de Tabla>
( Nom_Columna1  Tipo_de_Dato      [NULL | NOT NULL],
  Nom_Columna2  Tipo_de_Dato      [NULL | NOT NULL],
  Nom_Columna3  As formula ... )
GO
```

También puede crear sus tablas desde el Administrador Empresarial, para ello extienda la carpeta Tablas de la base de datos donde creará la tabla, haga clic derecho y seleccione Nueva Tabla, tal como lo indica la siguiente representación:



Aparecerá la siguiente caja de diálogo, complete de acuerdo a la representación:

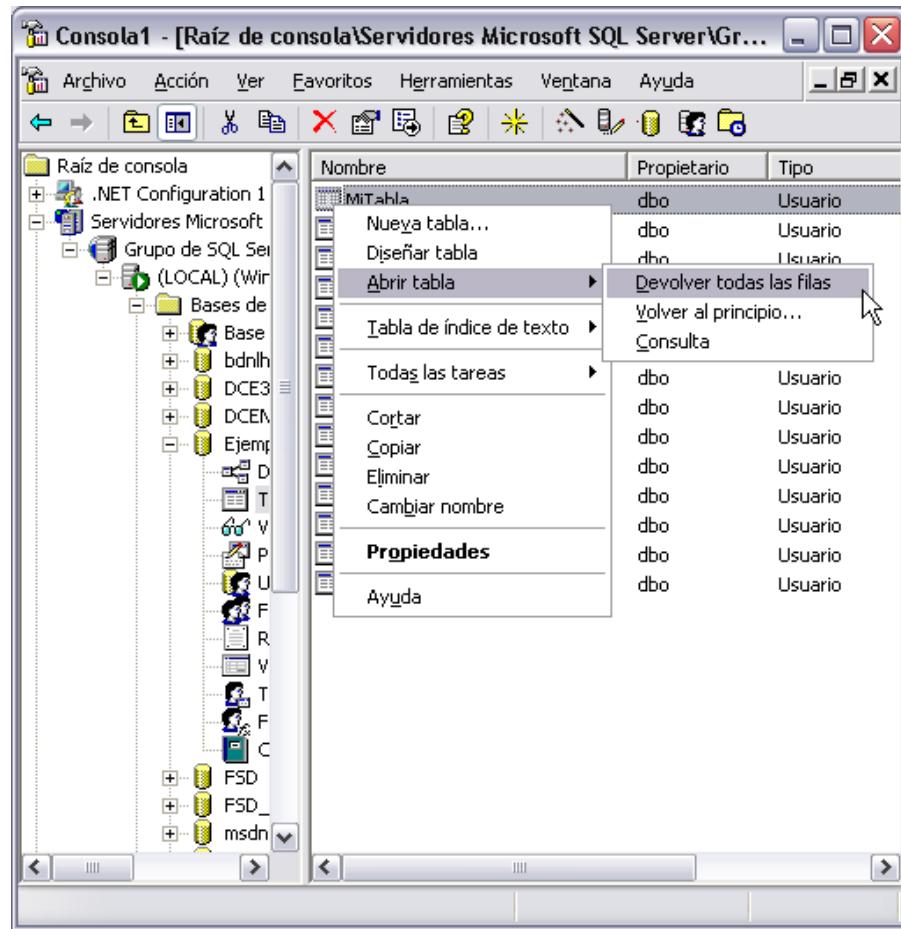


Cuando finalice pulse el icono de grabar y asigne el nombre de acuerdo a la representación:



Luego de pulsar Aceptar, pulse la combinación Ctrl-F4 y podrá observar que el ícono correspondiente a esta nueva tabla aparece en el panel de la derecha.

Para agregar los registros de prueba de esta tabla, haga clic derecho sobre la tabla DemoTabla, seleccione la opción Open table y luego un clic en Return all rows, tal como lo muestra la siguiente figura:



Agregar unos registros al finalizar pulse Ctrl-F4.

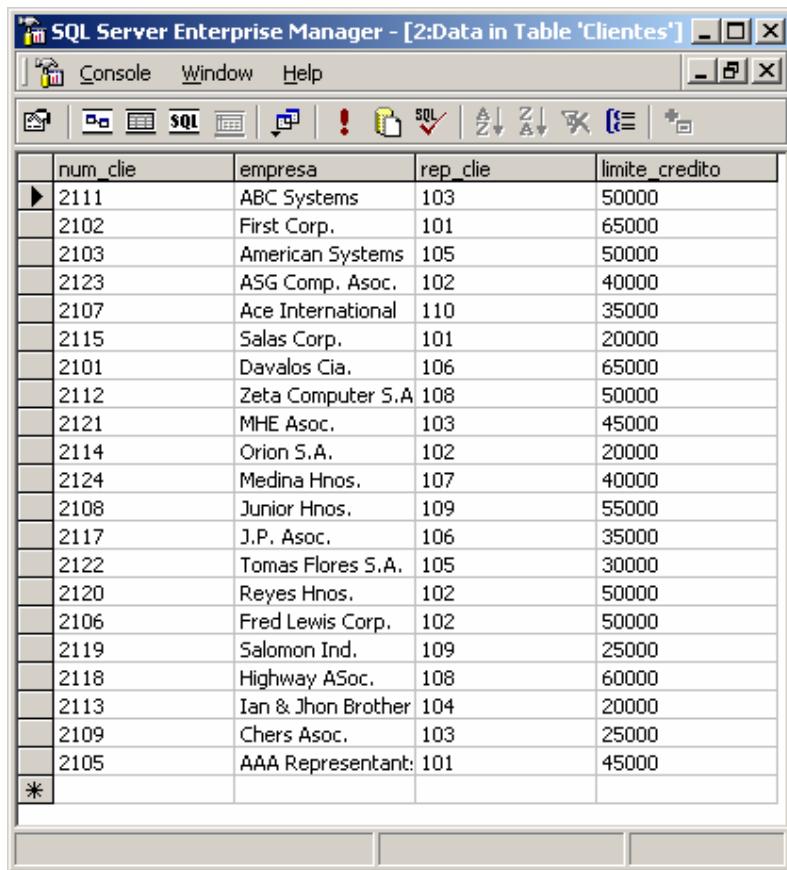
Ejercicios:

En la base de datos Ejemplo, crear las siguientes tablas:

CLIENTES

Nombre de Columna	Tipo de dato	Permite NULL
num_clie	integer	NOT NULL
empresa	varchar(20)	NOT NULL
rep_clie	Integer	NULL
limite_credito	Money	NULL

Agregar los siguientes registros a la tabla Clientes:



The screenshot shows the 'SQL Server Enterprise Manager - [2:Data in Table 'Clientes']' window. The table 'Clientes' has four columns: num_clie, empresa, rep_clie, and limite_credito. The data grid contains 21 rows of client information, starting with num_clie 2111 and ending with num_clie 2105. The 'rep_clie' column contains various employee IDs like 103, 101, 105, etc. The 'limite_credito' column contains monetary values ranging from 20000 to 65000. The bottom of the grid shows an asterisk (*) indicating more data.

	num_clie	empresa	rep_clie	limite_credito
▶	2111	ABC Systems	103	50000
	2102	First Corp.	101	65000
	2103	American Systems	105	50000
	2123	ASG Comp. Asoc.	102	40000
	2107	Ace International	110	35000
	2115	Salas Corp.	101	20000
	2101	Davalos Cia.	106	65000
	2112	Zeta Computer S.A	108	50000
	2121	MHE Asoc.	103	45000
	2114	Orion S.A.	102	20000
	2124	Medina Hnos.	107	40000
	2108	Junior Hnos.	109	55000
	2117	J.P. Asoc.	106	35000
	2122	Tomas Flores S.A.	105	30000
	2120	Reyes Hnos.	102	50000
	2106	Fred Lewis Corp.	102	50000
	2119	Salomon Ind.	109	25000
	2118	Highway ASoc.	108	60000
	2113	Ian & Jhon Brother	104	20000
	2109	Chers Asoc.	103	25000
	2105	AAA Representant.	101	45000
*				

Al terminar pulse CTRL-F4 con ello los registros permanecerán en la tabla.

RepVentas

Nombre de Columna	Tipo de dato	Permite NULL
num_empl	integer	NOT NULL
nombre	varchar(15)	NOT NULL
edad	integer	
oficina_rep	integer	

titulo	varchar(10)	
contrato	date	NOT NULL
director	integer	
cuota	money	
ventas	money	NOT NULL

También podemos crear tablas a partir de sentencias del Transact para ingrese al Analizador de Consultas y ejecute las siguientes instrucciones:

Use Ejemplo

GO

CREATE TABLE Oficinas

```
( oficina      integer not null,
  ciudad       varchar(15) not null,
  region       varchar(10) not null,
  dir          integer,
  objetivo     money,
  ventas       money not null)
```

GO

PEDIDOS

Nombre de Columna	Tipo de dato	Permite NULL
num_pedido	integer	NOT NULL
fecha_pedido	datetime	NOT NULL
clie	Integer	NOT NULL
rep	integer	
fab	char(3)	NOT NULL
producto	char(5)	NOT NULL
cant	integer	NOT NULL
importe	money	NOT NULL

PRODUCTOS

Nombre de Columna	Tipo de dato	Permite NULL
id_fab	char(3)	NOT NULL
id_producto	char(5)	NOT NULL
descripcion	varchar(20)	NOT NULL
precio	Money	NOT NULL
existencias	Integer	NOT NULL

Nota: Una vez que terminó de crear las tablas ejecute el script AgregaDatos.sql para poder poblar la información de las tablas.

Modificación de la estructura de las tablas

Con SQL Server 2000 se puede modificar la estructura de las tablas, se podrá agregar, eliminar o modificar las características de las columnas de la tabla.

Para demostrar el empleo de estas instrucciones emplearemos una tabla de prueba a la cual le daremos la siguiente estructura:

```
Use Ejemplo
```

```
GO
```

```
Create Table Prueba
```

```
( cod      char(1)    NOT NULL,  
  nom      char(20)   NULL,  
  pat      varchar(20) NOT NULL,  
  mat      varchar(20) NOT NULL)
```

```
GO
```

Verificar la creación de la tabla con la siguiente instrucción:

```
Sp_Help Prueba
```

```
GO
```

Ahora modificaremos el campo nom para asignarle como tipo de datos varchar con la misma longitud y que no permita valores NULL

```
ALTER TABLE Prueba
```

```
ALTER COLUMN nom varchar(20) NOT NULL
```

```
GO
```

Verifique empleando:

```
Sp_Help Prueba
```

```
GO
```

Luego agregaremos un campo llamado Sueldo de tipo money:

```
ALTER TABLE Prueba
```

```
ADD Sueldo money
```

```
GO
```

Una observación es que cuando agrega una columna con ALTER TABLE no puede utilizar NOT NULL, salvo que emplee la propiedad IDENTITY en una columna.

Verifique la modificación de la tabla:

```
Sp_Help Prueba
```

```
GO
```

Agregar la columna fecha_nac, de tipo datetime:

```
ALTER TABLE Prueba
```

```
ADD fecha_nac datetime
```

```
GO
```

```
Sp_Help Prueba  
GO
```

Ahora eliminaremos la columna sueldo:

```
ALTER TABLE Prueba  
DROP COLUMN sueldo  
GO
```

```
Sp_Help Prueba  
GO
```

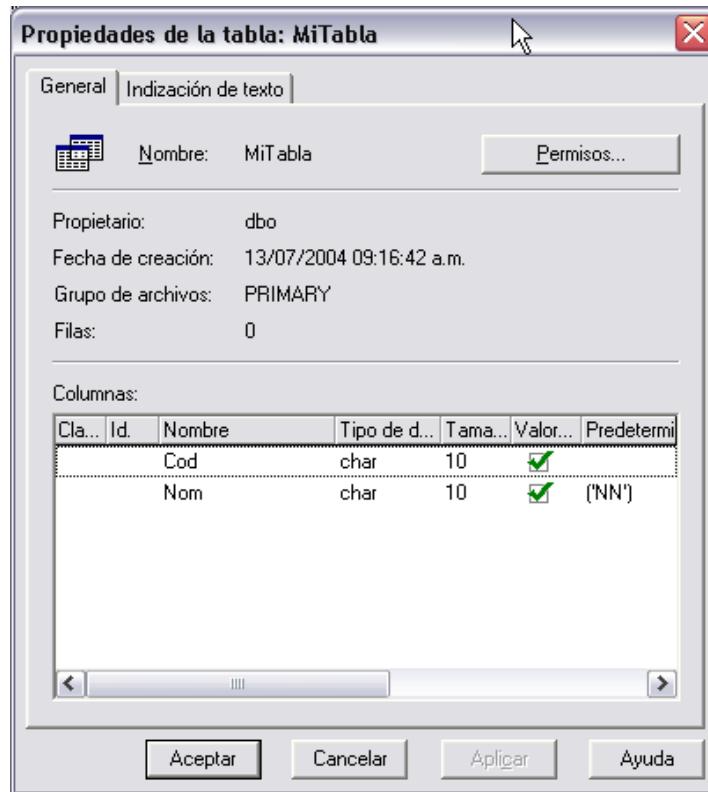
Una observación importante, antes de eliminar una columna deberá eliminar los índices basados en esa columna.

Ahora eliminaremos la columna cod:

```
ALTER TABLE Prueba  
DROP COLUMN cod  
GO
```

Otra de las formas de modificar la estructura de una tabla es desde el Administrador Empresarial, para ello haga clic derecho sobre la tabla a modificar y seleccione la opción **Diseñar Tabla**, tendrá una presentación similar a la que utilizó al momento de crear la tabla.

Si tan sólo desea observar la estructura haga doble clic sobre la tabla y aparecerá la siguiente presentación:



Valores autogenerados para las columnas

En SQL Server 2000 se puede definir columnas que obtengan valores generados por el sistema, para ello podemos hacer uso de:

Propiedad Identity

Permite generar valores secuenciales del sistema, este tipo de valores pueden ser utilizados en columnas que serán empleadas como primary key.

Para emplear esta propiedad debe especificar un valor de inicio y uno de incremento. Recuerde que este tipo de columnas no son editables.

Ejemplo:

```
USE EJEMPLO
GO
ALTER TABLE Prueba
ADD COLUMN cod integer Identity(1,1) NOT NULL
GO
```

Para comprobar la generación de los valores ejecute la siguiente secuencia de comandos:

```
USE EJEMPLO
```

```
GO  
INSERT PRUEBA VALUES ('JOSE', 'ROJAS', 'CALDERON',1000)  
GO  
INSERT PRUEBA VALUES ('ANA MARIA', 'SALAS', 'GUILLEN',1000)  
GO  
SELECT COD, NOM, PAT, MAT, SUELDO FROM PRUEBA  
GO
```

Para ver información sobre la columna IDENTITY puede utilizar las funciones:

```
Select Ident_Seed('Prueba') /* Retorna el valor de inicio de la columna  
identity */  
GO  
Select Ident_Incr('Prueba') /* Retorna el valor de incremento de la columna  
identity */  
GO
```

Función NEWID y Datos de tipo UNIQUEIDENTIFIER

El tipo de dato uniqueidentifier y la función NEWID trabajan unidas para poder generar valores en el formato GUID.

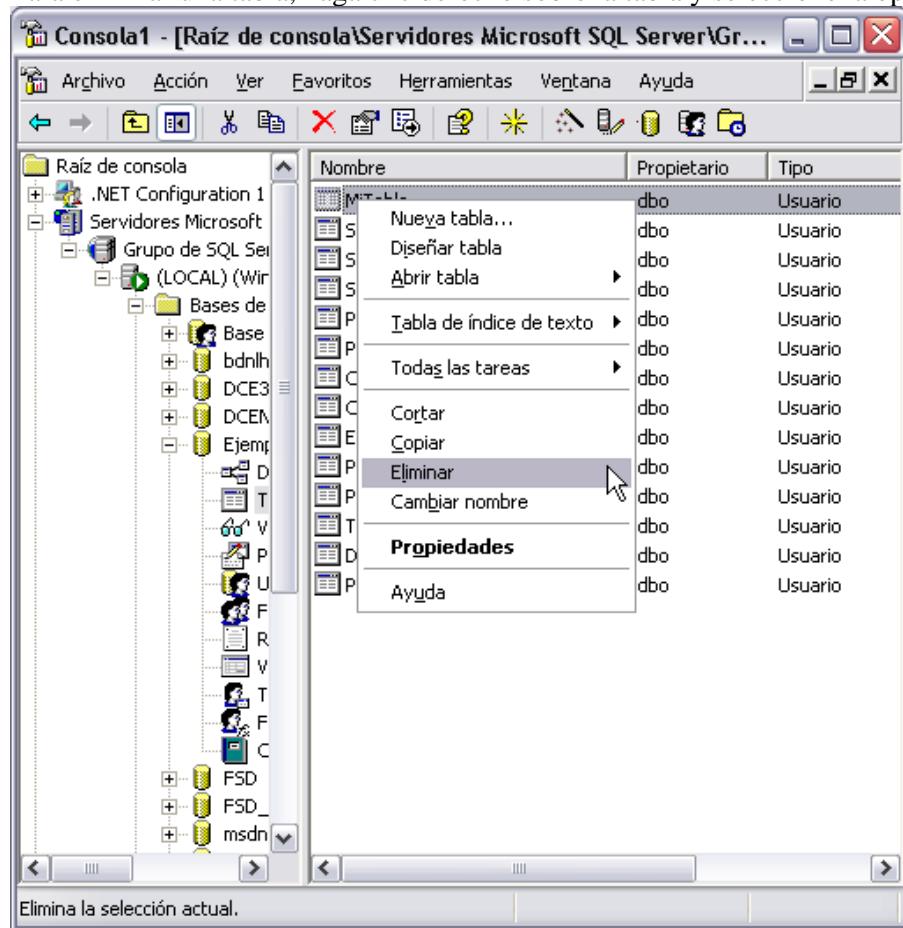
Este tipo de datos no genera valores automáticamente, sino que por el contrario hay que definirle un valor En forma predeterminada que especifique el empleo de la función NEWID.

Para poder observar un ejemplo de lo antes explicado, ejecute la siguiente secuencia de comandos:

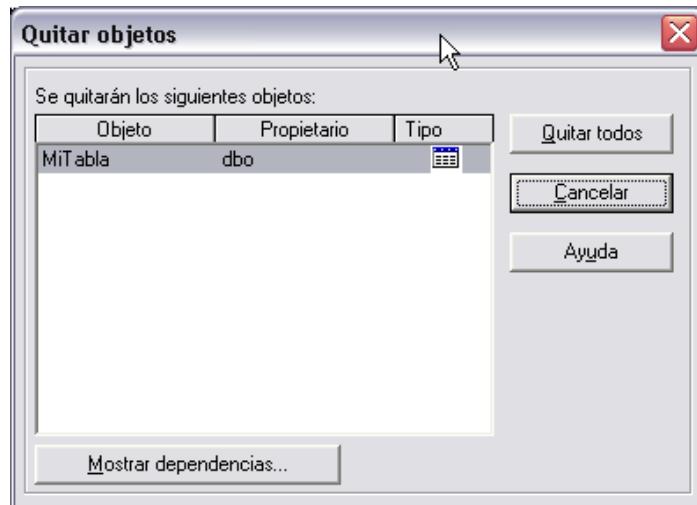
```
CREATE TABLE Prueba2  
( código uniqueidentifier NOT NULL DEFAULT NEWID(),  
  nombre char(20) NOT NULL)  
GO  
  
INSERT Prueba2 (nombre) VALUES ('Mauricio')  
GO  
INSERT Prueba2 (nombre) VALUES ('Gina')  
GO  
INSERT Prueba2 (nombre) VALUES ('Cristina')  
GO  
SELECT * FROM Prueba2  
GO
```

Eliminación de tablas

Para eliminar una tabla, haga clic derecho sobre la tabla y seleccione la opción Eliminar



aparecerá la siguiente caja de diálogo:



Pulse clic sobre *Quitar Todos* y con ello la tabla será retirada de la base de datos.

Otra forma es utilizando la sentencia DROP TABLE cuya sintaxis es la siguiente:

DROP TABLE <Nombre de la Tabla>

Para probar el empleo de esta instrucción utilice la siguiente sentencia:

```
DROP TABLE Prueba2  
GO
```

Compruebe que las tablas Prueba y Prueba2 están eliminadas, con la siguiente instrucción:

```
SELECT NAME FROM SYSOBJECTS WHERE TYPE='U'  
GO
```

Implementar Restricciones

Uno de los principales objetivos de una base de datos relacional es cuidar y controlar la integridad de datos, la cual podría perderse ante operaciones que modifican la información como: INSERT, UPDATE y DELETE.

Por ejemplo se puede perder la integridad de datos ante alguna de las siguientes situaciones:

- Se puede registrar un pedido de un producto no existente
- Podría modificarse los datos existentes son valores incorrectos
- Los cambios a la base de datos podrían aplicarse parcialmente, por ejemplo si se registra un pedido sin actualizar el stock del producto requerido.

SQL Server provee de múltiples medios para controlar la integridad de datos, como por ejemplo:

- Datos Requeridos, es una de las restricciones mas sencillas que especifican que columnas permiten valores nulos y que columnas no. Al momento de definir las columnas de una tabla podrá asignar a cada columna la especificación NULL o NOT NULL.
- Control de validez, permite controlar los valores que se le asignarán a una columna. Por ejemplo en una columna que guarde promedios de estudiantes se podría controlar que el rango de valores se encuentre entre 0 y 10.
- Integridad de entidad, referido a que una clave principal asegura la unicidad de cada registro.
- Integridad referencial, asegura las relaciones entre las claves primarias y claves foráneas, por ejemplo al agregar un pedido controlar que el código de producto que se especifica en el pedido exista en la tabla de productos.
- Reglas comerciales, las modificaciones en la base de datos deben cumplir con ciertos requisitos para poder mantener la información íntegra, por ejemplo en el caso anteriormente mencionado, el producto podría existir pero el stock encontrarse en 0, de tal manera que no debería registrarse el pedido.

SQL Server para poder forzar la integridad de datos propone dos modalidades:

- Integridad Declarativa
Se debe definir el criterio de consistencia como criterio de la definición del objeto.
Para utilizar integridad declarativa se puede emplear constraints, defaults y rules.
- Integridad por Procedimientos
Se pueden escribir procedimientos almacenados y desencadenadores (Triggers) para poder forzar la integridad de datos. Aunque las restricciones muy complejas podrían implementarse a través de lenguajes de programación y otras herramientas clientes.

En esta parte del capítulo revisaremos la integridad declarativa definiéndola a partir de los CONSTRAINTS.

Los CONSTRAINTS son un método estándar de forzar la integridad de datos, aseguran que los datos ingresados a las columnas sean válidos y que las relaciones entre las tablas se mantendrá.

Los constraints pueden definirse al momento de crear la tabla, aunque también es posible hacerlo después de que las tablas se han creado.

Los CONSTRAINTS se ejecutan antes que la información se registre en el log.

Tipos de Constraint

Tipo de Integridad	Tipo de Constraint
Controlar rango de valores sobre las columnas.	DEFAULT
	CHECK
Unicidad de registros y valores únicos.	PRIMARY KEY
	UNIQUE
Referencial	FOREIGN KEY
	CHECK

Definir restricción PRIMARY KEY

```
ALTER TABLE <Nombre de la Tabla>
ADD CONSTRAINT <Nombre del Constraint>
PRIMARY KEY (columna1, ...)
GO
```

Un constraint de tipo PRIMARY KEY asegura la unicidad de cada fila en la tabla, sólo se podrá definir uno por tabla y debemos recordar que no permite valores NULL.

En forma predeterminada crea un índice CLUSTERED.

Ejemplos:

Implementar un PRIMARY KEY Constraint que asegure la unicidad de cada cliente.

```
Use Ejemplo
GO
```

```
Select * From Clientes /* Note el orden de los códigos de clientes */
GO
```

```
ALTER TABLE Clientes
ADD CONSTRAINT PK_Cli_numclie
PRIMARY KEY (num_clie)
GO
```

```
Select * From Clientes /* Note que las filas aparecen ordenadas */
GO
```

Implementar un PRIMARY KEY Constraint que asegure la unicidad de cada representante de ventas.

```
Select * From RepVentas /* Note el orden de los códigos de empleados */  
GO
```

```
ALTER TABLE RepVentas  
ADD CONSTRAINT PK_num_clie  
PRIMARY KEY (num_empl)  
GO
```

```
Select * From RepVentas /* Note que las filas aparecen ordenadas */  
GO
```

Implementar un PRIMARY KEY Constraint que asegure la unicidad de cada oficina.

```
Select * From Oficinas /* Note el orden de los códigos de oficinas */  
GO
```

```
ALTER TABLE Oficinas  
ADD CONSTRAINT PK_Oficina  
PRIMARY KEY (Oficina)  
GO
```

```
Select * From Oficinas /* Note que las filas aparecen ordenadas */  
GO
```

Implementar un PRIMARY KEY Constraint que asegure la unicidad de cada pedido.

```
Select * From Pedidos /* Note el orden de los códigos de pedidos */  
GO
```

```
ALTER TABLE Pedidos  
ADD CONSTRAINT PK_num_pedido  
PRIMARY KEY (num_pedido)  
GO
```

```
Select * From Pedidos /* Note que las filas aparecen ordenadas */  
GO
```

Implementar un PRIMARY KEY Constraint que asegure la unicidad de cada producto.

```
Select * From Productos /* Note el orden de los códigos de producto */  
GO
```

```
ALTER TABLE Productos  
ADD CONSTRAINT PK_fab_prod  
PRIMARY KEY (id_fab, id_producto)  
GO
```

```
Select * From Productos /* Note que las filas aparecen ordenadas */
```

GO

Definir FOREIGN KEY Constraint

```
ALTER TABLE <Nombre de la Tabla>
ADD CONSTRAINT <Nombre del Constraint>
FOREIGN KEY (columna1, ...)
REFERENCES Tabla(columna, ...)
GO
```

Un foreign key constraint permite forzar la integridad de datos manteniendo la relación entre una llave primaria y una llave secundaria.

Para implementar este tipo de característica debemos recordar que el número de columnas y el tipo de datos referenciados en la cláusula FOREIGN KEY debe ser el mismo que el mencionado en la cláusula REFERENCES

Ejemplos:

Implementar un foreign key constraint que asegure que cada vez que asigne un representante de ventas a un cliente este exista.

```
USE Ejemplo
GO
```

```
ALTER TABLE Clientes
ADD CONSTRAINT FK_Cli_RepVentas
FOREIGN KEY (Rep_Clie)
REFERENCES RepVentas(Num_Empl)
GO
```

Implementar un foreign key constraint que asegure que cada vez que a un representante de ventas se le asigne un director, esté se encuentre registrado.

```
ALTER TABLE RepVentas
ADD CONSTRAINT FK_Dir_RepVentas
FOREIGN KEY (Director)
REFERENCES RepVentas(Num_Empl)
GO
```

Implementar un foreign key constraint que asegure que la oficina asignada al representante de ventas se encuentre en la tabla oficinas.

```
ALTER TABLE RepVentas
ADD CONSTRAINT FK_Ofi_Oficinas
FOREIGN KEY (oficina_rep)
REFERENCES Oficinas(Oficina)
GO
```

Implementar un foreign key constraint que verifique el código de director de la oficina.

```
ALTER TABLE Oficinas
ADD CONSTRAINT FK_Direc_RepVentas
FOREIGN KEY (dir)
REFERENCES RepVentas(num_empl)
GO
```

Implementar un foreign key constraint que verifique la existencia del representante de ventas que toma un pedido.

```
ALTER TABLE Pedidos
ADD CONSTRAINT FK_Rep_RepVentas
FOREIGN KEY (rep)
REFERENCES RepVentas(num_empl)
GO
```

Implementar un foreign key constraint que verifique la existencia de los productos que se indican al momento de tomar un pedido.

```
ALTER TABLE Pedidos
ADD CONSTRAINT FK_FabPro_Productos
FOREIGN KEY (fab, producto)
REFERENCES Productos(id_fab, id_producto)
GO
```

Definir CHECK CONSTRAINT

```
ALTER TABLE <Nombre de la tabla>
ADD CONSTRAINT <Nombre del Constraint>
CHECK <Regla a validar>
GO
```

Un Check Constraint restringe a los usuarios la posibilidad de ingresar valores inapropiados a una columna. Este constraint actúa cuando el usuario emplea una instrucción INSERT o UPDATE.

Ejemplos:

Implementar un check constraint que verifique que los códigos de los representantes de ventas sean mayores que 100.

```
ALTER TABLE RepVentas
ADD CONSTRAINT CK_RV_100
CHECK (Num_Empl > 100)
GO
```

Implementar un check constraint que verifique que los códigos de los pedidos sean mayores que 100000.

```
ALTER TABLE Pedidos  
ADD CONSTRAINT CK_Pedidos  
CHECK (num_pedido > 100000)  
GO
```

Implementar DEFAULT CONSTRAINTS

```
ALTER TABLE <Nombre de la tabla>  
ADD CONSTRAINT <Nombre del constraint>  
DEFAULT <Valor En forma predeterminada>  
FOR <columna>  
GO
```

Estos constraints trabajan al momento de utilizar la función INSERT y asignan un valor automáticamente a la columna que no se le asignó.

Ejemplo:

Asignar un valor en forma predeterminada a la columna DIRECTOR de la tabla que almacena los datos de los representantes de ventas haciendo que el código En forma predeterminada sea 106.

```
ALTER TABLE RepVentas  
ADD CONSTRAINT DF_RV_Director  
DEFAULT 106  
FOR Director  
GO
```

Como parte final de esta implementación emplearemos un conjunto de instrucciones para tratar de modificar la información de las distintas tablas y veremos como los constraints implementados realizan su trabajo.

Para ello ejecute las siguientes instrucciones desde el Analizador de Consultas

```
/* Intentemos agregar un cliente con el código 2113 */  
Insert Clientes Values (2113, 'Amago Sys.', 103, 15000)  
GO  
/* La sentencia falla debido a que el primary constraint 'PK_Cli_numclie'  
le impide incluir códigos duplicados */  
  
/* Ahora indicaremos un código de cliente apropiado pero el código de  
representante de ventas inexistente */  
Insert Clientes Values (3000, 'Amago Sys.', 250, 15000)  
GO  
  
/* En este caso el error se produce debido a que el foreign key constraint  
'FK_Cli_RepVentas' a detectado que el código del representante de ventas  
es inexistente */  
  
/* Ahora agregaremos un representante de ventas sin indicar el código de su  
director */  
Insert RepVentas Values (450, 'Karem Vigo', 33, 22, 'Rep.Ventas', '3/5/1991',  
DEFAULT, DEFAULT, 12000)  
GO  
  
Select num_empl, nombre, director From RepVentas Where num_empl = 450  
GO  
  
/* Luego de ejecutar estas instrucciones observará que el código de director  
asignado a Karem Vigo es el 106, código asignado por el Default Constraint */  
  
/* Ahora intentaremos agregar un representante de ventas con código menor  
que 100*/
```

```
Insert RepVentas Values (50, 'Sofia Quevedo', 33, 22, 'Rep.Ventas', '3/5/1991',
DEFAULT, DEFAULT, 12000)
GO
```

/* En este caso el error se produce debido a que el check constraint
a detectado que el código del representante de ventas es menor que
100 */

Diagrama de Base de Datos

Una vez que hemos terminado de implementar las restricciones especificadas anteriormente y luego que ya tenemos las restricciones funcionando podríamos dar un vistazo al diagrama de la base de datos:

Los diagramas representan gráficamente la estructura de la Base de Datos, podemos ver sus tablas y diseño, además de las relaciones entre ellas. También se convierte en una herramienta gráfica para crear, modificar e implementar integridad y constancia de datos.

Ejemplo:

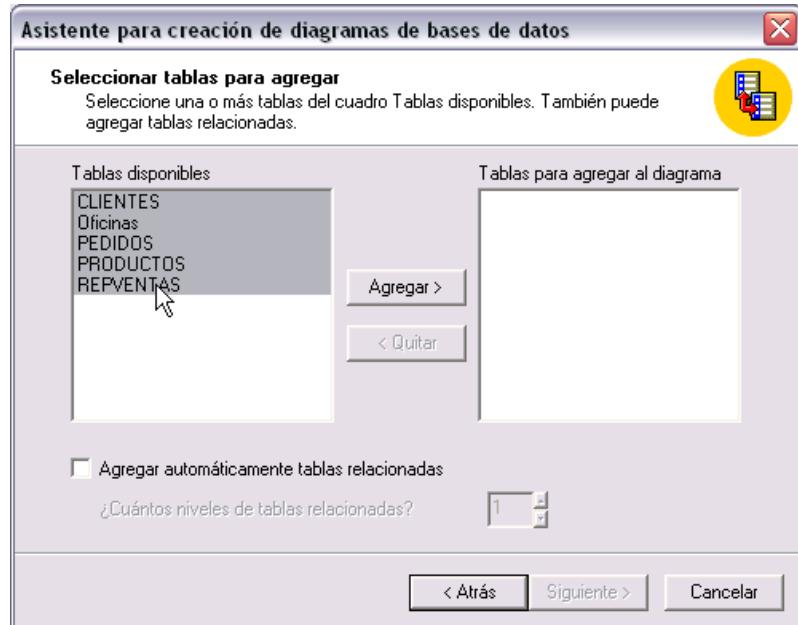
1. Por lo tanto, haga clic derecho sobre Diagramas y seleccione la opción Nuevo Diagrama de base de datos, tal como lo muestra la figura:



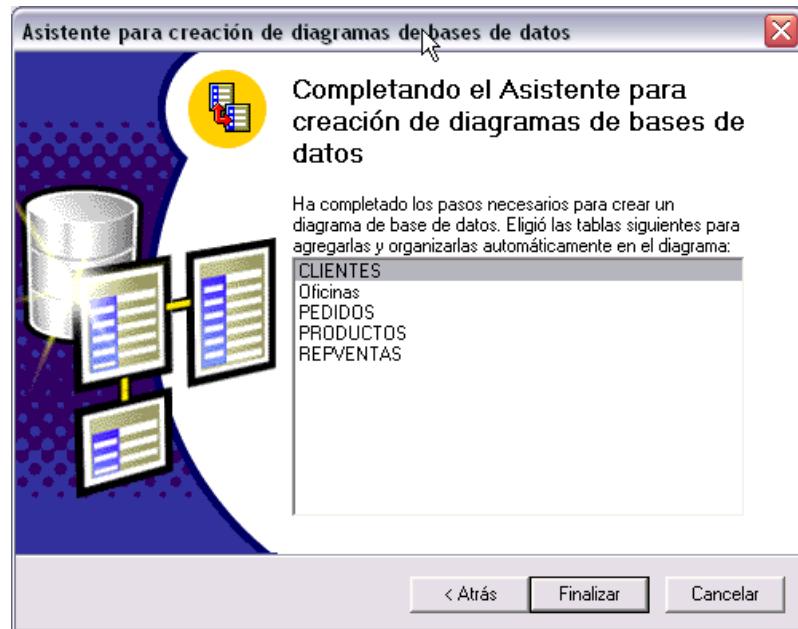
Luego de esto aparecerá un mensaje de bienvenida al asistente para definición del diagrama de la base de datos, tal como lo muestra la siguiente imagen:



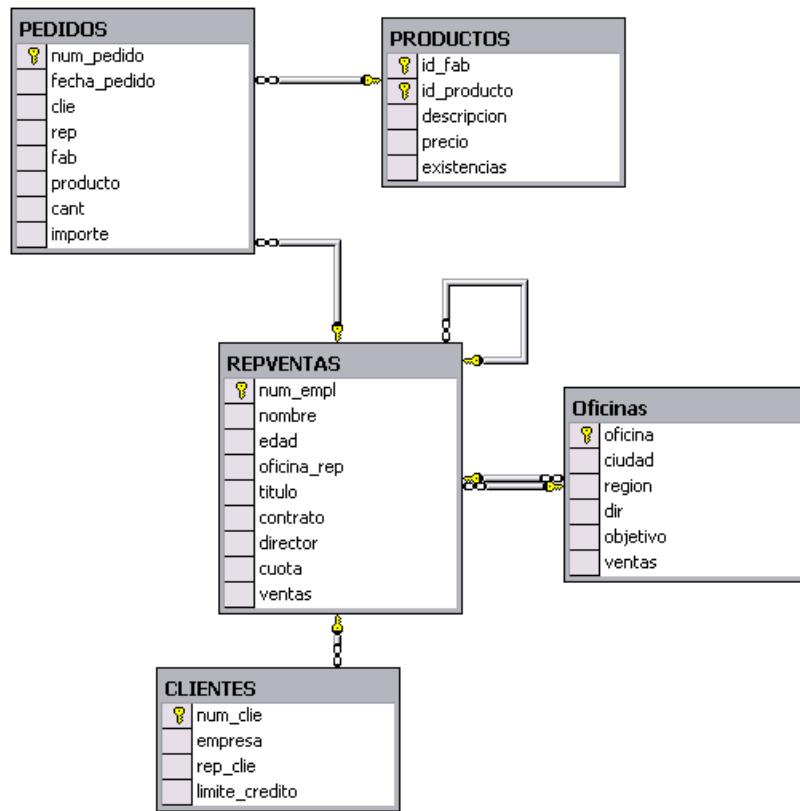
2. Pulse Siguiente y se presentará una caja de diálogo, donde debe seleccionar las tablas que se muestran en la siguiente representación:



3. Luego de pulsar el botón Agregar y Siguiente, aparecerá la siguiente pantalla:



Luego de pulsar Finalizar tendrá la siguiente representación en pantalla:



Recuperar Información

Objetivos:

Conocer los comandos DML

Realizar JOINS

Conocer la funcionalidad de los Desencadenadores

Temas:

- Sentencia SELECT
- Sentencia INSERT
- Sentencia UPDATE
- Sentencia DELETE
- Recuperar información de dos o más tablas
- Desencadenadores

Uno de los principales motivos por el cual se guarda información, es por que posteriormente la vamos a consultar, una de las principales razones por las cuales las bases de datos relacionales lograron gran aceptación fue por la forma tan sencilla de lograr acceder a los datos.

Y como parte de estas facilidades para poder realizar consultas, encontramos a la sentencia SELECT.

Select

Recupera información de la Base de Datos y permite la selección de una o más filas o columnas de una o muchas tablas. La sintaxis completa de la instrucción SELECT es compleja, sus cláusulas principales pueden ser resumidas de la siguiente manera.

```
SELECT lista_cols  
[INTO nueva_tabla]  
FROM tabla_origen  
[WHERE condición]
```

```
[GROUP BY columna1,...]  
[HAVING condición]  
[ORDER BY columna1, ... [ASC][DESC]]
```

lista_cols Específica las columnas a ser devueltas por el query.

Tener en cuenta las siguientes consideraciones:

- La lista de select recupera y muestra las columnas en el orden especificado.
- Separar los nombres de columnas con comas, excepto la última columna.
- Usar un asterisco en la lista de select para recuperar todas las columnas de la tabla.

INTO nueva_tabla Define la creación de una nueva tabla a partir de la respuesta a la consulta especificada. Esta operación no es registrada en el log.

FROM Determina la tabla o tablas de donde se muestra la información.

WHERE Establece un criterio de selección de filas

GROUP BY Establece la lista de columna por las cuales se agrupara la información.

HAVING Permite filtrar los grupos generados por GROUP BY

ORDER BY Permite ordenar la información de acuerdo a los requerimientos.

Ejemplos

En los siguientes ejemplos veremos, el uso del SELECT, la creación de campos calculados, el uso de operadores de funciones agregadas y de group by.

1. Mostrar las lista de las oficinas de ventas con sus objetivos y ventas reales

```
USE Ejemplo  
GO  
SELECT CIUDAD, OBJETIVO, VENTAS  
FROM Oficinas  
GO
```

2. Mostrar los nombres, oficinas y fechas de contrato de los vendedores

```
SELECT NOMBRE, OBJETIVO, VENTAS  
FROM REPVENTAS  
GO
```

3. Mostrar el nombre, cuota y ventas del empleado de código 107

```
SELECT NOMBRE, CUOTA, VENTAS  
FROM REPVENTAS  
WHERE NUM_EMPL = 107
```

GO

Nótese que en esta última consulta se ha empleado la cláusula WHERE para restringir el número de filas a devolver, a diferencia de la primera que le devolvía todas las filas.

4. Mostrar el monto de ventas promedio de los vendedores

```
SELECT AVG(VENTAS)
FROM REPVENTAS
GO
```

En este último ejemplo se obtiene un único valor que representa una pequeña tabla, aunque conste de una sola fila y una sola columna. Este valor es el resultado de sumar todos los valores del campo VENTAS y dividirlo entre el número de filas.

5. Mostrar los nombres y fechas de contratos de los vendedores que superaron la barrera de los 500,000

```
SELECT NOMBRE, CONTRATO
FROM REPVENTAS
WHERE VENTAS>500000
GO
```

En este último ejemplo no se obtienen filas, con lo cual queda demostrado que no siempre las consultas deben devolver filas, esto representa que ningún registro cumplió con la condición expresada en la cláusula WHERE.

6. Mostrar la CIUDAD, REGION y el IMPORTE de por encima o por debajo del OBJETIVO

```
SELECT CIUDAD, REGION, (VENTAS - OBJETIVO) As SITUACION
FROM OFICINAS
GO
```

Nótese que en el ejemplo se emplea la cláusula **As** que permite asignar un encabezado de columna al campo calculado (VENTAS – OBJETIVO).

De los resultados podemos observar que las ciudades de Chicago y Denver aun se encuentran por debajo del objetivo, mientras que las demás oficinas ya superaron el objetivo.

7. Mostrar el valor del inventario para cada producto

```
Select Id_Fab, Id_Producto, Descripcion, (Existencias*Precio) As Valor
From Productos
GO
```

Similar al ejemplo anterior se emplea un campo calculado al cual se le asigna un encabezado a través del empleo de la cláusula As.

8. Muestra que sucedería si se eleva el monto de la cuota decada vendedor en un 3% de sus ventas anuales

```
SELECT NOMBRE, CUOTA, (CUOTA*1.03) As CUOTAPROYECTADA  
FROM REPVENTAS  
GO
```

9. Mostrar el nombre, mes y año de contrato de cada representante de ventas

```
Select Nombre, Month(Contrato) As Mes, Year(Contrato) As Año  
FROM REPVENTAS  
GO
```

10. Mostrar los campos CIUDAD y VENTAS separados por la cadena ' tiene ventas de '

```
SELECT CIUDAD, ' tiene ventas de ', VENTAS  
FROM OFICINAS  
GO
```

11. Mostrar todos los campos de la tabla CLIENTES

```
SELECT * FROM CLIENTES  
GO
```

El carácter * permite recuperar todos los campo y todas las filas de la tabla que se está consultando.

12. Mostrar todos los campos de la tabla OFICINAS, así como también una columna que indique si se alcanzo o no el objetivo.

```
SELECT *, (VENTAS - OBJETIVO) as SITUACION  
FROM OFICINAS  
GO
```

13. Comparar las siguientes consultas:

```
SELECT DIR FROM OFICINAS  
GO
```

```
SELECT DISTINCT DIR FROM OFICINAS  
GO
```

Note que en el primer caso se muestran los 5 códigos de director uno por cada fila existente en la tabla, mientras que con la segunda no se muestran los duplicados.

14. Mostrar las oficinas que han superado el OBJETIVO trazado:

```
SELECT CIUDAD, VENTAS, OBJETIVO  
FROM OFICINAS  
WHERE VENTAS > OBJETIVO  
GO
```

15. Mostrar los vendedores que están dirigidos por Jorge Castro (código 104)

```
SELECT NOMBRE, VENTAS  
FROM REPVENTAS
```

```
WHERE DIRECTOR = 104  
GO
```

16. Muestre a los vendedores contratados antes de 1988

```
SELECT NOMBRE  
FROM REPVENTAS  
WHERE CONTRATO < '01-01-88'  
GO
```

17. Mostrar las oficinas que están por debajo del 80% del objetivo

```
SELECT CIUDAD, VENTAS, OBJETIVO  
FROM OFICINAS  
WHERE VENTAS < (0.8 * OBJETIVO)  
GO
```

18. Mostrar las oficinas que no están a cargo de Paola Marin

```
SELECT CIUDAD, DIR  
FROM OFICINAS  
WHERE DIR <> 108  
GO
```

19. Mostrar a los vendedores que superaron sus cuotas

```
SELECT NOMBRE  
FROM REPVENTAS  
WHERE VENTAS > CUOTA  
GO
```

20. Mostrar los clientes que tiene un límite de crédito entre 45000 y 60000

```
SELECT EMPRESA, LIMITE_CREDITO FROM CLIENTES  
WHERE LIMITE_CREDITO BETWEEN 45000 AND 60000  
GO
```

21. Mostrar a los vendedores que trabajan en New York, Denver, o Atlanta

```
SELECT NOMBRE, CUOTA, VENTAS FROM REPVENTAS  
WHERE OFICINA REP IN (11, 13, 22)  
GO
```

22. Mostrar a los clientes cuya razón social comienza con S

```
SELECT EMPRESA, LIMITE_CREDITO  
FROM CLIENTES  
WHERE EMPRESA LIKE 'S%'  
GO
```

23. Mostrar a los clientes cuya razón social incluye una Y en su nombre

```
SELECT EMPRESA, LIMITE_CREDITO  
FROM CLIENTES
```

```
WHERE EMPRESA LIKE '%Y%'  
GO
```

24. Mostrar los vendedores que no tiene asignada una oficina

```
SELECT NOMBRE  
FROM REPVENTAS  
WHERE OFICINA_REP IS NULL  
GO
```

25. Mostrar los vendedores que tienen asignada una oficina

```
SELECT NOMBRE  
FROM REPVENTAS  
WHERE OFICINA_REP IS NOT NULL  
GO
```

26. Mostrar a los vendedores que tienen ventas por debajo de sus cuotas o ventas menores a 300000

```
SELECT NOMBRE, CUOTA, VENTAS  
FROM REPVENTAS  
WHERE VENTAS < CUOTA  
      OR VENTAS < 300000  
GO
```

27. Mostrar a los vendedores que tienen ventas por debajo de sus cuotas y ventas menores a 300000

```
SELECT NOMBRE, CUOTA, VENTAS  
FROM REPVENTAS  
WHERE VENTAS < CUOTA  
      AND VENTAS < 300000  
GO
```

28. Mostrar la información de las oficinas ordenadas por Región

```
SELECT *  
FROM OFICINAS  
ORDER BY REGION  
GO
```

29. Mostrar las oficinas ordenadas por las ventas en forma descendente:

```
SELECT CIUDAD, REGION, VENTAS  
FROM OFICINAS  
ORDER BY VENTAS DESC  
GO
```

30. Mostrar las oficinas organizadas en forma descendente por el rendimiento de ventas

```
SELECT CIUDAD, REGION, (VENTAS-OBJETIVO) AS RENDIMIENTO  
FROM OFICINAS  
ORDER BY 3 DESC
```

GO

31. Mostrar las oficinas organizadas por REGION y dentro de cada región por el rendimiento de las VENTAS en forma descendente.

```
SELECT CIUDAD, REGION, (VENTAS-OBJETIVO) AS RENDIMIENTO  
FROM OFICINAS  
ORDER BY REGION, 3 DESC  
GO
```

32. Indicar la cuota promedio y las ventas promedio de los vendedores

```
SELECT AVG(CUOTA), AVG(VENTAS)  
FROM REPVENTAS  
GO
```

Nótese que la función AVG primero suma todos los valores de la columna especificada en el argumento y luego divide este total entre el número de filas.

33. Mostrar la suma total de las cuotas y de las ventas de todos los vendedores

```
SELECT SUM(CUOTA), SUM(VENTAS)  
FROM REPVENTAS  
GO
```

Nótese que la función SUM suma todos los valores de la columna especificada en el argumento.

34. Mostrar el importe promedio del cliente de código 103

```
SELECT AVG(IMPORTE)  
FROM PEDIDOS  
WHERE CLIE = 2103  
GO
```

35. Mostrar el mayor y menor monto de cuotas

```
SELECT MIN(CUOTA), MAX(CUOTA)  
FROM REPVENTAS  
GO
```

Nótese que la función MIN devuelve el menor valor de los datos almacenados en la columna especificada en el argumento, mientras que MAX devuelve el mayor valor.

36. Mostrar el número de clientes que existen.

```
SELECT COUNT(EMPRESA)  
FROM CLIENTES  
GO
```

Nótese que la función COUNT cuenta los registros en base al campo especificado en el argumento.

37. Mostrar cuantos pedidos superaron el importe de los 25000

```
SELECT COUNT(IMPORTE)
FROM PEDIDOS
WHERE IMPORTE > 25000
GO
```

38. Mostrar cual es el promedio de pedidos por cada vendedor

```
SELECT REP, AVG(IMPORTE)
FROM PEDIDOS
GROUP BY REP
GO
```

Nótese que la cláusula GROUP BY permite agrupar los valores con la finalidad de aplicarles alguna de las funciones agregadas (COUNT, SUM, AVG, MAX, MIN). En este caso se devolverá un valor promedio de importes por cada uno de los representantes de ventas.

39. Mostrar el rango de cuotas asignadas en cada oficina

```
SELECT OFICINA_REP, MIN(CUOTA), MAX(CUOTA)
FROM REPVENTAS
GROUP BY OFICINA_REP
GO
```

40. Mostrar el número de vendedores asignados a cada oficina

```
SELECT OFICINA_REP, COUNT(*)
FROM REPVENTAS
GROUP BY OFICINA_REP
GO
```

41. Mostrar los valores totales por cada cliente y por cada vendedor

```
SELECT REP, CLIE, SUM(IMPORTE)
FROM PEDIDOS
GROUP BY REP, CLIE
ORDER BY REP
GO
```

42. Mostrar un informe que calcule el total de importes por cada cliente, vendedor ordenados por vendedor y luego por cliente

```
SELECT REP, CLIE, IMPORTE
FROM PEDIDOS
ORDER BY REP, CLIE
COMPUTE SUM(IMPORTE) BY REP, CLIE
COMPUTE SUM(IMPORTE) , AVG(IMPORTE) BY REP
```

Para poder emplear la cláusula COMPUTE debe ordenar primero la información.

43. Mostrar los promedios de ventas que superan los 30000

```
SELECT REP, AVG(IMPORTE)
FROM PEDIDOS
```

```
GROUP BY REP  
HAVING SUM(IMPORTE) > 30000  
GO
```

Nótese que la cláusula HAVING permite filtrar los grupos generados por GROUP BY a diferencia de WHERE que filtra los registros que se agruparían.

Insert

Utilice la sentencia INSERT para agregar registros a una tabla.
La sintaxis reducida puede ser :

```
INSERT [INTO] <Nombre de la Tabla> VALUES (Valor1, ....)  
GO
```

Recuerde que si el valor que intenta agregar a una de las columnas no cumple con alguno de los constraints establecidos la operación abortará inmediatamente.

También es posible agregar múltiples filas a través del siguiente formato:

```
INSERT [INTO] <Nombre de la Tabla>  
SELECT <lista de campos> FROM <Tabla>
```

Ejemplos:

Insertar los siguientes registros a la tabla de Clientes

```
INSERT Clientes Values (500, 'Mauricio Hidalgo', 104, 45000)  
GO  
INSERT Clientes Values (501, 'Gaby Mansilla', 104, 45000)  
GO  
INSERT Clientes Values (502, 'Cristina Donayre', 104, 45000)  
GO  
Select * From Clientes  
GO
```

Update

Esta sentencia nos permite modificar la información de las tablas.

La sintaxis reducida puede ser:

```
UPDATE <Nombre de la Tabla>  
SET    <columna> = <Nuevo Valor>  
[WHERE <condición>]  
GO
```

Recuerde que si la actualización de una fila no cumple con una restricción o regla, infringe la configuración de valores NULL o si el nuevo valor es de un tipo de datos incompatible, se cancela la instrucción, se devuelve un error y no se actualiza ningún registro.

Ejemplos

Actualizar la información del registro del cliente de código 502

```
UPDATE Clientes  
SET empresa = 'Cristina Hidalgo'  
WHERE num_clie = 502  
GO
```

```
Select * From Clientes Where num_clie= 502  
GO
```

A cada código sumarle 500 para los códigos menores que 1000

```
UPDATE Clientes  
SET num_clie = num_clie + 500  
WHERE num_clie < 1000  
GO
```

```
Select * From Clientes  
GO
```

Crear una tabla llamada MejoresCli, con los registros de los clientes con un límite de crédito mayor que 60000, en esta nueva tabla incremente el límite de crédito en un 20%

```
SP_DBOPTION 'EJEMPLO', 'SELECT INTO/BULKCOPY', 'TRUE'  
GO  
SELECT *  
INTO MEJORESCLI  
FROM CLIENTES  
WHERE LIMITE_CREDITO > 60000  
GO  
UPDATE MEJORESCLI  
SET LIMITE_CREDITO = LIMITE_CREDITO * 1.2  
GO  
SELECT * FROM MEJORESCLI  
GO  
SP_DBOPTION 'EJEMPLO', 'SELECT INTO/BULKCOPY', 'FALSE'  
GO
```

Delete

Las instrucciones DELETE y TRUNCATE TABLE remueven filas de una tabla.
La sintaxis de DELETE puede ser:

```
DELETE <Nombre de la tabla>
```

[WHERE <Condición>]

Usar la instrucción DELETE para eliminar una o más filas de una tabla.

Tener en cuenta las siguientes consideraciones:

- El SQL Server borra todas las filas de una tabla a menos que se use la cláusula WHERE.
- Cada fila borrada genera historia en el Log de Transacciones.

Ejemplos:

Eliminar el registro de código 1000 en la tabla de clientes

```
DELETE Clientes  
WHERE Num_Clie = 1000  
GO
```

```
SELECT * FROM CLIENTES  
GO
```

Eliminar los registros cuyo código de cliente es menor que 2000

```
DELETE Clientes  
WHERE Num_Clie < 2000  
GO
```

```
SELECT * FROM CLIENTES  
GO
```

Para eliminar registros puede utilizar también la sentencia TRUNCATE TABLE, que resulta más rápida que DELETE puesto que no genera entradas en el log de transacciones.

Su sintaxis es:

TRUNCATE TABLE <Nombre de la Tabla>

Ejemplo:

```
TRUNCATE TABLE MejoresCli  
GO  
Select * From MejoresCli  
GO
```

Luego de probar este ejemplo elimine la tabla MejoresCli

```
DROP TABLE MejoresCli  
GO
```

Recuperar información de dos o más tablas (Joins)

Para muchas de las consultas que los usuarios realizan sobre la data almacenada en nuestra base de datos es necesario extraer información de más de una tabla, para ello es necesario emplear los **JOINS** que representan una operación producir un conjunto de resultados que incorporen filas y columnas de las tablas referidas en la consulta, esto lo hace basándose en columnas comunes a las tablas.

Cuando se ejecutan los **JOIN**, SQL Server compara los valores de las columnas especificadas fila por fila entonces usa los resultados de la comparación para combinar los valores que califican como nuevas filas.

```
SELECT <lista de columnas>
FROM <tabla o vista >
[INNER | LEFT|RIGHT|FULL [OUTER]] JOIN
<Tabla o Vista > ON <condición>
```

La lista de columnas puede incluir campos de diferentes tablas.

JOIN especifica las tablas involucradas en la consulta.

ON, establece la condición de unión de las tablas, a través de campos comunes.

Cuando se implementa los JOIN, debe tener en cuenta las siguientes consideraciones:

- Especificar las condiciones del JOIN en base a Primary Key y a Foreign Key.
- Si una tabla tiene un Primary Key compuesta, se debe referenciar a la clave entera en la cláusula ON del JOIN de tablas.
- Las columnas comunes a las tablas deben ser del mismo tipo de dato.
- Si dos o más columnas de las diferentes tablas que participan en el JOIN, tienen el mismo nombre, deberá de calificar dichas columnas usando el formato NombreTabla.Nombre.Columna.
- Limitar en lo posible el número de tablas en un JOIN, a más tablas, el SQL Server se tomará más tiempo para resolver la consulta.
- La cláusula LEFT OUTER JOIN nos permite observar todos los registros de la tabla que se referencia a la izquierda en una consulta, completa las filas con NULL en caso no exista un valor almacenado en la tabla de la derecha.
- La cláusula RIGHT OUTER JOIN nos permite observar todos los registros de la tabla que se referencia a la derecha en una consulta, completa las filas con NULL en caso no exista un valor almacenado en la tabla de la izquierda.
- La cláusula FULL OUTER JOIN nos muestra la combinación de todos los registros de la tabla de la izquierda con los registros de la tabla de la derecha.

Ejemplos:

- Mostrar los pedidos indicando el número de pedido, importe, nombre del cliente y el límite de crédito

```
SELECT Num_Pedido, Importe, Empresa, Limite_credito
FROM PEDIDOS INNER JOIN CLIENTES
ON CLIE = NUM_CLIE
GO
```

- Muestra la lista de vendedores con especificación de la ciudad y región a la cual pertenece

```
SELECT Nombre, Ciudad, Región  
FROM REPVENTAS INNER JOIN OFICINAS  
ON OFICINA REP = OFICINA  
GO
```

- Muestra la lista de oficinas con los nombres y títulos de sus directores.

```
SELECT Ciudad, Nombre, Titulo  
FROM OFICINAS INNER JOIN REPVENTAS  
ON DIR = NUM_EMPL  
GO
```

- Muestra la lista de las oficinas con un objetivo superior a 600000

```
SELECT CIUDAD, NOMBRE, TITULO  
FROM OFICINAS INNER JOIN REPVENTAS  
ON DIR = NUM_EMPL AND OBJEIVO > 600000  
GO
```

- Mostrar los pedidos indicando los importes y la descripción de los productos

```
SELECT NUM_PEDIDO, IMPORTE, DESCRIPCION  
FROM PEDIDOS INNER JOIN PRODUCTOS  
ON FAB = ID_FAB AND PRODUCTO = ID_PRODUCTO  
GO
```

- Mostrar los pedidos superiores a 25000 indicando el nombre del vendedor que tomó el pedido y el nombre del cliente

```
SELECT NUM_PEDIDO, IMPORTE, EMPRESA, NOMBRE  
FROM PEDIDOS INNER JOIN CLIENTES  
ON CLIE = NUM_CLIE  
INNER JOIN REPVENTAS ON REP = NUM_EMPL  
AND IMPORTE > 25000  
GO
```

- Mostrar los pedidos superiores a 25000 indicando el nombre del vendedor asignado al cliente y el nombre del cliente

```
SELECT NUM_PEDIDO, IMPORTE, EMPRESA, NOMBRE  
FROM PEDIDOS INNER JOIN CLIENTES  
ON CLIE = NUM_CLIE  
INNER JOIN REPVENTAS ON REP_CLIE = NUM_EMPL  
AND IMPORTE > 25000  
GO
```

Desencadenadores

Un Desencadenador (*Trigger*) es un tipo especial de procedimiento almacenado que se activa de forma controlada por sucesos antes que por llamadas directas. Los desencadenadores (*Triggers*) están asociados a tablas.

Son una gran herramienta para controlar las reglas de negocio más complejas que una simple integridad referencial, los desencadenadores (*Triggers*) y las sentencias que desencadenan su ejecución trabajan unidas como una transacción.

El grueso de instrucciones de la definición del Desencadenador deben ser INSERT, UPDATE o DELETE, aunque se puede utilizar SELECT, no es recomendable ya que el usuario no espera que se le devuelva registros luego de agregar o modificar información.

Los desencadenadores (*Triggers*) siempre toman acción después de que la operación fue registrada en el log.

Para crear un Desencadenador puede utilizar el siguiente formato:

```
CREATE DESENCADENADOR <Nombre del Desencadenador>
ON <Nombre de la Tabla>
FOR <INSERT I UPDATE I DELETE>
AS
      Sentencias....
GO
```

Para graficar con un ejemplo la idea de un Desencadenador implementaremos uno, piense en la siguiente situación:

Al agregar un nuevo pedido a la tabla de PEDIDOS se debe incrementar las ventas del representante que concreto el pedido, así como también debe reducirse el número de existencias.

Para ello debe crear el siguiente Desencadenador:

```
Use Ejemplo
GO
CREATE DESENCADENADOR NuevoPedido
ON Pedidos
FOR INSERT
AS
    UPDATE RepVentas
    SET VENTAS = VENTAS + INSERTED.IMPORTE
    FROM REPVENTAS INNER JOIN INSERTED
    ON REPVENTAS.NUM_EMPL = INSERTED.REP

    UPDATE PRODUCTOS
    SET EXISTENCIAS = EXISTENCIAS - INSERTED.CANT
    FROM PRODUCTOS INNER JOIN INSERTED
    ON PRODUCTOS.ID_FAB = INSERTED.FAB
    AND PRODUCTOS.ID_PRODUCTO = INSERTED.PRODUCTO
```

GO

Para comprobar la ejecución de este Desencadenador ejecute las siguientes sentencias:

/* Antes de ejecutar un INSERT de prueba, mostraremos la información con respecto a un producto */

Select * From Productos Where Id_Fab= 'ACI' AND Id_Producto='41001'
GO

Rpta:

id_fab	id_producto	descripción	precio	existencias
ACI	41001	Articulo Tipo 1	55.00	270

/* Ahora la de un representante de ventas */

Select nombre, cuota, ventas From RepVentas Where num_empl = 104
GO

Rpta:

nombre	cuota	ventas
Jorge Castro	200000	142594

/* Ahora agregaremos un pedido */

Insert Pedidos Values (111000, '5/15/1996', 2101, 104, 'ACI', '41001', 5, 275)
GO

/* Verifique los resultados anteriores */

Select * From Productos Where Id_Fab= 'ACI' AND Id_Producto='41001'
GO

Rpta:

id_fab	id_producto	descripcion	precio	existencias
ACI	41001	Articulo Tipo 1	55.00	265

Select nombre, cuota, ventas From RepVentas Where num_empl = 104
GO

Rpta:

nombre	cuota	ventas
Jorge Castro	200000	142869

Como comprobo al agregar un nuevo pedido automáticamente el Desencadenador funciona y actualiza las ventas para el representantes de ventas y reduce el número de existencias en stock.

Bien en general cuando trabaja con desencadenadores (Triggers), tiene que recordar que los CONSTRAINTS se verifican primero, de cumplirse con los datos solicitados se ejecutará el Desencadenador.

Un Desencadenador para inserción de registros genera automáticamente una tabla en el cache con la información que intenta añadir, esta tablita se denomina **INSERTED** y es a través de esta tabla que se pueden hacer comparaciones en otras tablas.

Un Desencadenador para eliminación de registros genera automáticamente una tabla en el cache con la información que intenta eliminar, esta tablita se denomina **DELETED** y es a través de esta tabla que se pueden hacer comparaciones en otras tablas.

Si se trata de un Desencadenador para actualización se generan ambas tablas **INSERTED** con los nuevos datos y **DELETED** con la información que será reemplazada.

Asignar Roles y/o Permisos – Comandos Dcl (Data Control Language)

Los comandos Data Control Language nos permiten asignar o negar derechos a los usuarios sobre los distintos objetos de la base de datos. Para esto se deben haber definido los usuarios que tendrán acceso a nuestra base de datos.

Los comandos DCL se pueden asignar por sentencias o por objetos.

Antes de empezar con las demostraciones, crearemos algunos inicios de sesión que nos permitan practicar, ejecute la siguiente secuencia de comandos:

```
USE MASTER  
GO  
Sp_AddInicio de sesión 'Usuario01', 'contraseña'  
GO  
Sp_AddInicio de sesión 'Usuario02', 'contraseña'  
GO  
Sp_AddInicio de sesión 'Usuario03', 'contraseña'  
GO  
Sp_AddInicio de sesión 'Usuario04', 'contraseña'  
GO  
Sp_AddInicio de sesión 'Usuario05', 'contraseña'  
GO
```

Luego de crear los inicios de sesión defina a cada uno de ellos como usuarios de la base de datos Ejemplo, realice la siguiente secuencia de comandos:

```
Use Ejemplo
GO
Sp_GrantDBAccess 'Usuario01'
GO
Sp_GrantDBAccess 'Usuario02'
GO
Sp_GrantDBAccess 'Usuario03'
GO
Sp_GrantDBAccess 'Usuario04'
GO
Sp_GrantDBAccess 'Usuario05'
GO
```

Creados los usuarios podemos asignar, revocar o negar permisos sobre cada uno de los objetos de la base de datos.

Para ello emplearemos instrucciones tales como:

GRANT Permite asignar permisos sobre los objetos de una base de datos y/o sentencias a utilizar

```
GRANT <sentencia>
ON    Objeto
TO    Usuarios/Rol
GO
```

REVOKE Remueve la asignación o negación del recurso de base de datos.

```
REVOKE <sentencia>
ON    Objeto
TO    Usuarios/Rol
GO
```

DENY Permite negar permisos sobre los objetos de una base de datos y/o sentencias a utilizar

```
DENY <sentencia>
ON    Objeto
TO    Usuarios/Rol
GO
```

Ejemplos:

Ingresar al Analizador de Consultas con la cuenta del “sa” para poder asignar permisos de lectura sobre las tablas pedidos y clientes a los usuarios USUARIO01 y USUARIO04

```
GRANT Select
ON    Pedidos
```

```
TO    Usuario01, Usuario04  
GO
```

```
GRANT Select  
ON    Clientes  
TO    Usuario01, Usuario04  
GO
```

Debe negar el acceso de inserción y eliminación de registros sobre las tablas antes mencionadas:

```
DENY INSERT, DELETE  
ON    Pedidos  
TO    Usuario01, Usuario04  
GO
```

```
DENY INSERT, DELETE  
ON    Clientes  
TO    Usuario01, Usuario04  
GO
```

Para comprobar las asignaciones realizadas, conectese al Analizador de Consultas con la cuenta del Usuario01, tal como lo muestra la siguiente figura:



Una vez que ingreso compruebe las siguientes instrucciones:

```
Select * From Pedidos /* debe observar la información */  
GO
```

```
Select * From Clientes  
GO  
/* SELECT permission denied on object 'Clientes', database 'Ejemplo', owner 'dbo'. */
```

Como observa el usuario sólo puede accesar a la información que se especificó con la instrucción GRANT y no podrá realizar ninguna de las operaciones indicadas en la sentencia DENY.

Otra de las características que brinda SQL Server es la posibilidad de crear roles, darles permisos específicos a estos y luego agregarles usuarios.

Como demostración conéctese a Analizador de Consultas como “sa” y ejecute la siguiente secuencia de instrucciones:

```
Use Ejemplo  
GO  
Sp_AddRole 'MisPermisos'  
GO
```

A continuación deberá establecer los derechos que tendrán los integrantes de este rol:

```
GRANT SELECT, INSERT  
ON     CLIENTES  
TO     MIS PERMISOS  
GO
```

```
DENY DELETE, UPDATE  
ON     CLIENTES  
TO     MISPERMISOS  
GO
```

Ahora que ya estableció los derechos, debe agregar los usuarios que conformaran dicho rol:

```
Sp_AddRoleMember 'MISPERMISOS', 'USUARIO02'  
GO  
Sp_AddRoleMember 'MISPERMISOS', 'USUARIO04'  
GO
```

Ahora que completo la secuencia, conéctese al Analizador de Consultas con la cuenta del USUARIO02 y ejecute la siguiente secuencia:

```
Use Ejemplo  
GO  
Select * From Clientes /* debe observar la información */  
GO
```

```
Delete Clientes where rep_clie = 101 /* permiso negado */  
GO
```

Ejecute la misma secuencia conéctandose como USUARIO04. Los resultados deben ser los mismos.

Luego debe conectarse con el USUARIO03 y ejecutar:

```
Use ejemplo  
go  
Select * From Clientes /* permiso negado */
```

GO

Otra de las formas de crear un rol personalizado es a través del Administrador Empresarial.

Para ello expanda la base de datos y haga clic derecho sobre la carpeta Funciones, pulse sobre Nueva Función de base de Datos..., tal como lo muestra la pantalla:



Se presentará la siguiente caja de diálogo, indique el nombre y agregue a los usuarios que considere necesarios:

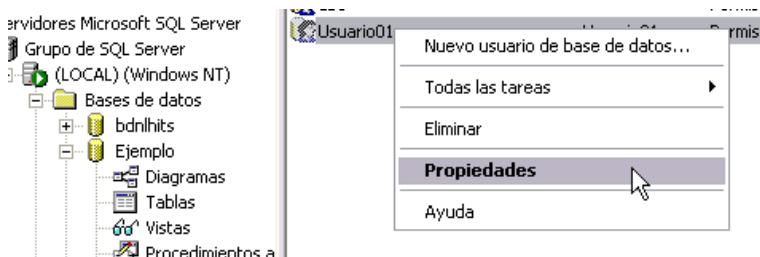


Una vez que pulso Aceptar podrá asignarle todos los derechos que considere necesarios para el rol, además que podrá agregar o retirar integrantes del rol. Estos roles son el equivalente de trabajar con grupos de usuarios en Windows NT.

Todos los permisos que se especifiquen se guardan en la tabla **SysProtects**, donde registran que usuarios tienen permisos para leer la información de determinadas tablas, que usuarios no podrán realizar modificaciones y en general todo lo que tenga que ver con los usuarios y roles.

Otra de las maneras que puede aprovechar para establecer derechos es a través del Administrador Empresarial donde puede hacer un clic derecho sobre cada uno de los usuarios y/o grupos de los cuales necesita establecer niveles de acceso.

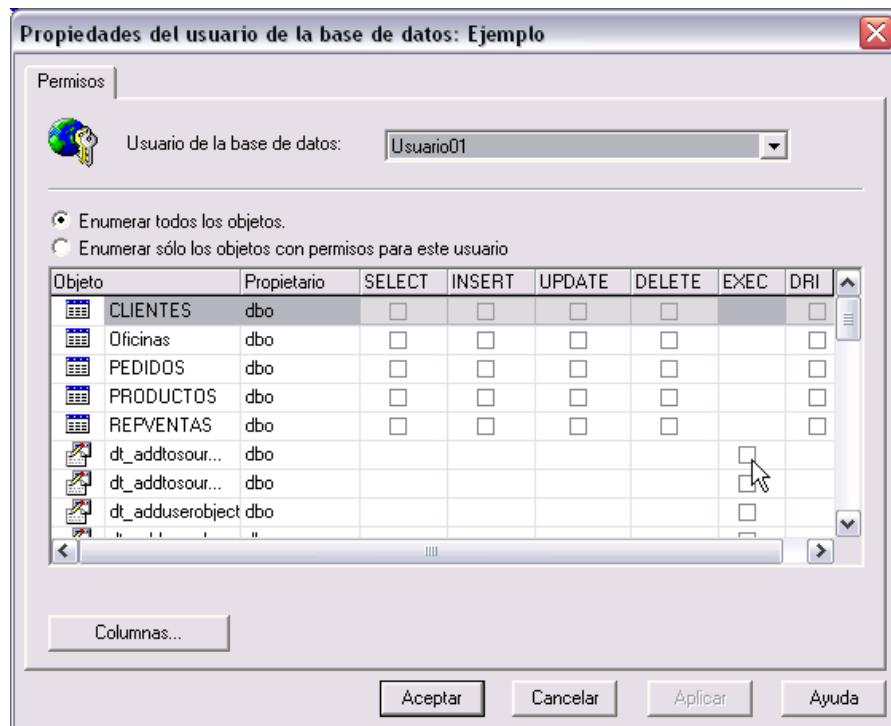
Para ello expanda la carpeta de Usuarios en la base de datos que desea trabajar y haga clic derecho sobre el usuario a asignar derechos, tal como lo muestra la figura siguiente:



Luego de pulsar clic en propiedades observará una caja de diálogo donde debe pulsar un clic en el botón Permissions:



Luego de ello podrá observar una ventana donde podrá permitir o negar accesos sobre los objetos. Tal como lo muestra la siguiente imagen:



EJERCICIOS PROPUESTOS

Crear la base de datos Matriculas con el archivo de datos de 10 Mb., tamaño máximo de 15 Mb. y un crecimiento de 1 Mb. El archivo de log debe tener inicialmente un tamaño de 3 Mb, con un tamaño máximo de 5 Mb. y un crecimiento de 1 Mb.

Una vez creada la base de datos defina la creación de las siguientes tablas:

ALUMNOS

Diseñar tabla 'Alumnos' en 'Matriculas' en '(LOCAL)'

Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
codalu	char	4	
nom	varchar	20	
pat	varchar	20	
mat	varchar	20	

Columnas

Descripción
Valor predeterminado
Precisión 0
Escala 0
Identidad No
Inicialización de identidad
Incremento de identidad
Es RowGuid No
Fórmula
Intercalación SQL_Latin1_General_CI_AS

CALIFICACIONES

Diseñar tabla 'Calificaciones' en 'Matriculas' en '(LOCAL)'

Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
codalu	char	4	
sec	char	4	
n1	int	4	
n2	int	4	

Columnas

Descripción
Valor predeterminado
Precisión 0
Escala 0
Identidad No
Inicialización de identidad
Incremento de identidad
Es RowGuid No
Fórmula
Intercalación SQL_Latin1_General_CI_AS

CATEGORIAS

Diseñar tabla 'Categorias' en 'Matriculas' en '(LOCAL)'

Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
categ	char	3	
pagoporhora	money	8	

Columnas

Descripción	
Valor predeterminado	
Precisión	0
Escala	0
Identidad	No
Inicialización de identidad	
Incremento de identidad	
Es RowGuid	No
Fórmula	
Intercalación	SQL_Latin1_General_CI_AS

INSCRITOS

Diseñar tabla 'Inscritos' en 'Matriculas' en '(LOCAL)'

Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
codalu	char	4	
sec	char	4	

Columnas

Descripción	
Valor predeterminado	
Precisión	0
Escala	0
Identidad	No
Inicialización de identidad	
Incremento de identidad	
Es RowGuid	No
Fórmula	
Intercalación	SQL_Latin1_General_CI_AS

PROFESORES

Diseñar tabla 'Profesores' en 'Matriculas' en '(LOCAL)'

Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
codprofe	char	4	
nombre	varchar	40	
categ	char	3	
fec_ing	datetime	8	
direc	varchar	50	
telef	char	8	

Columnas

Descripción	
Valor predeterminado	
Precisión	0
Escala	0
Identidad	No
Inicialización de identidad	
Incremento de identidad	
Es RowGuid	No
Fórmula	
Intercalación	SQL_Latin1_General_CI_AS

SECCIONES

Diseñar tabla 'Secciones' en 'Matriculas' en '(LOCAL)'

Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
sec	char	4	
codprofe	char	4	
inicio	datetime	8	
pension	money	8	
vacantes	int	4	
numhoras	int	4	

Columnas

Descripción	
Valor predeterminado	
Precisión	0
Escala	0
Identidad	No
Inicialización de identidad	
Incremento de identidad	
Es RowGuid	No
Fórmula	
Intercalación	SQL_Latin1_General_CI_AS

Una vez que termine recupere el script Alumnado.sql, revise el script y ejecútelo con la finalidad de poblar la base de datos.

Bien una vez que hemos creado las tablas y hemos cargado los valores, debemos implementar las restricciones (*CONSTRAINTS*) correspondientes a las reglas de integridad de cada tabla. Iniciaremos asignando las restricciones de tipo clave Principal (*PRIMARY KEY*):

```
Use Matriculas
GO
ALTER TABLE Alumnos
ADD CONSTRAINT PK_Alumnos_codalu
PRIMARY KEY (codalu)
GO

ALTER TABLE Secciones
ADD CONSTRAINT PK_Secciones_sec
PRIMARY KEY (sec)
GO

ALTER TABLE Profesores
ADD CONSTRAINT PK_Profesores_codprofe
PRIMARY KEY (codprofe)
GO
ALTER TABLE Categorias
ADD CONSTRAINT PK_Categorias_categ
PRIMARY KEY (categ)
GO

ALTER TABLE Inscritos
ADD CONSTRAINT PK_Inscritos_alusec
PRIMARY KEY (codalu, sec)
GO

ALTER TABLE Calificaciones
ADD CONSTRAINT PK_Calificaciones_alusec
PRIMARY KEY (codalu, sec)
GO
```

Después de crear las definiciones para asegurar la unicidad de cada fila, nos toca implementar las restricciones de claves foráneas (*FOREIGN KEY constraint*).

```
ALTER TABLE Inscritos
ADD CONSTRAINT FK_Alumnos_codalu
FOREIGN KEY (codalu)
REFERENCES Alumnos(codalu)
GO

ALTER TABLE Inscritos
ADD CONSTRAINT FK_Secciones_sec
FOREIGN KEY (sec)
REFERENCES Secciones(sec)
GO

ALTER TABLE Secciones
ADD CONSTRAINT FK_Profesores_codprofe
```

```

FOREIGN KEY (codprofe)
REFERENCES profesores(codprofe)
GO

```

```

ALTER TABLE Calificaciones
ADD CONSTRAINT FK_Inscritos_codalu
FOREIGN KEY (codalu, sec)
REFERENCES Inscritos(codalu, sec)
GO

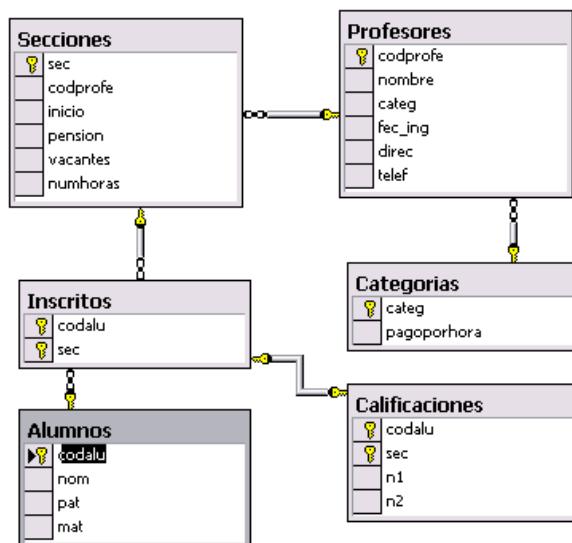
```

```

ALTER TABLE Profesores
ADD CONSTRAINT FK_Categorias_categ
FOREIGN KEY (categ)
REFERENCES Categorias(categ)
GO

```

Finalmente genere el diagrama de base de datos que debe observarse como el siguiente:



Para probar el funcionamiento de los constraints implementados ejecute las siguientes instrucciones:

```

Insert Inscritos Values ('A001', '1116')
GO

```

```

/* El problema se debe a que la llave de inscripcion es duplicada por tanto podemos
deducir que el alumno ya estudio en esa aula. En este caso trabaja PK_Inscritos_alusec.
*/

```

```

Insert Calificaciones Values ('A003', '2315', 10,10)
GO

```

```

/* El problema es que el alumno no ha sido inscrito en la sección especificada, en este
caso trabaja el FK_Inscritos_codalu. */

```

```

Insert Secciones Values ('2318', 'D009', '5/15/2001', 600, 15, 40)

```

GO

/* El problema es que el código del profesor no esta registrado en la tabla PROFESORES, en este caso trabaja el FK_Profesores_codprofe. */

Implementaremos un Desencadenador para inserción de registros en la tabla Inscritos, de forma que al momento de agregar un registro se valide la disponibilidad de vacantes, de ser así actualice el número de vacantes en la tabla secciones y agregar un registro en la tabla Calificaciones.

Ejecute el siguiente código:

```
Use Matriculas
GO
CREATE DESENCADENADOR Inscribe_Alumno
ON Inscritos
FOR INSERT
As
BEGIN
    DECLARE @Vacantes int
    SELECT @Vacantes = (Select Vacantes From Secciones INNER JOIN Inserted
On Secciones.Sec = Inserted.Sec)
    IF @Vacantes = 0
        BEGIN
            RAISERROR ('No hay vacantes en el aula', 10, 1)
            ROLLBACK TRANSACTION
        END
    ELSE
        BEGIN
            INSERT Calificaciones
            SELECT Inserted.Codalu, Inserted.Sec, 0, 0 From Inserted

            UPDATE Secciones
            SET Vacantes = Vacantes - 1
            FROM Secciones JOIN Inserted On Secciones.Sec = Inserted.Sec
        END
END
GO
```

Mostrar la información de las secciones

```
Select * From Secciones
GO
```

Tomemos como referencia la sección 1116 pues tiene una sola vacante inscribamos al alumno A019 en dicha sección

```
Insert Inscritos Values ('A019', '1116')
GO
```

Verifique el número de vacantes en la 1116

```
Select * From Secciones Where Sec='1116' -- Ahora es 0  
GO
```

Revisemos la tabla Calificaciones

```
Select * From Calificaciones Where codalu = 'A019'  
GO
```

Intentemos agregar otro alumno a la sección 1116

```
Insert Inscritos Values ('A016', '1116')  
GO
```

En este último intento debe mostrarse el mensaje “No hay vacantes”

Propuesto

Implementar el Desencadenador RetiraInscripcion sobre la tabla Inscritos el cual debe ejecutarse el eliminar un registro de la tabla Inscritos, este Desencadenador debe eliminar la entrada generada en la tabla Calificaciones además de incrementar el número de vacantes de la sección de la cual se retira el alumno.

Implementar el Desencadenador CambioSeccion sobre la tabla Inscritos, este se ejecutará al actualizar la información del inscrito con un cambio de sección, en este caso se verificará si en la nueva sección existen vacantes de haberlo debe actualizar el número de vacantes incrementando su valor en la antigua sección y reduciendolo en la nueva sección.

Ahora para practicar los querys implementaremos las siguientes consultas:

Mostrar la lista de alumnos de la sección 2315 ordenados alfabéticamente

```
SELECT Pat, Mat, Nom  
From Alumnos Inner Join Inscritos  
On Alumnos.codalu = Inscritos.codalu  
AND Inscritos.Sec = '2315'  
Order By Pat, Mat, Nom  
GO
```

El segundo carácter de la sección indica el curso tomado en cuenta lo siguiente:
[1] V.Basic [2] V.Fox [3] SQL Server.

Mostrar un reporte que muestre el código del alumno, el curso que estudia y el promedio

```
SELECT NomApe = (Nom + ' ' + Pat + ' ' + Mat),
```

```

Curso = Case
    When SubString(Inscritos.Sec, 2, 1)='1'
        Then 'V.Basic'
    When SubString(Inscritos.Sec, 2, 1)='2'
        Then 'V.Fox'
    When SubString(Inscritos.Sec, 2, 1)='3'
        Then 'SQL Server'
    End,
Promedio = (N1+N2)/2
From Alumnos INNER JOIN Inscritos
On Alumnos.codalu = Inscritos.codalu
INNER JOIN Calificaciones
On Inscritos.codalu = Calificaciones.codalu
    AND Inscritos.sec = Calificaciones.sec
GO

```

Mostrar todos los alumnos con indicación de los cursos que estudian con indicación de sus promedios en caso tenerlo.

```

SELECT NomApe = (Nom + ' ' + Pat + ' ' + Mat),
Curso = Case
    When SubString(Inscritos.Sec, 2, 1)='1'
        Then 'V.Basic'
    When SubString(Inscritos.Sec, 2, 1)='2'
        Then 'V.Fox'
    When SubString(Inscritos.Sec, 2, 1)='3'
        Then 'SQL Server'
    End,
Promedio = (N1+N2)/2
From Alumnos INNER JOIN Inscritos
On Alumnos.codalu = Inscritos.codalu
LEFT OUTER JOIN Calificaciones
On Inscritos.codalu = Calificaciones.codalu
    AND Inscritos.sec = Calificaciones.sec
GO

```

Mostrar el promedio general de los alumnos que completaron los tres cursos

```

SELECT NomApe = (Max(Nom) + ' ' + Max(Pat) + ' ' + Max(Mat)),
PromedioFinal = AVG((N1+N2)/2)
From Alumnos INNER JOIN Inscritos
On Alumnos.codalu = Inscritos.codalu
INNER JOIN Calificaciones
On Inscritos.codalu = Calificaciones.codalu
    AND Inscritos.sec = Calificaciones.sec
Group By Inscritos.codalu
Having count(Inscritos.codalu) = 3
GO

```

Mostrar el número de alumnos que estudia en cada sección

```
Select Sec, Alumnado=Count(codalu)
From Inscritos
Group By sec
GO
```

Mostrar a los alumnos cuyo promedio es mayor que 14, indique sección, curso y profesor

```
SELECT NomApe = (Nom + ' ' + Pat + ' ' + Mat), Inscritos.Sec,
       Curso = Case
           When SubString(Inscritos.Sec, 2, 1)='1'
               Then 'V.Basic'
           When SubString(Inscritos.Sec, 2, 1)='2'
               Then 'V.Fox'
           When SubString(Inscritos.Sec, 2, 1)='3'
               Then 'SQL Server'
       End,
       Promedio = (N1+N2)/2
From Alumnos INNER JOIN Inscritos
On Alumnos.codalu = Inscritos.codalu
LEFT OUTER JOIN Calificaciones
On Inscritos.codalu = Calificaciones.codalu
    AND Inscritos.sec = Calificaciones.sec
Where (N1+N2)/2 > 14
GO
```

Implementar Vistas y Procedimientos Almacenados

Objetivos:

- Entender que es una vista
- Entender que es un Procedimientos Almacenados
- Implementar Vistas y procedimientos

Temas:

1. ¿Qué es una vista?
2. Agregar, Modificar y Eliminar vistas
3. ¿Qué es un Procedimientos Almacenados?
4. Agregar, Modificar y Eliminar Procedimientos Almacenados
5. Funciones de usuario en SQL Server 2000

¿Qué es una vista?

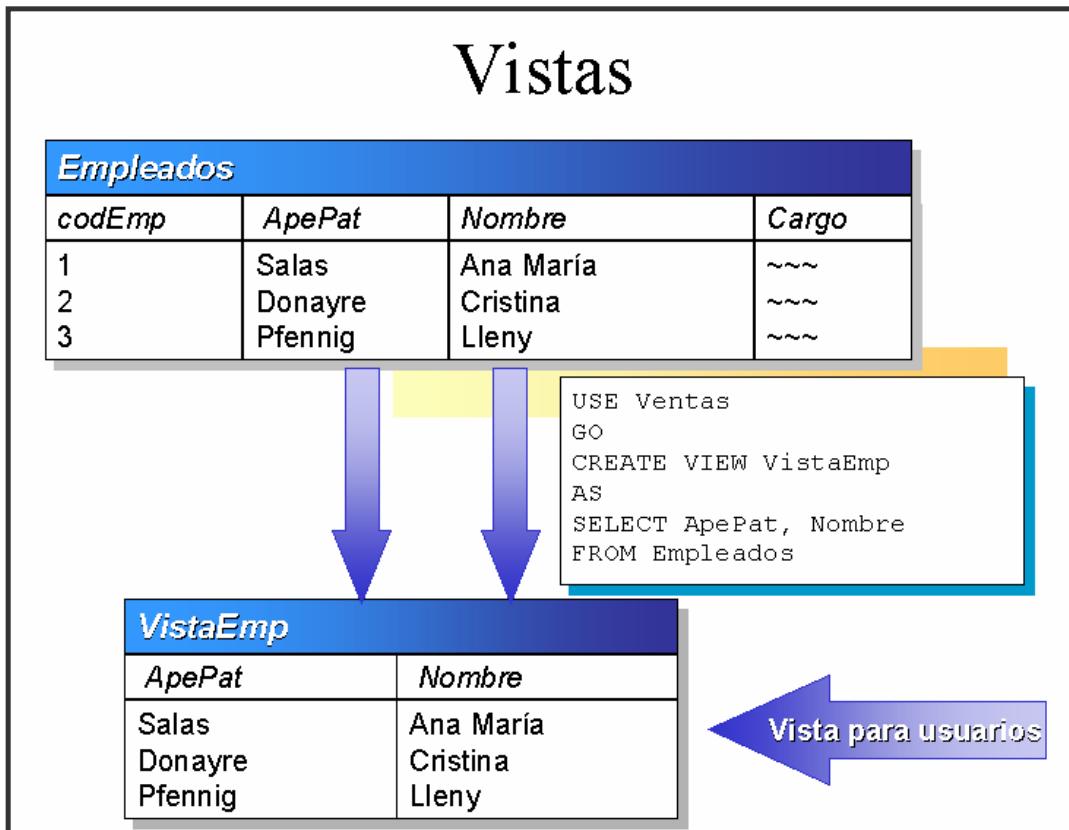
Una vista es una tabla virtual que muestra la información relevante para el usuario además que permite encapsular la complejidad de su implementación.

Una vista puede mostrar toda la información de una tabla o de la integración de información de más de una tabla. Es decir se puede afirmar que las vistas trabajan como especie de filtros de las tablas subyacentes que brindan la información presentada por la vista.

Entre las principales ventajas del empleo de las vistas podemos mencionar:

- Permite mostrar un subconjunto de filas y/o columnas de una tabla.
- Permite mostrar información de más de una tabla.
- Permite realizar uniones entre dos o más tablas.
- Permite mostrar informes resumen.

- Puede definirse a partir de otras vistas



Como observa en la representación anterior los usuarios podrán observar sólo las columnas que son importantes para su trabajo, esto da la ventaja de que no tengan que manipular datos innecesarios para su labor.

Otra de las ventajas de las vistas es que pueden generarse a partir de consultas distribuidas entre orígenes de datos heterogéneos, de tal manera que los usuarios invocan a las vistas en lugar de estar digitando complejos querys.

Antes de implementar sus vistas tenga en cuenta las siguientes consideraciones:

Sólo pueden crearse vistas en la base de datos activa, aunque las tablas y/o vistas que son parte de la definición puedan encontrarse en distintas bases de datos.

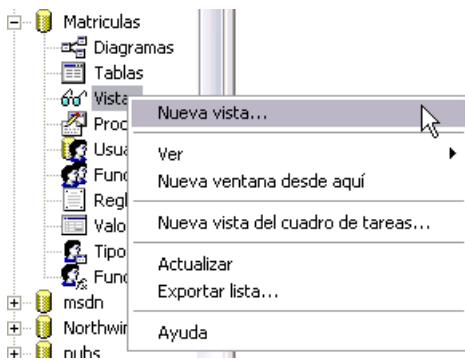
- Se pueden crear vistas a partir de otras vistas.
No se pueden asociar defaults, rule y/o desencadenadores (Triggers) a una vista.
- La consulta que forma la vista no puede incluir las cláusulas ORDER BY, COMPUTE o COMPUTE BY
No se pueden construir índices sobre las vistas
- No se pueden crear vistas temporales, ni vistas basadas en tablas temporales.
Cada una de las columnas empleadas en la vista debe tener un encabezado.

Agregar, Modificar y Eliminar una Vista

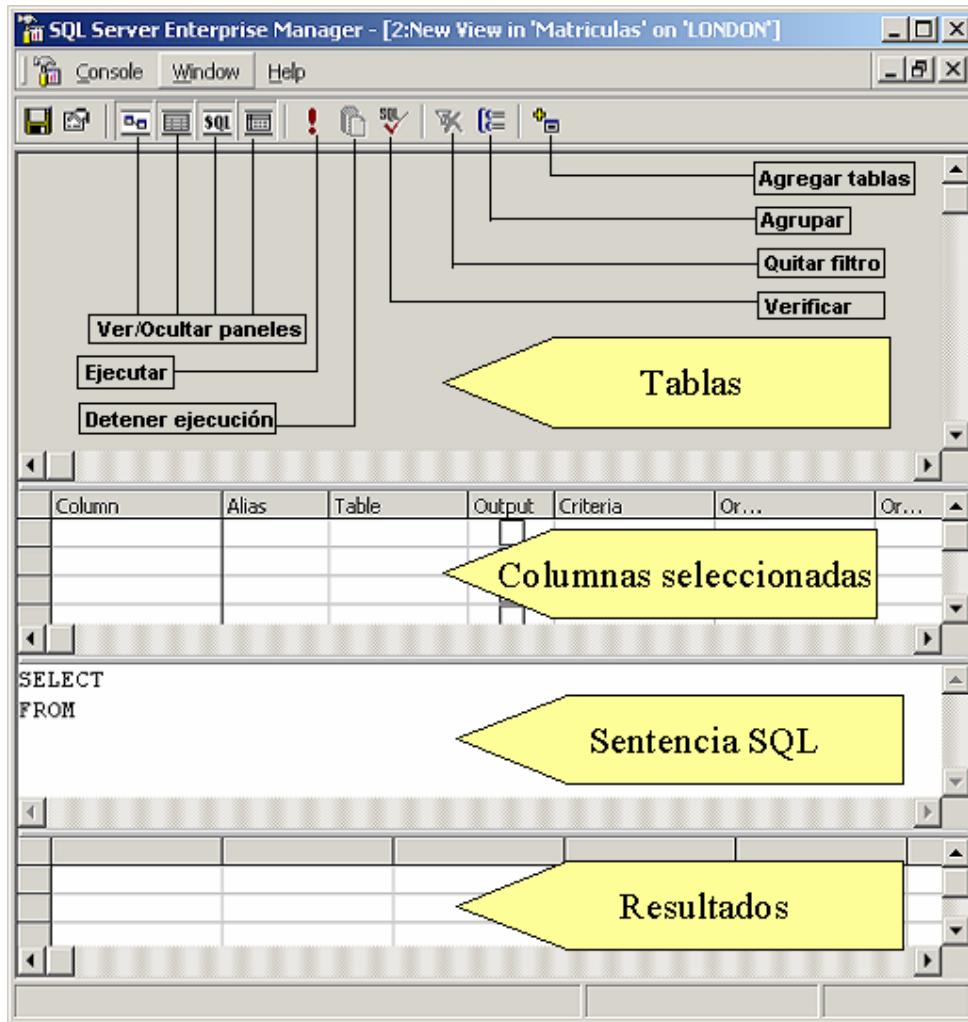
Crear Vistas

Para crear vistas podemos emplear el Administrador Empresarial, realizando el siguiente proceso:

Expanda la base de datos Matriculas y haga clic derecho sobre la carpeta Vistas, luego pulse clic sobre la opción Nueva Vista... tal como lo indica la siguiente figura:



Luego de ello aparecerá una pantalla como la siguiente:



Otra de las formas de crear una vista es a partir de la sentencia CREATE VIEW, cuya sintaxis es:

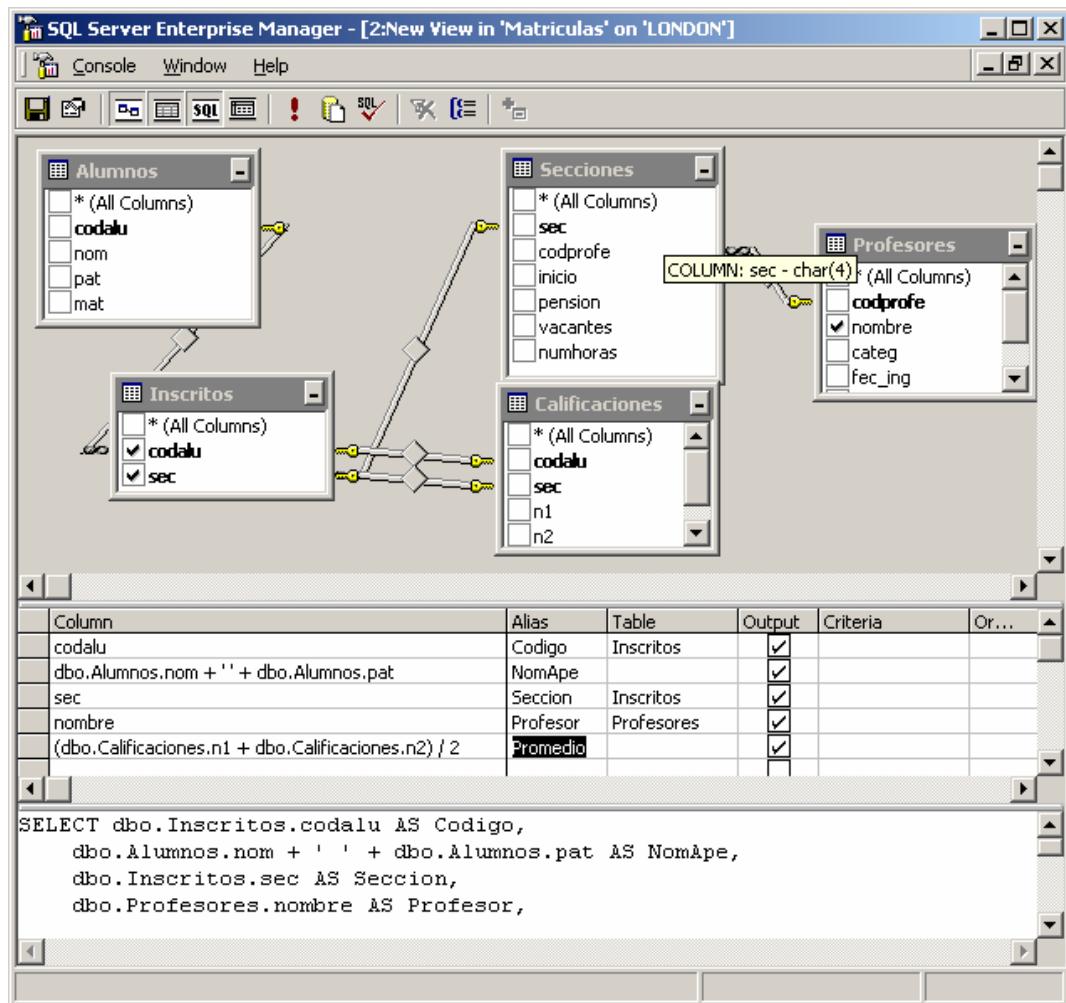
```
CREATE VIEW <Nombre de la vista> [Encabezado1, ....]  
[WITH ENCRYPTION]  
AS  
    Sentencias Select  
    [WITH CHECK OPTION]
```

La cláusula WITH ENCRYPTION, permite ocultar la implementación de la vista, la cual puede ser vista con el stored procedure del sistema **sp_helptext** o desde la tabla del sistema **syscomments**.

La cláusula WITH CHECK OPTION, garantiza que los cambios hechos desde la vista puedan ser observados una vez que la operación finaliza.

Ejemplos:

Utilice el Administrador Empresarial para definir una vista con las características que muestra la ilustración:



Una vez que termine de diseñar la vista pulse CTRL-F4, conteste afirmativamente a la pregunta y digite el nombre Relación de Promedios, pulse Enter para finalizar.

Para comprobar la creación de la vista en el Analizador de Consultas digite la siguiente instrucción:

```
Select * From [Relacion de Promedios]  
GO
```

Ver la sentencia que implementa la vista:

Sp_HelpText [Relacion de Promedios]

GO

Ver las columnas que se presentan en la vista

```
Sp_Depends [Relación de Promedios]  
GO
```

Desde el Query Analyzer debe crear una vista que nos permita mostrar un informe resumen por sección que muestre la sección, el nombre del profesor, el total de alumnos, el mayor y menor promedio, el promedio de la sección y el monto acumulado de las pensiones.

```
CREATE VIEW Resumen  
AS  
    Select  
        Inscritos.Sec As Seccion,  
        MAX(nombre) As Profesor,  
        Count(Inscritos.CodAlu) As Alumnado,  
        Max((N1+N2)/2) As MayorPromedio,  
        Min((N1+N2)/2) As MenorPromedio,  
        Avg((N1+N2)/2) As PromedioSec,  
        Sum(Pension) As Acumulado  
    From  
        Inscritos INNER JOIN Calificaciones  
        On Inscritos.codAlu = Calificaciones.codalu  
        AND Inscritos.sec = Calificaciones.Sec  
        INNER JOIN Secciones On Inscritos.Sec = Secciones.Sec  
        INNER JOIN Profesores On Secciones.CodProfe = Profesores.CodProfe  
    Group by Inscritos.Sec  
    GO
```

Compruebe la información que devuelve la vista:

```
Select * From Resumen  
GO
```

Mostrar la implementación de la vista:

```
Sp_HelpText Resumen  
GO
```

Modificar Vistas

Para modificar la vista utilice la siguiente sintaxis:

```
ALTER VIEW <Nombre de la Vista> [(Encabezado1, ...)]  
[WITH ENCRYPTION]
```

AS
<Sentencia SELECT>
[WITH CHECK OPTION]

Para poder comprobar el empleo de esta sentencia, ejecute el siguiente comando:

```
ALTER VIEW RESUMEN
WITH ENCRYPTION
AS
    Select
        Inscritos.Sec As Seccion,
        MAX(nombre) As Profesor,
        Count(Inscritos.CodAlu) As Alumnado,
        Max((N1+N2)/2) As MayorPromedio,
        Min((N1+N2)/2) As MenorPromedio,
        Avg((N1+N2)/2) As PromedioSec,
        Sum(Pension) As Acumulado
    From
        Inscritos INNER JOIN Calificaciones
        On Inscritos.codAlu = Calificaciones.codalu
        AND Inscritos.sec = Calificaciones.Sec
        INNER JOIN Secciones On Inscritos.Sec = Secciones.Sec
        INNER JOIN Profesores On Secciones.CodProfe = Profesores.CodProfe
    Group by Inscritos.Sec
    GO
```

Ahora muestre la implementación de la vista con el siguiente comando:

```
Sp_HelpText Resumen
GO
```

Como observará en el panel de resultados ahora ya no se muestra la implementación de la vista.

Lo mismo ocurrirá si consulta la información de la tabla del sistema SysComments.

```
Select * From SysComments
GO
```

Eliminar Vistas

Para eliminar una vista emplear la siguiente sintaxis:

DROP VIEW <Nombre de la vista> [,...n]

Como ejemplo podría utilizar la siguiente instrucción:

```
DROP VIEW [Relacion de Promedios]
GO
```

Select * From [Relacion de Promedios]
GO

Procedimientos Almacenados

Un stored procedure es una colección de sentencias del Transact-SQL las cuales organizadas lógicamente resuelven algunas de las operaciones transaccionales que requieren los usuarios, estos procedimientos se almacenan en la base de datos. Los procedimientos almacenados soportan el empleo de variables declaradas por el usuario, sentencias para toma de decisiones entre otras características.

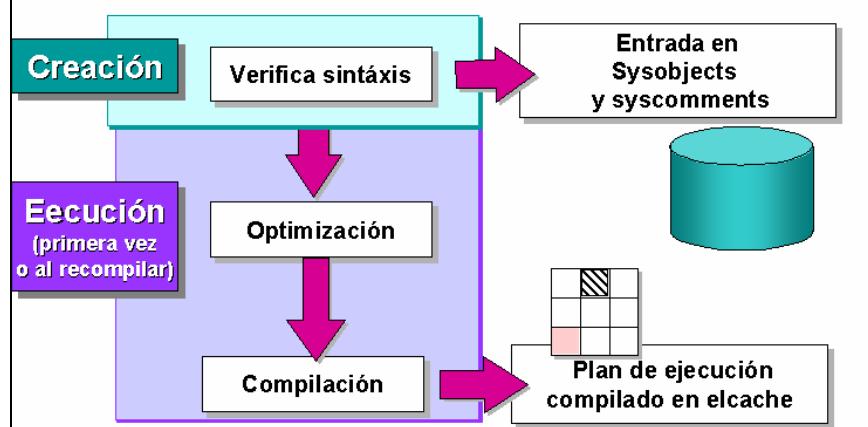
En SQL Server existen 5 tipos de procedimientos almacenados:

- Procedimientos del sistema, son los que se encuentran almacenados en la base de datos **master** y algunas en las bases de datos de usuario, estos procedimientos almacenados brindan información acerca de los datos y características del servidor. En el nombre usan como prefijo **sp_**.
- Procedimientos locales, son los procedimientos almacenados en una base de datos.
- Procedimientos temporales, son procedimientos locales y sus nombres empiezan con los prefijos # o ##, dependiendo si se desea que sea un procedimiento global a todas las conexiones o local a la conexión que lo define.
- Procedimientos remotos, son procedimientos almacenados en servidores distribuidos.
- Procedimientos extendidos, son aquellos que nos permiten aprovechar las funcionalidades de otras librerías externas a SQL Server. Estos procedimientos usan el prefijo **xp_** y se encuentran en la base de datos **master**.

Entre las principales características de un procedimiento almacenado podemos mencionar:

- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida al lote o al procedimiento que realiza la llamada.
- Contener instrucciones de programación que realicen operaciones en la base de datos, incluidas las llamadas a otros procedimientos.
- Devolver un valor de estado que indica si la operación se ha realizado correctamente o habido un error (y el motivo del mismo).
- Permiten una ejecución más rápida, ya que los procedimientos son analizados y optimizados en el momento de su creación, y es posible utilizar una versión del procedimiento que se encuentra en la memoria después de que se ejecute por primera vez.
- Pueden reducir el tráfico de red.
- Pueden utilizarse como mecanismo de seguridad, ya que se puede conceder permisos a los usuarios para ejecutar un procedimiento almacenado, incluso si no cuentan con permiso para ejecutar directamente las instrucciones del procedimiento.

Proceso inicial de un Stored Procedures



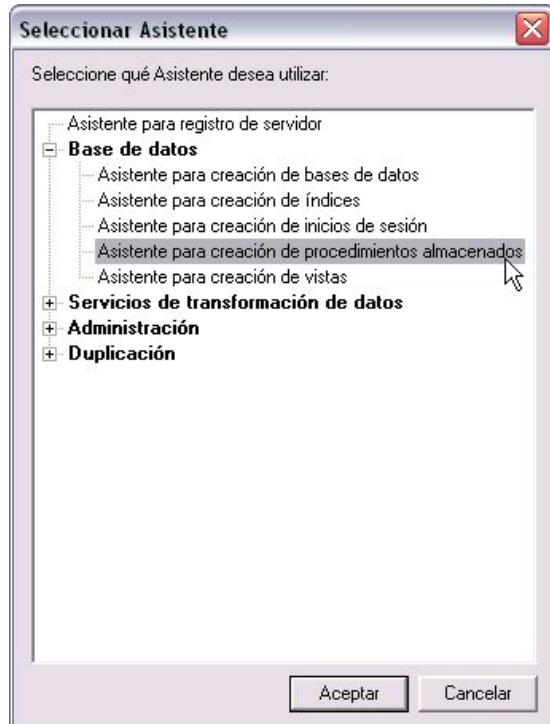
Crear, Modificar y Eliminar un **Procedimiento Almacenado**

Crear Procedimientos Almacenados

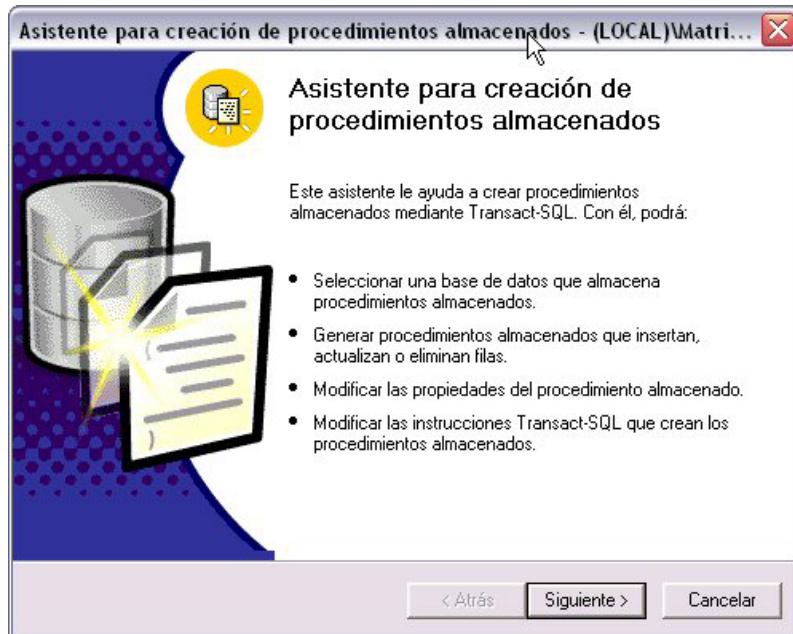
Para crear un stored procedure en SQL Server tiene la posibilidad de utilizar múltiples formas entre ellas un asistente para la creación de procedimientos para ingresar, eliminar y actualizar información en las tablas.

Siga las instrucciones para aprovechar el asistente:

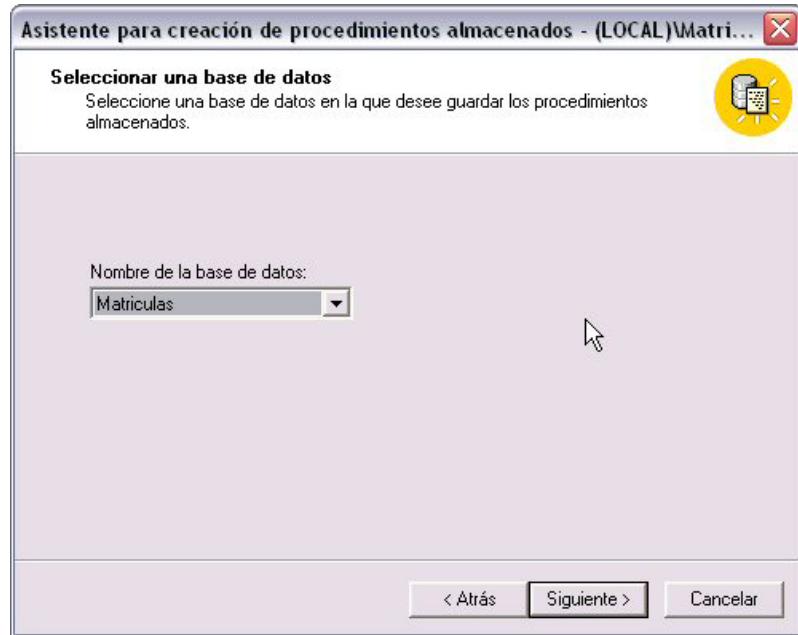
1. En el menú Herramientas haga clic sobre la opción Asistentes, y seleccione la opción que muestra la figura:



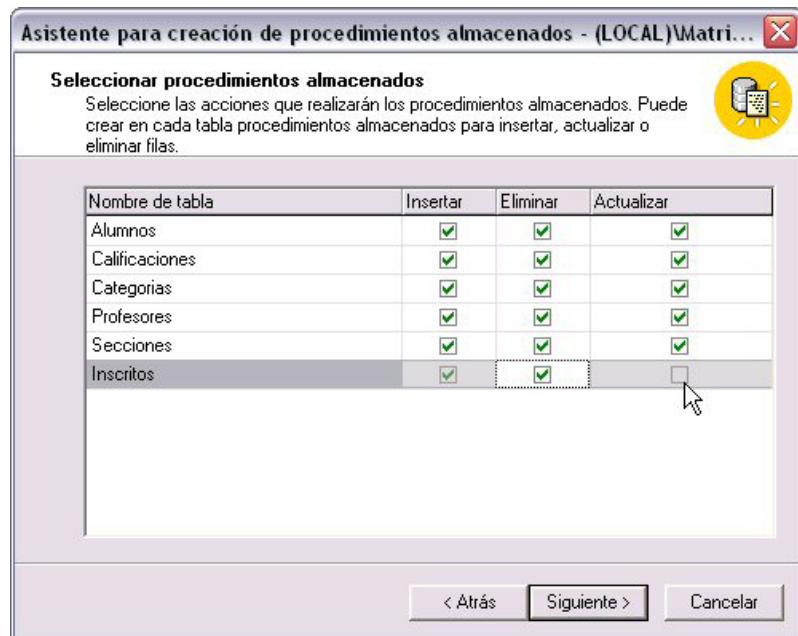
2. Luego de pulsar Aceptar, aparecerá una pantalla de bienvenida al asistente



3. Pulse Siguiente y tendrá la posibilidad de elegir que base de datos utilizará.



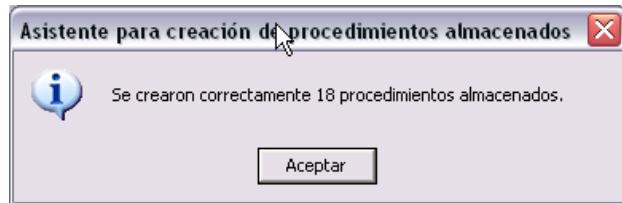
4. Pulse Siguiente y marque las casillas que indican que procedimientos creará.



Luego de pulsar Siguiente aparecerá una pantalla que indica el fin de los pasos requeridos para crear los procedimientos.



5. Al pulsar Finalizar aparecerá un mensaje indicando cuantos procedimientos se han creado.



También puede crear los procedimientos con la sentencia CREATE PROCEDURE.

```

CREATE PROC[EDURE] <Nombre Procedimiento>
[
{@parámetro tipoDatos} [=predeterminado] [OUTPUT]
]
[,...n]
[WITH
{
RECOMPILE
| ENCRYPTION
}
]
AS
Sentencias SQL [...n]

```

Argumentos

@parámetro

El usuario puede tener hasta máximo de 1024 parámetros. El nombre del parámetro debe comenzar con un signo (@) . Los parámetros son locales al procedimiento.

default

Es un valor predeterminado para el parámetro.

OUTPUT

Indica que se trata de un parámetro de salida. El valor de esta opción puede devolverse a EXECUTE. Utilice los parámetros OUTPUT para devolver información al procedimiento que llama. Los parámetros de texto no se pueden utilizar como parámetros OUTPUT.

{RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}

RECOMPILE indica que SQL Server no almacena en la caché un plan para este procedimiento, con lo que el procedimiento se vuelve a compilar cada vez que se ejecuta. Utilice la opción RECOMPILE cuando emplee valores atípicos o temporales para no anular el plan de ejecución que está almacenado en la memoria caché.

ENCRYPTION indica que SQL Server codifica la entrada de la tabla **syscomments** que contiene el texto de la instrucción CREATE PROCEDURE.

Entre otras observaciones podemos mencionar:

- El tamaño máximo de un procedimiento es de 128 Mb.
- Un procedimiento sólo puede crearse en la base de datos actual.
- Se puede crear otros objetos de base de datos dentro de un procedimiento almacenado. Puede hacer referencia a un objeto creado en el mismo procedimiento almacenado, siempre que se cree antes de que se haga referencia al objeto.
- Puede hacer referencia a tablas temporales dentro de un procedimiento almacenado.

Si crea una tabla temporal privada dentro de un procedimiento almacenado, la tabla temporal existirá únicamente para los fines del procedimiento; desaparecerá cuando éste finalice.

Si ejecuta un procedimiento almacenado que llama a otro procedimiento almacenado, el procedimiento al que se llama puede tener acceso a todos los objetos creados por el primer procedimiento, incluidas las tablas temporales.

Si se ejecuta un procedimiento almacenado remoto que realiza cambios en un servidor SQL Server remoto, los cambios no se podrán deshacer. Los procedimientos almacenados remotos no intervienen en las transacciones.

Las siguientes instrucciones no se pueden emplear dentro de un procedimiento.

CREATE
DEFAULT
CREATE
PROCEDURE
CREATE RULE

CREATE
DESENCADENADOR
CREATE VIEW

Para crear un procedimiento desde el Analizador de Consultas ejecute las siguientes instrucciones:

```
Use Matriculas
GO
CREATE PROCEDURE ListaPromedios
as
SELECT NomApe = (Nom + ' ' + Pat + ' ' + Mat), Inscritos.Sec,
    Curso = Case
        When SubString(Inscritos.Sec, 2, 1)='1'
            Then 'V.Basic'
        When SubString(Inscritos.Sec, 2, 1)='2'
            Then 'V.Fox'
        When SubString(Inscritos.Sec, 2, 1)='3'
            Then 'SQL Server'
    End,
    Promedio = (N1+N2)/2
From Alumnos INNER JOIN Inscritos
On Alumnos.codalu = Inscritos.codalu
    INNER JOIN Calificaciones
        On Inscritos.codalu = Calificaciones.codalu
            AND Inscritos.sec = Calificaciones.sec
```

GO

Para poder ejecutar el procedimiento emplearemos la siguiente sintaxis:

```
EXEC ListaPromedios
GO
```

Para ver la información de la implementación del procedimiento almacenado:

```
Sp_HelpText ListaPromedios
GO
```

Para ver cuales son las columnas que producen la información presentada por el procedimiento:

```
Sp_Depends ListaPromedios
GO
```

Modificar Procedimientos Almacenados

Si se desea modificar el procedimiento almacenado utilice la siguiente sintaxis:

```
ALTER PROC[EDURE] <Nombre Procedimiento>
[
{@parámetro tipoDatos} [=predeterminado] [OUTPUT]
]
[,...n]
[WITH
{
    RECOMPILE
    | ENCRYPTION
}
]
AS
Sentencias SQL [...n]
```

Para ver un ejemplo realice lo siguiente:

```
ALTER PROCEDURE ListaPromedios
WITH ENCRYPTION
as
SELECT NomApe = (Nom + ' ' + Pat + ' ' + Mat), Inscritos.Sec,
    Curso = Case
        When SubString(Inscritos.Sec, 2, 1)='1'
            Then 'V.Basic'
        When SubString(Inscritos.Sec, 2, 1)='2'
            Then 'V.Fox'
        When SubString(Inscritos.Sec, 2, 1)='3'
            Then 'SQL Server'
    End,
    Promedio = (N1+N2)/2
From Alumnos INNER JOIN Inscritos
On Alumnos.codalu = Inscritos.codalu
    INNER JOIN Calificaciones
    On Inscritos.codalu = Calificaciones.codalu
        AND Inscritos.sec = Calificaciones.sec
GO
```

Ejemplo:

Ahora implementaremos un procedimiento que muestre el promedio de cada alumno de acuerdo a la sección indicada en el argumento:

```
CREATE PROCEDURE PromPorSeccion
(@seccion char(4) = NULL)
as
```

```

IF @seccion IS NULL
BEGIN
    RAISERROR ('Debe indicar un codigo', 10, 1)
    RETURN
END
IF NOT EXISTS (Select Sec From Inscritos Where Sec=@seccion)
BEGIN
    RAISERROR ('La seccion no tiene alumnos', 10, 1)
    RETURN
END
SELECT NomApe = (Nom + ' ' + Pat + ' ' + Mat), Inscritos.Sec,
    Curso = Case
        When SubString(Inscritos.Sec, 2, 1)='1'
            Then 'V.Basic'
        When SubString(Inscritos.Sec, 2, 1)='2'
            Then 'V.Fox'
        When SubString(Inscritos.Sec, 2, 1)='3'
            Then 'SQL Server'
    End,
    Promedio = (N1+N2)/2
From Alumnos INNER JOIN Inscritos
On Alumnos.codalu = Inscritos.codalu
    INNER JOIN Calificaciones
        On Inscritos.codalu = Calificaciones.codalu
            AND Inscritos.sec = Calificaciones.sec
WHERE Inscritos.Sec = @Seccion
GO

```

Para ejecutar realizaremos lo siguiente:

```

EXEC PromPorSeccion
GO
EXEC PromPorSeccion '9090'
GO
EXEC PromPorSeccion '2315'
GO
Sp_HelpText PromPorSeccion
GO
Sp_Depends PromPorSeccion
GO

```

Eliminar Procedimientos Almacenados

Para eliminar un procedimiento utilice el siguiente formato:

DROP PROCEDURE <Nombre del procedimiento>

Funciones en SQL Server 2000 (1/2)

Microsoft agregó nuevas características a su producto SQL Server 2000, y una de las más sobresalientes es sin duda la capacidad de implementar funciones definidas por el usuario. Esta característica ayudará a solucionar los problemas de reutilización del código y dará mayor flexibilidad al programar las consultas de SQL.

Tipos de funciones

El servidor 2000 del SQL utiliza tres tipos de funciones: las funciones escalares, tabla en linea , funciones de tabla de multi sentencias. Los tres tipos de funciones aceptan parámetros de cualquier tipo excepto el rowversion. Las funciones escalares devuelven un solo valor, tabla en linea y Multisentencias devuelven un tipo de dato tabla.

Funciones Escalares

Las funciones escalares devuelven un tipo de los datos tal como int, money, varchar, real, etc. Pueden emplearse en cualquier lugar incluso incorporada dentro de sentencias SQL. La sintaxis para una función escalar es la siguiente:

```
CREATE FUNCTION [propietario] Nombre_de_Función
( [{ @parametro parametro_scalar [= default]} [..n]])
RETURNS tipo_de_dato_scalar_de_retorno
[WITH <opción> >::={SCHEMABINDING | ENCRYPTION}]
[AS]
BEGIN
Código de la función
RETURN expresión_scalar
END
```

Como ejemplo una función sencilla para obtener un número elevado al cubo:

```
CREATE FUNCTION dbo.Cubo( @Numero float)
RETURNS float
AS
BEGIN
RETURN(@fNumero * @fNumero * @fNumero)
END
```

Otra característica interesante es que las funciones de usuario soportan llamadas recursivas, como se muestra en el siguiente ejemplo, que calcula el factorial de un número:

```
CREATE FUNCTION dbo.Factorial ( @Numero int )
RETURNS INT
```

```

AS
BEGIN
DECLARE @i int

IF @Numero <= 1
SET @i = 1
ELSE
SET @i = @Numero * dbo.Factorial( @Numero - 1 )
RETURN (@i)
END

```

Funciones de tabla en línea

Las funciones de tabla en linea son las funciones que devuelven la salida de una simple declaración SELECT. La salida se puede utilizar adentro de joins o querys como si fuera un tabla de estándar. La sintaxis para una función de tabla en linea es como sigue:

```

CREATE FUNCTION [propietario] Nombre_de_Función
( [{ @parametro tipo_parametro_scalar [= default]} [...n]])
RETURNS TABLE
[WITH <opcion>::={SCHEMABINDING | ENCRYPTION}]
RETURN [(] sentencia_select [)]

```

Una función InLine podría retornar los autores de un estado en particular:

```

CREATE FUNCTION dbo.AutoresPorEstado (@State char(2) )
RETURNS TABLE
AS
RETURN (SELECT * FROM Authors WHERE state = @cState)

```

Las funciones de tabla de multi sentencias

Son similares a los procedimientos almacenados excepto que devuelven una tabla. Este tipo de función se usa en situaciones donde se requiere más lógica y proceso. Lo que sigue es la sintaxis para una función de tabla de multi sentencias:

```

CREATE FUNCTION [propietario] Nombre_de_Función
( [{ @parametro tipo_parametro_scalar [= default]} [...n]])
RETURNS TABLE
[WITH <opción>::={SCHEMABINDING | ENCRYPTION}]
[AS]
BEGIN
Código de la función
RETURN
END

```

Llamando Funciones

SQL 2000 proporciona algunas funciones definidas por el usuario a nivel sistema en la base de datos Master. Estas funciones del sistema se invocan con un sintaxis ligeramente distinta a las que usted puede crear. Las funciones del sistema que devuelven un tabla tienen la sintaxis siguiente:

Nombre de Funcion([argumento_expr], [...])

Las funciones escalares y las de conjunto de filas son invocadas con el siguiente formato:

[BD] propietario. Funcion ([argumento_expr], [...])

Limitaciones

Las funciones definidas por el usuario tienen algunas restricciones. No todas las sentencias SQL son válidas dentro de una función. Las listas siguientes enumeran las operaciones válidas e inválidas de la funciones:

Válido:

Las sentencias de asignación

Las sentencias de Control de Flujo

Sentencias SELECT y modificación de variables locales

Operaciones de cursores sobre variables locales Sentencias INSERT, UPDATE, DELETE con variables Locales

Inválidas:

Armar funciones no determinadas como GetDate()

Sentencias de modificación o actualización de tablas o vistas

Operaciones CURSOR FETCH que devuelven datos del cliente

Columnas computadas

Las funciones escalares se pueden utilizar para crear columnas calculadas en una definición de tabla. Los argumentos de las funciones calculadas, columnas de la ficha, constantes, o funciones incorporadas. Este ejemplo muestra un tabla que utilice una función del volumen para calcular el volumen de un envase

```
CREATE FUNCTION dbo.Volume ( @Height decimal(5,2),  
@Length decimal(5,2),
```

```
@Width decimal(5,2) )
RETURNS decimal (15,4)
AS
BEGIN
RETURN (@dHeight * @dLength * @dWidth )
END

CREATE TABLE dbo.Container
(
ContainerID int NOT NULL PRIMARY KEY,
MaterialID int NOT NULL REFERENCES Material(MaterialID),
ManufacturerID int NOT NULL REFERENCES Manufacturer(ManufacturerID)
Height decimal(5,2) NOT NULL,
Length decimal(5,2) NOT NULL,
Width decimal(5,2) NOT NULL,
Volume AS (Height * Length * Width), --Computed column
)
-- Volume is calculated as Height * Length * Width
```

Glosario

abstract data type (ADT)

Es un tipo de dato definido por el usuario en el cual se encapsula un rango de valores de datos y funciones. The functions are both defined on, y operadas en el set of values

alternate key

Columnas o columnas cuyo valor únicamente identifica a un registro en una tabla y no son llaves primarias en una columna.

business rule

Sentencia escrita en la cual se especifica como debe ser la información del sistema o como debe ser estructurada para soportar los negocios necesarios.

clustered index

Indice en el cual el orden físico y el orden lógico(indexado) es el mismo.

column

Estructura de datos que contiene un dato individual por registro, equivalente a un campo en un modelo de Base de Datos.

constraint

Relación que fuerza a verificar requerimientos de datos, valores En forma predeterminada o integridad referencial en una tabla o columna.

domain

Predetermina tipos de datos usados mas frecuentemente por los data item

extended attribute

Información Adicional que completa la definición de un objeto para la documentación propuesta o para el uso de una aplicación externa como un Lenguaje de Cuarta Generación(4GL)

FOREIGN KEY

Columna o columnas cuyo valores son dependientes y han sido migrados de una llave primaria o una llave alternativa desde otra tabla.

4gl

Aplicación externa basada en un Lenguaje de Cuarta Generación, usada usualmente para generar Aplicaciones Cliente / Servidor.

index

Estructura de datos basados sobre una llave, cuya finalidad es definir la velocidad de acceso a los datos de una tabla y controlar valores únicos.

odbc

Open Database Connectivity (ODBC), interface la cual provee a PowerDesigner acceso a la data de un Manejador de Base de Datos (DBMS)

odbc driver

Parte de el Open Database Connectivity (ODBC), interface que procesa llamadas de funciones del ODBC, recibe requerimientos SQL de un especifico data source, y retorna resultados a la aplicación.

PRIMARY KEY

Columna o columnas cuyos valores son identificados como valores únicos en el registro de una tabla.

REFERENCE

Relación entre una tabla padre y una tabla hijo. Una referencia puede relacionar tablas por llaves compartidas o por columnas específicas.

REFERENCIAL INTEGRITY

Reglas de consistencia de datos, específicamente las relaciones existentes entre primary keys y foreign keys de tablas diferentes.

TABLE

Colección de registros que tienen columnas asociadas.

DESENCADENADOR

Forma especial de Procedimientos Almacenados, el cual toma efecto cuando se realiza una transacción SQL en la Base de Datos ya sea un UPDATE, INSET o DELETE.

Referencia del Transact-SQL

TIPOS DE VALOR

Utilizar datos char y varchar

Los tipos de datos **char** y **varchar** almacenan datos compuestos de:

- Caracteres en mayúsculas o minúsculas, como, por ejemplo, a, b y C.
- Numerales, como 1, 2 ó 3, por ejemplo.
- Caracteres especiales, como, por ejemplo, @, & y !.

Los datos **char** o **varchar** pueden ser un carácter individual o una cadena de hasta 8.000 caracteres.

Las constantes de caracteres deben incluirse entre comillas simples ('') o comillas dobles (""). Siempre se puede incluir una constante de caracteres entre comillas simples; además, ésta es una práctica recomendable. Cuando la opción QUOTED IDENTIFIER está activada, algunas veces no se permite incluir una constante de carácter entre comillas dobles.

A continuación se muestra un ejemplo de Transact-SQL donde se asigna un valor a una variable de carácter.

```
DECLARE @MyCharVar CHAR(25)
SET @MyCharVar = 'Ricardo Adocicados'
```

Cuando use comillas simples para delimitar una constante de carácter que contenga una comilla simple incrustada, utilice dos comillas simples para representar la comilla simple incrustada, por ejemplo:

```
SET @MyCharVar = 'O''Leary'
```

Si los datos que se van a almacenar tienen una longitud mayor que el número de caracteres permitido, se truncan los datos. Por ejemplo, si una columna se define como **char(10)** y en la columna se almacena el valor “Esta es una cadena de caracteres realmente larga”, Microsoft® SQL Server™ trunca la cadena de caracteres a “Esta es un”.

El tipo de datos **char** es un tipo de datos de longitud fija cuando se especifica la cláusula NOT NULL. Si en una columna **char** NOT NULL se inserta un valor más corto que la longitud de la columna, el valor se rellena a la derecha con blancos hasta completar el tamaño de la columna. Por ejemplo, si una columna se define como **char(10)** y el dato que se va a almacenar es “música”, SQL Server almacena este dato como “música____” donde “_” indica un blanco.

El tipo de datos **varchar** es de longitud variable. Los valores más cortos que el tamaño de la columna no se llenan a la derecha hasta completar el tamaño de la misma. Si la opción ANSI_PADDING estaba desactivada cuando se creó la columna, cualquier

blanco a la derecha será recortado de los valores de carácter almacenados en la columna. Si ANSI_PADDING estaba activado cuando se creó la columna, los blancos a la derecha no se recortarán.

Si ANSI_PADDING está activado cuando se crea una columna **char** NULL, se comporta igual que una columna **char** NOT NULL y los valores son rellenados a la derecha hasta el tamaño de la columna. Si ANSI_PADDING está desactivado cuando se crea la columna **char** NULL, se comporta igual que una columna **varchar** con ANSI_PADDING desactivado, y se recortan los blancos de relleno.

La función CHAR se puede usar para convertir un código entero a un carácter ASCII. Esto resulta de utilidad cuando se intenta especificar caracteres de control como, por ejemplo, un retorno de carro o un avance de línea. Utilice CHAR(13) y CHAR(10) para colocar una nueva línea y un retorno de carro en una cadena de caracteres:

```
PRINT 'First line.' + CHAR(13) + CHAR(10) + 'Second line.'
```

La forma en que se interpretan los patrones de bit almacenados en los bytes de una cadena de caracteres se basa en la página de códigos de Microsoft SQL Server que se ha especificado durante la instalación. Un objeto **char** o **varchar** puede contener cualquier carácter de la página de códigos de SQL Server.

Las aplicaciones de bibliotecas de bases de datos y las aplicaciones que usan los controladores ODBC de SQL Server de las versiones SQL Server 6.5 o anteriores sólo admiten hasta 255 bytes de datos de caracteres. Si estas aplicaciones intentan recuperar parámetros de carácter de SQL Server versión 7.0 o columnas de conjuntos de resultados que contengan más de 255 bytes de datos, los datos de carácter se truncan a 255 bytes.

Utilizar datos de fecha y hora

Microsoft® SQL Server™ tiene los tipos de datos **datetime** y **smalldatetime** para almacenar datos de fecha y hora.

No hay tipos de datos diferentes de hora y fecha para almacenar sólo horas o sólo fechas. Si sólo se especifica una hora cuando se establece un valor **datetime** o **smalldatetime**, el valor predeterminado de la fecha es el 1 de enero de 1900. Si sólo se especifica una fecha, la hora será, de forma predeterminada, 12:00 a.m. (medianoche).

En los datos de fecha y hora puede realizar las siguientes operaciones:

- Escribir fechas nuevas o cambiar las existentes.
- Realizar cálculos de fecha y hora, como sumar o restar fechas.
- Buscar una hora y/o fecha determinada.

En los datos **datetime**, puede realizar algunos cálculos aritméticos con las funciones de fecha del sistema.

A continuación se muestran algunas directrices acerca de la utilización de datos de fecha y hora:

- Para buscar una coincidencia exacta tanto de fecha como de hora, use un signo igual (=). Microsoft SQL Server devuelve valores de fecha y hora que coincidan exactamente con el mes, día y año, y a la hora exacta 12:00:00:000 a.m. (de forma predeterminada).
- Para buscar un valor parcial de fecha u hora, use el operador LIKE. SQL Server convierte primero las fechas al formato **datetime** y, a continuación, a **varchar**. Puesto que los formatos de presentación estándar no incluyen segundos ni milisegundos, no puede buscarlos con LIKE y un patrón de coincidencia, a menos que utilice la función CONVERT con el parámetro *estilo* establecido en 9 ó 109.
- SQL Server 7.0 evalúa las constantes **datetime** en tiempo de ejecución. Una cadena de fecha que funcione para los formatos de fecha que espera un idioma puede resultar irreconocible si la consulta se ejecuta mediante una conexión que utiliza un idioma y configuración de formato de fechas diferentes. Por ejemplo, la vista siguiente funciona correctamente en conexiones realizadas con el idioma configurado como inglés de EE.UU., pero no funciona en las conexiones realizadas con otros idiomas:

```
CREATE VIEW USA_Dates AS
SELECT *
FROM Northwind.dbo.Orders
WHERE OrderDate < 'May 1, 1997'
```

Cuando utilice constantes **datetime** en consultas ejecutadas en conexiones que utilizan diferentes configuraciones de idioma, debe tomar precauciones para comprobar que las fechas se aceptan en todas las configuraciones de idioma. Debe tomar las mismas precauciones cuando utilice constantes **datetime** en objetos permanentes de bases de datos internacionales como, por ejemplo, las restricciones de tabla y las cláusulas WHERE de vistas..

SQL Server reconoce los datos de fecha y hora incluidos entre comillas simples ('') con los siguientes formatos:

- Formatos alfabéticos de fecha (por ejemplo, '15 de abril de 1998')
- Formatos numéricos de fecha (por ejemplo, '15/4/1998', '15 de abril de 1998')
- Formatos de cadenas sin separar (por ejemplo, '19981207', '12 de Diciembre de 1998')

Formato alfabético de las fechas

Microsoft® SQL Server™ permite especificar datos de fechas con un mes indicado con su nombre completo (por ejemplo, Abril) o la abreviación del mes dado (por ejemplo, Abr) en el idioma actual; las comas son opcionales y las mayúsculas se pasan por alto.

Algunas directrices para la utilización de formatos alfabéticos de fecha son:

- Incluya los datos de fecha y hora entre comillas simples (').
- Éstos son formatos alfabéticos válidos para los datos de fecha de SQL Server (los caracteres [] indican caracteres opcionales):

```

Apr[il] [15][,] 1996
Apr[il] 15[,][19]96
Apr[il] 1996 [15]
[15] Apr[il][,] 1996
15 Apr[il][,][19]96
15 [19]96 apr[il]
[15] 1996 apr[il]
1996 APR[IL] [15]
1996 [15] APR[IL]

```

- Si especifica solamente los dos últimos dígitos del año, los valores que son menores que los dos últimos dígitos de la opción de configuración **two digit year cutoff** (reducción del año a dos dígitos) pertenecen al mismo siglo que el año reducido. Los valores mayores o iguales que el valor de esta opción pertenecen al siglo anterior al año reducido. Por ejemplo, el valor de la opción **two digit year cutoff** es 2050 (valor predeterminado), 25 se interpreta como 2025 y 50 se interpreta como 1950. Para evitar la ambigüedad, utilice años de cuatro dígitos.
- Si falta el día, se supone el primer día del mes.
- La configuración de sesión SET DATEFORMAT no se aplica cuando se especifica el mes de forma alfabética.

Formato numérico de fecha

Microsoft® SQL Server™ permite especificar datos de fecha en la que aparece un mes en forma de número. Por ejemplo, 5/20/97 es el día 20 de mayo del año 1997. Cuando use el formato numérico de fecha, especifique el mes, día y año en una cadena con barras diagonales (/), guiones (-) o puntos (.) como separadores. Esta cadena debe aparecer de la forma siguiente:

número **separador** número **separador** número [hora] [hora]

Estos formatos numéricos son válidos:

```

[0]4/15/[19]96 -- (mdy)
[0]4-15-[19]96 -- (mdy)
[0]4.15.[19]96 -- (mdy)
[04]/[19]96/15 -- (myd)
15/[0]4/[19]96 -- (dmy)
15/[19]96/[0]4 -- (dym)
[19]96/15/[0]4 -- (ydm)
[19]96/[04]/15 -- (ymd)

```

Cuando el idioma está establecido a **us_english**, el orden predeterminado de la fecha es mda. Con la instrucción SET DATEFORMAT puede cambiar el orden de la fecha, lo que también puede afectar al formato de la hora, según el idioma.

La configuración de SET DATEFORMAT determina cómo se interpretan los valores de fecha. Si el orden no coincide con la configuración, los valores no se interpretan como fechas (puesto que se encuentran fuera del intervalo), o los valores se interpretan incorrectamente. Por ejemplo, 12/10/08 se puede interpretar de seis formas distintas, según la configuración de DATEFORMAT.

Formato de cadena sin separar

Microsoft® SQL Server™ permite especificar datos de fecha como una cadena sin separar. Los datos de fecha se pueden especificar con cuatro, seis u ocho dígitos, una cadena vacía o un valor de hora sin un valor de fecha.

La configuración de sesión SET DATEFORMAT no se aplica a las entradas de fecha totalmente numéricas (entradas numéricas sin separadores). Las cadenas de seis u ocho dígitos se interpretan siempre como año. El mes y el día deben ser siempre de dos dígitos.

Éste es el formato válido de una cadena sin separar:

[19]960415

Una cadena de sólo cuatro dígitos se interpreta como el año. El mes y el día se establecen a 1 de enero. Cuando se especifican sólo cuatro dígitos, es necesario incluir el siglo.

Formatos de hora

Microsoft® SQL Server™ reconoce los siguientes formatos de datos de hora. Incluya cada formato entre comillas simples ('').

14:30
14:30[:20:999]
14:30[:20.9]
4am
4 PM
[0]4[:30:20:500]AM

Puede especificar el sufijo AM o PM para indicar si el valor de la hora es anterior o posterior a las 12 del mediodía. No se distingue entre mayúsculas y minúsculas en AM o PM.

Las horas se pueden especificar con el reloj de 12 o de 24 horas. De forma predeterminada, las horas del intervalo de 0 a 12 son a.m. (AM), y las horas del intervalo de 13 a 23 son p.m. (PM). Las horas del intervalo de 1 a 12 representan horas antes del mediodía si se especifica AM y representan horas posteriores al mediodía si se especifica PM. Las horas del intervalo 13 a 23 representan horas posteriores al mediodía, independientemente de si se especifica PM. No es válido especificar PM en una hora del intervalo de 0 a 12. El valor 24:00 no es válido: use 12:00AM, 12:00 o 00:00 para representar la medianoche.

Los milisegundos se pueden preceder de dos puntos (:) o un punto (.). Si se preceden de dos puntos, el número significa milésimas de segundo. Si se precede de un punto, un único dígito significa décimas de segundo, dos dígitos significa centésimas de segundo y tres dígitos significa milésimas de segundo. Por ejemplo, 12:30:20:1 significa las 12:30, veinte segundos y una milésima; 12:30:20.1 significa las 12:30, veinte segundos y una décima.

Formato datetime de ODBC

La API de ODBC define secuencias de escape para representar valores de fecha y de hora, que ODBC llama datos de marca de hora. Este formato de marca de hora de ODBC lo admite también la definición del lenguaje de OLE DB (DBGUID-SQL) aceptada por Microsoft OLE DB Provider for SQL Server. Las aplicaciones que usan las API basadas en ODBC, OLE DB y ADO pueden usar este formato de marca de hora de ODBC para representar fechas y horas.

Las secuencias de escape de marcas de hora de ODBC tienen el formato:

```
{ tipoLiteral 'valorConstante' }
```

tipoLiteral

Especifica el tipo de la secuencia de escape. Las marcas de hora tienen tres indicadores *tipoLiteral*:

d = sólo fecha.

t = sólo hora.

ts = marca de hora (hora + fecha).

'valorConstante'

Es el valor de la secuencia de escape. *valorConstante* debe seguir estos formatos para cada *tipoLiteral*.

<i>tipoLiteral</i>	formato <i>valorConstante</i>
d	<i>aaaa-mm-dd</i>
t	<i>hh:mm:ss[.fff]</i>
ts	<i>aaaa-mm-dd hh:mm:ss[.fff]</i>

Los siguientes son ejemplos de constantes de hora y fecha de ODBC:

```
{ ts '1998-05-02 01:23:56.123' }
{ d '1990-10-02' }
{ t '13:33:41' }
```

No confunda el nombre del tipo de datos de marca de hora de ODBC y OLE DB (timestamp) con el nombre del tipo de datos **timestamp** de Transact-SQL. El tipo de datos de marca de hora de ODBC y OLE DB guarda fechas y horas. El tipo de datos **timestamp** de Transact-SQL es un tipo de datos binario con valores no relacionados con el tiempo.

Utilizar datos enteros

Los enteros son números completos. No contienen decimales o fracciones.

Microsoft® SQL Server™ tiene tres tamaños distintos de tipos de datos enteros:

- integer o int
Tiene una longitud de 4 bytes y almacena números entre -2.147.483.648 y 2.147.483.647.
- smallint
Tiene una longitud de 2 bytes y almacena números entre -32.768 y 32.767.
- tinyint
Tiene una longitud de 1 byte y almacena números entre 0 y 255.

Los objetos y expresiones enteras se pueden usar en cualquier operación matemática. Cualquier fracción generada por estas operaciones será truncada, no redondeada. Por ejemplo, SELECT 5/3 devuelve el valor 1, no el valor 2 que devolvería si se redondeara el resultado fraccionario.

Los tipos de datos enteros son los únicos que se pueden usar con la propiedad IDENTITY, que es un número que se incrementa automáticamente. La propiedad IDENTITY se usa normalmente para generar automáticamente números exclusivos de identificación o claves principales.

No es necesario incluir los datos enteros entre comillas simples como los datos de carácter o de fecha y hora.

Utilizar datos decimal, float y real

El tipo de datos **decimal** puede almacenar hasta 38 dígitos y todos pueden estar a la derecha del separador decimal. El tipo de datos **decimal** almacena una representación exacta del número; no hay una aproximación del valor almacenado.

Los dos atributos que definen las columnas, variables y parámetros **decimal** son:

- *p*
Especifica la precisión, o el número de dígitos que puede contener el objeto.
- *s*
Especifica la escala, o el número de dígitos que puede ir a la derecha del separador decimal.

p y *s* deben seguir la regla: $0 \leq s \leq p \leq 38$.

Use el tipo de datos **decimal** para almacenar números con decimales cuando los valores de datos se deban almacenar exactamente como se especifican.

En Transact-SQL, numeric es un sinónimo del tipo de datos decimal.

Utilizar datos float y real

Los tipos de datos **float** y **real** se conocen como tipos de datos aproximados. El comportamiento de **float** y **real** sigue la especificación IEEE 754 acerca de los tipos de datos numéricos aproximados.

Los tipos de datos numéricos aproximados no almacenan los valores exactos especificados para muchos números; almacenan una aproximación muy precisa del valor. Para muchas aplicaciones, la pequeña diferencia entre el valor especificado y la aproximación almacenada no es apreciable. Sin embargo, a veces la diferencia se hace notar. Debido a esta naturaleza aproximada de los tipos de datos **float** y **real**, no los use cuando necesite un comportamiento numérico exacto, como, por ejemplo, en aplicaciones financieras, en operaciones que conlleven un redondeo o en comprobaciones de igualdad. En su lugar, use los tipos de datos enteros, **decimal**, **money** o **smallmoney**.

Evite usar columnas **float** o **real** en las condiciones de búsqueda de la cláusula WHERE, especialmente los operadores = y <>. Es mejor limitar las columnas **float** y **real** a las comparaciones > o <.

La especificación IEEE 754 proporciona cuatro modos de redondeo: redondear al más cercano, redondear hacia arriba, redondear hacia abajo y redondear hacia cero. Microsoft® SQL Server™ usa el redondeo hacia arriba. Todos son precisos para garantizar la exactitud, aunque pueden dar como resultado valores en punto flotante ligeramente distintos. Puesto que la representación binaria de un número en punto flotante puede usar cualquiera de los esquemas válidos de redondeo, es imposible cuantificar de forma precisa un valor en punto flotante.

Utilizar datos text e image

Microsoft® SQL Server™ almacena cadenas de caracteres con más de 8.000 caracteres y datos binarios con más de 8.000 bytes en tipos de datos especiales llamados **text** e **image**. Las cadenas Unicode superiores a 4.000 caracteres se almacenan en el tipo de datos **ntext**.

Por ejemplo, suponga que tiene un archivo grande de texto (.TXT) de información de clientes que es necesario importar a la base de datos de SQL Server. Estos datos desea que se almacenen como un único dato, en lugar de integrarlos en las diversas columnas de las tablas de datos. Con este propósito puede crear una columna del tipo de datos **text**. Sin embargo, si necesita almacenar logotipos de empresa, almacenados actualmente como imágenes TIFF (.TIF) de 10 KB cada uno, cree una columna del tipo de datos **image**.

Si los datos del texto que desea almacenar se encuentran en formato Unicode, use el tipo de datos **ntext**. Por ejemplo, una carta modelo creada para clientes internacionales seguramente contendrá ortografías y caracteres internacionales usados en varias culturas distintas. Almacene estos datos en una columna **ntext**.

SQL Server interpreta los datos **text** como series de caracteres mediante la página de códigos instalada con SQL Server. SQL Server interpreta los datos **ntext** como series de caracteres con la especificación de Unicode.

Los datos de tipo **image** se almacenan como una cadena de bits y no son interpretados por SQL Server. Cualquier interpretación de los datos de una columna **image** debe ser realizada por la aplicación. Por ejemplo, una aplicación podría almacenar datos en una columna **image** con el formato BMP, TIFF, GIF o JPEG. Depende de la aplicación que lee los datos de la columna **image** reconocer o no el formato de los datos y mostrarlos correctamente. Todo lo que hace una columna **image** es proporcionar una ubicación para almacenar la secuencia de bits que conforman los datos de la imagen.

Utilizar Constantes

Una constante es un símbolo que representa el valor específico de un dato. El formato de las constantes depende del tipo de datos del valor que representa. Las constantes se llaman también literales. Algunos ejemplos de constantes son:

- Cadenas de caracteres:

```
'O'''Brien'  
'The level for job_id: %d should be between %d and %d.'
```

- Cadenas Unicode:

```
N'Michél'
```

- Constantes de cadenas binarias:

```
0x12Ef  
0x69048AEFDD010E
```

- Las constantes **bit** se representan con los números 0 o 1.

- Constantes **datetime**:

```
'April 15, 1998'  
'04/15/98'  
'14:30:24'  
'04:24 PM'
```

- Constantes **integer**:

```
1894  
2
```

- Constantes **decimal**:

```
1894.1204  
2.0
```

- Constantes **float** y **real**

```
101.5E5  
0.5E-2
```

- Constantes **money**:

```
$12  
$542023.14
```

- Constantes **uniqueidentifier**:

```
0xff19966f868b11d0b42d00c04fc964ff  
'6F9619FF-8B86-D011-B42D-00C04FC964FF'
```

Para las constantes numéricas, use los operadores unarios + y - cuando sea necesario especificar el signo del valor numérico:

```
+$156.45  
-73.52E8  
-129.42  
+442
```

Utilizar constantes en Transact-SQL

Las constantes se pueden usar de muchas formas en Transact-SQL. A continuación se muestran algunos ejemplos:

- Como un valor constante de una expresión aritmética:

```
SELECT Price + $.10  
      FROM MyTable
```

- Como el valor con el que se compara una columna en una cláusula

WHERE:

```
SELECT *  
      FROM MyTable  
     WHERE LastName = 'O''Brien'
```

- Como el valor que se va a colocar en una variable:

```
SET @DecimalVar = -1200.02
```

- Como el valor que debe colocarse en una columna de la fila actual.

Esto se especifica con la cláusula SET de la instrucción UPDATE o la cláusula VALUES de una instrucción INSERT:

```
UPDATE MyTable  
   SET Price = $99.99  
     WHERE PartNmb = 1234  
INSERT INTO MyTable VALUES (1235, $88.88)
```

- Como la cadena de caracteres que especifica el texto del mensaje emitido por una instrucción PRINT o RAISERROR:

```
PRINT 'This is a message.'
```

- Como el valor que se va a probar en una instrucción condicional, como, por ejemplo, una instrucción IF o funciones CASE:

```
IF (@@SALESTOTAL > $100000.00)  
EXECUTE Give_Bonus_Procedure
```

Funciones

Microsoft® SQL Server™ dispone de funciones integradas para realizar ciertas operaciones rápida y fácilmente. Las categorías en que se dividen las funciones son:

Funciones de agregado

Realizan operaciones que combinan varios valores en uno.
Ejemplos son COUNT, SUM, MIN y MAX.

Funciones de configuración

Son funciones escalares que devuelven información acerca de la configuración.

Funciones de cursos

Devuelven información acerca del estado de un cursor.

Funciones de fecha y hora

Tratan valores **datetime** y **smalldatetime**.

Funciones matemáticas

Realizan operaciones trigonométricas, geométricas y demás operaciones numéricas.

Funciones de metadatos

Devuelven información acerca de los atributos de las bases de datos y de los objetos de base de datos.

Funciones de conjuntos de filas

Devuelven conjuntos de filas que se pueden usar en el lugar de una referencia de tabla de una instrucción de Transact-SQL.

Funciones de seguridad

Devuelven información acerca de usuarios y funciones.

Funciones de cadena

Tratan valores **char**, **varchar**, **nchar**, **nvarchar**, **binary** y **varbinary**.

Funciones del sistema

Funcionan en o informan acerca de varias opciones y objetos del sistema.

Funciones de estadísticas del sistema

Devuelven información relacionada con el rendimiento de SQL Server.

Funciones de texto e imagen

Tratan valores **text** e **image**.

Las funciones se pueden usar o incluir en:

- La lista de selección de una consulta que usa una instrucción SELECT para devolver un valor.
`SELECT DB_NAME()`
- Una condición de búsqueda de una cláusula WHERE de una instrucción SELECT o de modificación de datos (SELECT, INSERT, DELETE o UPDATE) para limitar las filas adecuadas para la consulta.

```
SELECT *
FROM [Order Details]
WHERE Quantity =
(SELECT MAX(Quantity) FROM [Order Details])
```

- La condición de búsqueda (condición WHERE) de una vista para hacer que la vista se adapte dinámicamente al usuario o entorno en tiempo de ejecución.

```
CREATE VIEW ShowMyEmploymentInfo AS
SELECT * FROM Employees
WHERE EmployeeID = SUSUARIO_SID()
GO
```

- Cualquier expresión.
- Un desencadenador o restricción CHECK para comprobar los valores especificados cuando se insertan datos.

```
CREATE TABLE SalesContacts
(SalesRepID INT PRIMARY KEY CHECK (SalesRepID = SUSUARIO_SID() ) ,
ContactName VARCHAR(50) NULL,
ContactPhone VARCHAR(13) NULL)
```

- Un desencadenador o restricción DEFAULT para suministrar un valor en el caso de que no se especifique ninguno en una instrucción INSERT.

```
CREATE TABLE SalesContacts
(
SalesRepID INT PRIMARY KEY CHECK (SalesRepID = SUSUARIO_SID() ) ,
ContactName VARCHAR(50) NULL,
ContactPhone VARCHAR(13) NULL,
WhenCreated DATETIME DEFAULT GETDATE(),
Creator INT DEFAULT SUSUARIO_SID()
)
GO
```

Las funciones se usan siempre con paréntesis, incluso cuando no haya parámetros. Una excepción son las funciones niládicas (funciones que no toman parámetros) usadas con la palabra clave DEFAULT.

Algunas veces, los parámetros que especifican una base de datos, equipo, inicio de sesión o usuario de base de datos son opcionales. Si no se proporcionan, el valor predeterminado es el de la base de datos, equipo host, inicio de sesión o usuario de base de datos actual.

Las funciones se pueden anidar (una función se usa dentro de otra función).

Utilizar funciones del sistema

Las funciones del sistema permiten que tenga acceso a la información de las tablas del sistema de Microsoft® SQL Server™ sin tener acceso directamente a las tablas del sistema.

Este grupo de cinco pares de funciones del sistema para bases de datos, hosts, objetos, inicio de sesión y usuarios devuelven un nombre cuando se les proporciona un identificador y devuelven un identificador cuando se les proporciona un nombre:

- DB_ID y DB_NAME
- HOST_ID y HOST_NAME
- OBJECT_ID y OBJECT_NAME
- SUSUARIO_ID y SUSUARIO_NAME (o SUSUARIO_SID y SUSUARIO_SNAME)
- USUARIO_ID y USUARIO_NAME

Estas funciones ofrecen una forma fácil de convertir un nombre a un identificador, y un identificador a un nombre. Por ejemplo, use la función DB_ID para obtener un número de Id. de base de datos en lugar de ejecutar una instrucción SELECT de la tabla **sysobjects**.

En este ejemplo se recupera el nombre del usuario actual conectado (mediante Autenticación de SQL Server).

```
SELECT SUSUARIO_NAME()
```

Las siguientes funciones son similares, pero no se producen en pares complementarios y necesitan más de un parámetro de entrada:

- **COL_NAME**
Devuelve un nombre de columna.
- **COL_LENGTH**
Devuelve la longitud de una columna.
- **INDEX_COL**
Devuelve el nombre de la columna de un índice.
- **COL_LENGTH**
devuelve la longitud de una columna, no la longitud de ninguna cadena individual almacenada en la columna. Use la función DATALENGTH para determinar el número total de caracteres en un valor determinado.

En este ejemplo se devuelve la longitud de la columna y la longitud de los datos de la columna **LastNames** de la tabla **Employees**.

```
SELECT COL_LENGTH('Employees', 'LastNames') AS Col_Length,  
DATALENGTH(LastNames) AS DataLength  
FROM Employees  
WHERE EmployeeID > 6
```

Nota Se recomienda usar las funciones del sistema, las vistas del sistema de información o los procedimientos almacenados del sistema para obtener acceso a la información del sistema sin consultar directamente las tablas del sistema. Las tablas del sistema pueden cambiar significativamente entre versiones de SQL Server.

Utilizar funciones de cadena

Las funciones de cadena se usan para realizar varias operaciones en cadenas de caracteres y binarias, y devuelven valores que, normalmente, son necesarios para las operaciones con los datos de caracteres. La mayor parte de las funciones de cadena se pueden usar sólo en los tipos de datos **char**, **nchar**, **varchar** y **nvarchar** o en los tipos de datos que se convierten a ellos implícitamente. En los datos **binary** y **varbinary** se pueden usar también unas cuantas funciones de cadena.

Las funciones de cadena se pueden usar para:

- Recuperar sólo una parte de una cadena (SUBSTRING).

- Buscar similitudes en el sonido de una cadena de caracteres (SOUNDEX y DIFFERENCE).
- Buscar una posición de inicio de una cadena particular en una columna o expresión. Por ejemplo, la posición de la letra A en “¡Qué día tan bonito!”.
- Concatenar o combinar cadenas en una sola. Por ejemplo, combinar un nombre, apellido y segundo nombre o inicial en un nombre completo.
- Convertir un valor que no sea de cadena a un valor de cadena (como, por ejemplo, convertir el valor 15,7, almacenado como **float**, a **char**).
- Insertar una cadena específica en una cadena existente. Por ejemplo, insertar la cadena “una vez” en una cadena existente “Érase” para producir la cadena “Érase una vez”.

Utilizar SUBSTRING

La función SUBSTRING devuelve una parte de un carácter o cadena binaria, o una cadena de texto, y toma tres parámetros:

- 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 este ejemplo se muestra la primera inicial y el nombre de cada empleado, como, por ejemplo, A Fuller.

```
USE Northwind
SELECT SUBSTRING(FirstName, 1, 1), LastName
FROM Employees
```

En este ejemplo se muestran el segundo, tercero y cuarto caracteres de la constante de cadena abcdef.

```
SELECT x = SUBSTRING('abcdef', 2, 3)
x
-----
bcd
(1 row(s) affected)
```

Comparación de CHARINDEX y PATINDEX

Las funciones CHARINDEX y PATINDEX devuelven la posición de inicio del patrón que se especifique. PATINDEX puede usar caracteres comodín, mientras que CHARINDEX no puede.

Las funciones toman dos parámetros:

- El patrón cuya posición se desea obtener. Con PATINDEX, el patrón es una cadena de literales que puede contener caracteres comodín. Con CHARINDEX, el patrón es una cadena de literales (sin caracteres comodín).

- Una expresión que da como resultado una cadena, normalmente, un nombre de columna, en la que Microsoft® SQL Server™ busca la cadena especificada.

Por ejemplo, para buscar la posición en que comienza la cadena “wonderful” en una fila específica de la columna **notes** de la tabla **titles**.

```
USE pubs
SELECT CHARINDEX( 'wonderful' , notes )
FROM titles
WHERE title_id = 'TC3218'
Éste es el conjunto de
resultados:
-----
46
(1 row(s) affected)
```

Si no restringe las filas en las que buscar, la consulta devolverá todas las filas de la tabla e indicará valores distintos de cero para las filas en las que haya encontrado la cadena y cero para todas las restantes.

Por ejemplo, para utilizar caracteres comodín para encontrar la posición en la que comienza la cadena “breads” en cualquier fila de la columna **Description** de la tabla **Categories**.

```
USE Northwind
GO
SELECT CategoryID, PATINDEX( '%candies%' , LOWER(Description) )
FROM Categories
WHERE PATINDEX( '%candies%' , Description ) <> 0
```

Si no restringe las filas en las que buscar, la consulta devolverá todas las filas de la tabla e indicará valores distintos de cero para las filas en las que haya encontrado la cadena.

PATINDEX resulta de utilidad con los tipos de datos **text** y se puede emplear en una cláusula WHERE además de IS NULL, IS NOT NULL y LIKE (las únicas comparaciones adicionales válidas con **text** en una cláusula WHERE).

Utilizar STR

La función STR convierte números a caracteres, con parámetros opcionales para especificar la longitud total del resultado, incluidos el separador decimal y el número de posiciones que siguen al separador decimal.

Los parámetros longitud y decimal de STR (si se suministran) deben ser positivos. La longitud predeterminada es 10. De forma predeterminada o si el parámetro decimal es 0, el número se redondea a un entero. La longitud especificada debe ser mayor o igual que la longitud de la parte del número anterior al separador decimal, más el signo (si corresponde).

En este ejemplo se convierte la expresión float 123,45 a un carácter, con una longitud de 6 caracteres y 2 lugares decimales.

```
SELECT STR(123.45, 6, 2)
Este es el conjunto de resultados:
-----
123.45
(1 row(s) affected)
```

Si la parte entera de la expresión que se está convirtiendo a una cadena de caracteres excede de la longitud especificada en STR, STR devuelve ** para la longitud especificada. Por ejemplo, el número 1234567,89 tiene 7 dígitos a la izquierda del separador decimal. Si el parámetro que indica la longitud de STR es 7 o más, la cadena resultante contiene el entero y tantos decimales como quepan. Si el parámetro que indica la longitud de STR es 6 o menos, se devuelven asteriscos. Por ejemplo, el lote:

```
SELECT STR(1234567.89, 7, 2)
SELECT STR(1234567.89, 6, 2)
```

Devuelve:

```
-----
1234568
(1 row(s) affected)
-----
*****
(1 row(s) affected)
```

STR ofrece más flexibilidad que CAST al convertir tipos de datos **decimal** a tipos de datos de caracteres porque proporciona un control explícito del formato.

Utilizar STUFF

La función STUFF inserta una cadena en otra. Elimina una longitud determinada de caracteres de la primera cadena a partir de la posición de inicio y, a continuación, inserta la segunda cadena en la primera, en la posición de inicio.

Si la posición de inicio o la longitud es negativa, o si la posición de inicio es mayor que la longitud de la primera cadena, se devuelve una cadena Null. Si la longitud que se va a eliminar es mayor que la primera cadena, se elimina hasta el primer carácter de la primera cadena.

En este ejemplo se coloca la cadena de caracteres “xyz” a continuación del segundo carácter de la expresión de caracteres “abc” y sustituye un total de tres caracteres.
SELECT STUFF('abc', 2, 3, 'xyz')

Este es el conjunto de resultados:

```
-----
axyz
(1 row(s) affected)
```

Comparación de SOUNDEX y DIFFERENCE

La función SOUNDEX convierte una cadena de caracteres a un código de cuatro dígitos para ser utilizado en una comparación. En la comparación se pasan por alto las vocales.

Para determinar la comparación se usan caracteres no alfabéticos. Esta función siempre devuelve un valor.

En este ejemplo se muestran los resultados de la función SOUNDEX para las cadenas similares de caracteres “Smith” y “Smythe”. Cuando las cadenas de caracteres son similares, ambas tienen los mismos códigos SOUNDEX.

```
SELECT SOUNDEX ('smith'), SOUNDEX ('smythe')
```

Éste es el conjunto de resultados:

```
----- -----  
S530 S530  
(1 row(s) affected)
```

La función DIFFERENCE compara los valores SOUNDEX de dos cadenas, evalúa las similitudes entre ellas y devuelve un valor entre 0 y 4, donde 4 representa la mejor coincidencia. En este ejemplo se devuelve una DIFFERENCE de 4 para el primer SELECT porque “Smithers” y “Smothers” difieren sólo en un carácter.

```
SELECT DIFFERENCE('smithers', 'smothers')
```

Éste es el conjunto de resultados:

```
-----  
4  
(1 row(s) affected)
```

En este ejemplo se devuelve una DIFFERENCE de 3 que indica que las dos cadenas de caracteres tienen un sonido similar, aunque difieren en varios caracteres.

```
SELECT DIFFERENCE('Jeff', 'Geoffe')
```

Utilizar las funciones **text**, **ntext** e **image**

Hay dos funciones **text**, **ntext** e **image** utilizadas exclusivamente para realizar operaciones en datos **text**, **ntext** e **image**:

- TEXTPTR devuelve un objeto **binary(16)** que contiene un puntero a una instancia **text**, **ntext** o **image**. El puntero es válido hasta que se elimina la fila.
- La función TEXTVALID comprueba si un determinado puntero de texto es válido o no.

Los punteros de texto se pasan a las instrucciones READTEXT, UPDATETEXT, WRITETEXT, PATINDEX, DATALENGTH y SET TEXTSIZE de Transact-SQL que se usan para el tratamiento de datos **text**, **ntext**, e **image**.

En las instrucciones de Transact-SQL, se hace referencia siempre a los datos **text**, **ntext** e **image** mediante punteros o las direcciones de los datos.

En este ejemplo se usa la función TEXTPTR para localizar la columna **text (pr_info)** asociada con **pub_id** 0736 en la tabla **pub_info** de la base de datos **pubs**. Primero se declara la variable local **@val**. El siguiente puntero (una cadena binaria larga) se coloca

entonces en **@val** y se suministra como parámetro a la instrucción READTEXT, que devuelve 10 bytes, empezando en el quinto byte (desplazamiento de 4).

```
USE pubs
DECLARE @val varbinary(16)
SELECT @val = textptr(pr_info) FROM pub_info
WHERE pub_id = '0736'
READTEXT pub_info.pr_info @val 4 10
Éste es el conjunto de resultados:
(1 row(s) affected)
pr_info
-----
is sample
```

Con la función CAST se admiten las conversiones explícitas realizadas desde los tipos de datos **text** a **varchar**, **ntext** a **nvarchar** e **image** a **varbinary** o **binary**, aunque los datos **text** o **image** se truncan a 8.000 bytes y los datos **ntext** se truncan a 4.000 bytes. No se admite la conversión, implícita ni explícita, de **text**, **ntext** o **image** a otros tipos de datos. Sin embargo, se puede hacer una conversión indirecta de **text**, **ntext** o **image**, por ejemplo:

```
CAST( CAST( text_column_name AS VARCHAR(10) ) AS INT ).
```

Utilizar funciones matemáticas

Una función matemática realiza una operación matemática en expresiones numéricas y devuelve el resultado de la operación. Las funciones matemáticas operan en los datos numéricos suministrados por el sistema de Microsoft® SQL Server™ (**decimal**, **integer**, **float**, **real**, **money**, **smallmoney**, **smallint** y **tinyint**). La precisión de las operaciones integradas para el tipo de datos **float** es, de forma predeterminada, de seis lugares 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, como, por ejemplo, **float**. Por ejemplo, el valor devuelto por la función FLOOR tiene el tipo de datos del valor de entrada. La entrada de esta instrucción SELECT es de tipo **decimal** y FLOOR devuelve 123, que es un valor decimal:

```
SELECT FLOOR(123.45)
-----
123
(1 row(s) affected)
Pero, en este ejemplo se usa un valor float y FLOOR devuelve un valor float:
SELECT FLOOR (CONVERT (float, 123.45))
-----
123.00000
(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 encontrarse entre -1,00 y 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**, mientras que POWER(-10,0, 401) es menor que el valor mínimo (~ -2e+308) del intervalo para el tipo de datos **float**.

En esta 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 usar:

- SET ARITHABORT ON, que termina la consulta y sale de la transacción definida por el usuario. La configuración de SET ARITHABORT suplanta la configuración de SET ANSI_WARNINGS.
- SET ANSI_WARNINGS ON, que detiene el comando.
- SET ARITHIGNORE ON, que hace que no se muestre ningún mensaje de advertencia. Tanto la configuración de SET ARITHABORT como de SET ANSI_WARNINGS suplantan la configuración de SET ARITHIGNORE.

Si no se ha establecido ninguna de estas opciones, Microsoft® SQL Server™ devuelve NULL y muestra un mensaje de advertencia después de ejecutar la consulta.

La conversión interna a **float** puede provocar la pérdida de precisión si se usa alguno de los tipos de datos **money** o **numeric**.

Utilizar funciones trigonométricas

Microsoft® SQL Server™ proporciona funciones trigonométricas que devuelven radianes.

Funciones que devuelven radianes	Use radianes como valor de entrada
ACOS	TAN
COS	SIN

ATAN
ATN2
COT

ASIN

ACOS y COS

Tanto ACOS como COS son funciones trigonométricas. La función ACOS devuelve el ángulo, en radianes, cuyo coseno es la expresión **float** dada. La función COS devuelve el coseno del ángulo especificado, en radianes, dada la expresión **float**. Por ejemplo, la siguiente instrucción SELECT calcula el ACOS de -0,997 y el COS del valor 1,134:

```
SELECT ACOS(-.997), COS(1.134)
```

Así, el coseno del ángulo que mide 3,06411360866591 radianes es -0,997 y el coseno del ángulo que mide 1,134 radianes es 1,134.

El intervalo válido de ACOS es de -1 a 1.

ASIN y SIN

Tanto ASIN como SIN son funciones trigonométricas que usan una expresión **float**. La función ASIN calcula el ángulo, medido en radianes, cuyo seno es la expresión **float** dada. La función SIN calcula el valor del seno trigonométrico del ángulo, medido en radianes, como una expresión **float**.

En este ejemplo se calcula el ASIN de -0,7582 y el SIN de 5. El seno del ángulo que mide -0,860548023283932, en radianes, es -0,7582 y seno del ángulo que mide 5 radianes tiene un valor de -0,958924274663138.

```
SELECT ASIN(-.7582), SIN(5)
```

El intervalo válido de ASIN es de -1 a 1.

ATAN , ATN2, TAN y COT

Las funciones ATAN, ATN2, TAN y COT son funciones matemáticas. La función ATAN devuelve la medida del ángulo, en radianes, cuya tangente es la expresión **float** dada. Un ángulo que tenga un valor de tangente de -27,29 medirá -1,53416925536896 radianes.

La función ATN2 devuelve el ángulo, en radianes, cuya tangente se encuentra entre las dos expresiones **float** dadas. Un ángulo con una tangente entre 3,273 y 15 mide 0,214832755968629 radianes.

La función TAN devuelve la tangente trigonométrica de la expresión **float** dada. Un ángulo que mide 27,92 radianes tiene una tangente de -0,36994766163616.

La función COT devuelve la cotangente trigonométrica del ángulo especificado, en radianes, indicado en la expresión **float** dada. Un ángulo de 97,1928 radianes tiene un valor de cotangente de -5,02149424849997.

DEGREES

La función DEGREES devuelve una expresión **numeric**: la medida del ángulo, en grados, de la medida del ángulo en radianes. Un ángulo que mide -14,578 radianes mide -835,257873741714090000 grados.

```
SELECT DEGREES(-14.578)
```

RADIANS

La función RADIANS calcula el ángulo en radianes dada la medida del ángulo en grados. Para calcular la medida en radianes de un ángulo que mide 10,75 grados, use:

```
SELECT RADIANS(10.75)
```

Comparación de CEILING y FLOOR

La función CEILING devuelve el menor entero que sea mayor o igual que la expresión numérica dada. La función FLOOR devuelve el mayor entero que sea menor o igual que la expresión numérica dada. Por ejemplo, dada la expresión numérica 12,9273, CEILING devuelve 13, y FLOOR devuelve 12. El valor de retorno tanto de FLOOR como de CEILING tiene el mismo tipo de datos que la expresión numérica de entrada.

Comparación de LOG y LOG10

La función LOG devuelve el logaritmo natural de la expresión **float** dada. Los logaritmos naturales se calculan con el sistema de base 2. Sin embargo, la función LOG10 devuelve el logaritmo en base 10. Use ambas funciones, LOG y LOG10, para aplicaciones trigonométricas. Por ejemplo, la siguiente instrucción SELECT calcula el LOG y el LOG10 del valor 1,75.

```
SELECT LOG(1.75), LOG10(1.75)
```

Utilizar las funciones exponenciales POWER y EXP

La función POWER devuelve el valor de la expresión numérica dada elevado a la potencia especificada. POWER(2,3) devuelve 2 elevado a la tercera potencia: el valor 8. Se pueden especificar potencias negativas, con lo que POWER(2,000, -3) devuelve 0,125. Observe que el resultado de POWER(2, -3) es 0. Esto es así porque el resultado será del mismo tipo de datos que la expresión numérica dada. Así, si el resultado tiene tres lugares decimales, el número que se va a elevar a una cierta potencia debe tener también tres decimales.

La función EXP devuelve el valor exponencial, en notación científica, de la expresión **float** dada. Así, con el valor 198,1938327, la función EXP devuelve el valor 1,18710159597953e+086.

```
SELECT EXP(198.1938327)
```

Utilizar RAND

La función RAND calcula un número aleatorio de punto flotante entre 0 y 1, y puede tomar opcionalmente un valor **tinyint**, **int** o **smallint** para el punto de inicio del número aleatorio que se va a calcular.

Este ejemplo calcula dos números aleatorios. La primera función RAND() permite a Microsoft® SQL Server™ elegir el valor de inicio, y la segunda función RAND() usa el valor 3 para la posición de inicio.

```
SELECT RAND(), RAND(3)
```

La función RAND es un pseudogenerador de números aleatorios que opera de forma similar a la función **rand** de la biblioteca de tiempo de ejecución de C. Si no se proporciona el valor de inicio, el sistema generará sus propios valores variables. Si llama a RAND con un valor de inicio, debe usar valores de inicio variables para generar números aleatorios. Si llama a RAND varias veces con el mismo valor de inicio, devolverá el mismo valor generado. La secuencia de comandos siguiente devuelve el mismo valor en las llamadas a RAND porque todas usan el mismo valor de inicio:

```
SELECT RAND(159784)
SELECT RAND(159784)
SELECT RAND(159784)
```

Una forma habitual de generar números aleatorios con RAND es incluir algo relativamente variable como valor de inicio, como, por ejemplo, agregar varias partes de un GETDATE:

```
SELECT RAND( (DATEPART(mm, GETDATE()) * 100000 )
+ (DATEPART(ss, GETDATE()) * 1000 )
+ DATEPART(ms, GETDATE()) )
```

Cuando use un algoritmo basado en GETDATE para generar valores de inicio, RAND puede seguir generando valores duplicados si las llamadas a RAND se realizan en el intervalo de la parte menor de la fecha usada en el algoritmo. Esto es lo que ocurrirá con más probabilidad si las llamadas a RAND se incluyen en un único lote. En el mismo milisecondo, que es el incremento más pequeño de DATEPART, se pueden ejecutar múltiples llamadas a RAND en un único lote. En este caso, incorpore un valor basado en algo diferente al tiempo para generar los valores de inicio.

Funciones de fecha

Las funciones de fecha se utilizan para mostrar información acerca de fechas y horas. Se usan para el tratamiento de valores **datetime** y **smalldatetime** y para realizar operaciones aritméticas en los mismos. Las funciones de fecha se pueden usar en cualquier parte donde se pueda usar una expresión.

SQL Server reconoce una amplia variedad de formatos de entrada de datos **datetime**. Puede usar la instrucción SET DATEFORMAT para establecer el orden de las partes de la fecha (mes, día y año) para introducir datos **datetime** o **smalldatetime**. Cuando escriba valores **datetime** o **smalldatetime**, inclúyalos entre comillas simples.

Utilizar GETDATE

La función GETDATE produce la fecha y hora actual en el formato interno de Microsoft® SQL Server™ para los valores **datetime**. GETDATE acepta el parámetro Null ().

En este ejemplo se averigua la fecha y hora actuales del sistema.

```
SELECT GETDATE()
```

Éste es el conjunto de resultados:

```
-----
```

```
July 29 1995 2:50 PM
(1 row(s) affected)
```

Se puede usar GETDATE para diseñar un informe de manera que se impriman la fecha y hora actual cada vez que se genere el informe. GETDATE también es útil para llevar a cabo funciones como, por ejemplo, registrar la hora de una transacción realizada en una cuenta.

Puede usar GETDATE en cualquier parte para devolver la fecha actual del sistema. Por ejemplo, puede usar GETDATE como valor predeterminado para la entrada de datos, con una variable local, o para comparar la fecha de un pedido con la fecha de hoy.

Comparación de DATEPART y DATENAME

Las funciones DATEPART y DATENAME producen la parte especificada de un valor **datetime** (el año, trimestre, día, hora, etc.) como un entero o como una cadena ASCII. Puesto que **smalldatetime** sólo es preciso hasta los minutos, cuando se use un valor **smalldatetime** con alguna de estas funciones, los segundos y milisegundos devueltos son siempre cero.

Los siguientes ejemplos suponen la fecha 29 de mayo.

```
SELECT DATEPART(mes, GETDATE())
Éste es el
conjunto de
resultados:
-----
5
(1 row(s) affected)
SELECT DATENAME(mes, GETDATE())
Éste es el
conjunto de
resultados:
-----
Mayo
(1 row(s) affected)
```

Comparación de DATEADD y DATEDIFF

La función DATEADD agrega un intervalo a la fecha que especifique. Por ejemplo, si las fechas de publicación de todos los libros de la tabla **titles** se retrasan tres días, puede obtener una nueva fecha de publicación con esta instrucción:

```
USE pubs
SELECT DATEADD(day, 3, pubdate)
FROM titles
```

Si el parámetro fecha es un tipo de datos **smalldatetime**, el resultado es también un **smalldatetime**. Puede usar DATEADD para agregar segundos o milisegundos a un valor **smalldatetime**, aunque esto sólo tiene sentido si el tipo de datos devuelto por DATEADD cambia en un minuto, como mínimo.

La función DATEDIFF calcula la cantidad de tiempo en partes de fecha entre la segunda y la primera de las dos fechas que especifique. En otras palabras, encuentra un intervalo entre dos fechas. El resultado es un valor entero con signo, igual a $fecha2 - fecha1$ en partes de fecha.

Esta consulta usa la fecha 30 de noviembre de 1995 y busca el número de días que hay entre **pubdate** (fecha de publicación) y esa fecha.

```
USE pubs
SELECT DATEDIFF(day, pubdate, 'Nov 30 1995')
FROM titles
```

Para las filas de los títulos que tengan una **pubdate** de 21 de octubre de 1995, el resultado producido por la última consulta es 40. (Hay 40 días entre el 21 de octubre y el 30 de noviembre). Para calcular un intervalo en meses, utilice esta consulta:

```
USE pubs
SELECT interval = DATEDIFF(month, pubdate, 'Nov 30 1995')
FROM titles
```

Esta consulta produce el valor 1 para las filas con una **pubdate** en octubre y el valor 5 para las filas con una **pubdate** en junio.

Cuando la primera fecha de la función DATEDIFF sea posterior a la segunda fecha especificada, el valor resultante será negativo. Puesto que dos de las filas de **titles** tienen valores para **pubdate** que son asignados con la función GETDATE como valor predeterminado, estos valores se establecen a la fecha en la que la base de datos **pubs** fue creada, y se devuelven valores negativos en las dos consultas precedentes.

Si uno o ambos argumentos de fecha son un valor **smalldatetime**, para el cálculo serán convertidos internamente a valores **datetime**. Los segundos y milisegundos de los valores **smalldatetime** se establecen automáticamente a 0 por motivos que tienen que ver con los cálculos.

Funciones que devuelven identificadores y nombres de usuarios

Varias funciones del sistema devuelven identificadores y nombres de usuarios. Para comprender los parámetros y datos de salida de estas funciones es necesario comprender los tipos de nombres e identificadores utilizados en Microsoft® SQL Server™.

Cada usuario que se conecta a SQL Server tiene dos clases de nombres en SQL Server y cada nombre está asociado con un identificador exclusivo:

- Nombres de inicio de sesión

Cada usuario autorizado para conectarse a SQL Server tiene un nombre de inicio de sesión que le proporciona acceso a una instalación de SQL Server. Hay dos tipos de nombres de inicio de sesión:

- Nombres de cuentas de Microsoft Windows NT®
Los miembros de las funciones fijas de servidor **sysadmin** o **SecurityAdmin** pueden autorizar a cuentas de Windows NT de usuarios individuales o a grupos de Windows NT para iniciar una sesión en una instalación de SQL Server con **sp_grantInicio de sesión**. El usuario identificado por la cuenta de Windows NT o cualquier persona del grupo de Windows NT pueden conectarse con la instalación de SQL Server mediante la Autenticación de Windows NT. Cada nombre de cuenta o grupo de Windows NT se almacena en **master.dbo.sysxInicios de sesión.name**.

El *identificadorDeSeguridad (Seguridad_identifier)* de Windows NT de la cuenta o grupo de Windows NT está almacenado en **sysInicios de sesión.sid**.

- Nombres de inicio de sesión de SQL Server
Los utilizan los usuarios que inician una sesión con la Autenticación de SQL Server. Los miembros de las funciones fijas de servidor **sysadmin** o **SecurityAdmin** definen los nombres de inicio de sesión de SQL Server con **sp_addInicio de sesión**. Cada nombre de inicio de sesión de SQL Server está almacenado en **master.dbo.sysxInicios de sesión.name**. SQL Server genera un GUID que se utiliza como *identificadorDeSeguridad (Seguridad_identifier)* y lo almacena en **sysxInicios de sesión.sid**.

SQL Server utiliza **master.dbo.sysxInicios de sesión.sid** como *identificadorDeSeguridad* del nombre de inicio de sesión. Las versiones anteriores de SQL Server utilizaban, en su lugar, un *IdUsuarioServidor* de una tabla diferente: **master.dbo.sysInicios de sesión.suid**. Para mantener la compatibilidad con versiones anteriores, SQL Server implementa **sysInicios de sesión** como una vista de **sysxInicios de sesión** y **sysInicios de sesión.suid** es una columna calculada que genera un *IdUsuarioServidor* a partir del *identificadorDeSeguridad* de **sysxInicios de sesión**.

- Nombre de usuario de base de datos
Cada cuenta de Windows NT o inicio de sesión de SQL Server debe estar asociado con un nombre de usuario en cada base de datos a la que tengan autorización para obtener acceso. Los miembros de las funciones fijas de base de datos **db_owner** o **db_accessadmin** definen los nombres de usuario de base de datos, y se almacenan en la tabla **sysusuarios** de cada base de datos. Cada nombre de usuario de base de datos está asociado con un identificador de usuario de base de datos almacenado en **sysusuarios.uid**.

El *identificadorDeSeguridad* de cada usuario se almacena en **sysusuarios.sid**; por lo tanto, se puede asociar de nuevo los usuarios a sus inicios de sesión correspondientes. Resulta menos confuso si los miembros de las funciones **sysadmin**, **SecurityAdmin**, **db_owner** y **db_accessadmin** utilizan el mismo nombre de usuario de la base de datos en el inicio de sesión de SQL Server y en la cuenta de Windows NT; sin embargo, no es obligatorio.

Obtener identificadores o cuentas de inicio de sesión

Cuando se conecte a SQL Server, utilice:

- SUSUARIO_SNAME para obtener el nombre de inicio de sesión de SQL Server o cuenta de Windows NT asociados con un *identificadorDeSeguridad*.
- SUSUARIO_SID para obtener el *identificadorDeSeguridad* asociado con un nombre de inicio de sesión de SQL Server o cuenta de Windows NT.
- SUSUARIO_SID() (SUSUARIO_SID especificado sin el parámetro *cuentaInicioSesión*) para obtener el *identificadorDeSeguridad* de la conexión actual, independientemente de si se utiliza la Autenticación de SQL Server o la Autenticación de Windows NT.
- La función SYSTEM_USUARIO de SQL-92 para obtener la cuenta de Windows NT de una conexión de Autenticación de Windows NT o el nombre de inicio de sesión de SQL Server de una conexión de Autenticación de SQL Server. En Transact-SQL, SYSTEM_USUARIO se implementa como sinónimo de SUSUARIO_SNAME() (SUSUARIO_SNAME especificado sin el parámetro *identificadorDeSeguridad*).

En SQL Server, las funciones que devuelven nombres de inicio de sesión o cuentas funcionan de esta forma:

- **SUSUARIO_SNAME(*identificadorDeSeguridad*)**
SUSUARIO_SNAME puede tomar:
 - El *identificadorDeSeguridad* de una cuenta o grupo de Windows NT, en cuyo caso devuelve el nombre de la cuenta o grupo de Windows NT.
 - El falso *identificadorDeSeguridad* generado para un inicio de sesión de SQL Server, en cuyo caso devuelve el nombre de inicio de sesión de SQL Server.

Si no se especifica el *identificadorDeSeguridad* de una conexión realizada con la Autenticación de Windows NT, SUSUARIO_SNAME devuelve el nombre de la cuenta de Windows NT asociada con la conexión. Si se realizó la conexión con la Autenticación de SQL Server, SUSUARIO_SNAME devuelve el inicio de sesión de SQL Server asociado con la conexión.

Si se especificó un *IdUsuarioSistema* en lugar de un *identificadorDeSeguridad*, SUSUARIO_SNAME devuelve NULL.

- **SUSUARIO_NAME(*IdUsuarioSistema*)**
Esta función se admite para mantener la compatibilidad con versiones anteriores. Utilice SUSUARIO_SNAME cuando se conecte a SQL Server SUSUARIO_NAME puede tomar:
 - El IdUsuarioSistema generado para una cuenta o grupo de Windows NT, en cuyo caso devuelve el nombre de la cuenta o grupo de Windows NT.
 - El IdUsuarioSistema generado para un inicio de sesión de SQL Server, en cuyo caso devuelve el nombre de inicio de sesión de SQL Server.

Si no se especificó *IdUsuarioSistema* en una conexión realizada con Autenticación de Windows NT, SUSUARIO_NAME devuelve NULL. Si se realizó la conexión con Autenticación de SQL Server, SUSUARIO_NAME devuelve el inicio de sesión de SQL Server asociado con la conexión.

Si se especificó *identificadorDeSeguridad* en lugar de *IdUsuarioSistema*, SUSUARIO_NAME devuelve NULL.

- **SUSUARIO_SID('cuentaInicioSesión')**

Esta función se admite para mantener la compatibilidad con versiones anteriores. Utilice SUSUARIO_SID cuando se conecte a SQL Server *cuentaInicioSesión* puede ser:

- Un nombre de cuenta o grupo de Windows NT, en cuyo caso SUSUARIO_SID devuelve el *identificadorDeSeguridad* de Windows NT de la cuenta o grupo de Windows NT.
- Un nombre de inicio de sesión de SQL Server, en cuyo caso SUSUARIO_SID devuelve el falso *identificadorDeSeguridad* generado para ese nombre de inicio de sesión de SQL Server.

Si no se especifica *cuentaInicioSesión* en una conexión realizada con la Autenticación de Windows NT, SUSUARIO_SID devuelve el *identificadorDeSeguridad* de Windows NT asociado con la conexión. Si se realizó la conexión con la Autenticación de SQL Server, SUSUARIO_SNAME devuelve el falso *identificadorDeSeguridad* asociado con la conexión.

- **SYSTEM_USUARIO**

Esta función de SQL-92 se implementa como sinónimo de SUSUARIO_SNAME() (SUSUARIO_SNAME especificado sin el parámetro *identificadorDeSeguridad*) en Transact-SQL.

Obtener nombres de usuario de base de datos o identificadores de usuario

Cuando se conecte a SQL Server utilice:

- USUARIO_ID para obtener el identificador de usuario de base de datos asociado con un nombre de usuario de base de datos.
- USUARIO_ID() para obtener el identificador de usuario de base de datos asociado con la conexión actual.
- USUARIO_NAME para obtener el nombre de usuario de base de datos asociado con un identificador de usuario de base de datos.
- Una de las funciones de SQL-92, CURRENT_USUARIO o SESSION_USUARIO, para obtener el nombre de usuario de base de datos asociado con la conexión actual. En Transact-SQL, estas funciones se implementan como sinónimo de USUARIO_NAME() (USUARIO_NAME especificado sin el parámetro *IdUsuarioBaseDatos*). La función USUARIO de Transact-SQL también se implementa como sinónimo de USUARIO_NAME().

SQL-92 tiene en cuenta las instrucciones SQL que se codifican en módulos de SQL y pueden tener identificadores de autorización diferentes del identificador de autorización del usuario que se ha conectado a una base de datos de SQL. SQL-92

especifica que SESSION_USUARIO siempre devuelva el identificador de autorización del usuario que realizó la conexión. CURRENT_USUARIO devuelve el identificador de autorización del módulo de SQL para las instrucciones ejecutadas desde un módulo de SQL o del usuario que realizó la conexión si las instrucciones SQL no se ejecutaron desde un módulo de SQL.

Si el módulo de SQL no tiene un identificador de autorización independiente, SQL-92 especifica que CURRENT_USUARIO devuelva el mismo valor que SESSION_USUARIO. Microsoft SQL Server no tiene identificadores de autorización independientes para módulos de SQL; por lo tanto, CURRENT_USUARIO y SESSION_USUARIO siempre son el mismo. La función USUARIO está definida por SQL-92 como función para mantener la compatibilidad con versiones anteriores para aplicaciones escritas en versiones anteriores del estándar. Se especifica que devuelva el mismo valor que CURRENT_USUARIO.

En SQL Server, la función que devuelve nombres de inicio de sesión o cuentas funciona de esta forma:

- **USUARIO_ID('nombreUsuarioBaseDeDatos')**
USUARIO_ID devuelve el identificador de usuario de base de datos asociado con el nombre de usuario de base de datos especificado. Si no se especifica *nombreUsuarioBaseDeDatos*, USUARIO_ID devuelve el identificador de usuario de base de datos asociado con la conexión actual.
- **USUARIO_NAME(IdUsuarioBaseDeDatos)**
USUARIO_NAME devuelve el nombre de usuario de base de datos asociado con el identificador de usuario especificado. Si no se especifica *IdUsuarioBaseDeDatos*, USUARIO_NAME devuelve el nombre de usuario de base de datos asociado con la conexión actual.
- **CURRENT_USUARIO, SESSION_USUARIO, USUARIO**
Estas funciones son sinónimos de USUARIO_NAME() (USUARIO NAME especificado sin el parámetro *IdUsuarioBaseDeDatos*).

Funciones de conversión

Use las funciones de conversión, CAST y CONVERT, para convertir expresiones de un tipo de datos a otro, cuando Microsoft® SQL Server™ no realice automáticamente estas conversiones. Estas funciones de conversión se usan también para obtener varios formatos especiales de datos. Cualquiera de las funciones de conversión se puede utilizar en la lista de selección, en la cláusula WHERE y en cualquier lugar que se permita una expresión.

Use CAST en lugar de CONVERT si desea que el código del programa Transact-SQL cumpla con SQL-92. Use CONVERT en lugar de CAST para aprovechar la funcionalidad del estilo de CONVERT.

Cuando se usa CAST o CONVERT, se necesitan dos clases de información:

- La expresión que se va a convertir (por ejemplo, un informe de ventas necesitaría que los datos de ventas se convirtieran desde datos de moneda a datos de caracteres).
- El tipo de datos al que se va a convertir la expresión dada, por ejemplo, **varchar** o cualquier otro tipo de datos suministrado por SQL Server.

A menos que guarde el valor convertido, una conversión sólo será válida mientras dure la función CAST o CONVERT.

En este ejemplo se usa CAST en la primera instrucción SELECT y CONVERT en la segunda instrucción SELECT para convertir la columna **title** a una columna **char(50)** con el fin de hacer que los resultados sean más fáciles de leer.

```
USE pubs
SELECT CAST(title AS char(50), ytd_sales
FROM titles
WHERE type = 'trad_coAceptar'
      - O bien -
USE pubs
SELECT CONVERT(char(50), title), ytd_sales
FROM titles
WHERE type = 'trad_coAceptar'

Este es el conjunto de resultados de las consultas:
ytd_sales
-----
Onions, Leeks, and Garlic: CoAcceptaring Secrets of the 375
Fifty Years in Buckingham Palace Kitchens 15096
Sushi, Anyone? 4095
(3 row(s) affected)
```

En el siguiente ejemplo, la columna **ytd_sales**, una columna **int**, se convierte a una columna **char(20)** de forma que se pueda usar con el predicado LIKE.

```
USE pubs
SELECT title, ytd_sales
FROM titles
WHERE CAST(ytd_sales AS char(20)) LIKE '15%'
AND type = 'trad_coAceptar'
```

Este es el conjunto de resultados:

title	ytd_sales
Fifty Years in Buckingham Palace Kitchens	15096

(1 row(s) affected)

SQL Server controla automáticamente ciertas conversiones de tipos de datos. Por ejemplo, si compara una expresión **char** y una **datetime**, una **smallint** y una **int**, o expresiones **char** de distintas longitudes, SQL Server las convierte automáticamente. Esto se denomina conversión implícita. No es necesario que use la función CAST para realizar estas conversiones. Sin embargo, se puede usar CAST:

- Cuando dos expresiones son exactamente del mismo tipo de datos.

- Cuando dos expresiones son convertibles implícitamente.
- Cuando es necesario convertir explícitamente los tipos de datos.

Si intenta realizar una conversión que no es posible (por ejemplo, convertir una expresión **char** que incluya letras a **int**), SQL Server muestra un mensaje de error. Si no especifica ninguna longitud cuando se convierte al tipo de datos, SQL Server suministra automáticamente la longitud 30.

Al convertir a **datetime** o **smalldatetime**, SQL Server rechaza todos los valores que no reconoce como fechas (incluidas las fechas anteriores al 1 de enero de 1753). Pueden convertir valores **datetime** a **smalldatetime** cuando la fecha se encuentre en el intervalo adecuado (1 de enero de 1900 al 6 de junio del 2079). El valor de la hora se redondea al minuto más cercano.

Convertir a **bit** cambia cualquier valor distinto de cero a 1.

Cuando convierta a **money** o **smallmoney**, se supone que los enteros son unidades de moneda. Por ejemplo, el valor entero 4 se convierte al equivalente de moneda de 4 dólares (para **us_english**, el idioma predeterminado). Los números situados a la derecha del separador decimal de los valores de punto flotante se redondean a cuatro lugares decimales para los valores **money**. Las expresiones de los tipos de datos **char** o **varchar** que se están convirtiendo a un tipo de datos entero deben constar sólo de dígitos y un signo opcional más o menos (+ o -). Los blancos situados a la izquierda se pasan por alto. Las expresiones de los tipos de datos **char** o **varchar** que se convierten a **money** pueden incluir también un separador decimal opcional y un signo de dólar a la izquierda (\$).

Las expresiones de tipos de datos **char** o **varchar** que se están convirtiendo a **float** o **real** pueden incluir también notación exponencial opcional (e o E, seguido de un signo opcional + o - y, a continuación, un número).

Cuando se convierten expresiones de caracteres a un tipo de datos de un tamaño distinto, los valores demasiado grandes para el nuevo tipo de datos se truncan, y SQL Server muestra un asterisco (*) tanto en la herramienta **osql** como en el Analizador de consultas de SQL Server. Cuando las expresiones numéricas son demasiado grandes para que se presente el nuevo tipo de datos, se truncan los valores. A continuación se muestra un ejemplo del truncamiento de caracteres:

```
USE pubs
SELECT SUBSTRING(title, 1, 25) AS Title, CONVERT(char(2), ytd_sales)
FROM titles
WHERE type = 'trad_coAceptar'
Éste es el conjunto de resultados:
Title
-----
Onions, Leeks, and Garlic *
Fifty Years in Buckingham *
Sushi, Anyone? *
(3 row(s) affected)
```

Al convertir tipos de datos en los que el tipo de datos de destino tiene menos separadores decimales que el tipo de datos de origen, se trunca el valor. Por ejemplo, el resultado de CAST(10,3496 AS money) es \$10,35.

Puede convertir explícitamente datos **text** a **char** o **varchar**, y datos **image** a **binary** o **varbinary**. Puesto que estos tipos de datos están limitados a 8.000 caracteres, estará limitado a la longitud máxima de los tipos de datos **character** y **binary**, es decir, 8.000 caracteres. Puede convertir explícitamente datos **ntext** a **nchar** o **nvchar**, pero la longitud máxima es de 4.000 caracteres. Si no especifica la longitud, el valor convertido tiene una longitud predeterminada de 30 caracteres. No se admite la conversión implícita.

El parámetro *estilo*

El parámetro *estilo* de CONVERT proporciona una amplia variedad de formatos de presentación cuando se convierten datos **datetime** a **char** o **varchar**. El número que se suministra como el parámetro *estilo* determina la forma en la que se muestran los datos **datetime**. El año se puede mostrar con dos o con cuatro dígitos. De forma predeterminada, SQL Server suministra un año de cuatro dígitos. Para mostrar un año de cuatro dígitos que incluya el siglo (aaaa), incluso si el año se guardó con un formato de dos dígitos, agregue 100 al valor de *estilo* para obtener un año de cuatro cifras.

En este ejemplo se muestra CONVERT con el parámetro *estilo*.

```
SELECT CONVERT(char(12), GETDATE(), 3)
```

Esta instrucción convierte la fecha actual al estilo 3, dd/mm/aa.

Expresiones

Una expresión es una combinación de identificadores, valores y operadores que Microsoft® SQL Server™ puede evaluar para obtener un resultado. Los datos se pueden usar en varios sitios distintos cuando se cambian o se tiene acceso a los datos. Las expresiones se pueden usar, por ejemplo, como parte de los datos que se van a recuperar (en una consulta) o como una condición para buscar los datos que cumplen un conjunto de criterios.

Una expresión puede ser una:

- Constante.
- Función.
- Nombre de columna.
- Variable.
- Subconsulta.
- CASE, NULLIF o COALESCE.

Una expresión también puede generarse a partir de la combinación de estas entidades y operadores.

En la siguiente instrucción SELECT, por cada fila del conjunto de resultados, SQL Server puede resolver **LastNome** como un valor único, con lo que constituye una expresión.

```
SELECT LastName  
FROM Northwind..Employees
```

Una expresión puede ser también un cálculo, como, por ejemplo (**price** * 1.5) o (**price** + **sales_tax**).

En una expresión, incluya los valores de caracteres de fecha entre comillas simples. En la siguiente instrucción SELECT, el literal de carácter B% usado como patrón para la cláusula LIKE debe estar entre comillas simples:

```
SELECT LastName, FirstName  
FROM Northwind..Employees  
WHERE LastName LIKE 'B%'
```

En la siguiente instrucción SELECT, el valor de fecha se incluye entre comillas:

```
SELECT *  
FROM Northwind..Orders  
WHERE OrderDate = 'Sep 13 1996'
```

En este ejemplo, se usa más de una expresión en la consulta. Por ejemplo, **col1**, **SUBSTRING**, **col3**, **price** y 1.5 son expresiones.

```
SELECT col1, SUBSTRING('This is a long string', 1, 5), col3, price *  
1.5  
FROM mytable
```

Utilizar operadores en expresiones

Los operadores permiten realizar operaciones aritméticas, comparaciones, concatenaciones o asignaciones de valores. Por ejemplo, puede probar si la columna **country** con datos de clientes está rellena (o no es NULL).

En las consultas, cualquier persona que pueda ver los datos de la tabla que necesitan ser usados con algún tipo de operador puede realizar operaciones. Antes de poder cambiar los datos correctamente, necesita los permisos adecuados.

Los operadores se usan en Microsoft® SQL Server™ para:

- Cambiar datos, permanente o temporalmente.
- Buscar filas o columnas que cumplan una condición determinada.
- Implementar una decisión entre columnas de datos o entre expresiones.
- Probar determinadas condiciones antes de iniciar o confirmar una transacción, o antes de ejecutar determinadas líneas de código.

SQL Server tiene siete categorías de operadores.

Para realizar este tipo de operación	Use esta categoría de operador
Comparar un valor con otro valor o una expresión.	Operadores de comparación

Probar si una condición es cierta, como AND, OR, NOT, LIKE, ANY, ALL o IN.	Lógicos
Suma, resta, multiplicación, división, módulo.	Operadores aritméticos
Realizar una operación en un operando, como positivo o negativo, o el complementario.	Unario.
Convertir temporalmente valores numéricos normales (como 150) a enteros y realizar aritmética de bits (0 y 1).	Operadores binarios
Combinar permanentemente dos cadenas (de caracteres o de datos binarios) en una cadena.	Operador de concatenación de cadenas
Asignar un valor a una variable, o asociar una columna de un conjunto de resultados con un alias.	Asignación

Una expresión se puede generar a partir de varias expresiones más pequeñas combinadas mediante operadores. En estas expresiones complejas, los operadores se evalúan en un orden que se basa en la definición de SQL Server de la preferencia de operadores. Los operadores con mayor preferencia se ejecutan antes que los operadores con menor preferencia.

Operadores aritméticos

Los operadores aritméticos se pueden usar para realizar cualquier cálculo aritmético, como:

- Suma.
- Resta.
- Multiplicación.
- División.
- Módulo (el resto de una operación de división)

A continuación se proporciona información acerca de los operadores aritméticos:

- Cuando hay más de un operador aritmético en una expresión, primero se calculan las multiplicaciones, divisiones y módulos, y, después, las restas y las sumas.
- Cuando todos los operadores aritméticos de una expresión tienen el mismo nivel de preferencia, el orden de ejecución es de izquierda a derecha.
- Las expresiones entre paréntesis tienen preferencia sobre el resto de las operaciones.

La siguiente instrucción SELECT resta la parte de las ventas del año que recibe el autor (ventas * porcentaje de derechos de autor / 100) del total de ventas. El resultado es la cantidad de dinero que recibe el editor. El producto de **ytd_sales** y **royalty** se calcula

primero porque el operador es una multiplicación. A continuación, el total se divide por 100. Este resultado se resta de **ytd_sales**.

```
USE pubs
SELECT title_id, ytd_sales - ytd_sales * royalty / 100
FROM titles
```

Para conseguir una mayor claridad, puede usar paréntesis:

```
USE pubs
SELECT title_id, ytd_sales - ((ytd_sales * royalty) / 100)
FROM titles
```

También puede usar paréntesis para cambiar el orden de ejecución. Los cálculos entre paréntesis se evalúan primero. Si los paréntesis están anidados, el cálculo con una anidación más profunda es el que tiene preferencia. Por ejemplo, el resultado y significado de la consulta anterior se pueden cambiar si usa paréntesis para obligar a que la resta se evalúe antes que la multiplicación:

```
USE pubs
SELECT title_id, (ytd_sales - ytd_sales) * royalty / 100
FROM titles
```

Operadores binarios

Los operadores binarios se usan en los datos **int**, **smallint** o **tinyint**. El operador **~** (NOT binario) puede usar también datos **bit**. Todos los operadores binarios realizan una operación en uno o más valores enteros especificados en expresiones binarias de las instrucciones de Transact-SQL. Por ejemplo, el operador **~** (NOT binario) cambia los 1 binarios a 0, y los 0 a 1. Para comprobar las operaciones binarias, puede convertir o calcular los valores decimales.

Por ejemplo, suponga que desea sumar 150 y 75, pero no sólo desea el valor decimal de 225, sino que desea usar aritmética binaria (suma de 0 y 1). Para este fin, use el operador binario AND (&).

Si está almacenando datos de enteros (valores decimales normales, como el 150 y 75 mencionado anteriormente) y desea realizar una conversión interna para realizar cálculos binarios, use los operadores binarios. Los operadores binarios también son útiles para obtener un valor NOT que no sea necesariamente el opuesto exacto.

Operadores de comparación

Los operadores de comparación se usan con los datos de caracteres, numéricos o de fecha, y se pueden utilizar en las cláusulas WHERE o HAVING de una consulta. Los operadores de comparación dan como resultado un tipo de datos Boolean: devuelven TRUE o FALSE según el resultado de la condición probada.

Por ejemplo, para calcular una bonificación para aquellos empleados que han sido contratados antes del 15 de marzo de 1998, el cálculo de si la **hire_date** (fecha de contratación) de un empleado es menor o igual al 15 de marzo de 1998 proporcionará una lista de los empleados que deben recibir la bonificación.

Los operadores de comparación válidos son:

- > (mayor que).
- < (menor que).
- = (igual).
- <= (menor o igual que).
- >= (mayor o igual que).
- != (distinto de).
- != (distinto de).
- !< (no menor que).
- !> (no mayor que).

Los operadores de comparación se pueden usar también en la lógica del programa para comprobar una condición. Por ejemplo, si la columna del país es RU en lugar de España, puede que se apliquen distintas tarifas de envío. En este caso, se usan juntos una combinación de un operador de comparación, una expresión (el nombre de columna), un literal ('RU') y una palabra clave de programación de control de flujo (IF), para conseguir este propósito.

Cualquier persona que tenga acceso a los datos reales (para consultas) puede usar los operadores de comparación en consultas adicionales. En las instrucciones de modificación de datos, se recomienda que sólo use los operadores de comparación si sabe que dispone de los permisos adecuados y que los datos serán cambiados sólo por un pequeño grupo de personas (para mantener la integridad de los datos).

Las consultas usan también comparaciones de cadena para comparar el valor de una variable local, cursor o columna con una constante. Por ejemplo, para imprimir todas las filas de cliente si el país es el Reino Unido. En la tabla se muestran ejemplos de comparación de cadenas entre datos Unicode y no Unicode; ST1 es **char** y ST2 es **nchar**.

Comparación	Descripción
ST1 = ST2	Equivalente a CONVERT(nchar , ST1) = ST2 o CAST(ST1 como nchar) = ST2.
ST1 = 'cadena no Unicode'	Comparación normal de cadena de SQL-92.
ST2 = 'cadena no Unicode'	Equivalente a ST2 = CONVERT(nchar , 'cadena no Unicode') o ST2 = CAST('cadena no Unicode' AS nchar). Comparación Unicode.
ST2 = N'cadena Unicode'	
CONVERT(nchar , ST1) = ST2	
o	Comparación Unicode.
CAST(ST1 AS nchar) = ST2	
ST1 = CONVERT(char , ST2)	
o	
ST1 = CAST(ST2 AS char)	
N" (cadena vacía Unicode entre paréntesis) " (cadena vacía no Unicode)	Comparación normal de cadena de SQL-92.
	Cadena vacía.
	Una cadena vacía o una cadena que contiene un carácter blanco (dependiendo

de la configuración de SQL-92).

Operador de concatenación de cadenas

El operador de concatenación de cadenas es el signo más (+). Puede combinar, o concatenar, dos o más cadenas de caracteres en una única cadena. También puede concatenar cadenas binarias. A continuación se muestra un ejemplo de concatenación:

```
SELECT ('abc' + 'def')
```

Éste es el conjunto de resultados:

```
-----
abcdef
(1 row(s) affected)
```

Esta consulta muestra los nombres de los autores con direcciones de California bajo la columna **Moniker**, en el orden apellido, nombre, con una coma y un espacio detrás del apellido.

```
USE Northwind
GO
SELECT LastName + ', ' + FirstName AS Moniker
FROM Employees
WHERE Region = 'WA'
```

Éste es el conjunto de resultados:

```
Moniker
-----
Davolio, Nancy
Fuller, Andrew
Leverling, Janet
Peacock, Margaret
Callahan, Laura
(15 row(s) affected)
```

Otros tipos de datos, como **datetime** y **smalldatetime**, deben convertirse a cadenas de caracteres con la función de conversión CAST antes de que se puedan concatenar con una cadena.

```
USE pubs
SELECT 'The due date is ' + CAST(pubdate AS varchar(128))
FROM titles
WHERE title_id = 'BU1032'
```

Éste es el conjunto de resultados:

```
-----
The due date is Jun 12 1991 12:00AM
(1 row(s) affected)
```

La cadena vacía ("") se evalúa como un espacio individual:

```
SELECT 'abc' + '' + 'def'
```

Éste es el conjunto de resultados:

```
-----
abcdef
(1 row(s) affected)
```

Nota Que una cadena vacía ("") se interprete como un carácter blanco individual o como un carácter vacío depende de la configuración del nivel de compatibilidad de **sp_dbcmptlevel**. Para este ejemplo, si **sp_dbcmptlevel** es 65, los literales vacíos se tratan como un blanco individual.

Valores NULL

NULL indica que el valor es desconocido. Es distinto de un valor vacío o cero. Dos valores NULL no son iguales. La comparación entre dos valores NULL, o entre un valor NULL y cualquier otro valor, tiene un resultado desconocido porque el valor de cada NULL es desconocido.

Los valores NULL indican normalmente que el dato es desconocido, no es aplicable o que se agregará posteriormente. Por ejemplo, la inicial de un cliente puede que no sea conocida en el momento en que éste hace un pedido.

A continuación se muestra información acerca de los valores NULL:

- Para comprobar valores NULL en una consulta, use IS NULL o IS NOT NULL en la cláusula WHERE.
- Cuando se visualizan los resultados en el Analizador de consultas de SQL Server, los valores NULL se muestran como (**null**) en el conjunto de resultados.
- Los valores NULL se pueden insertar en una columna si se indica explícitamente NULL en una instrucción INSERT o UPDATE, se deja fuera una columna de una instrucción INSERT, o cuando se agrega una columna nueva a una tabla existente con la instrucción ALTER TABLE.
- Los valores NULL no se pueden usar en la información necesaria para distinguir una fila de una tabla de una fila de otra tabla (por ejemplo, en claves principales o externas).

En el código del programa, puede comprobar la existencia de valores NULL de forma que determinados cálculos sólo se realicen en filas con datos válidos (o no NULL). Por ejemplo, un informe puede imprimir la columna de seguridad social sólo si en la columna hay datos que no son NULL. La eliminación de los valores NULL cuando realice cálculos puede ser importante porque algunos (como, por ejemplo, un promedio) pueden ser incorrectos si se incluyen columnas NULL.

Es posible que haya valores NULL en los datos, por lo que es una buena práctica crear consultas e instrucciones de modificación de datos que eliminen los valores NULL o los transformen en algún otro valor (si no desea que aparezcan en los datos).

Importante Para minimizar las tareas de mantenimiento y los posibles efectos en las consultas o informes existentes, se recomienda que disminuya al mínimo el uso de los valores NULL. Planee sus consultas e instrucciones de modificación de datos de forma que los valores NULL tengan un efecto mínimo.

Cuando hay valores NULL en los datos, los operadores lógicos y de comparación pueden devolver un tercer resultado desconocido (UNKNOWN) en lugar de simplemente TRUE (verdadero) o FALSE (falso). Esta necesidad de una lógica de tres valores es el origen de muchos errores de la aplicación. Estas tablas destacan el efecto de escribir comparaciones con NULL.

AND con valor	TRUE	UNKNOWN	FALSE
TRUE	TRUE	UNKNOWN	FALSE
UNKNOWN	UNKNOWN	UNKNOWN	FALSE
FALSE	FALSE	FALSE	FALSE

OR con valor	TRUE	UNKNOWN	FALSE
TRUE	TRUE	TRUE	TRUE
UNKNOWN	TRUE	UNKNOWN	UNKNOWN
FALSE	TRUE	UNKNOWN	FALSE

NOT	Evalúa a
TRUE	FALSE
UNKNOWN	UNKNOWN
FALSE	TRUE

El estándar SQL-92 escribe las palabras clave IS NULL e IS NOT NULL para comprobar la presencia de valores NULL.

IS NULL	Evalúa a	IS NOT NULL	Evalúa a
TRUE	FALSE	TRUE	TRUE
NULL	TRUE	NULL	FALSE
FALSE	FALSE	FALSE	TRUE

Transact-SQL ofrece también una ampliación para el procesamiento de los valores NULL. Si la opción ANSI_NULLS está desactivada, las comparaciones entre los valores NULL, como NULL = NULL, da como resultado TRUE. Las comparaciones entre NULL y cualquier valor de datos da como resultado FALSE.

Miscelaneo

Utilizar comentarios

Los comentarios son cadenas de texto que no se ejecutan incluidas en el código de un programa; también se conocen como observaciones. Los comentarios se pueden usar para documentar código o partes deshabilitadas temporalmente de instrucciones y lotes de Transact-SQL que se están diagnosticando.

La utilización de comentarios hace más fácil el mantenimiento futuro del código del programa. Los comentarios se usan, a menudo, para guardar el nombre de un programa, el nombre del autor y las fechas de los cambios importantes del código. Los

comentarios se pueden usar para describir cálculos complicados o para explicar determinado método de programación.

Microsoft® SQL Server™ admite dos tipos de caracteres para indicar comentarios:

- (doble guión). Estos caracteres para los comentarios se pueden usar en la misma línea que el código que se va a ejecutar o en una línea aparte. Todo lo que se encuentre entre los dos guiones y el final de la línea es parte del comentario. En el caso de que un comentario ocupe varias líneas, los guiones dobles deben aparecer al principio de cada línea de comentarios.
- /* ... */ (par de caracteres barra diagonal y asterisco). Estos caracteres para los comentarios se pueden usar en la misma línea que el código que se va a ejecutar, en líneas separadas o, incluso, en el código ejecutable. Todo lo incluido entre el par de apertura de comentario /*) y el par de cierre de comentario (*/) se considera parte del comentario.

En un comentario de varias líneas, el par de caracteres de apertura de comentario /* debe iniciar el comentario, y el par de caracteres de cierre de comentario */) debe finalizarlo. Ningún otro carácter de comentario debe aparecer en ninguna línea del comentario.

Los comentarios /* */ de varias líneas no pueden dividirse en varios lotes. El comentario completo debe estar contenido en un único lote. Por ejemplo, en el Analizador de consultas de SQL Server y el programa osql, el comando GO indica el final de un lote. Cuando las herramientas leen los caracteres GO en los dos primeros bytes de una línea, envían todo el código desde el último GO al servidor como un lote. Si hay un GO al inicio de una línea entre los delimitadores /* y */, cualquier delimitador de comentario sin correspondencia será enviado con cada lote que, a su vez, desencadenará errores de sintaxis. Por ejemplo, la siguiente secuencia de comandos contiene errores de sintaxis:

```
USE Northwind
GO
SELECT * FROM Employees
/* The
GO in this comment causes it to be brAceptaren in half */
SELECT * FROM Products
GO
```

Éstos son algunos comentarios válidos:

```
USE Northwind
GO
-- First line of a multiple-line comment.
-- Second line of a multiple-line comment.
SELECT * FROM Employees
GO
/* First line of a multiple-line comment.
Second line of a multipl-line comment. */
SELECT * FROM Products
GO
-- Using a comment in a Transact-SQL statement
-- during diagnosis.
```

```
SELECT EmployeeID, /* FirstName, */ LastName
FROM Employees
-- Using a comment after the code on a line.
USE Northwind
GO
UPDATE Products
SET UnitPrice = UnitPrice * .9 -- Try to build market share.
GO
```

A continuación se muestra información básica acerca de los comentarios:

- En los comentarios se pueden usar todos los caracteres o símbolos alfanuméricos. SQL Server pasa por alto todos los caracteres de un comentario, aunque el Analizador de consultas de SQL Server, **osql** e **isql** buscarán GO como los primeros dos caracteres de las líneas de un comentario que ocupa varias.
- No hay longitud máxima para un comentario dentro de un lote. Un comentario se puede componer de una o más líneas.

Utilizar palabras clave reservadas

Microsoft® SQL Server™ reserva ciertas palabras clave para su uso exclusivo. Por ejemplo, la utilización de la palabra clave DUMP o BACKUP de Transact-SQL en una sesión del Analizador de consultas de SQL Server u **osql**, indica a SQL Server que haga una copia de seguridad de toda o de parte de una base de datos, o una copia de seguridad del registro.

No se permite incluir las palabras clave reservadas de una instrucción de Transact-SQL en ninguna parte que no sea donde define SQL Server. Ningún objeto de la base de datos debe recibir un nombre que coincida con una palabra clave reservada. Si existe tal nombre, siempre se debe hacer referencia al objeto usando identificadores delimitados. Aunque este método permite la existencia de objetos cuyos nombres son palabras reservadas, se recomienda que no ponga a los objetos de la base de datos ningún nombre que coincida con el de una palabra reservada.

Entre las funciones de los administradores de la base de datos y del sistema o del creador de la base de datos se encuentra, precisamente, la de comprobar la existencia de palabras clave reservadas en el código de Transact-SQL y en los nombres de las bases de datos.

Use una convención de nombres que evite la utilización de palabras clave reservadas. Si el nombre de un objeto se parece a una palabra reservada clave, se pueden quitar del mismo las consonantes o vocales; por ejemplo, un procedimiento que realiza instrucciones BACKUP para todas las bases de datos definidas por el usuario se puede llamar **bckup**.

Sinónimos

Los tipos de datos sinónimos se incluyen por compatibilidad con SQL-92.

Sinónimo	Tipo de datos asignado al sistema
binary varying	varbinary
char varying	varchar
Character	char
Character	char(1)
character(<i>n</i>)	char(<i>n</i>)
character varying(<i>n</i>)	varchar(<i>nn</i>)
Dec	decimal
double precision	float
float[<i>(n)</i>] para <i>n</i> = 1-7	real
float[<i>(n)</i>] para <i>n</i> = 8-15	float
Integer	int
national character(<i>n</i>)	nchar(<i>n</i>)
national char(<i>n</i>)	nchar(<i>n</i>)
national character varying(<i>n</i>)	nvarchar(<i>n</i>)
national char varying(<i>n</i>)	nvarchar(<i>n</i>)
national text	ntext
numeric	decimal