

CAPITULO 1

- Concepto de base de datos relacional.
- Sistemas administradores de bases de datos.
- ¿Cómo guarda los datos SQL Server 2014?
- Creación de bases de datos.
- Componentes lógicos: registro de datos y registro de transacciones.
- Componentes físicos: archivos y grupos de archivos de bases de datos.
- Creación de Esquemas
- Creación de Snapshots

CONCEPTO DE BASE DE DATOS RELACIONAL.

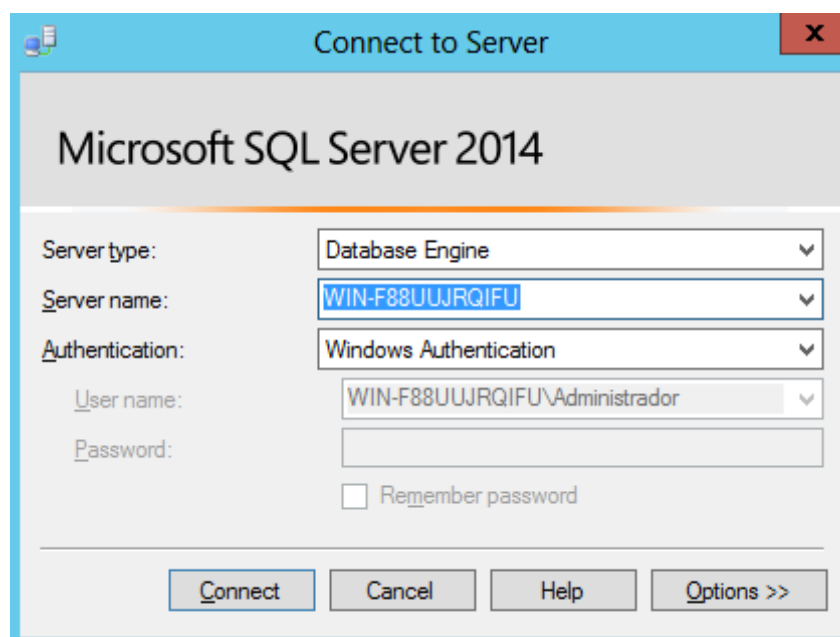
Una base de datos se puede definir como un conjunto de información que pertenece al mismo contexto, que se encuentra agrupada ó almacenada para su uso posterior. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta.

Sistemas administradores de bases de datos.

Los Sistemas de gestión de base de datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta.

En los textos que tratan este tema, o temas relacionados, se mencionan los términos SGBD y DBMS, siendo ambos equivalentes, y acrónimos, respectivamente, de Sistema Gestor de Bases de Datos y DataBase Management System.



¿Cómo guarda los datos SQL Server 2014?

Al crear una base de datos, es importante entender cómo SQL Server 2014 almacena los datos para que pueda calcular y especificar la cantidad de espacio en disco que debe asignar a los archivos de datos y registros de transacciones.

Considere los datos e instrucciones siguientes sobre el almacenamiento de datos:

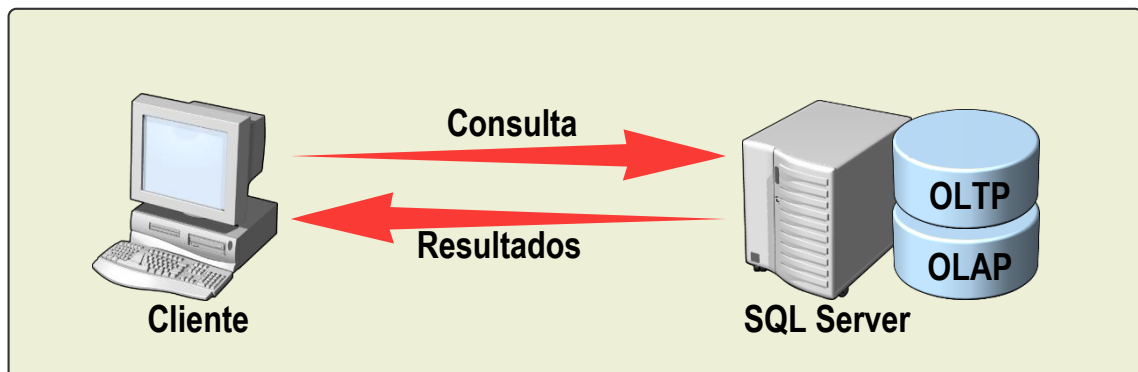
- Todas las bases de datos tienen un archivo de datos principal (.mdf) y uno o más archivos de registro de transacciones (.ldf). Una base de datos también puede tener archivos secundarios de datos (.ndf). Estos archivos físicos tienen los nombres de los archivos del sistema operativo y los nombres de los archivos lógicos que se pueden usar en las instrucciones Transact-SQL. La ubicación predeterminada para todos los archivos de datos y los registros de transacciones es:

C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA.

- Al crear una base de datos, una copia de la base de datos model, donde se incluyen las tablas del sistema, se copia en la base de datos y el resto de la base de datos se rellena con páginas vacías.
- Los datos se almacenan en bloques de 8 kilobytes (KB) de espacio en disco contiguo llamado páginas. Esto significa que una base de datos puede almacenar 128 páginas por megabyte (MB).

Creación de bases de datos.

Puede utilizar SQL Server para realizar procesamiento de transacciones, almacenar y analizar datos, y generar nuevas aplicaciones de base de datos.



SQL Server es una familia de productos y tecnologías que cumple los requisitos de almacenamiento de datos de los entornos de procesamiento de transacciones en línea (OLTP) y procesamiento analítico en línea (OLAP). SQL Server es un sistema de administración de bases de datos relacionales (RDBMS) que:

- Administra el almacenamiento de datos para las transacciones y los análisis.
- Almacena datos en una amplia gama de tipos de datos, incluyendo texto, numérico, lenguaje de marcado extensible (XML) y objetos grandes.
- Responde a las solicitudes de las aplicaciones cliente.
- Utiliza Transact-SQL, XML u otros comandos de SQL Server para enviar solicitudes entre una aplicación cliente y SQL Server.

El componente RDBMS de SQL Server es responsable de lo siguiente:

- Mantener las relaciones existentes entre los datos de una base de datos.

- Garantizar que los datos se almacenan correctamente y que no se infringen las reglas que definen las relaciones entre los datos.
- Recuperar todos los datos hasta un punto de coherencia conocida, en caso de que se produzca un error del sistema.

BASES DE DATOS OLTP

Las tablas relacionales organizan los datos de una base de datos OLTP para reducir la información redundante y aumentar la velocidad de las actualizaciones. SQL Server permite que un número elevado de usuarios realicen transacciones y modifiquen simultáneamente datos en tiempo real en bases de datos OLTP.

BASES DE DATOS OLAP

La tecnología OLAP se utiliza para organizar y resumir grandes cantidades de datos de manera que un analista pueda evaluar los datos rápidamente y en tiempo real. Microsoft SQL Server Analysis Services organiza estos datos para admitir una amplia variedad de soluciones empresariales, desde informes y análisis corporativos, hasta modelado de datos y ayuda a la toma de decisiones.

Consideraciones a la hora de planear una base de datos

A la hora de planear una nueva base de datos, debe tener en cuenta varios aspectos. Ellos incluyen, pero sin limitarse a, los siguientes:

- **Propósito del almacenamiento de datos.** Las bases de datos OLTP y OLAP tienen propósitos diferentes y, por tanto, tienen distintos requisitos de diseño.
- **Rendimiento de transacciones.** Las bases de datos OLTP suelen tener un requisito alto en cuanto al número de transacciones que se pueden procesar por minuto, hora o día. Un diseño eficiente con un nivel adecuado de normalización, índices y particiones de datos puede lograr un rendimiento de transacciones muy alto.
- **Crecimiento potencial del almacenamiento físico de datos.** Unas cantidades de datos elevadas requieren un hardware adecuado en cuanto a memoria, espacio en disco duro y capacidad de la unidad central de procesamiento (CPU). Una estimación de la cantidad de datos que su base de datos almacenará durante los meses y años venideros ayudará a garantizar que la base de datos siga funcionando eficazmente. Puede configurar las bases de datos para que los archivos crezcan automáticamente hasta alcanzar un tamaño máximo especificado. Sin embargo, el crecimiento automático de los archivos puede afectar el rendimiento. En la mayoría de las soluciones de bases de datos basadas en servidor, debe crear la base de datos con unos archivos correctamente dimensionados, supervisar el uso del espacio y reasignar más espacio sólo cuando sea necesario.
- **Ubicación de los archivos.** El lugar donde coloca los archivos de base de datos puede afectar el rendimiento. Si tiene la posibilidad de utilizar varias unidades de disco, puede repartir los archivos de base de datos en más de un disco. Esto permite a SQL Server aprovechar varias conexiones y varios cabezales de disco para lograr una lectura y una escritura eficientes de los datos.

EJEMPLO DE CREACIÓN DE UNA BASE DE DATOS

Puede crear una base de datos utilizando las herramientas visuales SQL Server Management Studio o la instrucción CREATE DATABASE de Transact-SQL. En el ejemplo siguiente se muestra cómo crear una base de datos utilizando Transact-SQL.

```
CREATE DATABASE sistemasuni_DB
ON (NAME = ' sistemasuni_DB_Data',
FILENAME = 'C:\DATOS\ sistemasuni_DB.mdf',
SIZE = 20 MB,
FILEGROWTH = 0)
LOG ON (NAME = ' sistemasuni_DB_Log',
FILENAME = 'C:\DATOS\ sistemasuni_DB_Log.ldf',
SIZE = 5 MB,
FILEGROWTH = 0)
```

Componentes lógicos: registro de datos y registro de transacciones.

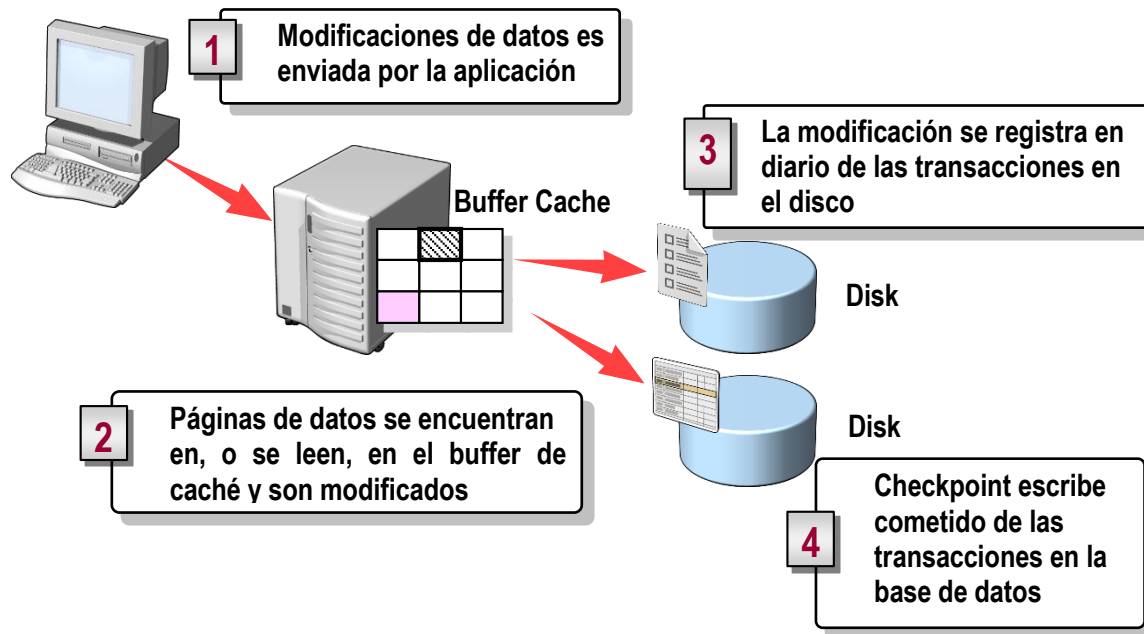
SQL Server graba todas las transacciones en un registro de transacciones para mantener la coherencia de la base de datos y facilitar la recuperación. Una transacción es una única unidad de trabajo en una base de datos. El acrónimo ACID se utiliza a menudo para describir las características de una transacción. ACID representa lo siguiente en inglés:

- **Atomicidad.** Una transacción es una unidad atómica; se completan todas las operaciones definidas en la transacción o no se completa ninguna de ellas.
- **Coherencia.** Una transacción siempre deja los datos en un estado coherente.
- **Aislamiento.** Una transacción se realiza aislada de otras actividades de base de datos; otras actividades simultáneas de base de datos no tienen ningún efecto sobre la transacción.
- **Durabilidad.** Cuando una transacción se confirma, los resultados se almacenan en un almacenamiento persistente y sobreviven a un posible error del sistema.

SQL Server admite transacciones implícitas para instrucciones individuales que modifican datos, y transacciones explícitas para varias instrucciones que deben ejecutarse como una unidad.

REGISTRO DE TRANSACCIONES

SQL Server registra todas las transacciones en un registro de transacciones de escritura anticipada para mantener la coherencia de la base de datos y ayudar en la recuperación de un posible error de la base de datos. El registro es un área de almacenamiento que realiza automáticamente un seguimiento de los cambios efectuados en una base de datos. SQL Server graba en el registro las modificaciones del disco a medida que se ejecutan las modificaciones y antes de que se escriban en la base de datos.



Proceso de registro

El proceso de registro es el siguiente:

1. La aplicación envía una modificación de datos.
2. Cuando se ejecuta una modificación, las páginas de datos afectadas se cargan desde el disco en la caché de búfer, siempre y cuando una consulta anterior no las haya puesto ya en la caché de búfer.
3. El registro graba todas las instrucciones de modificación de datos a medida que se realiza la modificación. El cambio siempre se graba en el registro y se escribe en el disco antes de realizar ese cambio en la base de datos. Este tipo de registro se denomina un registro de escritura anticipada.
4. Periódicamente, el proceso del punto de comprobación escribe en el disco todas las transacciones completadas en la base de datos.

Si se produce un error en el sistema, el proceso de recuperación automática utiliza el registro de transacciones para poner al día todas las transacciones confirmadas y revertir las transacciones incompletas.

El registro utiliza marcadores de transacción durante la recuperación automática para determinar los puntos inicial y final de una transacción. Se considera que una transacción está completa cuando el marcador **BEGIN TRANSACTION** tiene un marcador **COMMIT TRANSACTION** asociado. Las páginas de datos se escriben en el disco cuando se produce un punto de comprobación.

CONSIDERACIONES SOBRE LA UBICACIÓN DEL ARCHIVO DE REGISTRO

Para mejorar el rendimiento, se recomienda que ponga un archivo de registro de transacciones en un disco físico diferente que los archivos de datos. Esto reduce la contención y permite que un conjunto de cabezales de la unidad de disco grave las transacciones en el registro de transacciones mientras otros cabezales leen al mismo tiempo datos de los archivos de datos. La actualización de datos es rápida porque las

transacciones se pueden escribir inmediatamente en el disco sin esperar a que inalicen las lecturas de datos. Puesto que los archivos de registro se escriben secuencialmente, si el registro se almacena en un disco dedicado, los cabezales de disco permanecen en la posición correcta para la siguiente operación de escritura.

Orígenes de información de las bases de datos

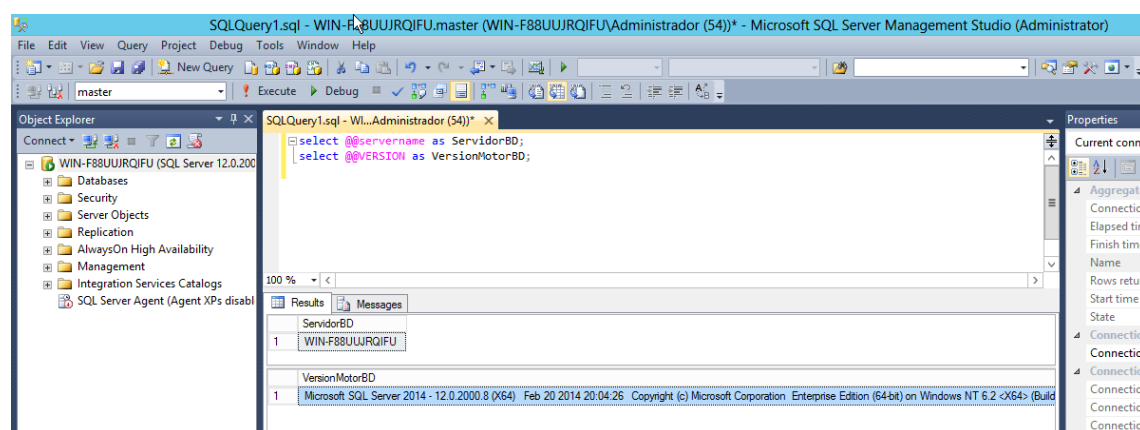
Hay dos formas distintas de ver metadatos del sistema sobre sus bases de datos. Cuando necesite ver información acerca de un objeto de base de datos, la manera más sencilla es utilizar SQL Server Management Studio. Cuando escriba aplicaciones que recuperen metadatos acerca de los objetos de base de datos, debe utilizar Transact-SQL para consultar las vistas de catálogo proporcionadas por el sistema, utilizar las funciones del sistema o ejecutar los procedimientos almacenados del sistema.

Origen de información	Descripción
SQL Server Management Studio	Herramienta visual que muestra los metadatos de la base de datos dentro del entorno de administración
Vistas de catálogo	Proporcionan metadatos acerca de objetos de base de datos que devuelven filas de información
Funciones de metadatos	Devuelven un valor único de información de metadatos por función
Procedimientos almacenados del sistema	Recuperan metadatos utilizando procedimientos almacenados

SQL SERVER MANAGEMENT STUDIO

SQL Server Management Studio proporciona herramientas visuales para mostrar metadatos de la base de datos dentro del entorno de administración. En la tabla siguiente se muestran las herramientas más utilizadas.

Herramienta de SQL Server Management Studio	Descripción
Explorador de objetos	El Explorador de objetos es una herramienta gráfica para buscar y administrar servidores, bases de datos y objetos de base de datos.
Ventana Propiedades	Dentro del Explorador de objetos, cada objeto de base de datos tiene una ventana Propiedades asociada a la que se puede tener acceso haciendo clic con el botón secundario del mouse en el objeto y haciendo clic después en Propiedades. La ventana Propiedades varía dependiendo del tipo de objeto seleccionado.
Informes	SQL Server Management Studio incluye un conjunto de informes para diversos nodos proporcionados dentro del Explorador de objetos por el motor del Servidor de informes de SQL Server. Entre los nodos utilizados con frecuencia que pueden mostrar informes se incluyen los siguientes: <ul style="list-style-type: none"> • Servidor • Base de datos • Service Broker, bajo un nodo Base de datos • Inicios de sesión, bajo el nodo Seguridad • Administración



VISTAS DE CATÁLOGO

Las vistas de catálogo le permiten consultar metadatos acerca de los objetos de base de datos de SQL Server, como tablas, procedimientos almacenados y restricciones. Algunas vistas de catálogo muestran información de todo el servidor, pero la mayoría es específica de la base de datos.

Las vistas de catálogo se muestran en la carpeta Vistas\Vistas del sistema para cada base de datos en el Explorador de objetos de SQL Server Management Studio. Aunque puede consultarlas utilizando la sintaxis estándar de Transact-SQL para las vistas definidas por el usuario, no se implementan realmente como vistas tradicionales en las tablas subyacentes, sino que consultan directamente los metadatos del sistema. Hay más de 200 vistas de catálogo y se definen en el esquema sys.

Las vistas de catálogo se organizan en categorías según su función. En la tabla siguiente se muestran algunas de las categorías clave y algunas de las vistas de catálogo más utilizadas dentro de cada categoría.

Categoría	Vista de catálogo	Descripción
Bases de datos y archivos	sys.databases	Devuelve una fila para cada base de datos del servidor
	sys.database_files	Devuelve una fila para cada archivo de una base de datos
Objetos	sys.columns	Devuelve una fila para cada columna de un objeto que contiene columnas (por ejemplo, una tabla o una vista)
	sys.events	Devuelve una fila para cada evento para el que se activa un desencadenador o una notificación
	sys.indexes	Devuelve una fila para cada índice o montón de un objeto tabular
	sys.tables	Devuelve una fila para cada tabla de la base de datos
	sys.views	Devuelve una fila para cada vista de la base de datos
Esquemas	sys.schemas	Devuelve una fila para cada esquema definido en la base de datos
Seguridad	sys.database_permissions	Devuelve una fila para cada permiso definido en la base de datos
	sys.database_principals	Devuelve una fila para cada entidad principal de seguridad de la base de datos
	sys.database_role_members	Devuelve una fila para cada miembro de cada función de base de datos

FUNCIONES DE METADATOS

SQL Server 2014 define varias categorías de funciones que devuelven información acerca de la base de datos y los objetos de base de datos. Mientras que las vistas de catálogo devuelven varias filas de información, estas funciones sólo devuelven un único valor y se conocen como funciones escalares.

En la lista siguiente se describen algunas funciones de metadatos que se utilizan con frecuencia:

- **DB_ID.** Devuelve el número de identificación (Id.) de base de datos para un nombre de base de datos especificado o para la base de datos actual si no se especifica ningún nombre.
- **DB_NAME.** Devuelve el nombre de base de datos para un Id. de base de datos especificado o para la base de datos actual si no se ha especificado ningún Id.
- **FILE_ID.** Devuelve el Id. de archivo para el nombre de archivo lógico indicado de la base de datos actual.
- **FILE_NAME.** Devuelve el nombre de archivo lógico para el Id. de archivo indicado.
- **FILEGROUP_ID.** Devuelve el Id. de grupo de archivos para un nombre de grupo de archivos especificado.
- **FILEGROUP_NAME.** Devuelve el nombre del grupo de archivos para el Id. de grupo de archivos especificado.

PROCEDIMIENTOS ALMACENADOS DEL SISTEMA

SQL Server 2014 proporciona numerosos procedimientos almacenados del sistema para recuperar metadatos de la base de datos. Estos procedimientos proporcionan una manera alternativa de consultar información proporcionada por las vistas de catálogo. Algunos aceptan argumentos para permitir la personalización de los conjuntos de resultados.

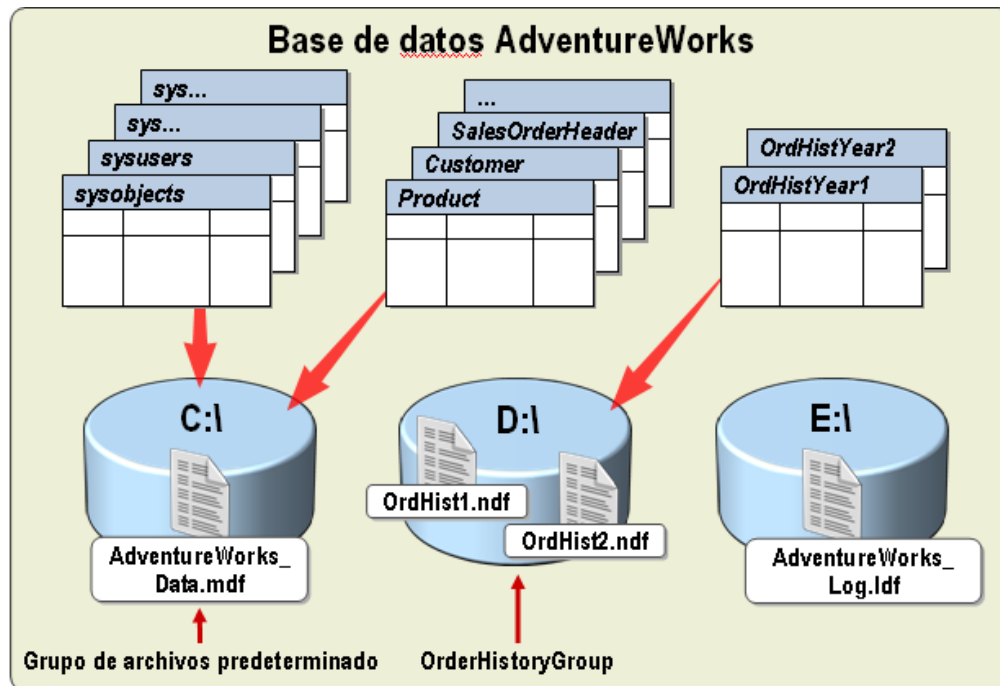
En la lista siguiente se describen algunos procedimientos almacenados del sistema utilizados con frecuencia de los centenares que hay disponibles:

- **sp_databases.** Muestra las bases de datos que hay disponibles dentro de una instancia de SQL Server o a las que se puede tener acceso a través de una puerta de enlace de base de datos.
- **sp_stored_procedures.** Devuelve una lista de los procedimientos almacenados de la base de datos actual.
- **sp_help.** Muestra información acerca de un objeto de base de datos, un tipo de datos definidos por el usuario o un tipo de datos proporcionado por SQL Server 2014.

Componentes físicos: archivos y grupos de archivos de bases de datos.

Un grupo de archivos es un conjunto lógico de archivos de datos que permite a los administradores controlar todos los archivos del grupo como un único elemento.

La posibilidad de controlar la posición física de los objetos individuales de la base de datos puede proporcionar diversas ventajas en cuanto a facilidad de administración y rendimiento. Por ejemplo, puede utilizar varios grupos de archivos para controlar cómo se almacenan físicamente los datos de una base de datos en dispositivos de almacenamiento, y para separar los datos de lectura y escritura de los datos de sólo lectura.



TIPOS DE GRUPOS DE ARCHIVOS

SQL Server 2014 tiene un grupo de archivos principal y también puede tener grupos de archivos definidos por el usuario.

- El grupo de archivos principal contiene el archivo principal de datos con las tablas del sistema. El archivo principal de datos utiliza normalmente la extensión .mdf.
- Un grupo de archivos definido por el usuario consta de archivos de datos agrupados con fines de asignación y administrativos. Estos otros archivos de datos se conocen como archivos secundarios de datos y suelen utilizar la extensión .ndf.

SITUACIÓN DE EJEMPLO PARA VARIOS GRUPOS DE ARCHIVOS

La ilustración proporciona un ejemplo de cómo podría colocar los archivos de base de datos en discos diferentes, como se describe en la lista siguiente:

- Puede crear grupos de archivos definidos por el usuario para separar los archivos que se consultan con mucha frecuencia de los que se modifican mucho. En la ilustración, los archivos OrdHist1.ndf y OrdHist2.ndf se colocan en un disco distinto que las tablas Product, Customer y SalesOrderHeader, ya que se consultan como ayuda para la toma de decisiones en lugar de actualizarse con información de pedidos actual.
- También podría colocar los archivos OrdHist1.ndf y OrdHist2.ndf en discos diferentes si ambos se consultaran con mucha frecuencia.
- No puede poner archivos de registro de transacciones en grupos de archivos. El espacio del registro de transacciones se administra por separado del espacio de datos. Los registros de transacciones suelen utilizar la extensión .ldf.

En el siguiente ejemplo de código Transact-SQL se utiliza la instrucción CREATE DATABASE para implementar esta situación de ejemplo.

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

También puede utilizar la instrucción ALTER DATABASE para agregar o eliminar archivos y grupos de archivos de bases de datos existentes.

CUÁNDO CREAR GRUPOS DE ARCHIVOS

Puede crear varios archivos de datos en discos diferentes y crear un grupo de archivos definido por el usuario para contener los archivos. Las dos razones principales para utilizar grupos de archivos son mejorar el rendimiento y controlar la colocación física de datos.



USO DE VARIOS ARCHIVOS EN UN ÚNICO GRUPO DE ARCHIVOS PARA MEJORAR EL RENDIMIENTO

Si bien Matriz redundante de discos independientes (RAID) es la manera preferida de mejorar el rendimiento de una base de datos, puede asignar varios archivos de discos distintos a un único grupo de archivos para mejorar el rendimiento implementando una forma de seccionamiento de datos dentro de SQL Server. Puesto que SQL Server utiliza una estrategia de relleno proporcional al escribir datos en un grupo de archivos, los datos se reparten entre los archivos y, por tanto, en las particiones físicas del disco. Este método permite tener un control más fino sobre el seccionamiento de datos del que se puede lograr al crear un conjunto de volúmenes seccionados en el sistema operativo Windows, o utilizando una controladora de matriz RAID.

Nota: En la mayoría de los casos, el uso de las funciones de seccionamiento de RAID Proporciona la misma ganancia de rendimiento que podría lograr utilizando grupos de archivos definidos por el usuario, sin la carga administrativa agregada que supone definir y administrar los grupos de archivos.

USO DE VARIOS GRUPOS DE ARCHIVOS PARA CONTROLAR LA COLOCACIÓN FÍSICA DE LOS DATOS

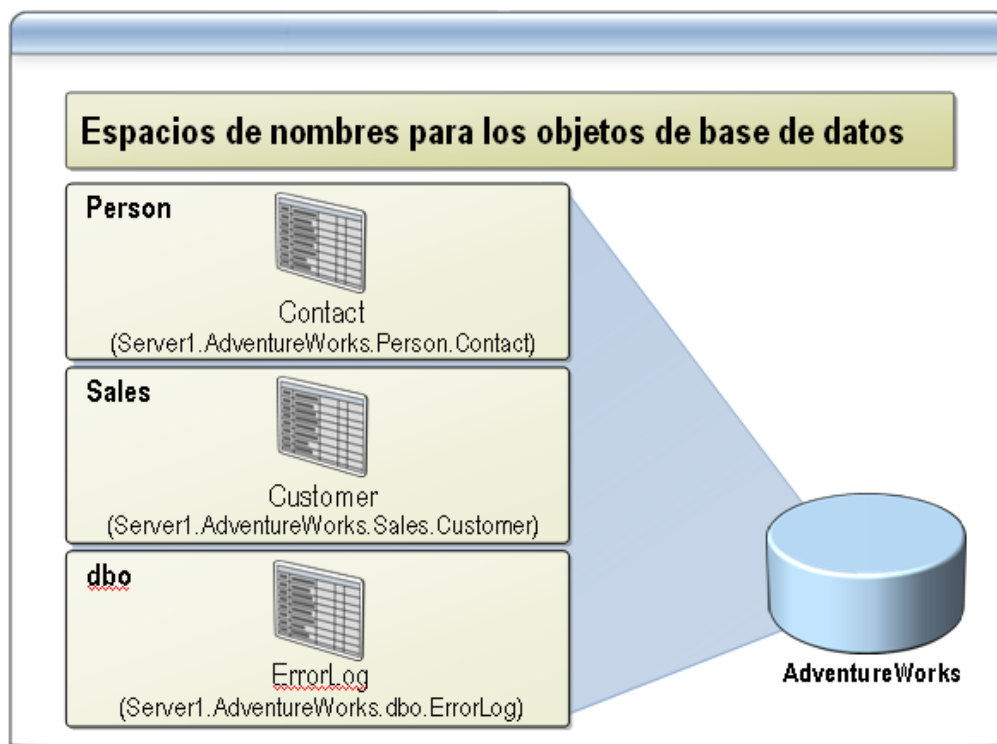
Para utilizar grupos de archivos con el fin de simplificar el mantenimiento o lograr objetivos de diseño, puede:

- Almacenar los datos de lectura y escritura separados de los datos de sólo lectura para mantener separados los diferentes tipos de actividad de E/S de disco.
- Almacenar los índices en discos diferentes que las tablas, lo que puede conducir a un mayor rendimiento.
- Hacer copia de seguridad o restaurar archivos individuales o grupos de archivos en lugar de hacer copia de seguridad o restaurar una base de datos entera. Puede ser necesario hacer copia de seguridad de archivos o de grupos de archivos para las bases de datos grandes con el fin de tener una estrategia eficaz de copia de seguridad y restauración.
- Agrupar en los mismos grupos de archivos las tablas y los índices que tienen unos requisitos de mantenimiento similares. Quizás desee realizar tareas de mantenimiento en algunos objetos con más frecuencia que en otros. Por ejemplo, si crea dos grupos de archivos y les asigna tablas, puede ejecutar las tareas diarias de mantenimiento en las tablas de un grupo diario y las tareas de mantenimiento semanales en las tablas de un grupo semanal. Esto limita la contención de disco entre los dos grupos de archivos.
- Separar las tablas de usuario y otros objetos de base de datos de las tablas del sistema en el grupo de archivos principal. También debe cambiar el grupo de archivos predeterminado para evitar que el crecimiento inesperado de las tablas restrinja las tablas del sistema del grupo de archivos principal.
- Almacenar las particiones de una tabla con particiones en varios grupos de archivos. Ésta es una buena forma de separar físicamente los datos que tienen necesidades de acceso diferentes dentro de una única tabla, y también puede proporcionar ventajas de facilidad de administración y rendimiento.

Creación de esquemas

Los desarrolladores que hayan trabajado con Microsoft .NET Framework o con XML estarán familiarizados con el concepto de espacios de nombres. Un espacio de nombres ayuda a agrupar los objetos relacionados, haciendo que las listas de objetos complejas sean más fáciles de administrar. SQL Server 2014 utiliza esquemas para implementar un concepto similar para los objetos de base de datos.

Los objetos de una base de datos (como tablas, vistas y procedimientos almacenados) se crean dentro de un esquema. Es esencial comprender lo que es un esquema antes de planear e implementar una base de datos de SQL Server 2014.



ESQUEMAS COMO ESPACIOS DE NOMBRES

Un esquema es un espacio de nombres para objetos de base de datos. Es decir, un esquema define un límite dentro del cual todos los nombres son únicos. Puesto que los nombres de esquema deben ser únicos dentro de la base de datos, cada objeto de una base de datos tiene un nombre completo único con el formato servidor.Base de datos.Eschema.Objeto. Dentro de una base de datos, puede acortarlo a esquema.objeto.

La ilustración anterior muestra tres esquemas de la base de datos AdventureWorks en una instancia de SQL Server denominada Server1. Los esquemas se denominan Person, Sales y dbo. Cada uno de estos esquemas contiene una tabla y el nombre completo de la tabla incluye el nombre del servidor, la base de datos y el esquema. Por ejemplo, el nombre completo de la tabla ErrorLog del esquema dbo es Server1.AdventureWorks.dbo.ErrorLog.

En versiones anteriores de SQL Server, el espacio de nombres de un objeto estaba determinado por el nombre de usuario de su propietario. En SQL Server 2014, los esquemas están separados de la propiedad de los objetos, lo que proporciona las ventajas siguientes:

- Mayor flexibilidad a la hora de organizar los objetos de base de datos en espacios de nombres, ya que la agrupación de objetos en esquemas no depende de la propiedad de los objetos.
- Administración de permisos más sencilla, ya que se puede otorgar permisos en el ámbito del esquema y en los objetos individuales.
- Facilidad de administración mejorada, porque al quitar a un usuario no es necesario cambiar el nombre de todos los objetos que ese usuario posee.

ESQUEMAS DE EJEMPLO

La base de datos AdventureWorks utiliza los esquemas siguientes para organizar sus objetos de base de datos en espacios de nombres:

- HumanResources
- Person
- Production
- Purchasing
- Sales

Por ejemplo, para hacer referencia a la tabla Employee del esquema HumanResources se utiliza HumanResources.Employee.

EL ESQUEMA DBO

Todas las bases de datos contienen un esquema denominado dbo. dbo es el esquema predeterminado para todos los usuarios que no tienen ningún otro esquema predeterminado definido explícitamente.

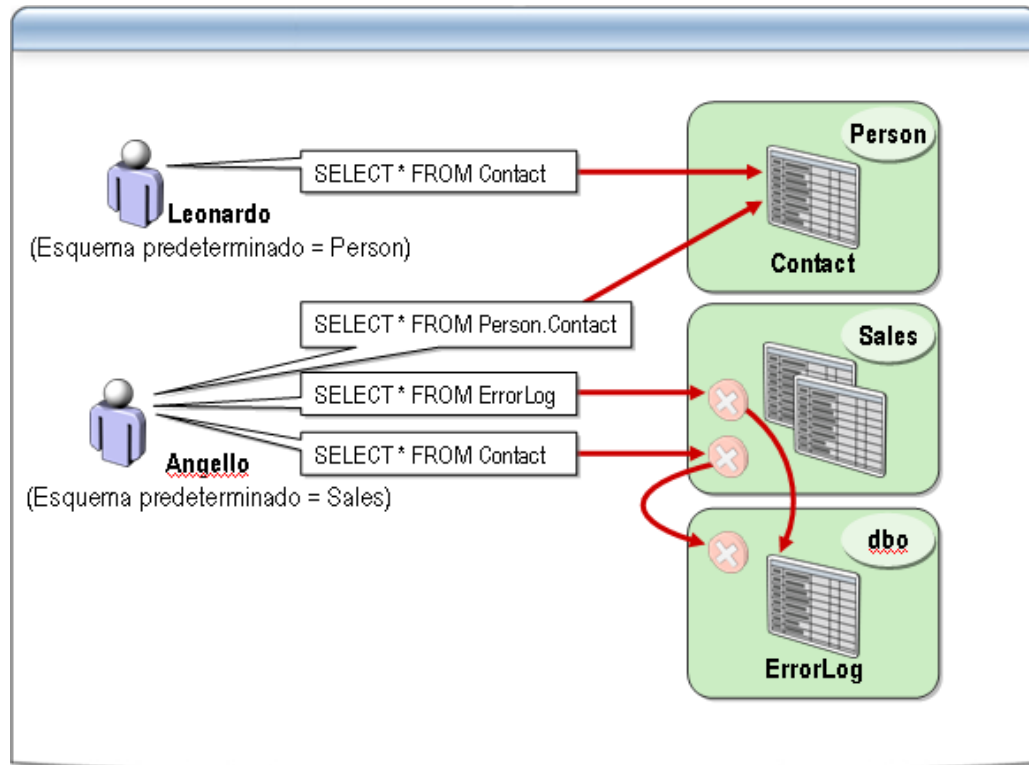
CREACIÓN DE UN ESQUEMA

Para crear un esquema, utilice el Explorador de objetos de SQL Server Management Studio o utilice la instrucción CREATE SCHEMA, como se muestra en el ejemplo siguiente.

```
Use AdventureWorks
GO
CREATE SCHEMA Sales
GO
```

Cómo funciona la resolución de nombres de objetos

Cuando una base de datos contiene varios esquemas, la resolución de nombres de objetos puede resultar confusa. Por ejemplo, una base de datos podría contener dos tablas denominadas Order en dos esquemas diferentes, Sales y dbo. Los nombres completos de los objetos dentro de la base de datos son inequívocos: Sales.Order y dbo.Order, respectivamente. Sin embargo, el uso del nombre incompleto Order puede producir resultados inesperados. Puede asignar a los usuarios un esquema predeterminado para controlar cómo se resuelven los nombres de objetos incompletos.



CÓMO FUNCIONA LA RESOLUCIÓN DE NOMBRES

SQL Server 2014 utiliza el proceso siguiente para resolver un nombre de objeto incompleto:

1. Si el usuario tiene un esquema predeterminado, SQL Server intenta encontrar el objeto en ese esquema predeterminado.
2. Si el objeto no se encuentra en el esquema predeterminado del usuario, si el usuario no tiene ningún esquema predeterminado, SQL Server intenta encontrar el objeto en el esquema dbo.

Por ejemplo, un usuario que tiene el esquema predeterminado Person ejecuta la siguiente instrucción Transact-SQL:

```
SELECT * FROM Contact
```

SQL Server 2014 intentará resolver primero el nombre de objeto como Person.Contact. Si el esquema Person no contiene un objeto denominado Contact, SQL Server intentará resolver el nombre de objeto como dbo.Contact.

Si un usuario que no tiene ningún esquema predeterminado definido ejecuta la misma instrucción, SQL Server resolverá inmediatamente el nombre de objeto como dbo.Contact.

ASIGNACIÓN DE UN ESQUEMA PREDETERMINADO

Puede asignar un esquema predeterminado a un usuario utilizando el cuadro de diálogo Propiedades de Usuario de la base de datos o especificando el nombre del esquema en

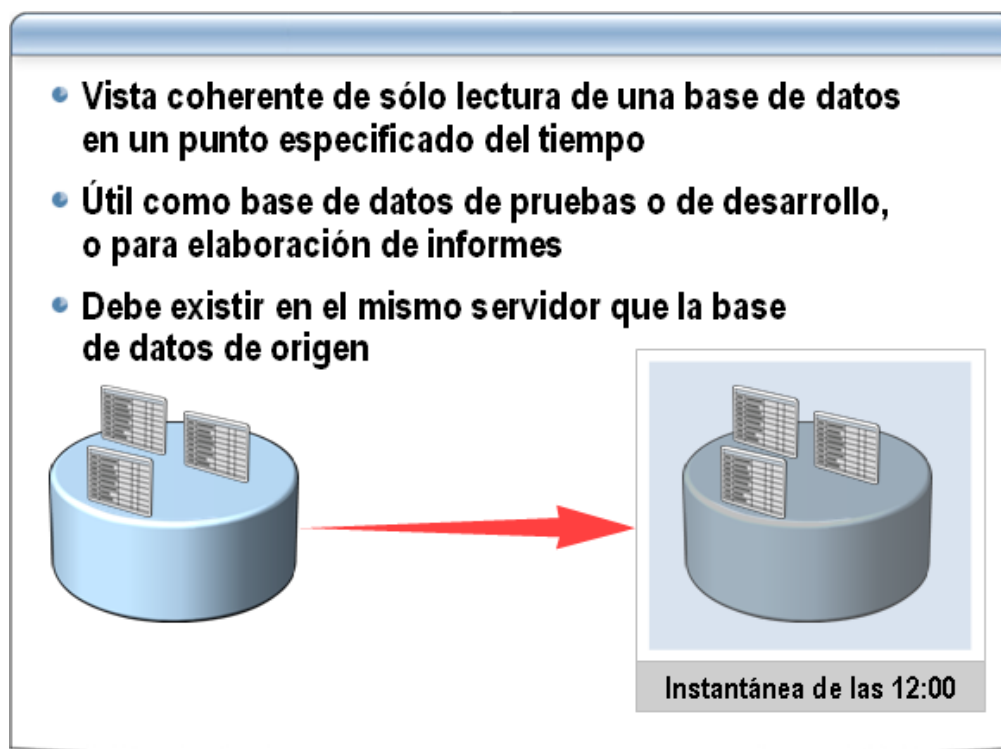
la cláusula `DEFAULT_SCHEMA` de la instrucción `CREATE USER` o `ALTER USER`. Por ejemplo, el siguiente código Transact-SQL asigna Sales como el esquema predeterminado para el usuario Angello:

```
ALTER USER Angello WITH DEFAULT_SCHEMA = Sales
```

Creación de instantáneas de base de datos

Como desarrollador de bases de datos de su organización, quizás tenga que crear una vista estática de una base de datos de SQL Server 2014. Este tipo de vista estática de los datos le permite trabajar con los datos tal y como aparecían en un punto del tiempo concreto, en lugar de reflejar el estado actual de los datos. Esta vista de punto en el tiempo podría ser útil para crear informes, o para fines de desarrollo o pruebas.

Hay muchas situaciones en las que una simple copia de la base de datos, conocida como una instantánea, es útil como base de datos en estado de espera, como base de datos de pruebas y desarrollo o simplemente como base de datos de informes. Para crear una instantánea de base de datos se utiliza la cláusula `AS SNAPSHOT OF` de la instrucción `CREATE DATABASE`.



Una instantánea de base de datos es una vista estática de sólo lectura de una base de datos en un punto especificado en el tiempo que no cambia después de la creación de la instantánea. La base de datos a partir de la cual se crea la instantánea se conoce como base de datos de origen.

Las instantáneas de base de datos pueden ser útiles como un punto de restauración rápido en caso de daño accidental o malintencionado a los datos de la base de datos.

Sin embargo, no pueden utilizarse como sustituto de las copias de seguridad, ya que una instantánea de base de datos no contiene todos los registros de la base de datos.

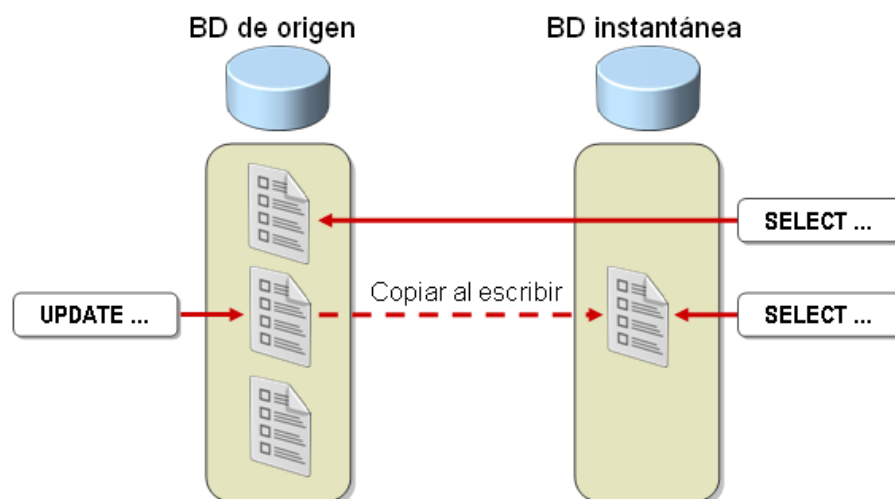
RESTRICCIONES PARA CREAR INSTANTÁNEAS

Una limitación de una instantánea de base de datos es que la instantánea debe estar en el mismo servidor que la base de datos de origen. Las restricciones siguientes también se aplican a las instantáneas de base de datos:

- No se pueden crear instantáneas para las bases de datos model, master o tempdb.
- No se puede hacer copia de seguridad ni restaurar instantáneas de base de datos.
- Las instantáneas de base de datos no se pueden adjuntar ni separar.
- No se pueden crear instantáneas de base de datos en particiones FAT32 o sin formato.
- Todas las instantáneas de base de datos creadas en una base de datos se deben quitar antes de quitar la propia base de datos.
- SQL Server Management Studio no permite la creación de instantáneas. Por tanto, las instantáneas de base de datos sólo se pueden crear mediante Transact-SQL.

CÓMO FUNCIONAN LAS INSTANTÁNEAS DE BASE DE DATOS

Las instantáneas de base de datos mantienen su vista estática de una base de datos de origen almacenando copias de los datos antes de la modificación cuando se realizan actualizaciones en la base de datos de origen. Esta información copiada se devuelve a continuación como parte de una consulta normal.



F. Crear una instantánea de base de datos

En el ejemplo siguiente se crea la instantánea de base de datos `sales_snapshot0600`.

Debido a que la instantánea de base de datos es de solo lectura, no se puede especificar un archivo de registro.

De acuerdo con la sintaxis, se especifican todos los archivos de la base de datos de origen, pero los grupos de archivos no se especifican.

La base de datos de origen en este ejemplo es la base de datos `Sales` creada en el ejemplo D.

[Copiar en](#)

```
USE master;
GO
CREATE DATABASE sales_snapshot0600 ON
    ( NAME = SPri1_dat, FILENAME = 'D:\SalesData\SPri1dat_0600.ss'),
    ( NAME = SPri2_dat, FILENAME = 'D:\SalesData\SPri2dt_0600.ss'),
    ( NAME = SGrp1Fil_dat, FILENAME = 'D:\SalesData\SG1Fi1dt_0600.ss'),
    ( NAME = SGrp1Fi2_dat, FILENAME = 'D:\SalesData\SG1Fi2dt_0600.ss'),
    ( NAME = SGrp2Fil_dat, FILENAME = 'D:\SalesData\SG2Fi1dt_0600.ss'),
    ( NAME = SGrp2Fi2_dat, FILENAME = 'D:\SalesData\SG2Fi2dt_0600.ss')
AS SNAPSHOT OF Sales ;
GO
```

MODIFICACIÓN DE DATOS

SQL Server 2014 utiliza la tecnología de copiar al escribir para implementar las instantáneas de base de datos sin incurrir en la sobrecarga que supone crear una copia completa de la base de datos. Una instantánea de base de datos está vacía inicialmente y se implementa físicamente como archivos dispersos NTFS, que son archivos para los que sólo se asigna espacio en el disco físico cuando es necesario. La primera vez que se actualiza una página en la base de datos de origen, la imagen original de esa página se copia a la instantánea de base de datos. Si una página no se modifica nunca, no se copia nunca.

SQL Server realiza realmente copias de páginas de datos enteras, aunque sólo se haya actualizado una fila. El uso de páginas en lugar de filas es más eficiente. El rendimiento de lectura y escritura para toda una página frente a toda una fila es muy similar. Una página puede contener varias filas, por lo que aunque se actualicen varias filas de una página de datos, SQL Server sólo necesita realizar una operación de copia. Este eficiente mecanismo de copia facilita la creación eficaz de instantáneas de base de datos incluso en una base de datos que se actualiza con frecuencia.

RECUPERACIÓN DE DATOS

Un usuario que tiene acceso a una instantánea de base de datos sólo verá la copia de una página de la instantánea si esa página ha cambiado desde que se creó la instantánea.

De lo contrario, se redirige al usuario a la página correspondiente en la base de datos de origen. Este redireccionamiento es transparente para el usuario.



CAPITULO 2

- Los tipos de datos del sistema que ofrece SQL Server 2014.
- Las tablas, relación y aplicación a las tablas de sus restricciones: primary key, foreign key, unique, check, default, identity y null.

CREACIÓN DE TIPOS DE DATOS Y TABLAS

Antes de poder crear una tabla, debe definir los tipos de datos para la tabla. Los tipos de datos especifican el tipo de información (caracteres, números o fechas) que una columna puede contener y cómo se almacenan los datos. SQL Server 2014 proporciona más de 30 tipos de datos del sistema. SQL Server 2014 también acepta tipos de datos de alias, que son tipos de datos definidos por el usuario basados en tipos de datos del sistema.

Los tipos de datos del sistema que ofrece SQL Server 2014.

Los tipos de datos definen los valores de datos permitidos para cada columna de una tabla de base de datos. SQL Server proporciona varios tipos de datos. Algunas categorías de tipos de datos que se utilizan normalmente en los lenguajes de programación tienen varios tipos de datos asociados de SQL Server. Debe utilizar el tipo de datos más pequeño que satisfaga sus necesidades. Esto ahorrará espacio en disco y permitirá que quepan más filas en una página de datos, lo que proporciona el máximo rendimiento.

bigint	binary	bit	char	CLR
cursor	date	datetime	datetime2	datetimeoffset
decimal	float	hierarchyid	image	int
money	nchar	ntext	numeric	nvarchar
real	rowversion	smalldatetime	smallint	smallmoney
sql_variant	table	text	time	timestamp
tinyint	varbinary	varchar	uniqueidentifier	xml

TIPOS DE DATOS SUMINISTRADOS POR EL SISTEMA

En la tabla siguiente se describen los tipos de datos suministrados por el sistema de SQL Server 2014 y se indican los sinónimos del American National Standards Institute (ANSI) para los tipos de datos estándar utilizados en sistemas de base de datos compatibles con ANSI. Al hacer referencia a tipos de datos en código Transact-SQL, puede utilizar el nombre del tipo de datos específico de SQL Server o el sinónimo de ANSI.

Categoría de tipo de datos	Tipos de datos suministrados por el sistema de SQL Server	Sinónimo de ANSI	Número de bytes
Entero	int bigint smallint, tinyint	integer — —	4 8 2, 1

Categoría de tipo de datos	Tipos de datos suministrados por el sistema de SQL Server	Sinónimo de ANSI	Número de bytes
Numéricos exactos	decimal[(p[, s])] numeric[(p[, s])]	dec —	2–17
Numéricos aproximados	float[(n)] real	double precision, float[(n)] for n=8–15 float[(n)] for n=1–7	8 4
Monetario	money, smallmoney	—	8, 4
Fecha y hora	datetime, smalldatetime, date, time	—	8, 4, 4, 4
Carácter no Unicode	char[(n)] varchar[(n)] varchar(max) text	character[(n)] char VARYING[(n)] character VARYING[(n)] char VARYING(max) character VARYING(max)	0–8,000 0–8,000 0-2 gigabytes (GB) (en fila o puntero de 16 bytes) 0-2 GB (puntero de 16 bytes)
Carácter Unicode	nchar [(n)] nvarchar [(n)] nvarchar (max) ntext	national char[(n)] national character[(n)] national char VARYING[(n)] national character VARYING[(n)] national char VARYING(max) national character VARYING(max)	0–8.000 (4.000 caracteres) 0–8.000 (4.000 caracteres) 0-2 GB (en fila o puntero de 16 bytes) 0-2 GB (puntero de 16 bytes)
Binario	binary[(n)] varbinary [(n)] varbinary (max)	— binary VARYING[(n)] binary VARYING(max)	0–8,000 0–8,000 0-2 GB (en fila o puntero de 16 bytes)
Imagen	image	—	0-2 GB (puntero de 16 bytes)
Identificador global	uniqueidentifier	—	16
XML	xml	—	0–2 GB
Especial	bit, cursor timestamp sysname table sql_variant	— rowversion — —	1, 0–8 8 256 0–8,016

TIPOS DE DATOS NUMÉRICOS EXACTOS Y NUMÉRICOS APROXIMADOS

La forma en que piense utilizar un tipo de datos debe determinar si elegirá un tipo de datos numérico exacto o numérico aproximado.

- Los tipos de datos numéricos exactos le permiten especificar exactamente la escala y la precisión que va a utilizar.

Por ejemplo, puede especificar tres dígitos a la derecha del signo decimal y cuatro a la izquierda. Una consulta siempre devuelve exactamente lo que escribió. SQL Server admite dos tipos de datos numéricos exactos por compatibilidad con ANSI: decimal y numeric. En general, los tipos de datos numéricos exactos se utilizan para aplicaciones financieras en las que desea representar los datos de forma coherente (siempre con dos posiciones decimales) y consultar en esa columna (por ejemplo, para encontrar todos los préstamos cuya tasa de interés sea del 8,75 por ciento).

- Los tipos de datos numéricos aproximados almacenan los datos con la máxima precisión posible.

Por ejemplo, la fracción un tercio se representa en un sistema decimal como 0,33333 (período). El número no se puede almacenar con precisión, por lo que se almacena una aproximación. SQL Server admite dos tipos de datos numéricos aproximados: float y real. Si va a redondear números o a realizar comprobaciones de calidad entre valores, debe evitar el uso de tipos de datos numéricos aproximados.

Sugerencia:

Es mejor evitar hacer referencia a columnas que tengan los tipos de datos float o real en cláusulas WHERE.

TIPOS DE DATOS FIJOS Y VARIABLES

Cuando sepa que todos los valores de una entidad de datos determinada tendrán exactamente el mismo tamaño, puede utilizar un tipo de datos de longitud fija como nchar e indicar el número de bytes que se deben reservar para cada valor. Cuando la longitud del valor puede variar, es más eficaz utilizar un tipo de datos de longitud variable como nvarchar y especificar la longitud máxima del valor. Cuando se especifica un tipo de datos de longitud variable, SQL Server reserva 2 bytes para cada valor de longitud variable como un marcador y sólo utiliza el espacio adicional necesario para cada valor.

VALORES DE DATOS GRANDES

Las columnas que se utilizarán para almacenar valores de datos medianos o grandes (normalmente de más de 8.000 bytes) pueden almacenarse utilizando un tipo de objeto grande (LOB) como text o image, o como una columna varchar, nvarchar o varbinary declarada con el especificador max. Los tipos de datos LOB se proporcionan principalmente por compatibilidad con versiones anteriores. Normalmente debe utilizar el especificador max para almacenar valores de datos grandes.

Qué son los tipos de datos de alias

Un tipo de datos de alias es un tipo de datos personalizado basado en un tipo de datos suministrado por el sistema. Un tipo de datos de alias:

- Le permite refinar aún más los tipos de datos para garantizar la coherencia al trabajar con elementos de datos comunes en varias tablas o bases de datos.
- Se define en una base de datos concreta.
- Debe tener un nombre único dentro de la base de datos. (Pero puede haber tipos de datos de alias con nombres diferentes y la misma definición.)

- Se basan en los tipos suministrados por el sistema
- Se usan para elementos de datos comunes con un formato específico
- Se crean con la instrucción CREATE TYPE

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

CUÁNDO CREAR TIPOS DE DATOS DE ALIAS

Debe crear un tipo de datos de alias cuando necesite definir un elemento de datos utilizado con frecuencia con un formato concreto. Por ejemplo, una columna en la que almacenará un código de país basado en las abreviaturas internacionales de nombre de país de dos caracteres de la Organización internacional de normalización (ISO), como JP para Japón y CH para Suiza, puede definirse como char(2). Sin embargo, si el código de país va a utilizarse con frecuencia en toda la base de datos, podría definir un tipo de datos CountryCode y utilizarlo en su lugar. Esto facilita la comprensión de las definiciones de los objetos y del código de su base de datos.

Sugerencia:

Los tipos de datos de alias que crea en la base de datos model se incluyen automáticamente en todas las bases de datos que se crean posteriormente.

EJEMPLO DE CREACIÓN DE UN TIPO DE DATOS DE ALIAS

Puede crear tipos de datos de alias utilizando el Explorador de objetos de SQL Server Management Studio o la instrucción CREATE TYPE de Transact-SQL.

En el ejemplo de código siguiente se muestra cómo crear un tipo de datos de alias denominado CountryCode.

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

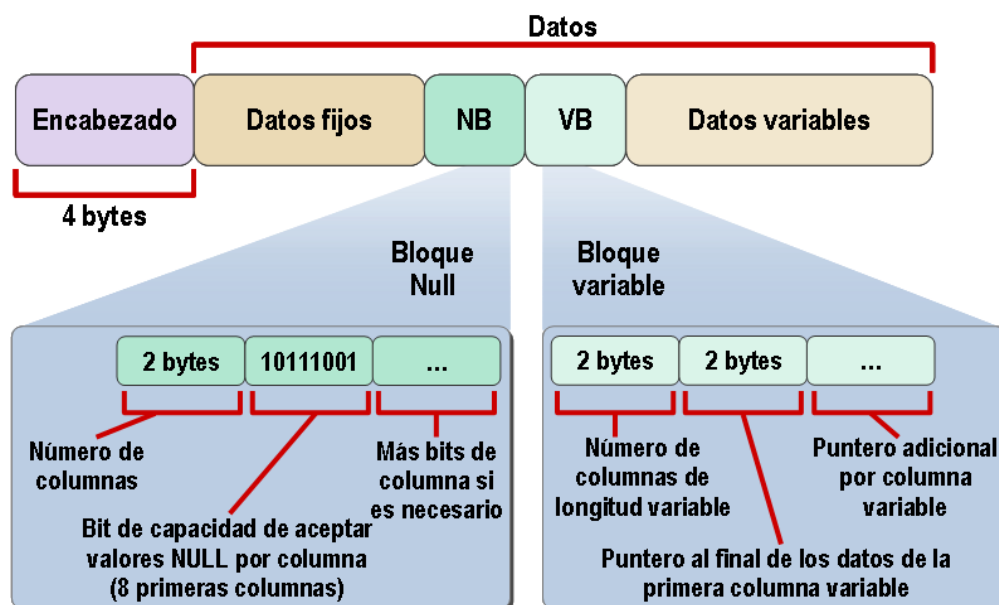
Tenga en cuenta que al crear un tipo de datos de alias puede especificar su capacidad de aceptar valores NULL. Un tipo de datos de alias creado con la opción NOT NULL no puede utilizarse nunca para almacenar un valor NULL. Es importante especificar la capacidad de aceptar valores NULL apropiada al crear un tipo de datos. Para cambiar un tipo de datos, debe utilizar la instrucción DROP TYPE para quitar el tipo de datos y volver a crear un nuevo tipo de datos para reemplazarlo. Además, puesto que no puede quitar un tipo de datos utilizado por tablas de la base de datos, también necesitaría modificar primero (con ALTER) cada tabla que utiliza el tipo de datos.

Las tablas, relación y aplicación a las tablas de sus restricciones.

Después de definir todos los tipos de datos que necesitará para su base de datos, puede crear las tablas necesarias para almacenar los datos. Al crear una tabla, tiene que entender cómo organiza SQL Server físicamente los datos, y cómo definir las columnas para lograr un almacenamiento y un rendimiento óptimos.

Cómo organiza SQL Server los datos en filas

Una fila de datos consta de un encabezado de fila y una parte de datos. Es importante comprender los elementos de la parte de datos de cada fila para estimar con precisión el tamaño de una tabla.



EL ENCABEZADO DE FILA

El encabezado de fila de 4 bytes contiene información acerca de las columnas de la fila de datos, como un puntero a la ubicación del final de la parte de datos fijos de la fila y si existen columnas de longitud variable en la fila.

LA PARTE DE DATOS

La parte de datos de una fila puede contener los elementos siguientes:

- **Datos de longitud fija.** Los datos de longitud fija se escriben en la página antes que los datos de longitud variable. Una fila de datos de longitud fija vacía ocupa tanto espacio como una fila de datos de longitud fija rellena. Una tabla que sólo tenga columnas de longitud fija siempre almacena el mismo número de filas en una página.
- **Bloque Null.** Un bloque null es un conjunto de bytes de longitud variable. Consta de 2 bytes que almacenan el número de columnas seguido de un mapa de bits null que indica si cada columna individual es null o no. El tamaño de un mapa de bits null es igual a 1 bit por columna, redondeado hasta el byte más cercano. De una a ocho columnas requieren un mapa de bits de 1 byte. De nueve a 16 columnas requieren un mapa de bits de 2 bytes.
- **Bloque variable.** Un bloque variable consta de 2 bytes que describen cuántas columnas de longitud variable hay presentes. 2 bytes adicionales por columna señalan el final de cada columna de longitud variable. Se omite el bloque variable si no hay ninguna columna de longitud variable.
- **Datos de longitud variable.** Los datos de longitud variable se escriben en la página después del bloque variable. Una fila de datos de longitud variable vacía no ocupa espacio. Una tabla con columnas de longitud variable puede tener pocas filas largas o muchas filas cortas.

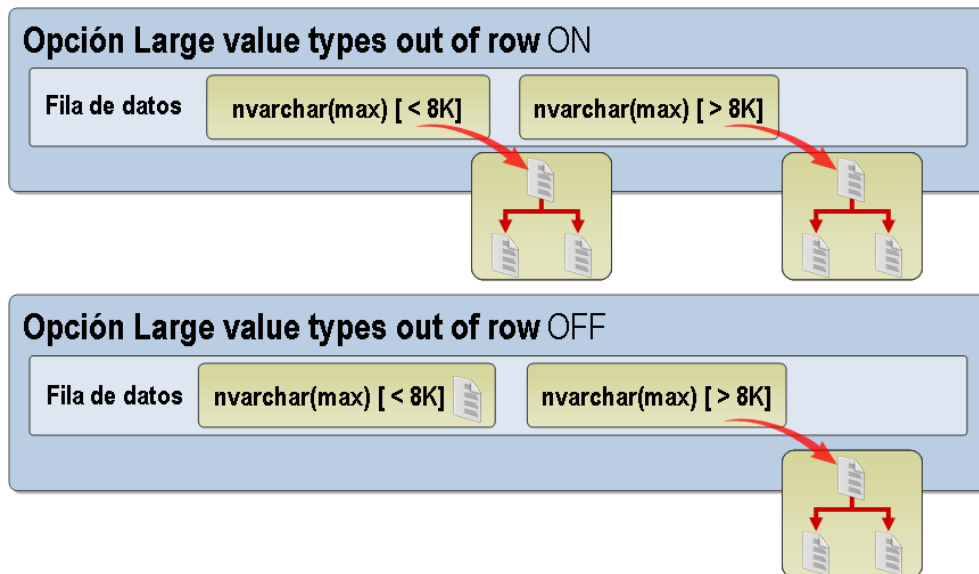
Cómo organiza SQL Server los valores de datos grandes

Una fila no puede ser mayor que una página, por lo que cuando una columna puede contener un valor de más de 8.000 bytes, SQL Server debe almacenar los datos por separado y almacenar en la fila real sólo un puntero a los datos.

DATOS DE OBJETOS GRANDES (LOB)

Los tipos de datos de objetos grandes (LOB) pueden almacenarse como un conjunto de páginas o en filas de datos. Entre los tipos de datos LOB se incluyen:

- **text.** El tipo de datos text puede contener 2.147.483.647 caracteres. El tipo de datos text no Unicode no puede utilizarse para variables en procedimientos almacenados.
- **ntext.** El tipo de datos ntext puede contener un máximo de 230 – 1 (1.073.741.823) caracteres o 231 – 1 bytes, que es 2.147.483.647 bytes de datos Unicode de longitud variable.
- **image.** El tipo de datos image puede contener de 0 a 2.147.483.647 bytes de datos binarios.



Puesto que los tipos de datos text, ntext e image suelen ser grandes, SQL Server los almacena fuera de filas. Un puntero de 16 bytes en la fila de datos señala a una estructura raíz que contiene los datos. La estructura raíz de texto forma el nodo raíz del árbol b, que señala a los bloques de datos. Si hay más de 32 kilobytes (KB) de datos, se agregan nodos intermedios del árbol b entre el nodo raíz y los bloques de datos.

Esto permite que la exploración rápida del árbol b empiece en medio de una cadena. Cuando hay contenido text, ntext e image de tamaño pequeño a mediano, SQL Server proporciona la opción de almacenar los valores en la fila de datos en lugar de almacenarlos en una estructura de árbol b independiente. Puede especificar esta opción text in row. También puede establecer el límite de la opción; el intervalo es de 24 a 7.000 bytes. Puede habilitar la opción text in row para una tabla mediante el procedimiento almacenado del sistema sp_tableoption.

EL ESPECIFICADOR MAX

El especificador max puede utilizarse con los tipos de datos varchar, nvarchar y varbinary. Esto permite utilizar estos tipos de datos para almacenar valores de datos mayores de 8.000 bytes del mismo modo que se almacenan los tipos de datos LOB, sin algunas de las limitaciones que se aplican a los tipos de datos LOB.

Como ocurre con los tipos de datos LOB, puede controlar cómo se almacenan las columnas definidas utilizando el especificador max. Puede establecer la opción large value types out of row con el procedimiento almacenado del sistema sp_tableoption.

Cuando esta opción está activada, se almacena en la fila un puntero al nodo raíz del árbol b para los datos; cuando la opción está desactivada, los valores que son suficientemente pequeños se almacenan directamente en la fila y sólo se utiliza un puntero para los valores que son demasiado grandes y no caben en la fila.

Consideraciones para la creación de tablas

Al crear una tabla, debe especificar el nombre de la tabla, los nombres de columna y los tipos de datos de las columnas. Los nombres de columna deben ser únicos en una tabla específica. Sin embargo, puede utilizar el mismo nombre de columna en tablas diferentes dentro de la misma base de datos. Debe especificar un tipo de datos para cada columna.

Puede tener:

- Más de 2.000 millones objetos por base de datos, incluyendo tablas.
- Hasta 1.024 columnas por tabla no ancha.
- Hasta 30,000 columnas por tabla ancha.
- Hasta 4,096 columnas por instrucción SELECT.
- Hasta 8.060 bytes por fila. (Esta longitud máxima aproximada no se aplica a los tipos de datos image, text y ntext ni a las columnas definidas utilizando el especificador max.)
- Hasta 32 niveles de procedimientos almacenados anidados.
- Hasta 2,100 parámetros por procedimiento almacenado y funciones definidas por el usuario.

Además del rol estándar de las tablas básicas definidas por el usuario, SQL Server proporciona los siguientes tipos de tabla que permiten llevar a cabo objetivos especiales en una base de datos:

Tablas con particiones

Las tablas con particiones son tablas cuyos datos se han dividido horizontalmente entre unidades que pueden repartirse por más de un grupo de archivos de una base de datos. Las particiones facilitan la administración de índices y tablas grandes al permitir el acceso y la administración de subconjuntos de datos rápidamente y con eficacia, mientras se mantiene la integridad de la colección global. De forma predeterminada, SQL Server 2014 admite hasta 15.000 particiones.

Tablas temporales

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

Tablas del sistema

SQL Server almacena los datos que definen la configuración del servidor y de todas sus tablas en un conjunto de tablas especial, conocido como tablas del sistema. Los usuarios no pueden consultar o actualizar directamente las tablas del sistema. La información de las tablas del sistema está disponible a través de las vistas del sistema.

Tablas anchas

Las tablas anchas usan las columnas dispersas para aumentar hasta 30.000 el número total de columnas permitidas. Las columnas dispersas son columnas normales que disponen de un almacenamiento optimizado para los valores NULL. Este tipo de columnas reducen los requisitos de espacio de los valores NULL a costa de una mayor sobrecarga a la hora de recuperar valores no NULL.

Una tabla ancha ha definido un conjunto de columnas, que es una representación XML sin tipo que combina todas las columnas dispersas de una tabla en una salida estructurada. El número de índices y estadísticas también se aumenta hasta 1.000 y 30.000, respectivamente. El tamaño máximo de una fila de una tabla ancha es de 8.019 bytes. Por consiguiente, la mayoría de los datos de cualquier fila deben ser NULL. El número máximo de columnas no dispersas más las columnas calculadas de una tabla ancha sigue siendo 1.024.

Las tablas anchas tienen las siguientes implicaciones de rendimiento.

- Las tablas anchas pueden aumentar el costo de mantenimiento de los índices de la tabla. Se recomienda que el número de índices de una tabla ancha se limite a los índices necesarios para la lógica de negocios. Si el número de índices aumenta, también lo harán el tiempo de compilación de DML y los requisitos de memoria. Los índices no clúster deben ser índices filtrados que se aplican a subconjuntos de datos.
- Las aplicaciones pueden agregar y quitar columnas de las tablas anchas de forma dinámica. Cuando se agregan o se quitan columnas, también se invalidan los planes de consulta compilados. Se recomienda diseñar una aplicación que se corresponda con la carga de trabajo prevista para minimizar los cambios de esquema.
- Cuando se agregan y se quitan datos de una tabla ancha, el rendimiento puede verse afectado. El diseño de las aplicaciones debe corresponderse con la carga de trabajo prevista para minimizar los cambios llevados a cabo en los datos de la tabla.
- Limite la ejecución de instrucciones DML en una tabla ancha destinadas a actualizar varias filas de una clave de agrupación en clústeres. La compilación y la ejecución de estas instrucciones pueden requerir recursos de memoria considerables.
- Las operaciones de cambio de partición en las tablas anchas pueden resultar lentas, y su procesamiento podría requerir grandes cantidades de memoria. Los requisitos de rendimiento y de memoria son proporcionales al número total de columnas existentes en las particiones de origen y de destino.
- Los cursores de actualización que actualizan columnas concretas de una tabla ancha deben enumerar las columnas de manera explícita en la cláusula FOR UPDATE. Esto ayudará a optimizar el rendimiento mientras se usan cursores.

INTERCALACIÓN DE COLUMNAS

Una intercalación es un criterio de ordenación para datos que determina el orden en que se muestran los valores cuando los datos se ordenan secuencialmente. Las distintas intercalaciones ordenan los datos con criterios diferentes dependiendo de si la intercalación distingue mayúsculas de minúsculas o no, las reglas de ordenación para las letras acentuadas y los caracteres especiales, y otras consideraciones. SQL Server permite almacenar entidades de datos con intercalaciones diferentes en la misma base de datos. Es posible especificar distintas intercalaciones de SQL Server en el nivel de columna, de forma que se pueda asignar una intercalación diferente a cada columna de una tabla.

CAPACIDAD DE ACEPTAR VALORES NULL

Puede especificar en la definición de tabla si se permiten valores nulos en cada columna. Si no especifica NULL o NOT NULL, SQL Server proporciona la característica NULL o NOT NULL basándose en el valor predeterminado de nivel de sesión o de nivel de base de datos. Sin embargo, estos valores predeterminados pueden cambiar, por lo que no debe confiar en ellos.

TIPOS DE COLUMNA ESPECIALES

Entre los tipos especiales de columna se incluyen los siguientes:

- **Columnas calculadas.** Una columna calculada es una columna virtual que no se almacena físicamente en la tabla. SQL Server utiliza una fórmula que usted crea para calcular este valor de columna utilizando otras columnas de la misma tabla. El uso de un nombre de columna calculada en una consulta puede simplificar la sintaxis de la consulta.
- **Columnas de identidad.** Puede utilizar la propiedad Identity para crear columnas (denominadas columnas de identidad) que contienen valores secuenciales generados por el sistema que identifican cada fila insertada en una tabla. Una columna de identidad suele utilizarse para los valores de clave principal. Hacer que SQL Server proporcione automáticamente los valores de clave puede reducir los costos y mejorar el rendimiento. Simplifica la programación, mantiene unos valores de clave principal cortos y reduce los cuellos de botella de transacciones del usuario.
- **Columnas timestamp.** Las columnas definidas con el tipo de datos timestamp tienen un valor predeterminado que consta de una marca de tiempo generada automáticamente que se garantiza que es única dentro de la base de datos.
- **Columnas uniqueidentifier.** Las columnas definidas con el tipo de datos uniqueidentifier pueden utilizarse para almacenar identificadores únicos globales (GUID), que se garantiza que son únicos universalmente. Puede generar un valor para una columna uniqueidentifier mediante la función NEWID de Transact-SQL.

EJEMPLO DE CREACIÓN DE UNA TABLA

Puede crear tablas utilizando el Explorador de objetos de SQL Server Management Studio o la instrucción CREATE TABLE de Transact-SQL. El siguiente ejemplo de código de Transact-SQL puede utilizarse para crear una tabla denominada CustomerOrders en un esquema denominado Sales. La tabla incluye una columna de identidad.

```
CREATE TABLE Sales.CustomerOrders
(OrderID int identity NOT NULL,
OrderDate datetime NOT NULL,
CustomerID int NOT NULL,
Notes nvarchar(200) NULL)
```


MODIFICACIÓN Y ELIMINACIÓN DE TABLAS

Puede modificar una definición de tabla en el Explorador de objetos o utilizando la instrucción ALTER TABLE de Transact-SQL. Por ejemplo, las siguientes instrucciones Transact-SQL muestran cómo agregar una columna y cambiar la capacidad de aceptar valores NULL de una columna en la tabla Sales.CustomerOrders definida previamente.

```
ALTER TABLE Sales.CustomerOrders
ADD SalesPersonID int NOT NULL
GO
ALTER TABLE Sales.CustomerOrders
ALTER COLUMN Notes nvarchar(200) NOT NULL
GO
```

Para quitar una tabla de la base de datos, puede eliminarla en el Explorador de objetos o utilizar la instrucción DROP TABLE de Transact-SQL.

EJEMPLO DE CREACIÓN DE UNA TABLA ANCHA

En este ejemplo, una tabla de documentos contiene un conjunto común que tiene las columnas DocID y Title. El grupo de producción desea tener una columna ProductionSpecification y una columna ProductionLocation para todos los documentos de producción. El grupo de marketing desea tener una columna MarketingSurveyGroup para los documentos de marketing. El código de este ejemplo crea una tabla que usa columnas dispersas, inserta dos filas en dicha tabla y, a continuación, selecciona datos en ella.

```
USE AdventureWorks2012;
GO

CREATE TABLE DocumentStore
  (DocID int PRIMARY KEY,
   Title varchar(200) NOT NULL,
   ProductionSpecification varchar(20) SPARSE NULL,
   ProductionLocation smallint SPARSE NULL,
   MarketingSurveyGroup varchar(20) SPARSE NULL );
GO

INSERT DocumentStore(DocID, Title, ProductionSpecification, ProductionLocation)
VALUES (1, 'Tire Spec 1', 'AXZZ217', 27);
GO

INSERT DocumentStore(DocID, Title, MarketingSurveyGroup)
VALUES (2, 'Survey 2142', 'Men 25 - 35');
GO
```

La selección de todas las columnas de la tabla devuelve un conjunto de resultados normal.

```
SELECT * FROM DocumentStore ;
```


El conjunto de resultados es el siguiente.

DocID	Title	ProductionSpecification	ProductionLocation	MarketingSurveyGroup
1	Tire Spec 1	AXZZ217	27	NULL
2	Survey 2142	NULL	NULL	Men 25-35

Dado que el departamento de producción no está interesado en los datos de marketing, desean usar una lista de columnas que devuelva solo las columnas de interés, como se muestra en la consulta siguiente.

```
SELECT DocID, Title, ProductionSpecification, ProductionLocation
FROM DocumentStore
WHERE ProductionSpecification IS NOT NULL ;
```

El conjunto de resultados es el siguiente.

DocID	Title	ProductionSpecification	ProductionLocation
1	Tire Spec 1	AXZZ217	27

Tipos de integridad de datos

Un paso importante en el diseño de una base de datos es decidir la mejor forma de implementar la integridad de los datos. La integridad de los datos hace referencia a la coherencia y la precisión de los datos que están almacenados en una base de datos. Los diferentes tipos de integridad de datos son los siguientes.

INTEGRIDAD DE DOMINIO

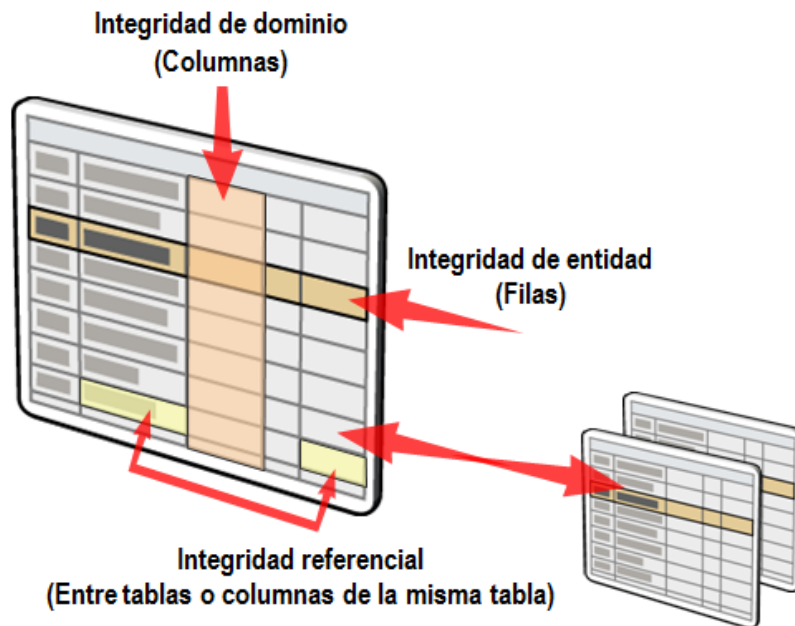
La integridad de dominio (o columna) especifica un conjunto de valores de datos que son válidos para una columna y determina si se permiten valores nulos. La integridad de dominio se suele implementar mediante el uso de comprobaciones de validez y, también, mediante la restricción del tipo de datos, el formato o el intervalo de los valores posibles permitidos en una columna.

INTEGRIDAD DE ENTIDAD

La integridad de entidad (o tabla) requiere que todas las filas de una tabla tengan un identificador exclusivo, conocido como clave principal. El que se pueda modificar el valor de la clave principal o eliminar la fila entera depende del nivel de integridad requerido entre la clave principal y cualquier otra tabla.

INTEGRIDAD REFERENCIAL

La integridad referencial asegura que siempre se mantienen las relaciones entre las claves principales (en la tabla a la que se hace referencia) y las claves externas (en las tablas que hacen referencia). No se puede eliminar una fila de una tabla a la que se hace referencia, ni se puede modificar la clave principal, si una clave externa hace referencia a la fila, salvo que se permita la acción en cascada. Puede definir relaciones de integridad referencial dentro de la misma tabla o entre tablas diferentes.



Creación de restricciones

Las restricciones se crean mediante la instrucción CREATE TABLE o ALTER TABLE. Puede agregar restricciones a una tabla con datos existentes y puede aplicar restricciones a una sola columna o a varias columnas:

- Si la restricción se aplica a una sola columna, se llama restricción de columna.
- Si la restricción hace referencia a varias columnas, se llama restricción de tabla, incluso si no hace referencia a todas las columnas de la tabla.

SINTAXIS PARCIAL

```
CREATE TABLE Tabla
( { < definiciónColumna >
  | < restricciónTabla > } [ ,...n ] )
< definiciónColumna > ::= { columnatipoDeDatos }
[ [ DEFAULT expresiónConstante ]
[ < restricciónColumna > ] [ ,...n ]
< restricciónColumna > ::=
[ CONSTRAINT nombreRestricción ]
[ [ { PRIMARY KEY | UNIQUE }
[ CLUSTERED | NONCLUSTERED ] ]
| [ [ FOREIGN KEY ] REFERENCES tablaRef [ ( columnaRef ) ]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ] ]
| CHECK ( expresiónLógica ) }

< restricciónTabla > ::=
[ CONSTRAINT nombreRestricción ]
{ [ { PRIMARY KEY | UNIQUE } [ CLUSTERED | NONCLUSTERED ]
{ ( columna [ ASC | DESC ] [ ,...n ] ) } ]
| FOREIGN KEY [ ( columna [ ,...n ] ) ]
REFERENCES tablaRef [ ( columnaRef [ ,...n ] ) ] }
```

```
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
| CHECK ( condicionesBúsqueda ) }
```

Tipo de integridad	Tipo de restricción	Descripción
Dominio	DEFAULT	Especifica el valor predeterminado de una columna
	CHECK	Especifica el valor permitido de una columna
	FOREIGN KEY	Especifica la columna en la que deben existir los valores
	NULL	Especifica si se permite NULL
Entidad	PRIMARY KEY	Identifica cada fila de manera única
	UNIQUE	Impide la duplicación de claves no principales
Referencial	FOREIGN KEY	Define columnas cuyo valor debe coincidir con la clave principal de esta tabla
	CHECK	Especifica el valor permitido para una columna basándose en el contenido de otra columna

Ejemplo

Este ejemplo crea la tabla Products, define columnas y define restricciones de columna y de tabla.

```
USE northwind
CREATE TABLE dbo.Products
(
    ProductID int IDENTITY (1,1) NOT NULL,
    ProductName nvarchar (40) NOT NULL,
    SupplierID int NULL,
    CategoryID int NULL,
    QuantityPerUnit nvarchar (20) NULL,
    UnitPrice money NULL CONSTRAINT DF_Products_UnitPrice DEFAULT(0),
    UnitsInStock smallint NULL CONSTRAINT DF_Products_UnitsInStock DEFAULT(0),
    UnitsOnOrder smallint NULL CONSTRAINT DF_Products_UnitsOnOrder DEFAULT(0),
    ReorderLevel smallint NULL CONSTRAINT DF_Products_ReorderLevel DEFAULT(0),
    Discontinued bit NOT NULL CONSTRAINT DF_Products_Discontinued DEFAULT(0),
    CONSTRAINT PK_Products PRIMARY KEY CLUSTERED (ProductID),
    CONSTRAINT FK_Products_Categories FOREIGN KEY (CategoryID)
    REFERENCES dbo.Categories (CategoryID) ON UPDATE CASCADE,
    CONSTRAINT FK_Products_Suppliers FOREIGN KEY (SupplierID)
    REFERENCES dbo.Suppliers (SupplierID) ON DELETE CASCADE,
    CONSTRAINT CK_Products_UnitPrice CHECK (UnitPrice >= 0),
    CONSTRAINT CK_ReorderLevel CHECK (ReorderLevel >= 0),
    CONSTRAINT CK_UnitsInStock CHECK (UnitsInStock >= 0),
    CONSTRAINT CK_UnitsOnOrder CHECK (UnitsOnOrder >= 0)
)
GO
```

Consideraciones para el uso de restricciones

Considere estos hechos cuando implemente o modifique restricciones:

- Puede crear, modificar y eliminar restricciones sin tener que eliminar y volver a crear una tabla.
- Tiene que generar lógica de control de errores en sus aplicaciones y transacciones para probar si se ha infringido una restricción.
- SQL Server comprueba los datos existentes cuando se agrega una restricción a una tabla.

Tiene que especificar los nombres de las restricciones cuando las cree, puesto que SQL Server proporciona nombres complicados, generados por el sistema. Los nombres tienen que ser exclusivos para el propietario del objeto de la base de datos y seguir las reglas de los identificadores de SQL Server.

Para obtener ayuda acerca de las restricciones, ejecute el procedimiento almacenado del sistema `sp_helpconstraint` o `sp_help`, o consulte las vistas del esquema de información, como `check_constraints`, `referential_constraints` y `table_constraints`.

Las tablas del sistema siguientes almacenan las definiciones de las restricciones: `syscomments`, `sysreferences` y `sysconstraints`.

Restricciones DEFAULT

La restricción DEFAULT escribe un valor en una columna cuando no se especifica en las instrucciones INSERT. Las restricciones DEFAULT implementan la integridad de dominio.

SINTAXIS PARCIAL

[CONSTRAINT nombreRestricción] DEFAULT expresiónConstante

EJEMPLO

En este ejemplo se agrega una restricción DEFAULT que inserta el valor UNKNOWN en la tabla `dbo.Customers` si no se proporciona el nombre de contacto.

```
USE Northwind ALTER TABLE dbo.Customers
ADD CONSTRAINT DF_contactname DEFAULT 'UNKNOWN'
FOR ContactName
```

Considere los hechos siguientes cuando aplique una restricción DEFAULT:

- La restricción comprueba los datos existentes en la tabla.
- Sólo se aplica a las instrucciones INSERT.
- Sólo se puede definir una restricción DEFAULT por cada columna.
- No se puede aplicar a columnas con la propiedad Identity o a columnas con el tipo de datos `rowversion`.
- Permite que se especifiquen algunos valores proporcionados por el sistema (`USER`, `CURRENT_USER`, `SESSION_USER`, `SYSTEM_USER` o `CURRENT_TIMESTAMP`) en lugar de valores definidos por el usuario. Dichos valores proporcionados por el sistema pueden ser útiles para obtener un registro de los usuarios que insertan los datos.

Restricciones CHECK

La restricción CHECK restringe los datos que los usuarios pueden escribir en una columna particular a unos valores específicos. Las restricciones CHECK son similares a las cláusulas WHERE donde se pueden especificar las condiciones bajo las que se aceptan los datos.

SINTAXIS PARCIAL

[CONSTRAINT nombreRestricción] CHECK (expresiónLógica)

EJEMPLO

Este ejemplo agrega una restricción CHECK para garantizar que una fecha de nacimiento cumpla un intervalo aceptable de fechas.

```
USE Northwind ALTER TABLE dbo.Employees
ADD CONSTRAINT CK_birthdate
CHECK (BirthDate > '01-01-1900' AND BirthDate < getdate())
```

Considere los hechos siguientes cuando aplique una restricción CHECK:

- La restricción comprueba los datos cada vez que se ejecuta una instrucción INSERT o UPDATE.
- Puede hacer referencia a otras columnas de la misma tabla.
- Por ejemplo, una columna salary podría hacer referencia a un valor de una columna job_grade.
- No se puede aplicar a columnas con el tipo de datos rowversion.
- No puede contener subconsultas.
- Si alguno de los datos infringe la restricción CHECK, puede ejecutar la instrucción DBCC CHECKCONSTRAINTS para ver las filas infractoras.

Restricciones PRIMARY KEY

La restricción PRIMARY KEY define una clave principal en una tabla para identificar de forma exclusiva cada una de sus filas. Implementa la integridad de entidad.

SINTAXIS PARCIAL

[CONSTRAINT nombreRestricción] PRIMARY KEY [CLUSTERED | NONCLUSTERED]
{ (columna[,...n]) }

EJEMPLO

En este ejemplo se agrega una restricción que especifica que la clave principal de la tabla dbo.Customers es la identificación del cliente e indica que se va a crear un índice no agrupado para implementar la restricción.

```
USE northwind
ALTER TABLE dbo.Customers
ADD
CONSTRAINT PK_Customers
PRIMARY KEY NONCLUSTERED (CustomerID)
```

Considere los hechos siguientes cuando aplique una restricción PRIMARY KEY:

- Sólo se puede definir una restricción PRIMARY KEY por tabla.
- Los valores escritos tienen que ser exclusivos.
- No se permiten valores nulos.
- Crea un índice exclusivo en las columnas especificadas. Puede especificar un índice agrupado o un índice no agrupado (el agrupado es el tipo predeterminado si no existe anteriormente).

NOTA: El índice creado para la restricción PRIMARY KEY no se puede eliminar directamente. Se elimina cuando se quita la restricción.

Restricciones UNIQUE

La restricción UNIQUE especifica que dos filas de una columna no pueden tener el mismo valor. Esta restricción implementa la integridad de entidad con un índice único.

La restricción UNIQUE es útil cuando ya se tiene una clave principal, como un número de empleado, pero se desea garantizar que otros identificadores, como el número del permiso de conducir de un empleado, también sean exclusivos.

SINTAXIS PARCIAL

[CONSTRAINT nombreRestricción] UNIQUE [CLUSTERED | NONCLUSTERED] { (columna[,...n]) }

EJEMPLO

Este ejemplo crea una restricción UNIQUE sobre la columna company name de la tabla dbo.Suppliers.

```
USE northwind
ALTER TABLE dbo.Suppliers
ADD
CONSTRAINT U_CompanyName
UNIQUE NONCLUSTERED (CompanyName)
```

Considere los hechos siguientes cuando aplique una restricción UNIQUE:

- Puede permitir un valor nulo.
- Puede aplicar varias restricciones UNIQUE en una misma tabla.
- Puede aplicar la restricción UNIQUE a una o varias columnas que tengan que tener valores exclusivos y no sean la clave principal de una tabla.
- La restricción UNIQUE se implementa mediante la creación de un índice exclusivo en la columna o columnas especificadas.

Restricciones FOREIGN KEY

La restricción FOREIGN KEY implementa la integridad referencial. La restricción FOREIGN KEY define una referencia a una columna con una restricción PRIMARY KEY o UNIQUE en la misma o en otra tabla.

SINTAXIS PARCIAL

[CONSTRAINT nombreRestricción] [FOREIGN KEY] [(columna[,...n])] REFERENCES tablaRef [(columnaRef [,...n])].

EJEMPLO

Este ejemplo utiliza una restricción FOREIGN KEY para garantizar que la identificación del cliente de la tabla dbo.Orders esté asociada con una identificación válida en la tabla dbo.Customers.

```
USE northwind
ALTER TABLE dbo.Orders
ADD CONSTRAINT FK_Orders_Customers
FOREIGN KEY (CustomerID)
REFERENCES dbo.Customers(CustomerID)
```

Considere los hechos y recomendaciones siguientes cuando aplique una restricción FOREIGN KEY:

- Proporciona integridad referencial de una o de varias columnas. El número de columnas y los tipos de datos que se especifican en la instrucción FOREIGN KEY tienen que coincidir con el número de columnas y los tipos de datos de la cláusula REFERENCES.
- Al contrario que las restricciones PRIMARY KEY o UNIQUE, las restricciones FOREIGN KEY no crean índices automáticamente. Sin embargo, si la base de datos utiliza muchas combinaciones, tiene que crear un índice para que FOREIGN KEY aumente el rendimiento en las combinaciones.
- Para modificar los datos, los usuarios deben tener permisos SELECT o REFERENCES en las tablas a las que se hace referencia en la restricción FOREIGN KEY.
- Sólo se puede utilizar la cláusula REFERENCES sin la cláusula FOREIGN KEY cuando se hace referencia a una columna de la misma tabla.

Deshabilitación de restricciones

Por motivos de rendimiento, algunas veces resulta aconsejable deshabilitar restricciones. Por ejemplo, es más conveniente permitir que se procesen grandes operaciones por lotes antes que habilitar restricciones. Esta sección describe el modo de deshabilitar la comprobación de restricciones, tanto si va a crear una nueva restricción o deshabilitar una existente.

Deshabilitación de la comprobación de las restricciones en los datos existentes

Cuando se define una restricción en una tabla que ya contiene datos, SQL Server los comprueba automáticamente para confirmar que cumplen los requisitos de la restricción. Sin embargo, puede deshabilitar la comprobación de restricciones en los datos existentes al agregar una restricción a una tabla.

Considere las recomendaciones siguientes para deshabilitar la comprobación de restricciones en los datos existentes:

- Sólo puede deshabilitar las restricciones CHECK y FOREIGN KEY. El resto de las restricciones se tienen que eliminar y volver a agregar.
- Para deshabilitar la comprobación de restricciones cuando se agrega una restricción CHECK o FOREIGN KEY a una tabla con datos existentes, incluya la opción WITH NOCHECK en la instrucción ALTER TABLE.
- Utilice la opción WITH NOCHECK si los datos existentes no van a cambiar. Los datos tienen que cumplir las restricciones CHECK si van a ser actualizados.
- Asegúrese de que deshabilitar la comprobación de la restricción es una acción apropiada. Puede ejecutar una consulta para cambiar los datos existentes antes de decidir agregar una restricción.

SINTAXIS PARCIAL

```
ALTER TABLE tabla [WITH CHECK□WITH NOCHECK]
ADD CONSTRAINT restricción
[FOREIGN KEY] [(columna[,...n])] REFERENCES tablaRef [(columnaRef [,...n])].
[CHECK (condicionesBúsqueda)]
```

EJEMPLO

En este ejemplo, se agrega una restricción FOREIGN KEY que comprueba que todos los empleados están asociados a un director válido. La restricción no se implementa en los datos existentes en el momento en que se agrega.

```
USE northwind
ALTER TABLE dbo.Employees
WITH NOCHECK
ADD CONSTRAINT FK_Employees_Employees
FOREIGN KEY (ReportsTo)
REFERENCES dbo.Employees(EmployeeID)
```

Deshabilitación de la comprobación de las restricciones al cargar datos nuevos

KEY existentes, de forma que sea posible modificar o agregar datos a una tabla sin comprobar la restricción.

Para evitar los costos de la comprobación de las restricciones, puede que le interese deshabilitar las restricciones cuando:

- Ya esté seguro de que los datos cumplen las restricciones.
- Desea cargar datos que no cumplan las restricciones. Posteriormente, puede ejecutar consultas para modificar los datos y volver a habilitar las restricciones.

La deshabilitación de restricciones en una tabla no afecta a las restricciones de otras tablas que hagan referencia a la tabla original. Las actualizaciones de una tabla siguen pudiendo generar errores de infracción de restricciones.

NOTA: La habilitación de una restricción que ha estado deshabilitada requiere la ejecución de otra instrucción ALTER TABLE que contenga una cláusula CHECK o CHECK ALL.

SINTAXIS PARCIAL

ALTER TABLE tabla {CHECK | NOCHECK} CONSTRAINT {ALL | restricción[,...n]}

Este ejemplo deshabilita la restricción FK_Employees_Employees. Se puede volver a habilitar si se ejecuta otra instrucción ALTER TABLE con la cláusula CHECK.

EJEMPLO

USE northwind

ALTER TABLE dbo.Employees

NOCHECK

CONSTRAINT FK_Employees_Employees

Para determinar si una restricción está habilitada o deshabilitada en una tabla, ejecute el procedimiento almacenado del sistema sp_help o utilice la propiedad CnstsDisabled de la función OBJECTPROPERTY.



CAPITULO 3

- Creación y optimización de índices
- Diseño de índices.
- Optimización de índices.

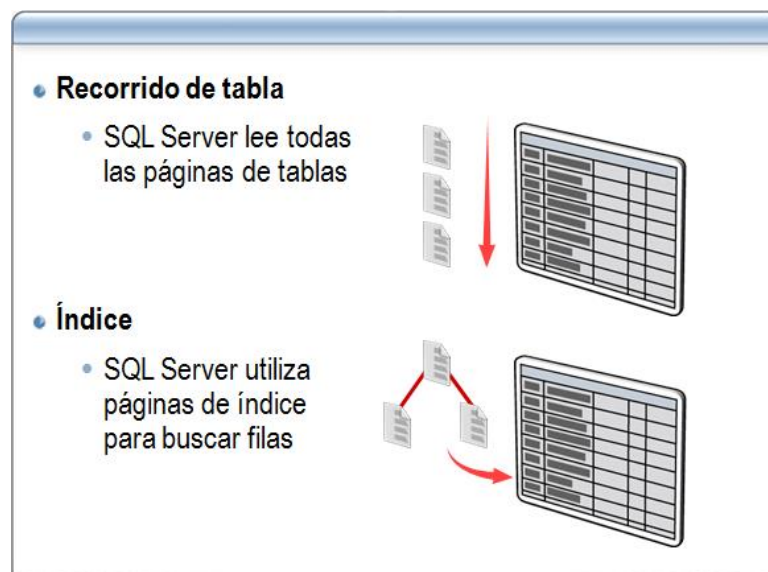
CREACIÓN DE ÍNDICES

Un índice es un conjunto de páginas asociado a una tabla (o a una vista) que se utiliza para acelerar la recuperación de filas de la tabla o para exigir unicidad. Por ejemplo, sin un índice tendría que recorrer todo un libro de página en página hasta encontrar información acerca de un tema. Microsoft® SQL Server™ 2014 utiliza índices para señalar la ubicación de una fila en una página de datos en lugar de tener que recorrer todas las páginas de datos de una tabla.

Un índice contiene claves formadas a partir de una o más columnas de la tabla. Estas claves se almacenan de tal forma que SQL Server 2014 puede encontrar las filas asociadas a los valores de clave rápida y eficazmente.

Diseño de Índices.

El diseño de índices útiles es uno de los aspectos más importantes de la mejora del rendimiento de las consultas. Requiere comprender tanto la estructura de los índices como la forma en que se utilizan los datos. Entender los fundamentos del almacenamiento y el acceso a los datos es el primer paso para comprender el funcionamiento de los índices, por qué son útiles y las razones por las que se utiliza cada una de las distintas opciones de indexado que ofrece SQL Server 2014.



¿Cómo tiene acceso SQL Server a los datos?

SQL Server tiene acceso a los datos de dos formas:

- Examinando todas las páginas de datos de una tabla, lo que se denomina recorrido de tabla. Cuando SQL Server realiza un recorrido de tabla:
 1. Empieza por el principio de la tabla.
 2. Avanza de página en página por todas las filas de la tabla.
 3. Extrae las filas que cumplen los criterios de la consulta.
- Utilizando índices. Cuando SQL Server utiliza un índice:

1. Recorre la estructura de árbol de índices para encontrar las filas solicitadas por la consulta.
2. Extrae únicamente las filas necesarias que cumplen los criterios
3. de la consulta.

En primer lugar, SQL Server averigua si existe un índice. Después, el optimizador de consultas (el componente responsable de generar el plan de ejecución óptimo de una consulta) determina qué es más eficiente para el acceso a los datos: recorrer una tabla o utilizar un índice.

QUÉ ES UN ÍNDICE AGRUPADO

Un índice agrupado ordena y almacena las filas de datos de la tabla según la clave del índice agrupado. El índice agrupado se implementa como un árbol b. Cada página de un árbol b se denomina un nodo de índice. El nodo superior del árbol b se denomina el nodo raíz. El nivel inferior de nodos del índice se llama nivel de hoja. Todos los niveles de índice que se encuentran entre el nodo raíz y los nodos de hoja se denominan colectivamente nodos intermedios. Cada página de los niveles intermedios o inferiores tiene un puntero a las páginas anterior y siguiente, formando una lista doblemente vinculada. Esta estructura proporciona un mecanismo muy eficiente para acelerar el proceso de búsqueda de datos.

En un índice agrupado, los nodos raíz e intermedios contienen páginas de índice que incluyen filas de índice. Cada fila de índice contiene un valor de clave y un puntero a una página de nivel intermedio del árbol b o a una fila de datos en el nivel de hoja del índice.

Las páginas de cada nivel del índice están vinculadas en una lista doblemente vinculada. Puesto que un índice agrupado determina el orden en que se almacenan realmente las filas de una tabla, cada tabla sólo puede tener un índice agrupado; las filas de una tabla no se pueden almacenar en más de un orden.

Importante Las columnas que tienen los tipos de datos siguientes no pueden utilizarse como clave de un índice agrupado: *ntext*, *text*, *varchar(max)*, *nvarchar(max)*, *varbinary(max)*, *xml* o *image*.

¿Cuándo se debe utilizar un índice agrupado ?

Puesto que sólo puede haber un índice agrupado por tabla, debe asegurarse de que lo utiliza para lograr las máximas ventajas posibles. Antes de crear un índice agrupado debe entender cómo se tendrá acceso a los datos. Como un índice agrupado determina el orden en el que SQL Server 2014 almacena las filas de datos de una tabla, los índices agrupados son más apropiados para determinados tipos de datos y patrones de uso.

Los índices agrupados son más eficaces cuando se utilizan en consultas que hacen lo siguiente:

- Devuelven un intervalo de valores utilizando operadores como **BETWEEN**, **>**, **>=**, **<** y **<=**. Como los datos de la tabla se almacenan físicamente según el orden del índice, cuando se encuentra la fila con el primer valor utilizando el índice agrupado, se sabe con total seguridad que las filas con valores indizados subsiguientes están adyacentes físicamente.

- Devuelven datos ordenados utilizando la cláusula ORDER BY o GROUP BY. Un índice en las columnas especificadas en la cláusula ORDER BY o GROUP BY puede eliminar la necesidad de que el motor de base de datos ordene los datos, puesto que las filas ya están ordenadas. Esto mejora el rendimiento de las consultas.
- Devuelven datos combinados mediante cláusulas JOIN; normalmente son columnas de clave externa.
- Devuelven conjuntos de resultados grandes.

¿Cuándo no se deben utilizar índices agrupados?

Los índices agrupados no son adecuados cuando:

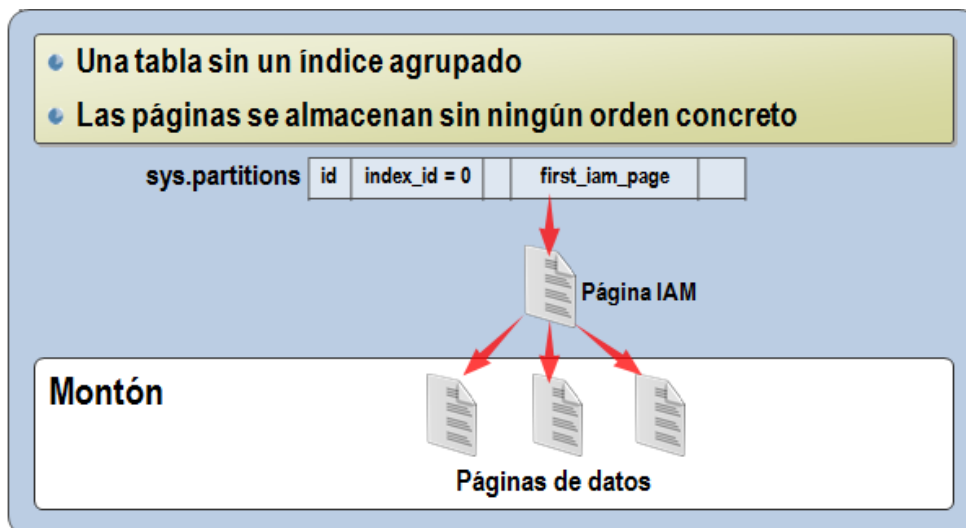
- Los datos de las columnas indizadas cambian con frecuencia. Los cambios en un índice agrupado hacen que haya que mover toda la fila de datos porque el motor de base de datos debe mantener los valores de datos de una fila ordenados físicamente. Ésta es una consideración importante en los sistemas de procesamiento con volúmenes elevados de transacciones donde los datos suelen ser volátiles.
- Las claves de índice son amplias. Las claves amplias son claves compuestas de varias columnas o varias columnas de gran tamaño. Todos los índices no agrupados utilizan los valores de clave del índice agrupado como claves de búsqueda. Los índices no agrupados definidos en la misma tabla serán mucho mayores porque las entradas de índices no agrupados contienen la clave de agrupación y también las columnas de clave definidas para ese índice no agrupado.

Qué es un montón

Un montón es una tabla sin un índice agrupado. Las filas de datos no se almacenan en ningún orden concreto y no hay ningún orden específico para la secuencia de las páginas de datos. Las páginas de datos no están vinculadas en una lista vinculada. SQL Server siempre mantiene las páginas de datos en un montón a menos que se defina un índice agrupado en la tabla.

SQL Server utiliza páginas Mapa de asignación de índices (IAM) para mantener los montones. Las páginas IAM:

- Contienen información acerca de dónde almacena SQL Server las extensiones de un montón. La tabla del sistema sys.partitions almacena un puntero a la primera página IAM asociada a un montón. Será un registro con index_id = 0.
- Permiten recorrer el montón para encontrar espacio disponible cuando se insertan nuevas filas en la tabla.
- Asocian páginas de datos a la tabla. Las páginas de datos y las filas que contienen no están en ningún orden concreto y no están vinculadas. La única asociación lógica entre las páginas de datos es lo que se registra en las páginas IAM.



Cuándo utilizar un montón

Utilizará un montón de forma predeterminada siempre que no defina un índice agrupado en una tabla. Debe utilizar un montón cuando la tabla contenga datos que están estructurados o que se utilizan de una forma que no sea adecuada para la implementación de un índice agrupado. Puede seguir implementando índices no agrupados en una tabla que utilice un montón.

Considere la posibilidad de utilizar un montón para las tablas que:

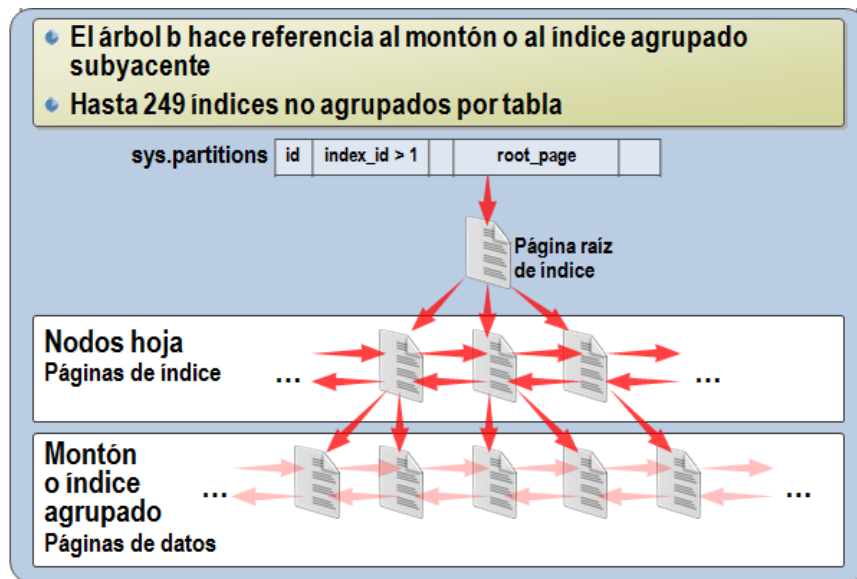
- Contengan datos volátiles donde se agregan, eliminan y actualizan filas con frecuencia. La sobrecarga que supone el mantenimiento de índices puede resultar más costosa que sus ventajas.
- Contengan pequeñas cantidades de datos. El uso de un recorrido de tabla para encontrar datos puede ser más rápido que el mantenimiento y el uso de un índice.
- Contengan principalmente filas de datos duplicados. Un recorrido de tabla puede ser más rápido que una búsqueda en un índice.
- Contengan datos que se escriben y raramente se leen, como un registro de auditoría. Un índice puede suponer una sobrecarga innecesaria de almacenamiento y mantenimiento.

Importante Cuando crea una restricción PRIMARY KEY en una tabla, se crea automáticamente un índice único en una columna (o en varias columnas). De forma predeterminada, este índice es agrupado, lo que significa que la tabla ya no se almacenará como un montón. Puede especificar un índice no agrupado cuando cree la restricción utilizando el argumento NONCLUSTERED.

QUÉ ES UN ÍNDICE NO AGRUPADO

Los índices no agrupados tienen la misma estructura de árbol b que los índices agrupados, excepto que las filas de datos de la tabla subyacente no se ordenan y se almacenan según sus claves no agrupadas. En el índice no agrupado, los datos y el

índice se almacenan por separado, y el nivel de hoja del índice consta de páginas de índice en lugar de páginas de datos.



Las filas del índice no agrupado se almacenan según el orden de los valores de la clave de índice, pero no se garantiza que las filas de datos a las que se hace referencia estén en un orden concreto a menos que se cree un índice agrupado en la tabla. Cada fila de índice del índice no agrupado contiene el valor de clave no agrupada y un localizador de fila. Este localizador señala a la fila de datos si la tabla es un montón y contiene la clave de índice agrupado para la fila si la tabla tiene un índice agrupado.

Si la tabla indizada tiene un índice agrupado, la columna o las columnas definidas en el índice agrupado se anexan automáticamente al final de cada índice no agrupado de la tabla. Esto puede producir una consulta cubierta sin especificar las columnas de índice agrupado en la definición del índice no agrupado. Por ejemplo, si una tabla tiene un índice agrupado en la columna C, un índice no agrupado en las columnas B y A tendrá como sus valores de clave las columnas B, A y C.

Una tabla puede tener hasta 249 índices no agrupados. Es posible definir índices no agrupados en una tabla independientemente de que la tabla utilice un índice agrupado o un montón.

Importante Las columnas que tienen los tipos de datos siguientes no pueden utilizarse como clave de un índice no agrupado: *ntext*, *text*, *varchar(max)*, *nvarchar(max)*, *varbinary(max)*, *xml* o *image*.

Cuándo se debe utilizar un índice no agrupado

Los índices no agrupados son útiles cuando los usuarios necesitan disponer de varias formas de buscar datos. Por ejemplo, puede que un usuario busque con frecuencia en una base de datos de jardinería los nombres comunes y científicos de las plantas.

Podría utilizar un índice no agrupado para recuperar los nombres científicos y un índice agrupado para recuperar los nombres comunes.

Los índices no agrupados están diseñados para mejorar el rendimiento de las consultas utilizadas con frecuencia que no están cubiertas por un índice agrupado. Si su tabla ya tiene un índice agrupado y necesita indizar otra columna, la única opción que tiene es utilizar un índice no agrupado. Antes de crear un índice no agrupado debe entender cómo se tendrá acceso a los datos.

El máximo rendimiento de las consultas se consigue cuando un índice contiene todas las columnas de una consulta. El término cobertura se refiere al número de columnas que participan en una consulta respaldadas por un índice. Con una cobertura total, el optimizador de consultas puede encontrar todos los valores de columnas dentro del índice, lo que significa que no se tiene acceso a los datos del montón o del índice agrupado y, por tanto, se realizan menos operaciones de E/S de disco. No obstante, las claves amplias y la existencia de demasiados índices pueden ser también perjudiciales para el rendimiento general de la base de datos.

Creación de índices

Una vez que haya decidido lo que desea indizar y si desea utilizar un índice agrupado o un índice no agrupado, debe crear el índice. Dispone de muchas opciones para crear un índice, y estas opciones pueden afectar considerablemente el rendimiento y la facilidad de mantenimiento del índice.

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
INDEX nombreDeÍndice ON { tabla | vista } ( columna [ ASC |
DESC ] [ ,...n ] )
INCLUDE ( columna [ ,...n ] )
[WITH option [ ,...n ] ]
[ON {esquemaDePartición (columna) | grupoDeArchivos |
"default" } ]
```

Opción WITH	Propósito
ALLOW_ROW_LOCKS	Habilita/deshabilita bloqueos de nivel de fila en el índice
ALLOW_PAGE_LOCKS	Habilita/deshabilita bloqueos de nivel de página en el índice
ONLINE	Habilita/deshabilita el acceso al índice durante la creación
FILLFACTOR	Controla el espacio libre en páginas de nivel de hoja
PAD_INDEX	Controla el espacio libre en páginas que no son de nivel de hoja

Advertencia No indice las columnas definidas mediante los tipos de datos ntext, text, varchar(max), nvarchar(max), varbinary(max), xml o image. Las columnas que tienen estos tipos de datos no se pueden indizar.

Para crear un índice mediante Transact-SQL, utilice la instrucción CREATE INDEX. La instrucción CREATE INDEX tiene la sintaxis siguiente.

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
INDEX nombreDeÍndice ON { tabla | vista } ( columna [ ASC | DESC ] [ ,...n ] )
```

```
INCLUDE ( columna [ ,...n ] )  
[WITH  
[PAD_INDEX = { ON | OFF }]  
[[,] FILLFACTOR = factorDeRelleno ]  
[[,] IGNORE_DUP_KEY = { ON | OFF }]  
[[,] ONLINE = { ON | OFF }]  
[[,] ALLOW_ROW_LOCKS = { ON | OFF }]  
[[,] ALLOW_PAGE_LOCKS = { ON | OFF }]]  
[ON {partition_scheme (columna) | grupoDeArchivos | "default" } ]
```

En el ejemplo de código siguiente se crea un índice no agrupado ascendente denominado `AK_Employee_LoginID` en la columna `LoginID` de la tabla `HumanResources.Employee` de la base de datos `AdventureWorks`.

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

Qué son los índices únicos

Un índice único es un índice que garantiza que todos los datos de una columna indizada son únicos y no contiene valores duplicados.

Cuando existe un índice único, el motor de base de datos comprueba si hay valores duplicados cada vez que se agregan datos utilizando una operación de inserción. Las operaciones de inserción que generarían valores de clave duplicados se deshacen y el motor de base de datos muestra un mensaje de error. Esto es cierto incluso aunque la operación de inserción cambie muchas filas y sólo produzca un duplicado. Sin embargo, si se intenta introducir datos para los que hay un índice único y la cláusula `IGNORE_DUP_KEY` se establece en `ON` en la instrucción `CREATE INDEX`, sólo producirán error las filas que infrinjan el índice único.

Cuando crea un índice en una tabla que ya contiene datos, el motor de base de datos se asegura de que no haya valores duplicados.

Debe crear un índice único para los índices agrupados o no agrupados cuando los datos propiamente dichos sean únicos de forma inherente. Cree índices únicos sólo en las columnas en las que se pueda exigir integridad de entidad. Por ejemplo, no debe crear un índice único en la columna `LastName` de la tabla `Person.Contact` de la base de datos `AdventureWorks`, ya que puede haber varios empleados con el mismo apellido. Si se debe exigir la unicidad, cree restricciones `PRIMARY KEY` o `UNIQUE` en la columna en lugar de crear un índice único.

Garantiza que no hay valores duplicados en la clave de índice

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

EmployeeID	LoginID	Gender	MaritalStatus	...
216	mike0	M	S	...
231	fukiko0	M	M	...
242	pat0	M	S	...
...				
291	pat0	F	S	...

Valor de clave duplicado no permitido

Puede crear un índice único en SQL Server Management Studio si activa la casilla de verificación Único en el cuadro de diálogo Nuevo índice al crear el índice.

En el ejemplo siguiente se muestra el código Transact-SQL necesario para crear un índice único no agrupado ascendente denominado AK_Employee_LoginID en la columna LoginID de la tabla HumanResources.Employee de la base de datos AdventureWorks.

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

Nota Si una tabla tiene una restricción PRIMARY KEY o UNIQUE, SQL Server crea automáticamente un índice único al ejecutar la instrucción CREATE TABLE o ALTER TABLE. De forma predeterminada será un índice agrupado, pero puede forzar que sea no agrupado si utiliza la opción NONCLUSTERED de la instrucción CREATE TABLE.

Consideraciones para crear índices con varias columnas

Un índice compuesto especifica más de una columna como valor de clave. El rendimiento de las consultas mejora si se utilizan índices compuestos, especialmente cuando los usuarios suelen buscar información de más de una forma distinta. Sin embargo, las claves amplias aumentan los requisitos de almacenamiento de un índice.

• Índices compuestos

- Incluya hasta 16 columnas y 900 bytes en la clave
- Defina primero la columna que sea más única

```
CREATE NONCLUSTERED INDEX K_Contact_LastName_FirstName  
ON Person.Contact ( LastName ASC, FirstName ASC)
```

• Columnas incluidas

- Columnas que no son claves incluidas en el índice
- Mejoran la “cobertura” de la consulta y por tanto el rendimiento

```
CREATE NONCLUSTERED INDEX AK_Employee_LoginID  
ON HumanResources.Employee ( LoginID ASC)  
INCLUDE ( ContactID, NationalIDNumber)
```

Los índices compuestos tienen los requisitos y las limitaciones siguientes:

- Es posible combinar hasta 16 columnas para formar un único índice compuesto.
- La suma de las longitudes de las columnas que forman el índice compuesto no puede superar los 900 bytes.
- La cláusula WHERE de una consulta debe hacer referencia a la primera columna del índice compuesto para que el optimizador de consultas utilice el índice compuesto. Los índices de cobertura no están sujetos a esta restricción.
- Todas las columnas de un índice compuesto deben proceder de la misma tabla, excepto cuando se crea un índice en una vista, en cuyo caso deben proceder todas de la misma vista.
- Un índice compuesto en (columna1, columna2) no es igual que un índice en (columna2, columna1); cada uno tiene un orden de columna distinto.

Nota Cuando un índice contiene todas las columnas a las que hace referencia una consulta, suele decirse que cubre la consulta. Las consultas cubiertas reducen las operaciones de E/S de disco y pueden mejorar considerablemente el rendimiento, ya que el optimizador de consultas puede obtener todos los resultados de las búsquedas de las páginas de índice, sin necesidad de obtener datos adicionales de las páginas de datos.

Uso de procedimientos almacenados del sistema para obtener información de índice

El procedimiento almacenado `sp_helpindex` devuelve detalles de los índices creados en una tabla especificada. La información devuelta para cada índice es el nombre de índice, su descripción y las claves. El procedimiento almacenado `sp_help` devuelve más detalles de una tabla, pero también incluye la misma información acerca de los índices que devuelve `sp_helpindex`.

En el ejemplo siguiente se muestra cómo obtener información de índice de la tabla `Production.Product` utilizando el procedimiento almacenado `sp_helpindex`.

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

En la tabla siguiente se muestran los resultados.

index_name	index_description	index_keys
AK_Product_Name	no agrupado, único, ubicado en PRIMARY	Name
AK_Product_ProductNumber	no agrupado, único, ubicado en PRIMARY	ProductNumber
AK_Product_rowguid	no agrupado, único, ubicado en PRIMARY	rowguid
PK_Product_ProductID	agrupado, único, clave principal ubicada en PRIMARY	ProductID



CAPITULO 4

- Consultas básicas: La sentencia SELECT,
- Precedencia de modificadores, campos calculados, funciones SQL.
- Mantenimiento de datos: Insert, Update y Delete.

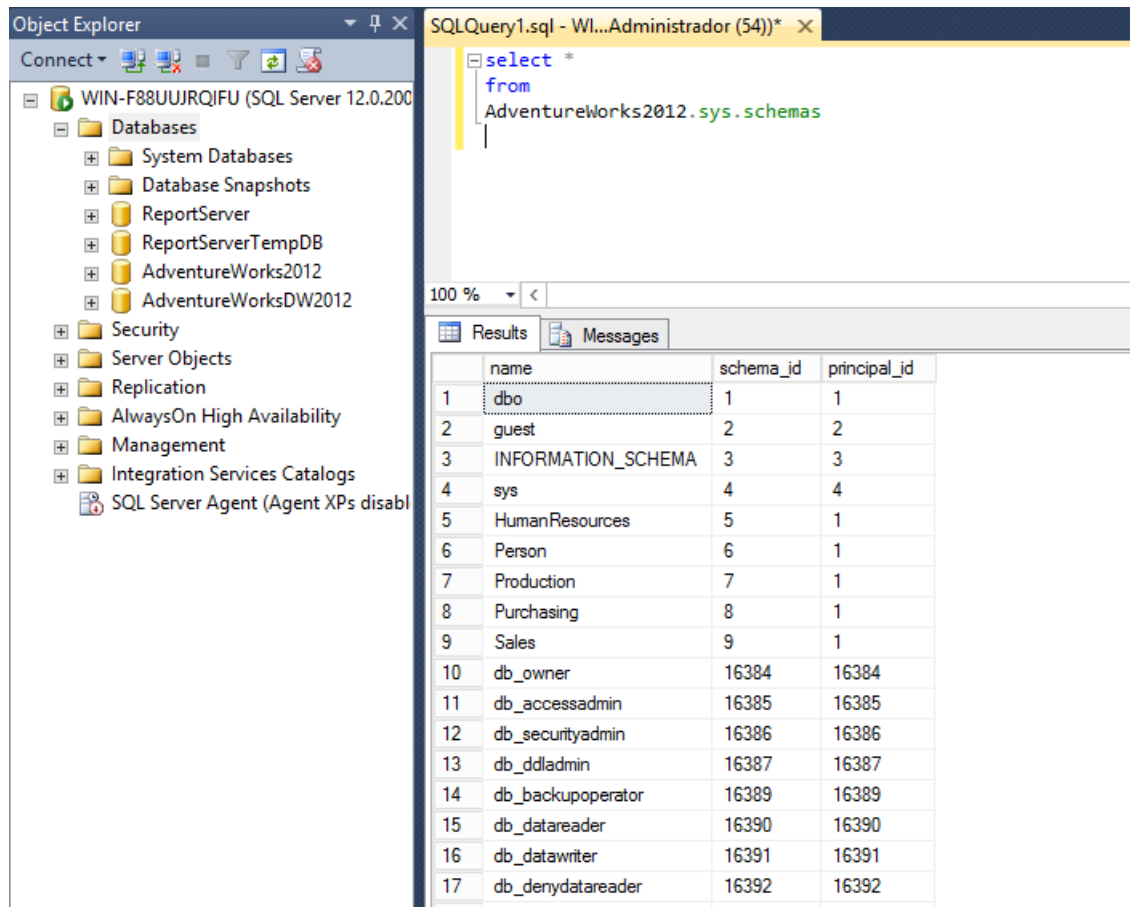
CONSULTAS BÁSICAS

La sentencia **SELECT**

La instrucción SELECT permite recuperar datos.

SINTAXIS PARCIAL

```
SELECT [ ALL | DISTINCT ] <listaSelección> FROM {<tablaOrigen>} [, ...n] [
WHERE <condiciónBúsqueda> ]
```



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' pane displays the 'Databases' folder expanded, showing 'AdventureWorks2012' and 'AdventureWorksDW2012'. The main window shows a SQL query in the 'SQLQuery1.sql' editor:

```
select *
from
AdventureWorks2012.sys.schemas
```

Below the query editor, the 'Results' pane displays the output of the query as a table with 17 rows and 4 columns: 'name', 'schema_id', and 'principal_id' (the fourth column is partially obscured). The data is as follows:

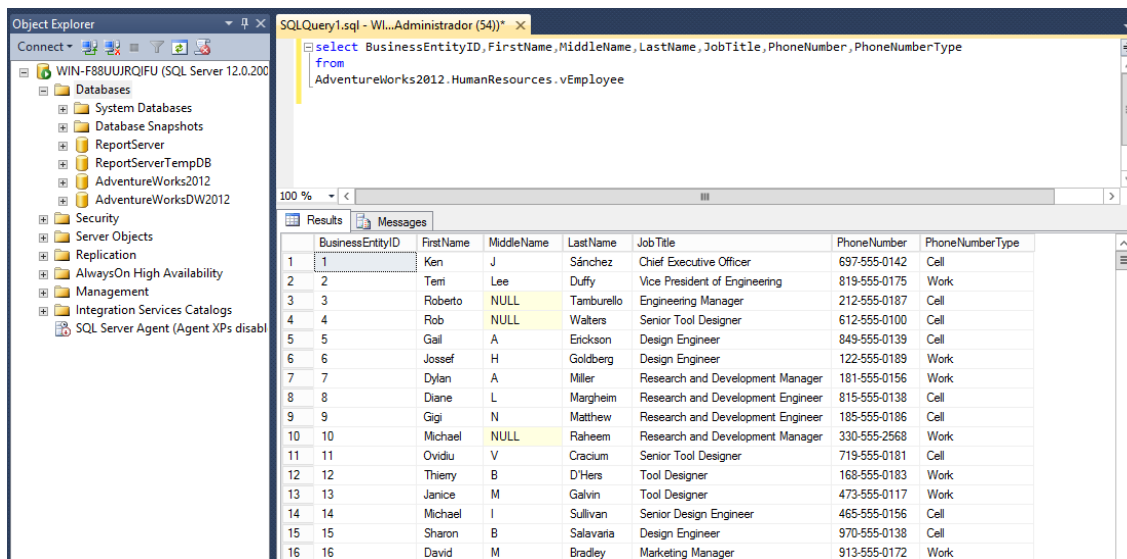
	name	schema_id	principal_id
1	dbo	1	1
2	guest	2	2
3	INFORMATION_SCHEMA	3	3
4	sys	4	4
5	HumanResources	5	1
6	Person	6	1
7	Production	7	1
8	Purchasing	8	1
9	Sales	9	1
10	db_owner	16384	16384
11	db_accessadmin	16385	16385
12	db_securityadmin	16386	16386
13	db_ddladmin	16387	16387
14	db_backupoperator	16389	16389
15	db_datareader	16390	16390
16	db_datawriter	16391	16391
17	db_denydatareader	16392	16392

Puede utilizar la instrucción SELECT para especificar las filas y columnas que desea obtener de una tabla:

- La lista de selección especifica las columnas que se deben obtener.
- La cláusula WHERE especifica las filas que se deben obtener. Cuando se utilizan condiciones de búsqueda en la cláusula WHERE, se puede restringir el número de filas devueltas mediante operadores de comparación, cadenas de caracteres y operadores lógicos como condiciones de búsqueda.
- La cláusula FROM especifica la tabla de la que se obtienen las filas y columnas.

Al especificar las columnas que desea obtener, tenga en cuenta los siguientes hechos e instrucciones:

- La lista de selección obtiene y muestra las columnas en el orden especificado.
- Separe los nombres de columna con comas, excepto después del último.
- Evite o reduzca al mínimo el uso del asterisco (*) en la lista de selección. El asterisco se emplea para obtener todas las columnas de una tabla.



Object Explorer: Connect to WIN-F88UJRIQIFU (SQL Server 12.0.2008) > Databases > AdventureWorks2012 > HumanResources > vEmployee

SQLQuery1.sql - Wl...Administrador (54))

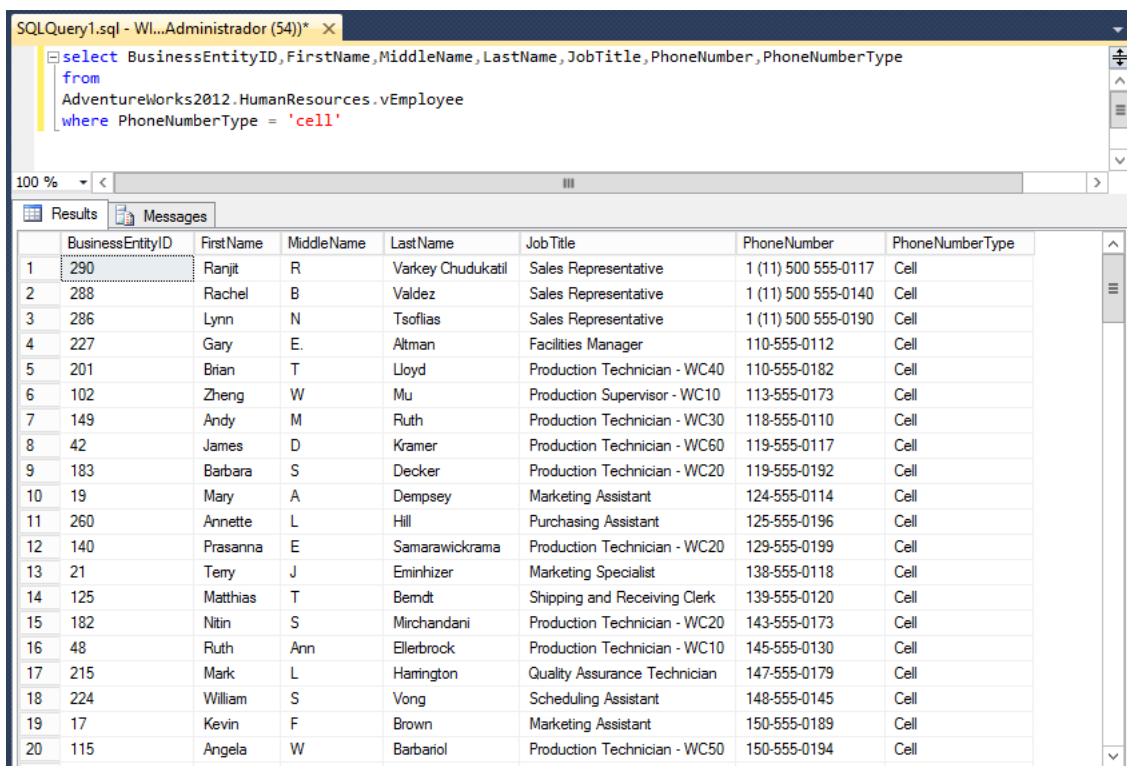
```
select BusinessEntityID, FirstName, MiddleName, LastName, JobTitle, PhoneNumber, PhoneNumberType
from AdventureWorks2012.HumanResources.vEmployee
```

Results

	BusinessEntityID	FirstName	MiddleName	LastName	JobTitle	PhoneNumber	PhoneNumberType
1	1	Ken	J	Sánchez	Chief Executive Officer	697-555-0142	Cell
2	2	Terri	Lee	Duffy	Vice President of Engineering	819-555-0175	Work
3	3	Roberto	NULL	Tamburello	Engineering Manager	212-555-0187	Cell
4	4	Rob	NULL	Walters	Senior Tool Designer	612-555-0100	Cell
5	5	Gail	A	Erickson	Design Engineer	849-555-0139	Cell
6	6	Joséph	H	Goldberg	Design Engineer	122-555-0189	Work
7	7	Dylan	A	Miller	Research and Development Manager	181-555-0156	Work
8	8	Diane	L	Margheim	Research and Development Engineer	815-555-0138	Cell
9	9	Gigi	N	Matthew	Research and Development Engineer	185-555-0186	Cell
10	10	Michael	NULL	Raheem	Research and Development Manager	330-555-2568	Work
11	11	Ovidiu	V	Craciun	Senior Tool Designer	719-555-0181	Cell
12	12	Thierry	B	D'Hers	Tool Designer	168-555-0183	Work
13	13	Janice	M	Galvin	Tool Designer	473-555-0117	Work
14	14	Michael	I	Sullivan	Senior Design Engineer	465-555-0156	Cell
15	15	Sharon	B	Salavaria	Design Engineer	970-555-0138	Cell
16	16	David	M	Bradley	Marketing Manager	913-555-0172	Work

LA CLÁUSULA WHERE

El uso de la cláusula WHERE permite obtener filas en función de las condiciones de búsqueda que se especifiquen. Las condiciones de búsqueda de la cláusula WHERE pueden contener una lista ilimitada de predicados.



SQLQuery1.sql - Wl...Administrador (54))

```
select BusinessEntityID, FirstName, MiddleName, LastName, JobTitle, PhoneNumber, PhoneNumberType
from AdventureWorks2012.HumanResources.vEmployee
where PhoneNumberType = 'cell'
```

Results

	BusinessEntityID	FirstName	MiddleName	LastName	JobTitle	PhoneNumber	PhoneNumberType
1	290	Ranjit	R	Varkey Chudukatil	Sales Representative	1 (11) 500 555-0117	Cell
2	288	Rachel	B	Valdez	Sales Representative	1 (11) 500 555-0140	Cell
3	286	Lynn	N	Tsofilas	Sales Representative	1 (11) 500 555-0190	Cell
4	227	Gary	E.	Altman	Facilities Manager	110-555-0112	Cell
5	201	Brian	T	Lloyd	Production Technician - WC40	110-555-0182	Cell
6	102	Zheng	W	Mu	Production Supervisor - WC10	113-555-0173	Cell
7	149	Andy	M	Ruth	Production Technician - WC30	118-555-0110	Cell
8	42	James	D	Kramer	Production Technician - WC60	119-555-0117	Cell
9	183	Barbara	S	Decker	Production Technician - WC20	119-555-0192	Cell
10	19	Mary	A	Dempsey	Marketing Assistant	124-555-0114	Cell
11	260	Annette	L	Hill	Purchasing Assistant	125-555-0196	Cell
12	140	Prasanna	E	Samarawickrama	Production Technician - WC20	129-555-0199	Cell
13	21	Terry	J	Eminhizer	Marketing Specialist	138-555-0118	Cell
14	125	Matthias	T	Bemdt	Shipping and Receiving Clerk	139-555-0120	Cell
15	182	Nitin	S	Mirchandani	Production Technician - WC20	143-555-0173	Cell
16	48	Ruth	Ann	Ellerbrock	Production Technician - WC10	145-555-0130	Cell
17	215	Mark	L	Harrington	Quality Assurance Technician	147-555-0179	Cell
18	224	William	S	Vong	Scheduling Assistant	148-555-0145	Cell
19	17	Kevin	F	Brown	Marketing Assistant	150-555-0189	Cell
20	115	Angela	W	Barbariol	Production Technician - WC50	150-555-0194	Cell

El marcador de posición predicado indica las expresiones que pueden incluirse en la cláusula WHERE. Un predicado puede contener las opciones siguientes:

<predicado> ::= { expresión { = | < > | > | >= | < | <= } expresión | expresiónCadena [NOT] LIKE expresiónCadena [ESCAPE 'carácterEscape'] | expresión [NOT] BETWEEN expresión AND expresión | expresión IS [NOT] NULL | CONTAINS ({columna | * }, '<contieneCondiciónBúsqueda>') | FREETEXT ({columna | * }, 'cadenaTextoLibre') | expresión [NOT] IN (subconsulta | expresión [, ...n]) | expresión { = | < > | > | >= | < | <= } {ALL | SOME | ANY} (subconsulta) | EXISTS (subconsulta) }

Al especificar filas con la cláusula WHERE, tenga en cuenta los siguientes hechos e instrucciones:

- Utilice comillas simples para todos los datos de tipo char, nchar, varchar, nvarchar, text, datetime y smalldatetime.
- Utilice una cláusula WHERE para limitar el número de filas que se devuelven al utilizar la instrucción SELECT.

A veces se desea limitar el conjunto de resultados que devuelve una consulta. Para limitar los resultados se puede especificar condiciones de búsqueda en una cláusula WHERE con las que filtrar los datos. No hay límite en el número de condiciones de búsqueda que se pueden incluir en una instrucción SELECT. La tabla siguiente describe el tipo de filtro y la correspondiente condición de búsqueda que se puede usar para filtrar los datos.

Tipo de filtro	Condición de búsqueda
Operadores de comparación	=, >, <, >=, <= y <>
Comparaciones de cadenas	LIKE y NOT LIKE
Operadores lógicos: combinación de condiciones	AND, OR
Operador lógico: negación	NOT
Intervalo de valores	BETWEEN y NOT BETWEEN
Listas de valores	IN y NOT IN
Valores desconocidos	IS NULL e IS NOT NULL

OPERADORES DE COMPARACIÓN

Los operadores de comparación permiten comparar los valores de una tabla con un valor o expresión especificados. También puede utilizarlos para comprobar si se cumple una condición. Los operadores de comparación comparan columnas o variables de tipos de datos compatibles.

SQLQuery1.sql - W...HVHL\win7pro (51))

```
USE ADVENTUREWORKS2012
GO

SELECT * FROM HumanResources.Employee
WHERE Gender='M'
```

100 %

Resultados Mensajes

	BusinessEntityID	NationalIDNumber	LoginID	OrganizationNode	OrganizationLevel	Job Title	BirthDate	MaritalStatus	Gender	HireDate
1	1	295847284	adventure-works\ken0	0x	0	Chief Executive Officer	1963-03-02	S	M	2003-02-15
2	3	509647174	adventure-works\roberto0	0x5AC0	2	Engineering Manager	1968-12-13	M	M	2001-12-12
3	4	112457891	adventure-works\rob0	0x5AD6	3	Senior Tool Designer	1969-01-23	S	M	2002-01-05
4	6	998320692	adventure-works\jossef0	0x5ADE	3	Design Engineer	1953-04-11	M	M	2002-02-24
5	7	134969118	adventure-works\dylan0	0x5AE1	3	Research and Development Manager	1981-03-27	M	M	2003-03-12
6	10	879342154	adventure-works\michael6	0x5AE178	4	Research and Development Manager	1979-01-01	M	M	2003-06-04
7	11	974026903	adventure-works\ovidiu0	0x5AE3	3	Senior Tool Designer	1972-02-18	S	M	2005-01-05
8	12	480168528	adventure-works\thieny0	0x5AE358	4	Tool Designer	1953-08-29	M	M	2002-01-11
9	14	42487730	adventure-works\michael8	0x5AE5	3	Senior Design Engineer	1973-07-17	S	M	2005-01-30
10	16	24756624	adventure-works\dauid0	0x68	1	Marketing Manager	1969-04-19	S	M	2002-01-20
11	17	253022876	adventure-works\kevin0	0x6AC0	2	Marketing Assistant	1981-06-03	S	M	2001-02-26
12	18	222969461	adventure-works\john5	0x6B40	2	Marketing Specialist	1972-04-06	S	M	2005-03-10
13	21	243322160	adventure-works\terry0	0x6C60	2	Marketing Specialist	1980-03-07	M	M	2003-04-03
14	22	95958330	adventure-works\arinya0	0x6CA0	2	Marketing Specialist	1981-06-21	S	M	2003-01-13
15	25	519899904	adventure-works\james1	0x78	1	Vice President of Production	1977-02-07	S	M	2003-03-07
16	26	277173473	adventure-works\staney0	0x7AC0	2	Production Control Manager	1976-12-04	M	M	2003-01-07

Consulta ejecutada correctamente. WIN-BVUQ9LUHVHL (11.0 RTM) WIN-BVUQ9LUHVHL\win7pr... AdventureWorks2012 00:00:00 206 filas

USO DE COMPARACIONES DE CADENAS

Puede utilizar la condición de búsqueda LIKE en combinación con caracteres comodín para seleccionar filas por comparación entre cadenas de caracteres. Al utilizar la condición de búsqueda LIKE, tenga en cuenta los hechos siguientes:

- Todos los caracteres de la cadena modelo son significativos, incluidos los espacios en blanco al comienzo o al final.
- LIKE sólo puede utilizarse con datos de tipo char, nchar, varchar, nvarchar o datetime.

SQLQuery1.sql - W...HVHL\win7pro (51))

```
SELECT NationalIDNumber, LoginID, JobTitle, BirthDate, MaritalStatus
FROM HumanResources.Employee
WHERE JobTitle LIKE '%DESIGN%'
```

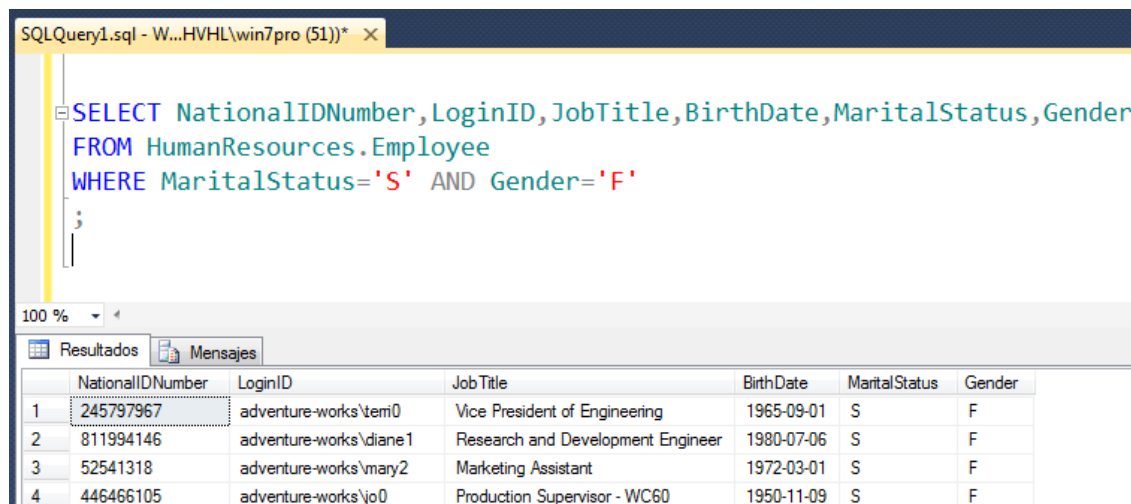
100 %

Resultados Mensajes

	NationalIDNumber	LoginID	Job Title	BirthDate	MaritalStatus
1	112457891	adventure-works\rob0	Senior Tool Designer	1969-01-23	S
2	695256908	adventure-works\gail0	Design Engineer	1946-10-29	M
3	998320692	adventure-works\jossef0	Design Engineer	1953-04-11	M
4	974026903	adventure-works\ovidiu0	Senior Tool Designer	1972-02-18	S
5	480168528	adventure-works\thieny0	Tool Designer	1953-08-29	M
6	486228782	adventure-works\janice0	Tool Designer	1983-06-29	M
7	42487730	adventure-works\michael8	Senior Design Engineer	1973-07-17	S
8	56920285	adventure-works\sharon0	Design Engineer	1955-06-03	M

USO DE OPERADORES LÓGICOS

Puede utilizar los operadores lógicos AND, OR y NOT para combinar un conjunto de expresiones y afinar el proceso de las consultas. Los resultados de una consulta pueden variar según el agrupamiento de las expresiones y el orden de las condiciones de búsqueda.



The screenshot shows a SQL query window with the following query:

```
SELECT NationalIDNumber, LoginID, JobTitle, BirthDate, MaritalStatus, Gender
FROM HumanResources.Employee
WHERE MaritalStatus='S' AND Gender='F'
```

Below the query, the results are displayed in a table with the following columns: NationalIDNumber, LoginID, JobTitle, BirthDate, MaritalStatus, and Gender. The results show four rows of data.

	NationalIDNumber	LoginID	JobTitle	BirthDate	MaritalStatus	Gender
1	245797967	adventure-works\tem0	Vice President of Engineering	1965-09-01	S	F
2	811994146	adventure-works\diane1	Research and Development Engineer	1980-07-06	S	F
3	52541318	adventure-works\mary2	Marketing Assistant	1972-03-01	S	F
4	446466105	adventure-works\jo0	Production Supervisor - WC60	1950-11-09	S	F

Al utilizar operadores lógicos, tenga en cuenta las instrucciones siguientes:

- Utilice el operador AND para obtener las filas que cumplan todos los criterios de búsqueda.
- Utilice el operador OR para obtener las filas que cumplan alguno de los criterios de búsqueda.
- Utilice el operador NOT para negar la expresión especificada a continuación.

USO DE LOS PARÉNTESIS

Utilice paréntesis cuando especifique dos o más expresiones como criterio de búsqueda.

El uso de paréntesis permite:

- Agrupar expresiones.
- Cambiar el orden de evaluación.
- Hacer más legibles las expresiones.

OBTENCIÓN DE UN INTERVALO DE VALORES

Puede utilizar la condición de búsqueda BETWEEN en la cláusula WHERE para obtener las filas contenidas en un intervalo de valores específico. Al utilizar la condición de búsqueda BETWEEN, tenga en cuenta los siguientes hechos e instrucciones:

- SQL Server incluye los valores inicial y final en el conjunto de resultados.
- Es preferible el uso de BETWEEN al de una expresión que incluya el operador AND y dos operadores de comparación ($> x$ AND $< = y$). Sin embargo, para buscar en un intervalo exclusivo en el que las filas devueltas no contengan los valores inicial y final, debe utilizar una expresión de este tipo ($> x$ AND $< y$).
- Puede utilizar la condición de búsqueda NOT BETWEEN para obtener las filas que estén fuera del intervalo especificado. Tenga en cuenta que el uso de condiciones NOT puede ralentizar la recuperación de los datos.

SQLQuery1.sql - W...HVHL\win7pro (51))

```
USE ADVENTUREWORKS2012
GO

SELECT * FROM HumanResources.Employee
WHERE BusinessEntityID BETWEEN 7 AND 13
```

100 %

Resultados Mensajes

	BusinessEntityID	NationalIDNumber	LoginID	OrganizationNode	OrganizationLevel	JobTitle	BirthDate	MaritalStatus	Gender	HireDate
1	7	134969118	adventure-works\dylan0	0x5AE1	3	Research and Development Manager	1981-03-27	M	M	2003-03-12
2	8	811994146	adventure-works\diane1	0x5AE158	4	Research and Development Engineer	1980-07-06	S	F	2003-01-30
3	9	658797903	adventure-works\gigi0	0x5AE168	4	Research and Development Engineer	1973-02-21	M	F	2003-02-17
4	10	879342154	adventure-works\vmichael6	0x5AE178	4	Research and Development Manager	1979-01-01	M	M	2003-06-04
5	11	974026903	adventure-works\ovidiu0	0x5AE3	3	Senior Tool Designer	1972-02-18	S	M	2005-01-05
6	12	480168528	adventure-works\thierry0	0x5AE358	4	Tool Designer	1953-08-29	M	M	2002-01-11
7	13	486228782	adventure-works\janice0	0x5AE368	4	Tool Designer	1983-06-29	M	F	2005-01-23

USO DE UNA LISTA DE VALORES COMO CRITERIO DE BÚSQUEDA

Puede utilizar la condición de búsqueda IN en la cláusula WHERE para obtener las filas que coincidan con una lista de valores especificada. Al utilizar la condición de búsqueda IN, tenga en cuenta las instrucciones siguientes:

SQLQuery1.sql - W...HVHL\win7pro (51))

```
SELECT BusinessEntityID,FirstName,LastName,AddressLine1,City,StateProvinceName,CountryRegionName
FROM Sales.vIndividualCustomer
WHERE CountryRegionName IN ('FRANCE','GERMANY')
```

100 %

Resultados Mensajes

	BusinessEntityID	FirstName	LastName	AddressLine1	City	StateProvinceName	CountryRegionName
1	10312	Allison	Adams	112, rue Faubourg St Antoine	Roubaix	Nord	France
2	4503	Dalton	Adams	3738, chaussée de Tournai	Bobigny	Seine Saint Denis	France
3	4747	Ian	Adams	7963 Elk Dr	Versailles	Yveline	France
4	10295	Jenna	Adams	3403bis, boulevard Saint Germain	Villeneuve-d'Ascq	Nord	France
5	16862	Luis	Adams	Moritzstr 45	Hof	Bayern	Germany
6	16884	Luke	Adams	Hellweg 4754	Paderborn	Nordrhein-Westfalen	Germany
7	10259	Morgan	Adams	810, rue des Rosiers	Saint-Denis	Seine Saint Denis	France
8	12041	Alaha	Alan	8128, rue Lamarck	Paris	Seine (Paris)	France
9	6184	Cheryl	Alan	4680, rue Villedo	Croix	Nord	France
10	10543	Kari	Alan	Zimmerstr 371	Berlin	Hessen	Germany
11	3107	Kelvin	Alan	Alte Landstr 9	Duesseldorf	Hessen	Germany
12	15970	Brian	Albrecht	Heidenweg 4624	Saarbrücken	Saarland	Germany
13	13220	Antonio	Alexander	Parise Straße 155	Darmstadt	Hessen	Germany
14	19672	Grace	Alexander	Höhenstr 2462	Werne	Nordrhein-Westfalen	Germany
15	5354	Jackson	Alexander	Carlsplatz 90	Darmstadt	Hessen	Germany
16	3288	Maria	Alexander	161, rue de Cambrai	Dunkerque	Nord	France

Consulta ejecutada correctamente. WIN-BVUQ9LUHVHL (11.0 RTM) | WIN-BVUQ9LUHVHL\win7pr... | AdventureWorks2012 | 00:00:00 | 3590 filas

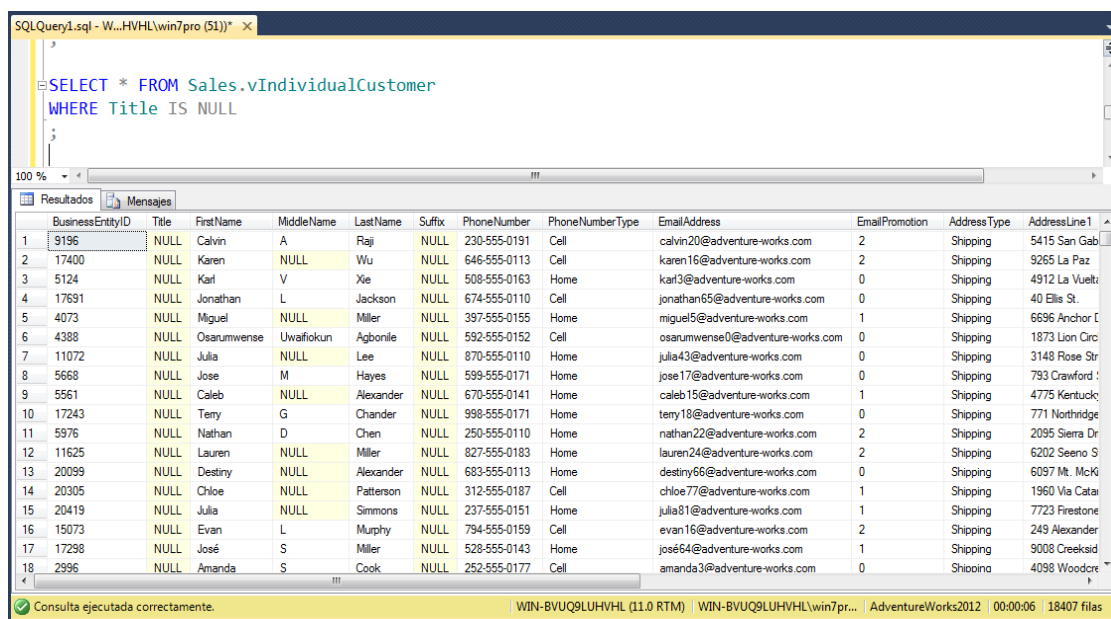
Puede utilizar la condición de búsqueda IN o un conjunto de expresiones de comparación conectadas con el operador OR. SQL Server las resuelve de la misma forma y los conjuntos de resultados devueltos son idénticos.

- No incluya un valor NULL en la condición de búsqueda. Un valor NULL en la lista de condiciones de búsqueda da como resultado la expresión = NULL. Puede producir conjuntos de resultados inesperados.

- Utilice la condición NOT IN para obtener las filas cuyos valores no se encuentren en la lista especificada. Tenga en cuenta que el uso de condiciones NOT puede hacer que los datos se recuperen más lentamente.

OBTENCIÓN DE VALORES DESCONOCIDOS

Una columna tiene un valor NULL cuando no se ha especificado ningún valor para ella durante la entrada de datos y no tiene definido un valor predeterminado. Un valor NULL no es lo mismo que cero (que es un valor numérico) o blanco (que es un valor de carácter).



SQLQuery1.sql - W...HVHL\win7pro (51)*

```
SELECT * FROM Sales.vIndividualCustomer
WHERE Title IS NULL
```

	BusinessEntityID	Title	FirstName	MiddleName	LastName	Suffix	PhoneNumber	PhoneNumberType	EmailAddress	EmailPromotion	AddressType	AddressLine1
1	9196	NULL	Calvin	A	Raji	NULL	230-555-0191	Cell	calvin20@adventure-works.com	2	Shipping	5415 San Gab
2	17400	NULL	Karen	NULL	Wu	NULL	646-555-0113	Cell	karen16@adventure-works.com	2	Shipping	9265 La Paz
3	5124	NULL	Karl	V	Xie	NULL	508-555-0163	Home	karl3@adventure-works.com	0	Shipping	4912 La Vuel
4	17691	NULL	Jonathan	L	Jackson	NULL	674-555-0110	Cell	jonathan65@adventure-works.com	0	Shipping	40 Ella St
5	4073	NULL	Miguel	NULL	Miller	NULL	397-555-0155	Home	miguel5@adventure-works.com	1	Shipping	6696 Anchor
6	4388	NULL	Osarumwense	Uwafokun	Agbonile	NULL	592-555-0152	Cell	osarumwense0@adventure-works.com	0	Shipping	1873 Lion Circ
7	11072	NULL	Julia	NULL	Lee	NULL	870-555-0110	Home	julia43@adventure-works.com	0	Shipping	3148 Rose Str
8	5668	NULL	Jose	M	Hayes	NULL	599-555-0171	Home	jose17@adventure-works.com	0	Shipping	793 Crawford
9	5561	NULL	Caleb	NULL	Alexander	NULL	670-555-0141	Home	caleb15@adventure-works.com	1	Shipping	4775 Kentuck
10	17243	NULL	Terry	G	Chander	NULL	998-555-0171	Home	terry18@adventure-works.com	0	Shipping	771 Northridge
11	5976	NULL	Nathan	D	Chen	NULL	250-555-0110	Home	nathan22@adventure-works.com	2	Shipping	2095 Sierra Dr
12	11625	NULL	Lauren	NULL	Miller	NULL	827-555-0183	Home	lauren24@adventure-works.com	2	Shipping	6202 Seeno S
13	20099	NULL	Destiny	NULL	Alexander	NULL	683-555-0113	Home	destiny66@adventure-works.com	0	Shipping	6097 Mt. McKi
14	20305	NULL	Chloe	NULL	Patterson	NULL	312-555-0187	Cell	chloe77@adventure-works.com	1	Shipping	1960 Via Catai
15	20419	NULL	Julia	NULL	Simmons	NULL	237-555-0151	Home	julia81@adventure-works.com	1	Shipping	7723 Firestone
16	15073	NULL	Evan	L	Murphy	NULL	794-555-0159	Cell	evan16@adventure-works.com	2	Shipping	249 Alexander
17	17298	NULL	José	S	Miller	NULL	528-555-0143	Home	jose64@adventure-works.com	1	Shipping	9008 Creeksid
18	2996	NULL	Amanda	S	Cook	NULL	252-555-0177	Cell	amanda3@adventure-works.com	0	Shipping	4098 Woodcote

Consulta ejecutada correctamente. WIN-BVUQ9LUHVHL (11.0 RTM) WIN-BVUQ9LUHVHL\win7pr... AdventureWorks2012 00:00:06 18407 filas

Puede utilizar la condición de búsqueda IS NULL para obtener las filas en las que falte información en una columna específica. Al obtener las filas con valores desconocidos, tenga en cuenta los siguientes hechos e instrucciones:

- Los valores NULL causan errores en todas las comparaciones porque no se evalúan como iguales entre sí.
- En la instrucción CREATE TABLE se define si las columnas permiten valores NULL.
- Puede utilizar la condición de búsqueda IS NOT NULL para obtener las filas con valores conocidos en las columnas especificadas.

ORDENACIÓN DE LOS DATOS

Puede utilizar la cláusula ORDER BY para ordenar las filas del conjunto de resultados de forma ascendente (ASC) o descendente (DESC).

SQLQuery1.sql - W...HVHL\win7pro (51) -

```

SELECT BusinessEntityID,FirstName,LastName,AddressLine1,City,StateProvinceName,CountryRegionName
FROM Sales.vIndividualCustomer
ORDER BY CountryRegionName

```

100 %

Resultados Mensajes

	BusinessEntityID	FirstName	LastName	AddressLine1	City	StateProvinceName	CountryRegionName
1	10292	Amber	Adams	9720 Morning Glory Dr.	Brisbane	Queensland	Australia
2	4970	Devin	Adams	8590 High Maple Court	Cranbourne	Victoria	Australia
3	10261	Isabella	Adams	1933 Rock Creek Pl.	Townsville	Queensland	Australia
4	10309	Savannah	Adams	8825 Walters Way	Sunbury	Victoria	Australia
5	10265	Sydney	Adams	8128 Kane Circle	Hervey Bay	Queensland	Australia
6	16003	Bob	Alan	2371 Deerfield Dr.	South Melbourne	Victoria	Australia
7	14644	Jamie	Alan	5492 Hacienda Drive	Townsville	Queensland	Australia
8	18784	Meghan	Alan	9730 Krueger Drive	Lavender Bay	New South Wales	Australia
9	20216	Alyssa	Alexander	4654 Blackridge Drive	Cranbourne	Victoria	Australia
10	6266	Austin	Alexander	5953 Pinewood Court	Warrnambool	Victoria	Australia
11	6675	David	Alexander	7334 Sterling Hill	Hobart	Tasmania	Australia
12	12690	Elizabeth	Alexander	1183 Tono Lane	Townsville	Queensland	Australia
13	5861	Gabriel	Alexander	1247 Violet Ct.	Geelong	Victoria	Australia
14	3211	Hailey	Alexander	2614 Park Glen Ct.	Hobart	Tasmania	Australia
15	12208	Gail	Alexander	918 Park Lane	Melton	Victoria	Australia
16	19371	Olivia	Alexander	4223 Las Trampas Road	Melbourne	Victoria	Australia
17	10915	Alexandra	Allen	1053 Rain Drop Circle	Warrnambool	Victoria	Australia
18	4955	Devin	Allen	7026 Trail Way	Wollongong	New South Wales	Australia
19	10923	Hailey	Allen	493 Tennyson Lane	Caloundra	Queensland	Australia

Consulta ejecutada correctamente. WIN-BVUQ9LUHVHL (11.0 RTM) WIN-BVUQ9LUHVHL\win7pr... AdventureWorks2012 00:00:01 18508 filas

Cuando utilice la cláusula ORDER BY, tenga en cuenta los siguientes hechos e instrucciones:

- El tipo de orden se especifica cuando se instala SQL Server. Puede ejecutar el procedimiento almacenado del sistema sp_helpsort para determinar el tipo de orden definido para la base de datos durante la instalación.
- SQL Server no garantiza ningún orden en el conjunto de resultados a menos que se especifique uno con la cláusula ORDER BY.
- De forma predeterminada, SQL Server ordena los datos de forma ascendente.
- No es necesario que las columnas incluidas en la cláusula ORDER BY aparezcan en la lista de selección.
- Las columnas especificadas en la cláusula ORDER BY no pueden tener más de 8060 bytes.
- Es posible ordenar por nombres de columna, valores calculados o expresiones.
- En la cláusula ORDER BY, puede hacer referencia a las columnas por su posición en la lista de selección. Las columnas se evalúan de la misma forma y el conjunto de resultados es el mismo.
- La cláusula ORDER BY no debe utilizarse para columnas de tipo text o image.

SQLQuery1.sql - W...HVHL\win7pro (51)*

```

SELECT BusinessEntityID,FirstName,LastName,AddressLine1,City,StateProvinceName,CountryRegionName
FROM Sales.vIndividualCustomer
ORDER BY CountryRegionName DESC
;

```

100 %

Resultados Mensajes

	BusinessEntityID	FirstName	LastName	AddressLine1	City	StateProvinceName	CountryRegionName
1	16867	Aaron	Adams	4116 Stanbridge Ct.	Downey	California	United States
2	16901	Adam	Adams	9381 Bayside Way	Newport Beach	California	United States
3	16724	Alex	Adams	237 Bellwood Dr.	Lake Oswego	Oregon	United States
4	10263	Alexandra	Adams	3215 Polson Court	Burlingame	California	United States
5	10314	Andrea	Adams	4767 Detroit Ave.	West Covina	California	United States
6	16699	Angel	Adams	9556 Lyman Rd.	Burlingame	California	United States
7	10299	Bailey	Adams	1817 Adobe Drive	Kirkland	Washington	United States
8	1770	Ben	Adams	1534 Land Ave	Bremerton	Washington	United States
9	4194	Blake	Adams	6055 Broadway Street	Beverly Hills	California	United States
10	10274	Amanda	Adams	6730 Saddlehill Lane	Fremont	California	United States
11	4891	Charles	Adams	1258 Steven Way	Tacoma	Washington	United States
12	10251	Chloe	Adams	3001 N. 48th Street	Marysville	Washington	United States
13	5055	Eduardo	Adams	4283 Meaham Drive	San Diego	California	United States
14	3731	Edward	Adams	9108 San Miguel Drive	Berkeley	California	United States
15	16858	Elijah	Adams	6385 Mark Twain	Seattle	Washington	United States
16	16902	Eric	Adams	2840 Oak Wood Court	Redwood City	California	United States
17	16730	Evan	Adams	4428 Jones Rd.	Everett	Washington	United States
18	3889	Fernando	Adams	4787 R St.	Fremont	California	United States
19	10302	Gabriella	Adams	9426 Georgia Street	Kirkland	Washington	United States

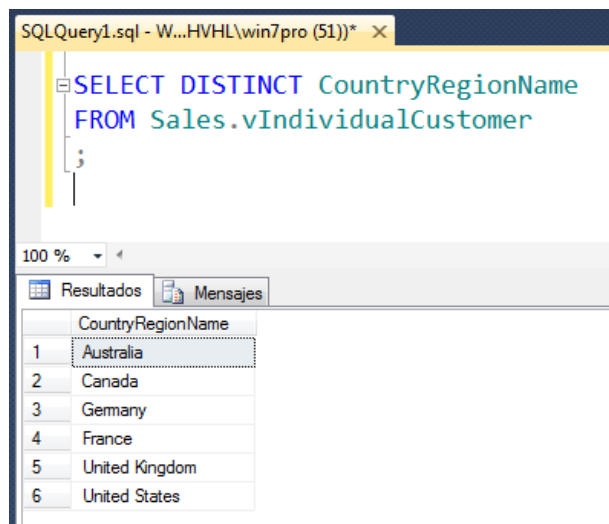
Consulta ejecutada correctamente. WIN-BVUQ9LUHVHL (11.0 RTM) WIN-BVUQ9LUHVHL\win7pr... AdventureWorks2012 00:00:01 18508 filas

NOTA: El uso de índices adecuados puede hacer más eficientes las ordenaciones con ORDER BY.

ELIMINACIÓN DE FILAS DUPLICADAS

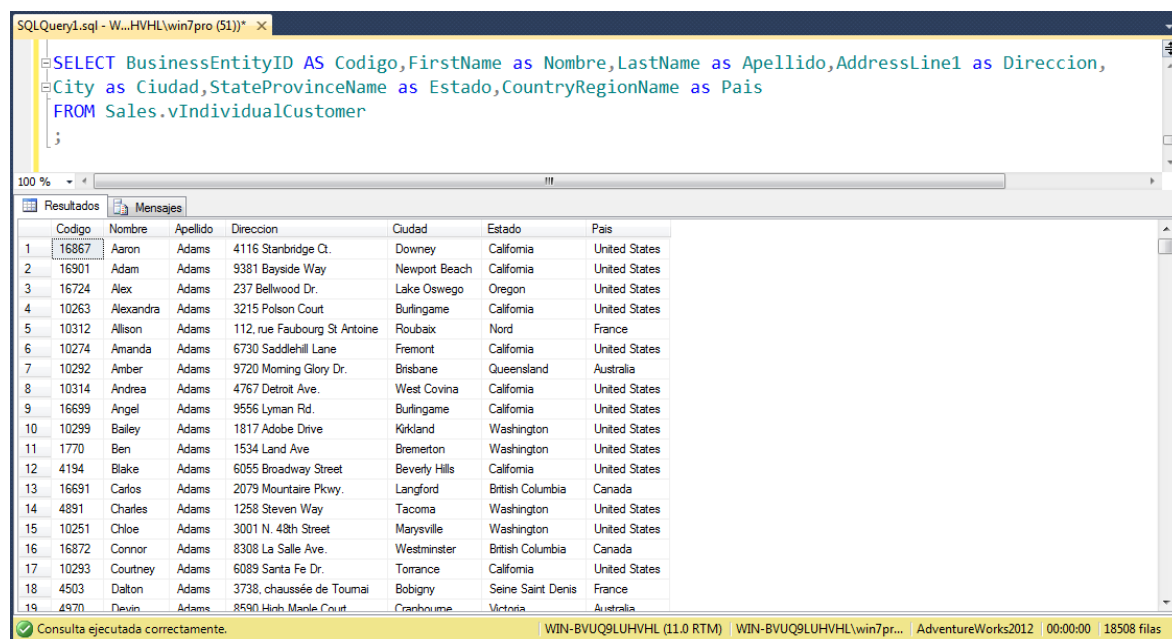
Si necesita obtener una lista de valores únicos, puede utilizar la cláusula DISTINCT para eliminar las filas duplicadas del conjunto de resultados. Al utilizar la cláusula DISTINCT, tenga en cuenta los hechos siguientes:

- El conjunto de resultados incluirá todas las filas que cumplan la condición de búsqueda de la instrucción SELECT, a menos que se especifique una cláusula DISTINCT.
- La combinación de valores de la lista de selección determina si las filas están duplicadas o no.
- Las filas que contengan una combinación única de valores se incluyen en el conjunto de resultados.
- La cláusula DISTINCT utiliza un orden aleatorio en el conjunto de resultados, a menos que se especifique también la cláusula ORDER BY.
- Si se especifica una cláusula DISTINCT, la cláusula ORDER BY debe incluir columnas que aparezcan en el conjunto de resultados.



CAMBIO DEL NOMBRE DE LAS COLUMNAS

Si desea crear nombres de columnas más legibles, puede utilizar la palabra clave AS para reemplazar los nombres predeterminados por alias en la lista de selección.



SINTAXIS PARCIAL

SELECT nombreColumna | expresión AS títuloColumna FROM nombreTabla

Al cambiar los nombres de las columnas, tenga en cuenta los siguientes hechos e instrucciones:

- De forma predeterminada, en el conjunto de resultados se muestran los nombres de columna designados en la instrucción CREATE TABLE.
- Utilice comillas simples para los nombres de columna que contengan espacios o que no se ajusten a las convenciones de denominación de objetos de SQL Server.

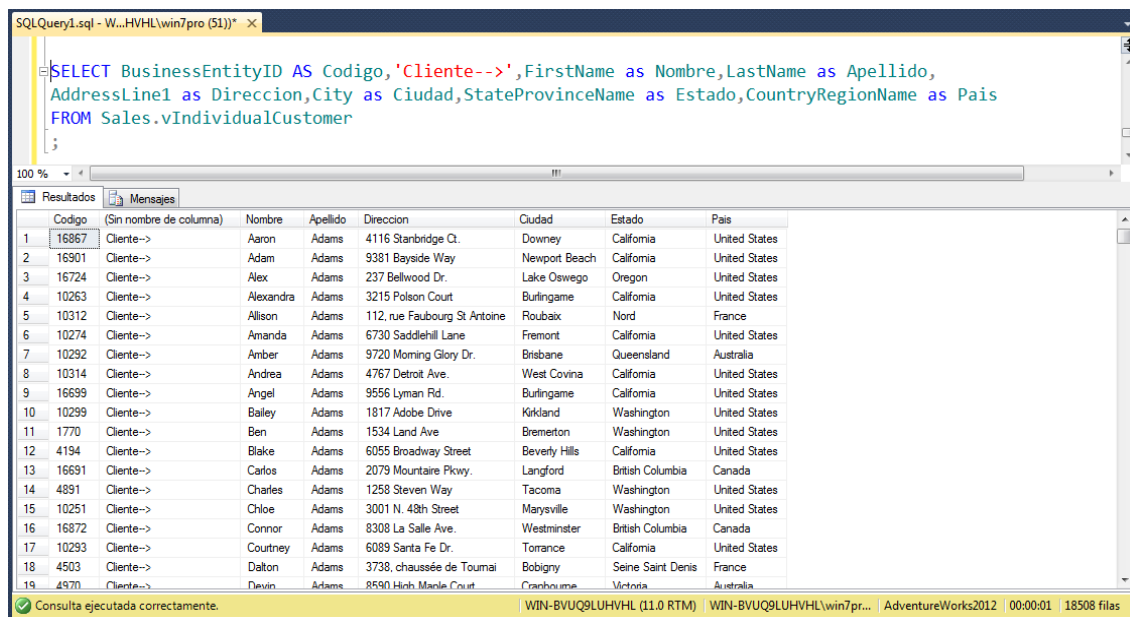
- Puede crear alias de columna para columnas calculadas que contengan funciones o literales de cadena.
- Cada alias de columna puede contener hasta 128 caracteres.

USO DE LITERALES

Los literales son letras, números o símbolos que se utilizan como valores específicos en un conjunto de resultados. Puede incluirlos en la lista de selección para hacer más legible el resultado.

SINTAXIS PARCIAL

```
SELECT nombreColumna | 'literalCadena' [, nombreColumna | 'literalCadena'...]
FROM nombreTabla
```



```
SELECT BusinessEntityID AS Codigo, 'Cliente-->', FirstName as Nombre, LastName as Apellido,
AddressLine1 as Direccion, City as Ciudad, StateProvinceName as Estado, CountryRegionName as Pais
FROM Sales.vIndividualCustomer
```

	Codigo	(Sin nombre de columna)	Nombre	Apellido	Direccion	Ciudad	Estado	Pais
1	16867	Cliente-->	Aaron	Adams	4116 Stanbridge Ct.	Downey	California	United States
2	16901	Cliente-->	Adam	Adams	9381 Bayside Way	Newport Beach	California	United States
3	16724	Cliente-->	Alex	Adams	237 Bellwood Dr.	Lake Oswego	Oregon	United States
4	10263	Cliente-->	Alexandra	Adams	3215 Polson Court	Burlingame	California	United States
5	10312	Cliente-->	Allison	Adams	112, rue Faubourg St Antoine	Roubaix	Nord	France
6	10274	Cliente-->	Amanda	Adams	6730 Saddlehill Lane	Fremont	California	United States
7	10292	Cliente-->	Amber	Adams	9720 Morning Glory Dr.	Brisbane	Queensland	Australia
8	10314	Cliente-->	Andrea	Adams	4767 Detroit Ave.	West Covina	California	United States
9	16699	Cliente-->	Angel	Adams	9556 Lyman Rd.	Burlingame	California	United States
10	10299	Cliente-->	Bailey	Adams	1817 Adobe Drive	Kirkland	Washington	United States
11	1770	Cliente-->	Ben	Adams	1534 Land Ave	Bremerton	Washington	United States
12	4194	Cliente-->	Blake	Adams	6055 Broadway Street	Beverly Hills	California	United States
13	16691	Cliente-->	Carlos	Adams	2079 Mountaire Pkwy.	Langford	British Columbia	Canada
14	4891	Cliente-->	Charles	Adams	1258 Steven Way	Tacoma	Washington	United States
15	10251	Cliente-->	Chloe	Adams	3001 N. 48th Street	Mayesville	Washington	United States
16	16872	Cliente-->	Connor	Adams	8308 La Salle Ave.	Westminster	British Columbia	Canada
17	10293	Cliente-->	Courtney	Adams	6089 Santa Fe Dr.	Torrance	California	United States
18	4503	Cliente-->	Dalton	Adams	3738, chaussée de Tournai	Bobigny	Seine Saint Denis	France
19	4970	Cliente-->	Devlin	Adams	8590 Hoch Mankie Court	Cranbourne	Victoria	Australia

Consulta ejecutada correctamente. WIN-BVUQ9LUHVHL (11.0 RTM) WIN-BVUQ9LUHVHL\win7pr... AdventureWorks2012 00:00:01 18508 filas

Mantenimiento de datos: Insert, Update y Delete directo y desde otras tablas.

Inserción de una fila de datos mediante valores

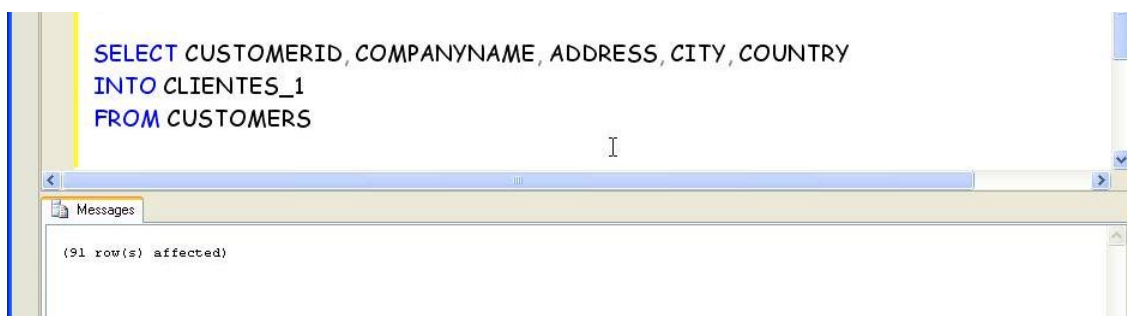
La instrucción INSERT agrega filas a una tabla.

Sintaxis parcial

```
INSERT [INTO] { nombreTabla | nombreVista } { [(listaColumnas)] { VALUES ( {  
DEFAULT | NULL | expresión}[...n]) | DEFAULT VALUES
```

La instrucción INSERT con la cláusula VALUES permite agregar filas a una tabla. Al insertar filas, tenga en cuenta los siguientes hechos e instrucciones:

- Debe atenerse a las restricciones de destino o la transacción INSERT fallará.
- Utilice listaColumnas para especificar las columnas en las que se va a almacenar cada valor especificado. listaColumnas debe especificarse entre paréntesis y delimitarse con comas. Si especifica valores para todas las columnas, listaColumnas es opcional.
- Para especificar los valores que desea insertar, utilice la cláusula VALUES. Esta cláusula se requiere para cada columna de la tabla o de listaColumnas.
- El orden y el tipo de los nuevos datos debe corresponder al orden y al tipo de las columnas de la tabla. Muchos tipos de datos tienen un formato de entrada asociado. Por ejemplo, los datos de carácter y las fechas deben encerrarse entre comillas simples.



Uso de la instrucción INSERT...SELECT

La instrucción INSERT...SELECT agrega filas a una tabla mediante la inserción del conjunto de resultados de una instrucción SELECT.

Puede emplear la instrucción INSERT...SELECT para agregar filas de otros orígenes a una tabla existente. Utilizar la instrucción INSERT...SELECT es más eficiente que escribir varias instrucciones INSERT de una sola fila. Al utilizar la instrucción INSERT...SELECT, tenga en cuenta los siguientes hechos e instrucciones:

- Todas las filas que cumplan la instrucción SELECT se insertan en la tabla más externa de la consulta.

- Debe comprobar que la tabla que recibe las nuevas filas existe en la base de datos.
- Debe asegurarse de que las columnas de la tabla que recibe los nuevos valores tienen tipos de datos compatibles con las columnas de la tabla de origen.
- Debe determinar si existe un valor predeterminado o si se permiten valores NULL en alguna de las columnas que se omiten. Si no se permiten valores NULL, debe proporcionar valores para esas columnas.



Creación de una tabla mediante la instrucción SELECT INTO

Mediante la instrucción SELECT INTO, el conjunto de resultados de cualquier consulta se puede colocar en una tabla nueva.

Utilice la instrucción SELECT INTO para llenar tablas nuevas de una base de datos con datos importados. También se puede utilizar la instrucción SELECT INTO para resolver problemas complejos que requieran un conjunto de datos de varios orígenes. Si crea primero una tabla temporal, las consultas que ejecute sobre ella serán más sencillas que las que ejecutaría sobre varias tablas o bases de datos.

Al utilizar la instrucción SELECT INTO, tenga en cuenta los siguientes hechos e instrucciones:

- Puede utilizar la instrucción SELECT INTO para crear una tabla e insertar filas en la tabla en una sola operación.
- Asegúrese de que el nombre de la tabla que se especifique en la instrucción SELECT INTO sea único. Si ya existe una tabla con el mismo nombre, la instrucción SELECT INTO producirá un error.
- Puede crear una tabla temporal local o global.
- Para crear una tabla temporal local, preceda el nombre de la tabla con el signo de almohadilla (#) o con dos signos de almohadilla (##) para crear una tabla temporal global.

Una tabla temporal local sólo es visible en la sesión actual. Una tabla temporal global es visible en todas las sesiones:

- El espacio de las tablas temporales locales se recupera cuando el usuario finaliza la sesión.
- El espacio de las tablas temporales locales se recupera cuando la tabla deja de utilizarse en todas las sesiones.

Uso de la instrucción DELETE

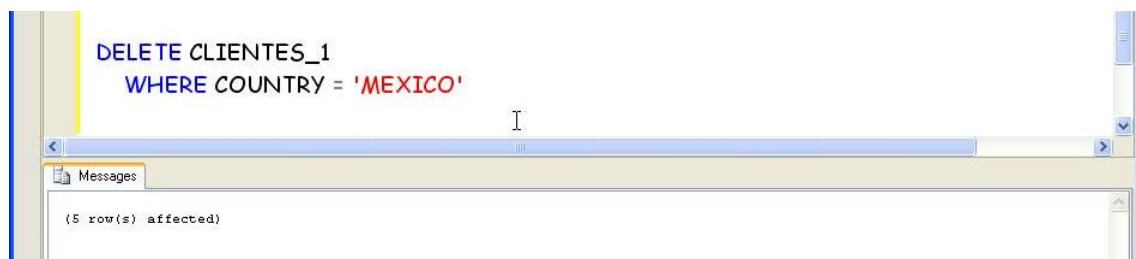
La instrucción DELETE quita filas de las tablas. La instrucción DELETE permite quitar una o varias filas de una tabla.

Sintaxis parcial

```
DELETE [from] {nombreTabla|nombreVista}
WHERE condicionesBúsqueda
```

Al utilizar la instrucción DELETE, tenga en cuenta los hechos siguientes:

- SQL Server eliminará todas las filas de la tabla si no incluye una cláusula WHERE en la instrucción DELETE.
- Cada fila eliminada se almacena en el registro de transacciones.



Uso de la instrucción TRUNCATE TABLE

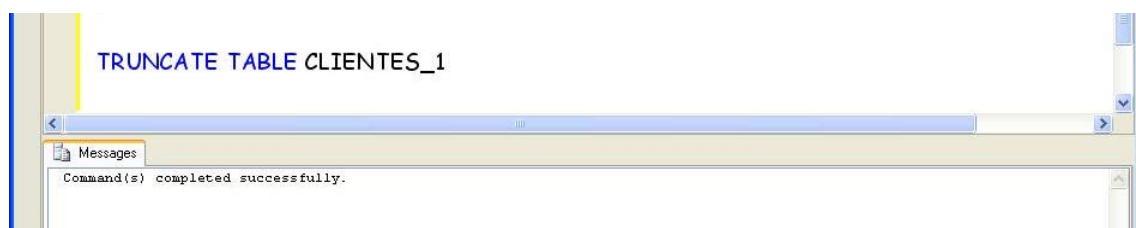
La instrucción TRUNCATE TABLE quita todos los datos de una tabla. Utilice la instrucción TRUNCATE TABLE para realizar una eliminación no registrada de todas las filas.

Sintaxis

```
TRUNCATE TABLE [[baseDatos.]propietario.]nombreTabla
```

Al utilizar la instrucción TRUNCATE TABLE, tenga en cuenta los hechos siguientes:

- SQL Server elimina todas las filas pero conserva la estructura de la tabla y los objetos asociados a ella.
- La instrucción TRUNCATE TABLE se ejecuta más rápidamente que la instrucción DELETE, ya que SQL Server sólo registra la cancelación de la asignación de las páginas de datos.
- Si una tabla tiene una columna IDENTITY, la instrucción TRUNCATE TABLE restablece el valor de inicio.



Actualización de filas basada en datos de la tabla

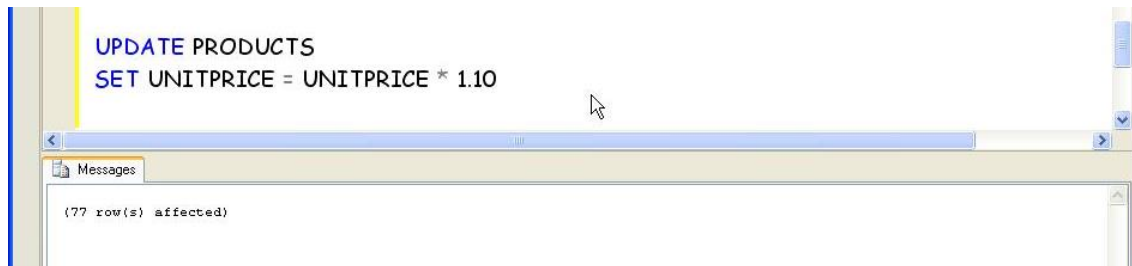
La instrucción UPDATE modifica datos existentes.

Sintaxis parcial

UPDATE {nombreTabla | nombreVista}

SET { nombreColumna = {expresión | DEFAULT | NULL} | @variable=expresión}[, ...n]

WHERE {condicionesBúsqueda}



La instrucción UPDATE permite cambiar filas individuales, grupos de filas o todas las filas de una tabla. Al actualizar filas, tenga en cuenta los hechos e instrucciones siguientes:

- Especifique las filas que desea actualizar con la cláusula WHERE.
- Especifique los nuevos valores con la cláusula SET.
- Compruebe que los valores de entrada tienen los mismos tipos de datos que los definidos para las columnas.
- SQL Server no hará actualizaciones que infrinjan alguna restricción de integridad. En ese caso, no se producirán los cambios y la instrucción se deshacerá.
- Sólo es posible cambiar los datos de una tabla cada vez.
- Puede establecer una expresión en una o varias columnas o variables. Por ejemplo, una expresión puede ser un cálculo (como price * 2) o la suma de dos columnas.

Actualización de filas basada en otras tablas

Puede utilizar la instrucción UPDATE con una cláusula FROM para modificar una tabla en función de los valores de otras tablas.

Al utilizar combinaciones y subconsultas con la instrucción UPDATE, tenga en cuenta los siguientes hechos e instrucciones:

- SQL Server nunca actualiza la misma fila dos veces en una única instrucción UPDATE. Ésta es una restricción integrada que disminuye la cantidad de registros que se genera durante las actualizaciones.
- Utilice la palabra clave SET para presentar la lista de columnas o nombres de variables que deben actualizarse. Las columnas a las que se haga referencia con la palabra clave SET no deben ser ambiguas. Por ejemplo, puede utilizar un prefijo de tabla para eliminar ambigüedades.

SINTAXIS PARCIAL

UPDATE {nombreTabla | nombreVista}

SET { nombreColumna={expresión | DEFAULT | NULL} |@variable=expresión}[, ...n]

[FROM { <origenTabla> } [WHERE condicionesBúsqueda]

ESPECIFICACIÓN DE FILAS PARA ACTUALIZAR CON COMBINACIONES

Al emplear combinaciones para actualizar filas, utilice la cláusula FROM para especificar las combinaciones en la instrucción UPDATE.



CAPITULO 5

- Agrupamiento de datos
- Subconsultas
- Uniones
- Uso de herramientas para creación de consultas y vistas.

AGrupamiento de Datos

Presentación de los primeros n valores

Utilice la palabra clave TOP n para presentar sólo las n primeras filas o el n por ciento de un conjunto de resultados. Aunque la palabra clave TOP n no es un estándar ANSI, resulta útil, por ejemplo, para presentar los productos más vendidos de una compañía.

SQLQuery1.sql - W...HVHL\win7pro (51) *

```

select top(7) * from Production.Product
order by ListPrice desc

```

100 %

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice	Size	SizeUnitMeasureCode	WeightUnitMeasureCode
1	749	Road-150 Red, 62	BK-R93R-62	1	1	Red	100	75	2171.2942	3578.27	62	CM	LB
2	750	Road-150 Red, 44	BK-R93R-44	1	1	Red	100	75	2171.2942	3578.27	44	CM	LB
3	751	Road-150 Red, 48	BK-R93R-48	1	1	Red	100	75	2171.2942	3578.27	48	CM	LB
4	752	Road-150 Red, 52	BK-R93R-52	1	1	Red	100	75	2171.2942	3578.27	52	CM	LB
5	753	Road-150 Red, 56	BK-R93R-56	1	1	Red	100	75	2171.2942	3578.27	56	CM	LB
6	771	Mountain-100 Silver, 38	BK-M82S-38	1	1	Silver	100	75	1912.1544	3399.99	38	CM	LB
7	772	Mountain-100 Silver, 42	BK-M82S-42	1	1	Silver	100	75	1912.1544	3399.99	42	CM	LB

Cuando utilice la palabra clave TOP n o TOP n PERCENT, considere los hechos e instrucciones siguientes:

- Especifique el intervalo de valores en la cláusula ORDER BY. Si no utiliza una cláusula ORDER BY, Microsoft® SQL Server™ 2000 devuelve las filas que cumplen la cláusula WHERE sin ningún orden concreto.
- Utilice un entero sin signo a continuación de la palabra clave TOP.
- Si la palabra clave TOP n PERCENT produce un número no entero de filas, SQL Server redondea la cantidad no entera al siguiente valor entero.
- Utilice la cláusula WITH TIES para incluir las filas iguales en el conjunto de resultados. Las filas iguales se producen cuando hay dos o más filas con valores iguales a los de la última fila devuelta según la cláusula ORDER BY. Por lo tanto, el conjunto de resultados puede incluir cualquier número de filas.

SQLQuery1.sql - W...HVHL\win7pro (51) *

```

select top(7) with ties * from Production.Product
order by ListPrice desc

```

100 %

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice	Size	SizeUnitMeasureCode	WeightUnitMeasureCode
1	749	Road-150 Red, 62	BK-R93R-62	1	1	Red	100	75	2171.2942	3578.27	62	CM	LB
2	750	Road-150 Red, 44	BK-R93R-44	1	1	Red	100	75	2171.2942	3578.27	44	CM	LB
3	751	Road-150 Red, 48	BK-R93R-48	1	1	Red	100	75	2171.2942	3578.27	48	CM	LB
4	752	Road-150 Red, 52	BK-R93R-52	1	1	Red	100	75	2171.2942	3578.27	52	CM	LB
5	753	Road-150 Red, 56	BK-R93R-56	1	1	Red	100	75	2171.2942	3578.27	56	CM	LB
6	771	Mountain-100 Silver, 38	BK-M82S-38	1	1	Silver	100	75	1912.1544	3399.99	38	CM	LB
7	772	Mountain-100 Silver, 42	BK-M82S-42	1	1	Silver	100	75	1912.1544	3399.99	42	CM	LB
8	773	Mountain-100 Silver, 44	BK-M82S-44	1	1	Silver	100	75	1912.1544	3399.99	44	CM	LB
9	774	Mountain-100 Silver, 48	BK-M82S-48	1	1	Silver	100	75	1912.1544	3399.99	48	CM	LB

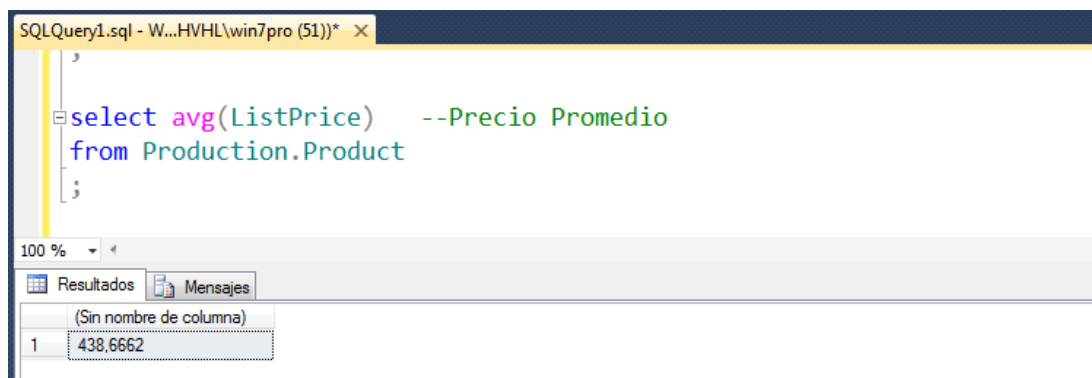
NOTA: Sólo se puede utilizar la cláusula WITH TIES cuando existe una cláusula ORDER BY.

Uso de funciones de agregado

Las funciones que calculan promedios y sumas se llaman funciones de agregado. Cuando se ejecuta una función de agregado, SQL Server resume los valores de toda una tabla o de grupos de columnas de una tabla, y produce un valor por cada conjunto de filas para las columnas especificadas:

- Las funciones de agregado se pueden utilizar en la instrucción SELECT o en combinación con la cláusula GROUP BY.
- Con la excepción de la función COUNT(*), todas las funciones de agregado devuelven NULL si ninguna fila cumple la cláusula WHERE. La función COUNT(*) devuelve el valor cero si ninguna fila cumple la cláusula WHERE.

Función de agregado	Descripción
AVG	Promedio de valores en una expresión numérica
COUNT	Número de valores en una expresión
COUNT (*)	Número de filas seleccionadas
MAX	Valor más alto en la expresión
MIN	Valor más bajo en la expresión
SUM	Valores totales en una expresión numérica
STDEV	Desviación estadística de todos los valores
STDEVP	Desviación estadística para la población
VAR	Varianza estadística de todos los valores
VARP	Varianza estadística de todos los valores para la población



The screenshot shows a SQL query window titled 'SQLQuery1.sql - W...HVHL\win7pro (51)*'. The query is:

```
select avg(ListPrice) --Precio Promedio
from Production.Product
;
```

The results pane shows a single row with the value 438,6662 under the column header '(Sin nombre de columna)'.

Uso de la cláusula GROUP BY

Utilice la cláusula GROUP BY en columnas o expresiones para organizar filas en grupos y para resumir dichos grupos. Por ejemplo, utilice la cláusula GROUP BY para determinar la cantidad de cada producto pedida en todos los pedidos.

Cuando utilice la cláusula GROUP BY, considere los hechos e instrucciones siguientes:

- SQL Server produce una columna de valores por cada grupo definido.
- SQL Server sólo devuelve filas por cada grupo especificado; no devuelve información de detalle.
- Todas las columnas que se especifican en la cláusula GROUP BY tienen que estar incluidas en la lista de selección.

- Si incluye una cláusula WHERE, SQL Server sólo agrupa las filas que cumplen las condiciones de la cláusula WHERE.
- En la lista de columnas de la cláusula GROUP BY puede haber hasta 8.060 bytes.
- No utilice la cláusula GROUP BY en columnas que contengan varios valores nulos, porque los valores nulos se procesan como otro grupo.
- Utilice la palabra clave ALL con la cláusula GROUP BY para presentar todas las filas que tengan valores nulos en las columnas de agregado, independientemente de si las filas cumplen la condición de la cláusula WHERE.

SQLQuery1.sql - W...HVHL\win7pro (51))*

```

select size,SizeUnitMeasureCode,count(*) as Cantidad
from Production.Product
group by size,SizeUnitMeasureCode

```

100 %

Resultados Mensajes

	size	SizeUnitMeasureCode	Cantidad
1	NULL	NULL	293
2	70	NULL	1
3	L	NULL	11
4	M	NULL	11
5	S	NULL	9
6	XL	NULL	3
7	38	CM	12
8	40	CM	11
9	42	CM	15
10	44	CM	29
11	46	CM	11
12	48	CM	25
13	50	CM	9
14	52	CM	16
15	54	CM	9
16	56	CM	2
17	58	CM	15
18	60	CM	11
19	62	CM	11

Consulta ejecutada correctamente. WIN-BVUQ9LUHVHL (11.0 RTM) WIN-BVUQ9LUHVHL\win7pr... AdventureWorks2012 00:00:00 19 filas

Uso de la cláusula GROUP BY con la cláusula HAVING

Utilice la cláusula HAVING en columnas o expresiones para establecer condiciones en los grupos incluidos en un conjunto de resultados. La cláusula HAVING establece condiciones en la cláusula GROUP BY de una forma muy similar a como interactúa la cláusula WHERE con la instrucción SELECT.

SQLQuery1.sql - W...HVHL\win7pro (51))*

```

select size,SizeUnitMeasureCode,count(*) as Cantidad
from Production.Product
group by size,SizeUnitMeasureCode having count(*) > 33

```

100 %

Resultados Mensajes

	size	SizeUnitMeasureCode	Cantidad
1	NULL	NULL	293

Cuando utilice la cláusula HAVING, considere los hechos e instrucciones siguientes:

- Utilice la cláusula HAVING sólo con la cláusula GROUP BY para restringir los agrupamientos. El uso de la cláusula HAVING sin la cláusula GROUP BY no tiene sentido.
- En una cláusula HAVING puede haber hasta 128 condiciones. Cuando utilice varias condiciones, tiene que combinarlas con operadores lógicos (AND, OR o NOT).
- Puede hacer referencia a cualquiera de las columnas que aparezcan en la lista de selección.
- No utilice la palabra clave ALL con la cláusula HAVING, porque la cláusula HAVING pasa por alto la palabra clave ALL y sólo devuelve los grupos que cumplen la cláusula HAVING.

Sub consultas

Una subconsulta es una instrucción SELECT anidada en una instrucción SELECT, INSERT, UPDATE o DELETE, o en otra subconsulta. A menudo puede escribir las subconsultas como combinaciones y utilizarlas en lugar de una expresión.

Una expresión es una combinación de identificadores, valores y operadores que evalúa SQL Server para obtener un resultado.

Por qué utilizar subconsultas

Las subconsultas se utilizan para dividir una consulta compleja en varios pasos lógicos y, como resultado, resolver un problema con una única instrucción. Las subconsultas son útiles cuando la consulta depende de los resultados de otra consulta.

Por qué utilizar combinaciones en lugar de subconsultas

A menudo, una consulta que contiene subconsultas se puede escribir como una combinación. En general, el rendimiento de una consulta puede ser similar con una combinación y con una subconsulta. El optimizador de consultas optimiza habitualmente subconsultas mediante el uso del plan de ejecución de ejemplo que utilizaría una combinación semánticamente equivalente. La diferencia consiste en que la subconsulta puede requerir que el optimizador de consultas realice pasos adicionales, como ordenar, lo que puede influir en la estrategia del proceso.

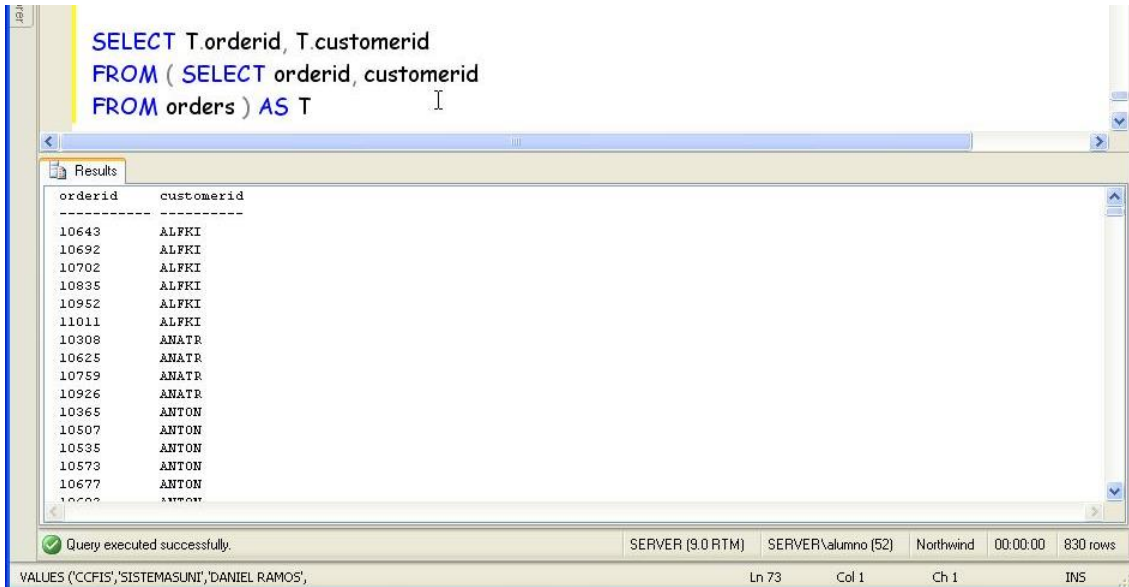
Normalmente, utilizar combinaciones permite al optimizador de consultas recuperar datos de forma más eficiente. Si una consulta no requiere varios pasos, puede que no sea necesario utilizar una subconsulta.

CÓMO UTILIZAR SUBCONSULTAS

Cuando decida utilizar subconsultas, tenga en cuenta los siguientes hechos e instrucciones:

- Las subconsultas se deben incluir entre paréntesis.
- Se pueden utilizar subconsultas en lugar de una expresión siempre y cuando se devuelva un solo valor o una lista de valores. Se pueden utilizar subconsultas que devuelvan un conjunto de registros de varias columnas en lugar de una tabla o para realizar la misma función que una combinación.

- No se pueden utilizar subconsultas que recuperen columnas con tipos de datos text e image.
- Puede tener subconsultas dentro de subconsultas, con una anidación de hasta 32 niveles. El límite varía según la cantidad de memoria disponible y la complejidad de las otras expresiones de la consulta. Las consultas individuales pueden no admitir una anidación de hasta 32 niveles.



The screenshot shows a SQL query window with the following text:

```
SELECT T.orderid, T.customerid
FROM ( SELECT orderid, customerid
FROM orders ) AS T
```

Below the query window, the 'Results' pane displays the following data:

orderid	customerid
10643	ALFKI
10692	ALFKI
10702	ALFKI
10835	ALFKI
10952	ALFKI
11011	ALFKI
10308	ANATR
10625	ANATR
10759	ANATR
10926	ANATR
10365	ANTON
10507	ANTON
10535	ANTON
10573	ANTON
10677	ANTON
10693	ANTON

At the bottom of the window, a status bar indicates: 'Query executed successfully. SERVER (9.0 RTM) SERVER\valumno (52) Northwind 00:00:00 830 rows'. The bottom-most status bar shows: 'VALUES ('CCFIS','SISTEMASUNI','DANIEL RAMOS', Ln 73 Col 1 Ch 1 INS'.

Uso de una subconsulta como una expresión

En Transact-SQL, puede sustituir una subconsulta donde utilice una expresión. La subconsulta debe producir un valor escalar o una lista de valores de una sola columna. Las subconsultas que devuelven una lista de valores sustituyen a una expresión en una cláusula WHERE que contiene la palabra clave IN.

Si se utiliza como expresión, tenga en cuenta que una subconsulta:

- Se evalúa y trata como una expresión. Con frecuencia, el optimizador de consultas evalúa una expresión como equivalente a una combinación que conecta con una tabla que tiene una fila.
- Se ejecuta una vez para la instrucción entera.



The screenshot shows a SQL query window with the following text:

```
USE PUBS
SELECT title, price, ( SELECT AVG(price) FROM titles ) AS average ,
price-(SELECT AVG(price) FROM titles) AS difference
FROM titles WHERE type='popular_comp'
```

Below the query window, the 'Results' tab is active, displaying a table with the following data:

	title	price	average	difference
1	But Is It User Friendly?	22.95	14,7662	8,1838
2	Secrets of Silicon Valley	20.00	14,7662	5,2338
3	Net Etiquette	NULL	14,7662	NULL

At the bottom of the window, a status bar indicates: 'Query executed successfully. SERVER (9.0 RTM) SERVER\alumno (52) pubs 00:00:00 3 rows'.

Uniones de Tablas (Join)

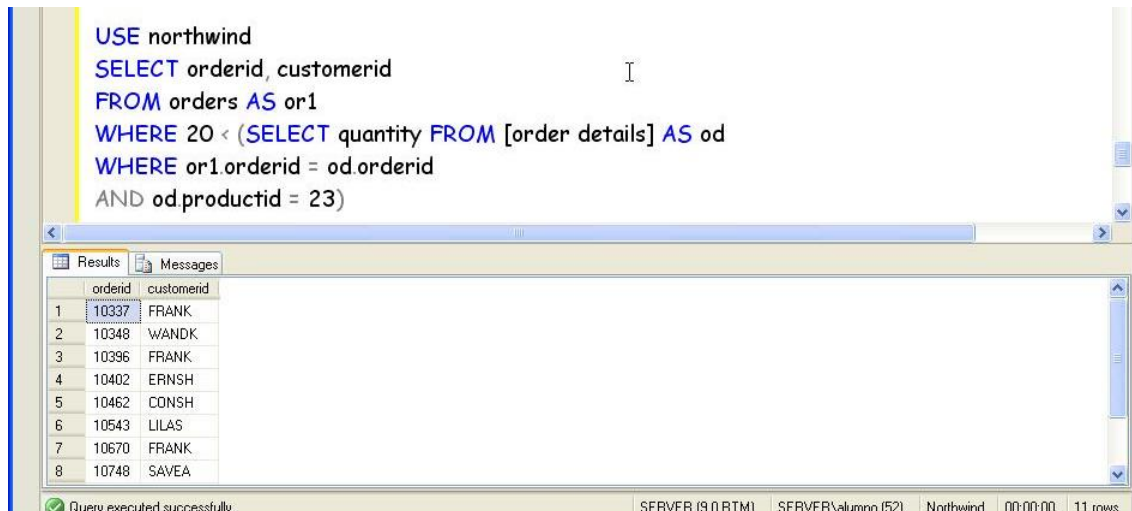
Puede utilizar una subconsulta correlacionada como una expresión dinámica que cambia en cada fila de una consulta externa.

El procesador de consultas realiza la subconsulta para cada fila de la consulta externa, una fila a la vez, que a su vez se evalúa como una expresión para esa fila y se pasa a la consulta externa. La subconsulta correlacionada es, de hecho, una COMBINACIÓN entre la subconsulta ejecutada dinámicamente y la fila de la consulta externa.

Normalmente, puede escribir una consulta de varias maneras y aun así obtener los mismos resultados. Las subconsultas correlacionadas dividen consultas complejas en dos o más consultas simples relacionadas.

Cuando crea una subconsulta correlacionada, las subconsultas internas se evalúan repetidamente, una vez por cada fila de la consulta externa:

- SQL Server ejecuta la consulta interna por cada fila que selecciona la consulta externa.
- SQL Server compara los resultados de la subconsulta con los resultados externos a ella.



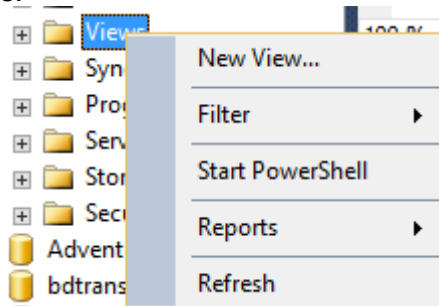
valores por cada fila especificada en la cláusula FROM de la consulta externa. Los pasos siguientes describen cómo se evalúa la subconsulta correlacionada del ejemplo:

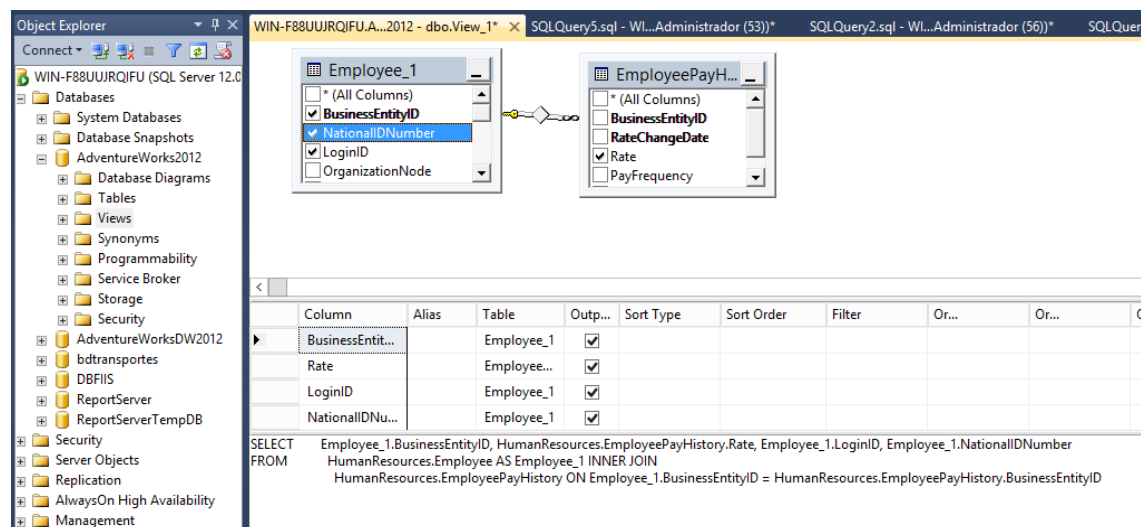
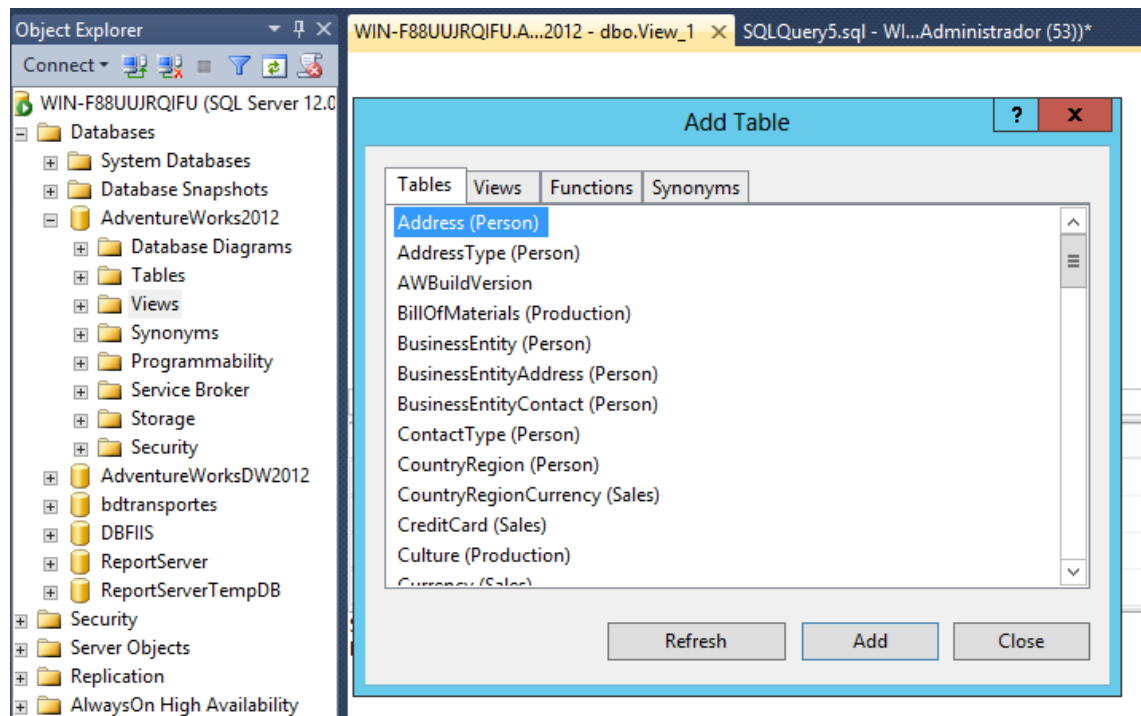
1. La consulta externa pasa un valor de columna a la consulta interna.
El valor de columna que la consulta externa pasa a la consulta interna es orderid. La consulta externa pasa el primer orderid de la tabla orders a la consulta interna.
2. La consulta interna utiliza los valores que pasa la consulta externa.
Cada orderid de la tabla orders se evalúa para determinar si existe un orderid idéntico en la tabla order details. Si el primer orderid coincide con un orderid de la tabla order details y ese orderid ha adquirido el producto número 23, la consulta interna devuelve ese orderid a la consulta externa.
3. La consulta interna devuelve un valor a la consulta externa.
La cláusula WHERE de la consulta externa evalúa posteriormente el orderid que adquirió el producto número 23 para determinar si la cantidad pedida es mayor de 20.
4. Este proceso se repite para la fila siguiente de la consulta externa.
La consulta externa pasa el segundo orderid de la tabla orders a la consulta interna y SQL Server repite el proceso de evaluación para esa fila.

Uso de herramientas para creación de vistas.

Cada vez que se quiera crear una vista, deberemos primero, tener en cuenta las relaciones existentes entre las tablas que las conforman.

Lo primero que vamos realizando es: hacer un click derecho sobre el icono de la views, dentro de la Base de Datos.





CAPITULO 6

- Programación en SQL Server 2014.
- Conceptos y estructuras de programación.
- Funciones definidas por el usuario.
- Procedimientos almacenados: Definición, estructura.

PROCEDIMIENTOS ALMACENADOS .

Un procedimiento almacenado es una colección con nombre de instrucciones de Transact-SQL que se almacena en el servidor. Los procedimientos almacenados son un método para encapsular tareas repetitivas. Admiten variables declaradas por el usuario, ejecución condicional y otras características de programación muy eficaces.

SQL Server admite cinco tipos de procedimientos almacenados:

PROCEDIMIENTOS ALMACENADOS DEL SISTEMA (SP_)

Almacenados en la base de datos master e identificados mediante el prefijo sp_, los procedimientos almacenados del sistema proporcionan un método efectivo de recuperar información de las tablas del sistema. Permiten a los administradores del sistema realizar tareas de administración de la base de datos que actualizan las tablas del sistema aunque éstos no tengan permiso para actualizar las tablas subyacentes directamente. Los procedimientos almacenados del sistema se pueden ejecutar en cualquier base de datos.

PROCEDIMIENTOS ALMACENADOS LOCALES

Los procedimientos almacenados locales se crean en las bases de datos de los usuarios individuales.

PROCEDIMIENTOS ALMACENADOS TEMPORALES

Los procedimientos almacenados temporales pueden ser locales, con nombres que comienzan por un signo de almohadilla (#), o globales, con nombres que comienzan por un signo de almohadilla doble (##). Los procedimientos almacenados temporales locales están disponibles en la sesión de un único usuario, mientras que los procedimientos almacenados temporales globales están disponibles para las sesiones de todos los usuarios.

PROCEDIMIENTOS ALMACENADOS REMOTOS

Los procedimientos almacenados remotos son una característica anterior de SQL Server. Las consultas distribuidas admiten ahora esta funcionalidad.

PROCEDIMIENTOS ALMACENADOS EXTENDIDOS (XP_)

Los procedimientos almacenados extendidos se implementan como bibliotecas de vínculos dinámicos (DLL, Dynamic-Link Libraries) que se ejecutan fuera del entorno de SQL Server. Normalmente, se identifican mediante el prefijo xp_. Se ejecutan de forma similar a los procedimientos almacenados.

Ventajas de los procedimientos almacenados

Los procedimientos almacenados ofrecen numerosas ventajas con respecto a la ejecución de consultas Transact-SQL ad-hoc. Pueden:

- Encapsular funcionalidad de negocios y crear lógica de aplicación reutilizable. Las reglas o directivas de negocios encapsuladas en procedimientos almacenados se pueden modificar en una única ubicación. Todos los clientes pueden utilizar los mismos procedimientos almacenados para garantizar un acceso y una modificación coherentes de los datos.
- Evitar la exposición de los usuarios a los detalles de las tablas de la base de datos. Si un conjunto de procedimientos almacenados acepta todas las

funciones de negocios que los usuarios deben realizar, éstos nunca necesitan tener acceso a las tablas directamente.

- Proporcionar mecanismos de seguridad. Se puede conceder permiso a los usuarios para ejecutar un procedimiento almacenado incluso aunque no tengan permiso de acceso a las tablas o a las vistas a las que hace referencia el procedimiento almacenado.
- Mejorar el rendimiento. Los procedimientos almacenados implementan muchas tareas como una serie de instrucciones Transact-SQL. Es posible aplicar lógica condicional al resultado de las primeras instrucciones Transact-SQL para determinar qué instrucciones Transact-SQL posteriores se ejecutarán. Todas estas instrucciones Transact-SQL y la lógica condicional forman parte de un único plan de ejecución en el servidor.
- Reducir el tráfico de red. En lugar de enviar cientos de instrucciones Transact-SQL a través de la red, los usuarios pueden realizar una operación compleja enviando una única instrucción, lo que reduce el número de solicitudes que pasan entre el cliente y el servidor.
- Reducir la vulnerabilidad a ataques de inserción de SQL. El uso de parámetros definidos explícitamente en código de SQL elimina la posibilidad de que un pirata pueda enviar instrucciones SQL incrustadas en los valores de los parámetros

Ejecución (por primera vez o recompilación)

La primera vez que se ejecuta un procedimiento almacenado o si el procedimiento almacenado se debe volver a compilar, el procesador de consultas lo lee en un proceso llamado resolución.

Ciertos cambios en una base de datos pueden hacer que un plan de ejecución sea ineficaz o deje de ser válido. SQL Server detecta estos cambios y vuelve a compilarlo automáticamente cuando se produce alguna de las situaciones siguientes:

- Se realiza algún cambio estructural en una tabla o vista a la que hace referencia la consulta (ALTER TABLE y ALTER VIEW).
- Se generan nuevas estadísticas de distribución, bien de forma explícita a partir de una instrucción, como en UPDATE STATISTICS, o automáticamente.
- Se quita un índice usado por el plan de ejecución.
- Se realizan cambios importantes en las claves (la instrucción INSERT o DELETE) de una tabla a la que hace referencia una consulta.

OPTIMIZACIÓN

Cuando un procedimiento almacenado pasa correctamente la etapa de resolución, el optimizador de consultas de SQL Server analiza las instrucciones de Transact-SQL del procedimiento almacenado y crea un plan que contiene el método más rápido para obtener acceso a los datos. Para ello, el optimizador de consultas tiene en cuenta lo siguiente:

- La cantidad de datos de las tablas.
- La presencia y naturaleza de los índices de las tablas, y la distribución de los datos en las columnas indizadas.
- Los operadores de comparación y los valores de comparación que se usan en las condiciones de la cláusula WHERE.
- La presencia de combinaciones y las cláusulas UNION, GROUP BY u ORDER BY.

COMPILACIÓN

La compilación hace referencia al proceso consistente en analizar el procedimiento almacenado y crear un plan de ejecución que se encuentra en la caché de procedimientos. La caché de procedimientos contiene los planes de ejecución de los procedimientos almacenados más importantes. Entre los factores que aumentan el valor de un plan se incluyen los siguientes:

- Tiempo requerido para volver a compilar (costo de compilación alto)
- Frecuencia de uso

Creación de procedimientos almacenados

Sólo se puede crear un procedimiento almacenado en la base de datos activa, excepto en el caso de los procedimientos almacenados temporales, que se crean siempre en la base de datos tempdb. La creación de un procedimiento almacenado es similar a la creación de una vista. Primero, escriba y pruebe las instrucciones de Transact-SQL que desea incluir en el procedimiento almacenado. A continuación, si recibe los resultados esperados, cree el procedimiento almacenado.

La sintaxis no permite especificar el nombre de la base de datos como prefijo del nombre del objeto.

Uso de CREATE PROCEDURE

Los procedimientos almacenados se crean con la instrucción CREATE PROCEDURE. Considere los hechos siguientes cuando cree procedimientos almacenados:

- Los procedimientos almacenados pueden hacer referencia a tablas, vistas, funciones definidas por el usuario y otros procedimientos almacenados, así como a tablas temporales.
- Si un procedimiento almacenado crea una tabla local temporal, la tabla temporal sólo existe para atender al procedimiento almacenado y desaparece cuando finaliza la ejecución del mismo.
- Una instrucción CREATE PROCEDURE no se puede combinar con otras instrucciones de Transact-SQL en un solo proceso por lotes.
- La definición de CREATE PROCEDURE puede incluir cualquier número y tipo de instrucciones de Transact-SQL, con la excepción de las siguientes instrucciones de creación de objetos: CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER y CREATE VIEW. En un procedimiento almacenado se pueden crear otros objetos de la base de datos y deben calificarse con el nombre del propietario del objeto.
- Para ejecutar la instrucción CREATE PROCEDURE, debe ser miembro de la función de administradores del sistema (sysadmin), de la función de propietario de la base de datos (db_owner) o de la función de administrador del lenguaje de definición de datos (db_ddladmin), o debe haber recibido el permiso CREATE PROCEDURE.
- El tamaño máximo de un procedimiento almacenado es 128 megabytes (MB), según la memoria disponible.

Sintaxis parcial para crear un procedimiento almacenado

La instrucción CREATE PROCEDURE contiene muchas opciones posibles, como se muestra en la siguiente sintaxis parcial.

```
CREATE { PROC | PROCEDURE } [nombreDeEsquema.] nombreDeProcedimiento
[ { @parámetro [ nombreDeEsquemaConTipo. ] tipoDeDatos }
[ VARYING ] [ = valorPredeterminado ] [ [ OUT [ PUT ] ]
[ ,...n ]
[ WITH <opciónDeProcedimiento> [ ,...n ]
AS instrucciónSql [;][ ...n ]
<opciónDeProcedimiento> =
[ ENCRYPTION ]
[ RECOMPILE ]
[ EXECUTE_AS_Clause ]
```

EJEMPLO DE CREACIÓN DE UN PROCEDIMIENTO ALMACENADO SENCILLO

En el ejemplo siguiente se muestra cómo se puede crear un procedimiento almacenado sencillo que devuelve un conjunto de filas de todos los productos que tardan más de un día en fabricarse.

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

El ejemplo anterior crea un procedimiento denominado ProductosComplejos dentro del esquema Production. Se incluye un comando GO para resaltar el hecho de que las instrucciones CREATE PROCEDURE deben declararse dentro de un único lote.

EJEMPLO DE LLAMADA A UN PROCEDIMIENTO ALMACENADO

En el ejemplo siguiente se muestra cómo llamar al procedimiento almacenado LongLeadProducts.

```
EXEC Production.ProductosComplejos
```

Recomendaciones para la creación de procedimientos almacenados

Considere las recomendaciones siguientes cuando cree procedimientos almacenados:

- Para evitar situaciones en las que el propietario de un procedimiento almacenado y el propietario de las tablas subyacentes sean distintos, se recomienda que el

usuario dbo (propietario de base de datos) sea el propietario de todos los objetos de una base de datos. Como un usuario puede ser miembro de varias funciones, debe especificar siempre el usuario dbo como propietario al crear el objeto. En caso contrario, el objeto se creará con su nombre de usuario como propietario:

- También debe tener los permisos adecuados en todas las tablas o vistas a las que se hace referencia en el procedimiento almacenado.
- Evite situaciones en las que el propietario de un procedimiento almacenado y el propietario de las tablas subyacentes sean distintos.
- Diseñe cada procedimiento almacenado para realizar una única tarea.
- Cree, pruebe y solucione los problemas del procedimiento almacenado en el servidor; a continuación, pruébelo desde el cliente.
- Para distinguir fácilmente los procedimientos almacenados del sistema, evite utilizar el prefijo sp_ cuando nombre los procedimientos almacenados locales
- No elimine nunca directamente las entradas de la tabla del sistema syscomments. Si no desea que los usuarios puedan ver el texto de los procedimientos almacenados, debe crearlos usando la opción WITH ENCRYPTION. Si no utiliza WITH ENCRYPTION, los usuarios pueden usar el Administrador corporativo de SQL Server o ejecutar el procedimiento almacenado del sistema sp_helptext para ver el texto de los procedimientos almacenados que se encuentran en la tabla del sistema syscomments.

Sintaxis para modificar y quitar procedimientos almacenados

Los procedimientos almacenados suelen modificarse como respuesta a solicitudes de los usuarios o a cambios en las definiciones de tablas subyacentes. Para modificar un procedimiento almacenado existente y conservar las asignaciones de permisos, utilice la instrucción ALTER PROCEDURE. SQL Server reemplaza la definición anterior del procedimiento almacenado cuando se modifica mediante ALTER PROCEDURE.

Tenga en cuenta lo siguiente cuando utilice la instrucción ALTER PROCEDURE: n

- Si desea modificar un procedimiento almacenado creado mediante una opción, como WITH ENCRYPTION, debe incluir la opción en la instrucción ALTER PROCEDURE para conservar la funcionalidad que proporciona la opción.
- ALTER PROCEDURE sólo cambia un único procedimiento. Si su procedimiento llama a otros procedimientos almacenados, los procedimientos anidados no se ven afectados.

EJEMPLO DE MODIFICACIÓN DE UN PROCEDIMIENTO ALMACENADO

En el ejemplo siguiente se modifica el procedimiento almacenado LongLeadProducts para seleccionar una columna adicional y ordenar el conjunto de resultados utilizando una cláusula ORDER BY.

```
ALTER PROC Production.ProductosComplejos
AS
SELECT Name, ProductNumber, DaysToManufacture
FROM Production.Product
WHERE DaysToManufacture >= 1
ORDER BY DaysToManufacture DESC, Name
GO
```

Quitar procedimientos almacenados

Utilice la instrucción DROP PROCEDURE para quitar procedimientos almacenados definidos por el usuario de la base de datos actual.

Antes de quitar un procedimiento almacenado, ejecute el procedimiento almacenado sp_depends para determinar si hay objetos que dependen del procedimiento almacenado, como se muestra en el ejemplo siguiente.

```
EXEC sp_depends @objname = N'Production. ProductosComplejos '
```

En el ejemplo siguiente se quita el procedimiento almacenado ProductosComplejos.

```
DROP PROC Production. ProductosComplejos
```

Creación de procedimientos almacenados parametrizados

Un procedimiento almacenado se comunica con el programa que llama al procedimiento mediante una lista de hasta 2100 parámetros. Los parámetros de entrada permiten pasar información a un procedimiento almacenado; estos valores pueden utilizarse entonces como variables locales dentro del procedimiento.

Directrices para utilizar parámetros de entrada

Para definir un procedimiento almacenado que acepta parámetros de entrada, debe declarar una o más variables como parámetros en la instrucción CREATE PROCEDURE.

Tenga en cuenta lo siguiente cuando utilice parámetros de entrada:

- Proporcione valores predeterminados para un parámetro cuando sea apropiado. Si se define un valor predeterminado, un usuario puede ejecutar el procedimiento almacenado sin especificar un valor para ese parámetro.

- Valide todos los valores de parámetros de entrada al principio de un procedimiento almacenado para interceptar pronto los valores que falten o que no sean válidos. Esto puede incluir la comprobación de si el parámetro es NULL.

EJEMPLO DE USO DE PARÁMETROS DE ENTRADA

En el ejemplo siguiente se agrega un parámetro @MinimumLength al procedimiento almacenado ProductosComplejos. Esto permite que la cláusula WHERE sea más flexible que la mostrada anteriormente al permitir que la aplicación que llama defina qué período de tiempo se considera apropiado.

```
ALTER PROC Production.ProductosComplejos
@MinimumLength int = 1 -- default value
AS
IF (@MinimumLength < 0) -- validate
BEGIN
RAISERROR('intervalo NO válido', 14, 1)
RETURN
END
SELECT Name, ProductNumber, DaysToManufacture
FROM Production.Product
WHERE DaysToManufacture >= @MinimumLength
ORDER BY DaysToManufacture DESC, Name
GO
```

El procedimiento almacenado define un valor de parámetro predeterminado de 1, por lo que las aplicaciones que llaman pueden ejecutar el procedimiento sin especificar un argumento. Si se pasa un valor a @MinimumLength, se valida para asegurarse de que el valor es apropiado para la finalidad de la instrucción SELECT.

Si el valor es menor que cero, se producirá un error y el procedimiento almacenado volverá inmediatamente sin ejecutar la instrucción SELECT.

Llamada a procedimientos almacenados parametrizados

Puede establecer el valor de un parámetro si pasa el valor al procedimiento almacenado por nombre de parámetro o por posición. No debe mezclar distintos formatos al suministrar valores.

La especificación de un parámetro en una instrucción EXECUTE con el formato @parámetro = valor se denomina pasar por nombre de parámetro. Cuando pasa valores por nombre de parámetro, es posible especificar los valores de parámetro en cualquier orden y puede omitir los parámetros que permitan valores nulos o que tengan un valor predeterminado.

En el ejemplo siguiente se llama al procedimiento almacenado ProductosComplejos y se especifica el nombre del parámetro.

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

Pasar sólo valores (sin hacer referencia a los nombres de parámetro a los que se pasan) se conoce como pasar valores por posición. Cuando sólo especifica un valor, los valores de los parámetros deben indicarse en el orden en que se definen en la instrucción CREATE PROCEDURE. Cuando pasa valores por posición, puede omitir los parámetros para los que existan valores predeterminados, pero no puede interrumpir la secuencia.

Por ejemplo, si un procedimiento almacenado tiene cinco parámetros, puede omitir el cuarto y el quinto parámetros, pero no puede omitir el cuarto y especificar el quinto.

En el ejemplo siguiente se llama al procedimiento almacenado ProductosComplejos y se especifica el parámetro utilizando la posición únicamente:

```
EXEC Production. ProductosComplejos 4
```

Uso de los valores predeterminados de un parámetro

El valor predeterminado de un parámetro, si se define para el parámetro en el procedimiento almacenado, se utiliza cuando:

- No se especifica ningún valor para el parámetro cuando se ejecuta el procedimiento almacenado.
- Se especifica la palabra clave DEFAULT como el valor del parámetro.

Parámetros de salida y valores devueltos

Los procedimientos almacenados pueden devolver información al procedimiento almacenado o al cliente que llama utilizando parámetros de salida y un valor devuelto.

Características de los parámetros de salida

Los parámetros de salida permiten conservar los cambios al parámetro que resultan de la ejecución del procedimiento almacenado, incluso después de que se complete la ejecución del procedimiento almacenado.

Para utilizar un parámetro de salida dentro de Transact-SQL, debe especificar la palabra clave OUTPUT en las instrucciones CREATE PROCEDURE y EXECUTE. Si se omite la palabra clave OUTPUT cuando se ejecuta el procedimiento almacenado, éste se ejecutará pero no devolverá el valor modificado. En la mayoría de los lenguajes de programación cliente, como Microsoft Visual C#®, la dirección de los parámetros es de entrada de forma predeterminada, por lo que debe indicar la dirección del parámetro en el cliente.

EJEMPLO DE USO DE PARÁMETROS DE SALIDA

En el ejemplo siguiente se crea un procedimiento almacenado que inserta un nuevo departamento en la tabla HumanResources.Department de la base de datos AdventureWorks.

```
CREATE PROC HumanResources.AddDepartment
@Name nvarchar(50), @GroupName nvarchar(50),
@DeptID smallint OUTPUT
AS
INSERT INTO HumanResources.Department (Name, GroupName)
VALUES (@Name, @GroupName)
SET @DeptID = SCOPE_IDENTITY()
```

El parámetro de salida @DeptID almacena la identidad del nuevo registro llamando a la función SCOPE_IDENTITY de forma que una aplicación que llama pueda tener acceso inmediatamente al número de Id. generado automáticamente.

En el ejemplo siguiente se muestra cómo la aplicación que llama puede almacenar el resultado de la ejecución del procedimiento almacenado utilizando la variable local @dept.

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

Valores devueltos También puede devolver información de un procedimiento almacenado mediante la instrucción RETURN. Este método es más restrictivo que el uso de parámetros de salida, ya que sólo devuelve un único valor entero. La instrucción RETURN suele utilizarse para devolver un resultado de estado o un código de error de un procedimiento.

En el ejemplo siguiente se modifica el procedimiento almacenado AddDepartment para devolver un valor de éxito o de error.

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

Si se pasa una cadena vacía al procedimiento para el parámetro @Name o @GroupName, se devolverá un valor -1 para indicar un error. Si la instrucción INSERT tiene éxito, se devolverá un valor 0 para indicar el éxito.

En el ejemplo siguiente se muestra cómo la aplicación que llama puede almacenar el valor devuelto de la ejecución del procedimiento almacenado utilizando la variable local @result.

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

Nota: SQL Server devuelve automáticamente un 0 de los procedimientos almacenados si no especifica su propio valor RETURN.

USO DE FUNCIONES

Las funciones son rutinas que se utilizan para encapsular lógica que se realiza con frecuencia. En lugar de tener que repetir toda la lógica de la función, cualquier código que deba realizar la lógica puede llamar a la función.

Las funciones son rutinas que constan de una o más instrucciones Transact-SQL que pueden utilizarse para encapsular código con el fin de reutilizarlo. Una función toma cero o más parámetros de entrada y devuelve un valor escalar o una tabla.

Los parámetros de entrada pueden ser de cualquier tipo de datos excepto timestamp, cursor o table, pero las funciones no aceptan parámetros de salida.

FUNCIONES ESCALARES

Las funciones escalares devuelven un único valor de datos del tipo definido en una cláusula RETURNS. Estos tipos de funciones son muy similares sintácticamente a las funciones integradas del sistema como COUNT o MAX.

FUNCIONES CON VALORES DE TABLA EN LÍNEA

Una función con valores de tabla en línea devuelve una tabla que es el resultado de una única instrucción SELECT. Es similar a una vista, pero ofrece más flexibilidad que las vistas porque puede proporcionar parámetros a la función.

FUNCIONES CON VALORES DE TABLA DE VARIAS INSTRUCCIONES

Una función con valores de tabla de varias instrucciones devuelve una tabla generada por una o varias instrucciones Transact-SQL y es similar a un procedimiento almacenado. A diferencia de un procedimiento almacenado, se puede hacer referencia a una función con valores de tabla de varias instrucciones en la cláusula FROM de una instrucción SELECT como si fuera una vista o una tabla.

Función Escalar

Una función escalar devuelve un único valor de datos del tipo definido en una cláusula RETURNS. El cuerpo de la función, definido en un bloque BEGIN...END, contiene la serie de instrucciones Transact-SQL que devuelven el valor.

En el ejemplo siguiente se crea una función escalar que calcula el total de las ventas de un producto determinado de la base de datos AdventureWorks y devuelve el total como un valor int.

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

Nota. Al modificar o quitar una función, se utiliza una sintaxis similar que para modificar y quitar otros objetos de base de datos. Utilice ALTER FUNCTION para modificar las funciones después de haberlas creado y DROP FUNCTION para quitar las funciones de la base de datos.

Invocación de funciones escalares

Una función definida por el usuario que devuelva un valor escalar puede invocarse en cualquier lugar de una expresión escalar del mismo tipo de datos permitido en las instrucciones Transact-SQL. La tabla siguiente contiene ejemplos de dónde puede utilizar funciones escalares.

Área	Ejemplo
Consultas	<ul style="list-style-type: none"> • Como una <i>expresión</i> en la <i>listaDeSelección</i> de una instrucción SELECT. • Como una <i>expresión</i> o una <i>expresiónDeCadena</i> en una cláusula WHERE o HAVING. • Como una <i>expresiónAgruparPor</i> en una cláusula GROUP BY. • Como una <i>expresiónOrdenarPor</i> en una cláusula ORDER BY. • Como una <i>expresión</i> en la cláusula SET de una instrucción UPDATE. • Como una <i>expresión</i> en la cláusula VALUES de una instrucción INSERT.
Definición de tabla	<ul style="list-style-type: none"> • Restricciones CHECK. Las funciones sólo pueden hacer referencia a columnas de la misma tabla. • Definiciones DEFAULT. Las funciones sólo pueden contener constantes.

	<ul style="list-style-type: none"> • Columnas calculadas. Las funciones sólo pueden hacer referencia a columnas de la misma tabla.
Instrucciones Transact-SQL	<ul style="list-style-type: none"> • En operadores de asignación. • En expresiones booleanas de instrucciones de control de flujo. • En expresiones CASE. • En instrucciones PRINT (sólo para funciones que devuelven una cadena de caracteres).
Funciones y procedimientos almacenados	<ul style="list-style-type: none"> • Como argumentos de función. • Como una instrucción RETURN de un procedimiento almacenado (sólo para las funciones escalares que devuelven un entero). • Como una instrucción RETURN de una función definida por el usuario, siempre y cuando el valor devuelto por dicha función se pueda convertir implícitamente en el tipo de datos devuelto de la función que realiza la llamada.

En el ejemplo siguiente se ejecuta una instrucción SELECT que recupera el ProductID, el Name y el resultado de la función escalar SumSold para cada registro de producto de AdventureWorks.

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

Función con valores de tabla en línea

Puede utilizar funciones en línea para lograr la funcionalidad de las vistas parametrizadas. Una de las limitaciones de una vista es que al crearla no puede incluir un parámetro proporcionado por el usuario dentro de la vista. Normalmente puede resolver este problema proporcionando una cláusula WHERE al llamar a la vista. Sin embargo, esto puede requerir la creación de una cadena para su ejecución dinámica, lo que puede aumentar la complejidad de la aplicación. Puede lograr la funcionalidad de una vista parametrizada si utiliza una función con valores de tabla en línea.

Tenga en cuenta las características siguientes de las funciones definidas por el usuario con valores de tabla en línea:

- La instrucción RETURNS especifica table como el tipo de datos devuelto.
- El conjunto de resultados de la instrucción SELECT define el formato de una variable de retorno.
- La cláusula RETURN contiene una única instrucción SELECT entre paréntesis. La instrucción SELECT utilizada en una función en línea está sujeta a las mismas restricciones que las instrucciones SELECT empleadas en las vistas.
- El cuerpo de la función no tiene por qué aparecer dentro de un bloque BEGIN...END.

EJEMPLO DE UNA FUNCIÓN CON VALORES DE TABLA EN LÍNEA

En el ejemplo siguiente se crea una función con valores de tabla en línea que devuelve los nombres de los empleados de un director particular de la base de datos AdventureWorks.

```
CREATE FUNCTION HumanResources.EmployeesForManager
(@ManagerId int)
RETURNS TABLE
AS
RETURN (
SELECT FirstName, LastName
FROM HumanResources.Employee Employee INNER JOIN
Person.Contact Contact
ON Employee.ContactID = Contact.ContactID
WHERE ManagerID = @ManagerId )
```

INVOCACIÓN DE FUNCIONES CON VALORES DE TABLA EN LÍNEA

Utilice una función con valores de tabla en línea en cualquier lugar donde utilizaría normalmente una vista, como en la cláusula FROM de una instrucción SELECT. En el ejemplo siguiente se recuperan todos los nombres de los empleados de dos directores.

```
SELECT * FROM HumanResources.EmployeesForManager(3)
-- OR
SELECT * FROM HumanResources.EmployeesForManager(6)
```

Función con valores de tabla de varias instrucciones

Una función con valores de tabla de varias instrucciones es una combinación de una vista y un procedimiento almacenado. Puede utilizar funciones definidas por el usuario que devuelvan una tabla para reemplazar procedimientos almacenados o vistas.

Una función con valores de tabla (como un procedimiento almacenado) puede utilizar lógica compleja y varias instrucciones Transact-SQL para generar una tabla. De la misma forma en que utiliza una vista, puede utilizar una función con valores de tabla en la cláusula FROM de una instrucción Transact-SQL.

Tenga en cuenta las características siguientes de las funciones con valores de tabla de varias instrucciones:

- La instrucción RETURNS especifica table como el tipo de datos devuelto, y devuelve un nombre para la tabla y el formato.
- Un bloque BEGIN...END delimita el cuerpo de la función.

EJEMPLO DE UNA FUNCIÓN CON VALORES DE TABLA DE VARIAS INSTRUCCIONES

En el ejemplo siguiente se crea una variable denominada @tbl_Employees con dos columnas. La segunda columna cambia dependiendo del valor del parámetro @format solicitado.

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

INVOCACIÓN DE FUNCIONES CON VALORES DE TABLA DE VARIAS INSTRUCCIONES

Puede llamar a la función en la cláusula FROM en lugar de utilizar una tabla o una vista. En el ejemplo siguiente se recuperan los nombres de los empleados en formato largo y en formato corto.

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

Control de errores , Uso del Try..Catch

El control de excepciones estructurado es una forma habitual de controlar excepciones en muchos lenguajes de programación muy conocidos, como Microsoft Visual Basic® y Visual C#. SQL Server 2014 le permite utilizar control de excepciones estructurado en cualquier situación transaccional, como un procedimiento almacenado. Esto hace que el código sea más legible y más fácil de mantener.

Sintaxis para el control de errores estructurado

Utilice bloques TRY...CATCH para implementar control de excepciones estructurado. El bloque TRY contiene el código transaccional que podría producir un error. El bloque CATCH contiene el código que se ejecuta si se produce un error en el bloque TRY.

TRY...CATCH tiene la sintaxis siguiente.

```
BEGIN TRY
{ instrucciónSql | bloqueDeInstrucciones }
END TRY
BEGIN CATCH
{ instrucciónSql | bloqueDeInstrucciones }
END CATCH
```

La parte instrucciónSql o bloqueDeInstrucciones de la sintaxis es cualquier instrucción o grupo de instrucciones Transact-SQL.

EJEMPLO DE TRY...CATCH

En este ejemplo, un procedimiento almacenado denominado AddData intenta insertar dos valores en una tabla denominada TestData. La primera columna de la tabla TestData es una clave principal entera y la segunda columna tiene el tipo de datos integer. Un bloque TRY...CATCH dentro del procedimiento almacenado AddData protege la instrucción INSERT TestData, y devuelve el número y el mensaje de error como parte de la lógica del bloque CATCH utilizando las funciones ERROR_NUMBER y ERROR_MESSAGE.

```
CREATE TABLE dbo.TableWithKey (ColA int PRIMARY KEY, ColB int)
GO
CREATE PROCEDURE dbo.AddData @a int, @b int
AS
BEGIN TRY
INSERT INTO TableWithKey VALUES (@a, @b)
END TRY
BEGIN CATCH
SELECT ERROR_NUMBER() ErrorNumber, ERROR_MESSAGE() [Message]
END CATCH
GO
EXEC dbo.AddData 1, 1
EXEC dbo.AddData 2, 2
EXEC dbo.AddData 1, 3 --viola la llave primaria
```

Directrices para controlar errores

Creación del bloque CATCH

Cree el bloque CATCH inmediatamente después de la instrucción END TRY utilizando las instrucciones BEGIN CATCH y END CATCH. No puede incluir ninguna otra instrucción entre END TRY y BEGIN CATCH.

El ejemplo siguiente no se compilará.

```
BEGIN TRY
-- INSERT INTO ...
END TRY
SELECT * FROM TableWithKey -- NOT ALLOWED
BEGIN CATCH
-- SELECT ERROR_NUMBER()
END CATCH
```

DESHACER TRANSACCIONES CON ERROR

El uso de transacciones le permite agrupar varias instrucciones de forma que todas ellas se completen correctamente o que ninguna de ellas se complete correctamente.

Considere el ejemplo siguiente, que no utiliza transacciones.

```
CREATE TABLE dbo.TableNoKey (ColA int, ColB int)
CREATE TABLE dbo.TableWithKey (ColA int PRIMARY KEY, ColB int)
GO
CREATE PROCEDURE dbo.AddData @a int, @b int
AS
BEGIN TRY
INSERT dbo.TableNoKey VALUES (@a, @b)
INSERT dbo.TableWithKey VALUES (@a, @b)
END TRY
BEGIN CATCH
SELECT ERROR_NUMBER() ErrorNumber, ERROR_MESSAGE() [Message]
END CATCH
GO
EXEC dbo.AddData 1, 1
EXEC dbo.AddData 2, 2
EXEC dbo.AddData 1, 3
```

Este ejemplo realiza dos inserciones secuenciales en dos tablas diferentes.

La primera inserción siempre se realizará correctamente porque no hay ninguna restricción de clave principal en la tabla. La segunda inserción producirá un error siempre que se inserte un valor ColA duplicado. Como en este ejemplo no se utilizan transacciones, la primera inserción se realiza correctamente cuando se produce un error en la segunda transacción, lo que puede producir resultados inesperados.

En el ejemplo siguiente se utilizan transacciones para asegurarse de que no se realizará ninguna inserción si cualquiera de las inserciones falla. Para ello se utilizan instrucciones BEGIN TRAN y COMMIT TRAN dentro del bloque TRY, y una instrucción ROLLBACK TRAN dentro del bloque CATCH.

```
ALTER PROCEDURE dbo.AddData @a int, @b int
AS
BEGIN TRY
BEGIN TRAN
INSERT dbo.TableNoKey VALUES (@a, @b)
INSERT dbo.TableWithKey VALUES (@a, @b)
COMMIT TRAN
END TRY
BEGIN CATCH
ROLLBACK TRAN
SELECT ERROR_NUMBER() ErrorNumber, ERROR_MESSAGE() [Message]
END CATCH
GO
```

Uso de XACT_ABORT y XACT_STATE

La opción XACT_ABORT especifica si SQL Server debe deshacer automáticamente la transacción actual cuando una instrucción Transact-SQL produzca un error en tiempo de ejecución. Sin embargo, si el error se produce dentro de un bloque TRY, la transacción no se deshace automáticamente, sino que se convierte en no confirmable. El código que hay dentro de un bloque CATCH debe comprobar el estado de una transacción mediante la función XACT_STATE. XACT_STATE devuelve -1 si hay una transacción no confirmable en la sesión actual. El bloque CATCH no debe intentar confirmar la transacción y debe deshacerla manualmente. Un valor devuelto de 1 de XACT_STATE significa que hay una transacción que se puede confirmar de manera segura. Un valor devuelto de 0 significa que no hay ninguna transacción actual.

En el ejemplo siguiente se establece XACT_ABORT ON y se prueba el estado de la transacción dentro del bloque CATCH.

```
SET XACT_ABORT ON
BEGIN TRY
BEGIN TRAN
...
COMMIT TRAN
END TRY
BEGIN CATCH
IF (XACT_STATE()) = -1 -- uncommittable
ROLLBACK TRAN
ELSE IF (XACT_STATE()) = 1 -- committable
COMMIT TRAN
END CATCH
```

CAPTURA DE INFORMACIÓN SOBRE EL ERROR SI ES NECESARIO

SQL Server proporciona varias funciones relacionadas con los errores a las que puede llamar dentro del bloque CATCH para registrar información acerca de los errores.

Por ejemplo, puede llamar a estos métodos y almacenar sus resultados en una tabla de registro de errores.

En la tabla siguiente se muestran las funciones de error.

Para obtener más información Para obtener más información acerca de los niveles de gravedad de los errores, vea “Niveles de gravedad de error del motor de base de datos” en los Libros en pantalla de SQL Server.

Función	Descripción
ERROR_LINE	Devuelve el número de línea donde se produjo el error que hizo que se ejecutara el código del bloque CATCH.
ERROR_MESSAGE	Devuelve información de diagnóstico acerca de la causa del error. Muchos mensajes de error tienen variables de sustitución donde se pone información, como el nombre del objeto que genera el error.
ERROR_NUMBER	Devuelve el número único del error que se produjo.
ERROR_PROCEDURE	Devuelve el nombre del procedimiento almacenado o del desencadenador donde se produjo el error.
ERROR_SEVERITY	Devuelve un valor que indica la gravedad del error. Los errores con una gravedad baja, como 1 ó 2, son mensajes informativos o advertencias de bajo nivel. Los errores con una gravedad alta indican problemas que deben corregirse lo antes posible.
ERROR_STATE	Devuelve el valor de estado. Algunos mensajes de error pueden producirse en varios lugares del código para el motor de base de datos. Cada condición específica que produce un error asigna un código de estado único. Esta información puede ser útil cuando está trabajando con artículos de Microsoft Knowledge Base para averiguar si el problema registrado es el mismo que el error que se ha producido.

[illegible]

CAPITULO 7

- Implementación de desencadenadores
- Uso de XML
- Recuperación de XML mediante FOR XML.
- Novedades en SQL 2012 y 2014

IMPLEMENTACIÓN DE DESENCADENADORES (TRIGGERS)

Un desencadenador es un tipo especial de procedimiento almacenado que se ejecuta cuando una instrucción INSERT, UPDATE o DELETE modifica los datos de una tabla especificada. Un desencadenador puede consultar otras tablas y puede incluir instrucciones Transact-SQL complejas. Los desencadenadores se crean normalmente para exigir integridad referencial o coherencia entre datos relacionados de forma lógica en tablas diferentes.

Puesto que los usuarios no pueden eludir los desencadenadores y tiene acceso a las características de Transact-SQL, puede utilizar desencadenadores para exigir lógica de negocios compleja que sea difícil o imposible de exigir mediante otros mecanismos de integridad de datos.

Tenga en cuenta lo siguiente acerca de los desencadenadores:

- El desencadenador y la instrucción que lo activa se tratan como una única transacción, que se puede deshacer desde dentro del desencadenador. Si se detecta un error grave (por ejemplo, espacio insuficiente en el disco), se deshace automáticamente toda la transacción.
- Los desencadenadores pueden realizar cambios en cascada en tablas relacionadas de la base de datos; sin embargo, estos cambios se pueden ejecutar de manera más eficiente mediante el uso de restricciones de integridad referencial en cascada.
- Los desencadenadores pueden proteger frente a operaciones de inserción, actualización y eliminación malintencionadas o incorrectas, y pueden exigir otras restricciones más complejas que las definidas mediante restricciones CHECK.
- Los desencadenadores pueden hacer referencia a columnas de otras tablas, a diferencia de las restricciones CHECK. Por ejemplo, un desencadenador puede utilizar una instrucción SELECT de otra tabla para comparar con los datos insertados o actualizados y realizar acciones adicionales, como modificar los datos o mostrar un mensaje de error definido por el usuario.
- Los desencadenadores pueden evaluar el estado de una tabla antes y después de una modificación de datos, y realizar acciones basándose en esa diferencia.
- Varios desencadenadores del mismo tipo (INSERT, UPDATE o DELETE) en una tabla permiten realizar distintas acciones como respuesta a la misma instrucción de modificación.

CREACIÓN DE DESENCADENADORES

Puede crear desencadenadores mediante la instrucción CREATE TRIGGER de Transact-SQL. La instrucción CREATE TRIGGER tiene la sintaxis siguiente.

```
CREATE TRIGGER [ nombreDeEsquema . ] nombreDeDesencadenador
ON { table | view }
[ WITH <opciónDesencadenadorDml> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { instrucciónSql [ ; ] [ ...n ] | EXTERNAL NAME <especificadorDeMétodo>
```

[;] > }

TIPOS DE DESENCADENADORES

Hay dos categorías de desencadenadores DML:

DESENCADENADORES AFTER. Los desencadenadores AFTER se ejecutan después de realizarse la acción de la instrucción INSERT, UPDATE o DELETE. Especificar AFTER es igual que especificar FOR, que es la única opción disponible en las versiones anteriores de Microsoft SQL Server. Sólo puede definir desencadenadores AFTER en tablas.

DESENCADENADORES INSTEAD OF. Los desencadenadores INSTEAD OF se ejecutan en lugar de la acción de desencadenamiento habitual. Los desencadenadores INSTEAD OF también pueden definirse en vistas con una o más tablas base, donde pueden extender los tipos de actualizaciones que una vista puede aceptar.

DESENCADENADORES FRENTE A RESTRICCIONES

La ventaja principal de los desencadenadores es que pueden contener lógica de procesamiento compleja que utilice código Transact-SQL. Por tanto, los desencadenadores pueden aceptar un superconjunto de la funcionalidad de las restricciones. Sin embargo, la integridad de los datos siempre se debe exigir en el menor nivel posible mediante restricciones, siempre y cuando las restricciones cumplan los requisitos de la solución.

Los desencadenadores son más útiles cuando las características compatibles con las restricciones no pueden satisfacer las necesidades funcionales de la aplicación.

Por ejemplo:

- Las restricciones FOREIGN KEY pueden validar el valor de una columna sólo con una coincidencia exacta con un valor de otra columna, a menos que la cláusula REFERENCES defina una acción referencial en cascada.
- Las restricciones pueden informar de los errores a través de mensajes de error del sistema estándar. Si su aplicación requiere un control de errores más complejo, debe utilizar un desencadenador.
- Los desencadenadores puede realizar cambios en cascada en tablas relacionadas de la base de datos; sin embargo, estos cambios se pueden ejecutar de manera más eficiente mediante el uso de restricciones de integridad referencial en cascada:
- Los desencadenadores pueden impedir o deshacer los cambios que infrinjan la integridad referencial, cancelando de esta forma la modificación de datos que se intentaba realizar. Ese tipo de desencadenador puede entrar en vigor cuando cambia una clave externa y el nuevo valor no coincide con su clave principal. Sin embargo, las restricciones FOREIGN KEY suelen utilizarse para este fin.
- Si existen restricciones en la tabla de desencadenadores, se comprueban después de la ejecución del desencadenador INSTEAD OF pero antes de que se ejecute el desencadenador AFTER. Si se infringen las restricciones, se deshacen las acciones del desencadenador INSTEAD OF y no se ejecuta el desencadenador AFTER.

Cómo funciona un desencadenador INSERT

Un desencadenador INSERT es un desencadenador que se ejecuta cuando una instrucción INSERT inserta datos en una tabla o en una vista en la que se ha configurado el desencadenador.

Cuando un desencadenador INSERT se activa, se agregan nuevas filas tanto a la tabla de desencadenadores como a la tabla inserted. La tabla inserted es una tabla lógica que contiene una copia de las filas que se han insertado. La tabla inserted contiene la actividad de inserción registrada de la instrucción INSERT. La tabla inserted le permite hacer referencia a datos registrados de la instrucción INSERT de inicialización.

El desencadenador puede examinar la tabla inserted para determinar si se deben ejecutar las acciones del desencadenador y, en caso afirmativo, cómo deben realizarse.

Las filas de la tabla inserted siempre son duplicados de una o más filas de la tabla de desencadenadores.

Se registra toda la actividad de modificación de datos (instrucciones INSERT, UPDATE y DELETE), pero la información del registro de transacciones es ilegible. No obstante, la tabla inserted le permite hacer referencia a los cambios registrados producidos por la instrucción INSERT. Puede comparar entonces los cambios con los datos insertados para comprobarlos o para realizar otra acción. También puede hacer referencia a datos insertados sin necesidad de almacenar la información en variables.

IMPLEMENTACIÓN DE UN DESENCADENADOR INSERT

En el código siguiente se muestra la creación de un desencadenador INSERT denominado insrtWorkOrder en la tabla Production.WorkOrder de la base de datos AdventureWorks. Observe el uso de la tabla inserted para trabajar con los valores que han producido la ejecución del desencadenador.

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

Cómo funciona un desencadenador DELETE

Un desencadenador DELETE es un tipo especial de procedimiento almacenado que se ejecuta siempre que una instrucción DELETE elimina datos de una tabla o de una vista donde se ha configurado el desencadenador.

Cuando un desencadenador DELETE se activa, las filas eliminadas de la tabla afectada se colocan en una tabla deleted especial. La tabla deleted es una tabla lógica que contiene una copia de las filas que se han eliminado. La tabla deleted le permite hacer referencia a datos registrados de la instrucción DELETE de inicialización.

Tenga en cuenta lo siguiente cuando utilice el desencadenador DELETE:

- Cuando se anexa una fila a la tabla deleted, deja de existir en la tabla de la base de datos; por tanto, la tabla deleted y las tablas de la base de datos no tienen filas en común.
- Se asigna espacio de la memoria para crear la tabla deleted. La tabla deleted siempre está en la memoria caché.
- Un desencadenador definido para una acción DELETE no se ejecuta para la instrucción TRUNCATE TABLE porque TRUNCATE TABLE no se registra.

IMPLEMENTACIÓN DE UN DESENCADENADOR DELETE

En el código siguiente se muestra la creación de un desencadenador DELETE denominado delCustomer en la tabla Sales.Customer de la base de datos AdventureWorks.

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

Cómo funciona un desencadenador UPDATE

Un desencadenador UPDATE es un desencadenador que se ejecuta siempre que una instrucción UPDATE modifica datos en una tabla o en una vista en la que se ha configurado el desencadenador.

Se puede considerar que un desencadenador UPDATE consta de dos pasos:

1. El paso DELETE que captura la imagen anterior de los datos.
2. El paso INSERT que captura la imagen posterior de los datos.

Cuando se ejecuta una instrucción UPDATE en una tabla que tiene definido un desencadenador, las filas originales (imagen anterior) se mueven a la tabla deleted y las filas actualizadas (imagen posterior) se insertan en la tabla inserted.

El desencadenador puede examinar las tablas deleted e inserted, así como la tabla updated, para determinar si se han actualizado varias filas y cómo deben realizarse las acciones del desencadenador.

Puede definir un desencadenador para supervisar las actualizaciones de datos de una columna específica mediante la instrucción IF UPDATE. Esto permite al desencadenador aislar fácilmente la actividad para una columna concreta. Cuando detecta que la columna específica se ha actualizado, puede realizar la acción apropiada,

como mostrar un mensaje de error que indique que no se puede actualizar la columna o procesar una serie de instrucciones basándose en el valor de la columna recién actualizada.

IMPLEMENTACIÓN DE UN DESENCADENADOR UPDATE

En el código siguiente se muestra la creación de un desencadenador UPDATE denominado updtProductReview en la tabla Production.ProductReview de la base de datos AdventureWorks.

```
CREATE TRIGGER [updtProductReview] ON [Production].[ProductReview]
AFTER UPDATE NOT FOR REPLICATION AS
BEGIN
SET NOCOUNT ON;
UPDATE [Production].[ProductReview]
SET [Production].[ProductReview].[ModifiedDate] = GETDATE()
FROM inserted
WHERE inserted.[ProductReviewID] =
[Production].[ProductReview].[ProductReviewID];
```

Cómo funciona un desencadenador INSTEAD OF

Un desencadenador INSTEAD OF se ejecuta en lugar de la acción de desencadenamiento habitual. Los desencadenadores INSTEAD OF también pueden definirse en vistas con una o más tablas base, donde pueden extender los tipos de actualizaciones que una vista puede aceptar.

Este desencadenador se ejecuta en lugar de la acción de desencadenamiento original. Los desencadenadores INSTEAD OF aumentan la variedad de los tipos de actualizaciones que puede realizar en una vista. Cada tabla o cada vista está limitada a un desencadenador INSTEAD OF para cada acción de desencadenamiento (INSERT, UPDATE o DELETE).

Puede especificar un desencadenador INSTEAD OF tanto en tablas como en vistas. No se puede crear un desencadenador INSTEAD OF en vistas que tengan definido WITH CHECK OPTION.

En la lista siguiente se describen las ventajas principales de los desencadenadores INSTEAD OF:

- Permiten que las vistas formadas por varias tablas base acepten inserciones, actualizaciones y eliminaciones que hagan referencia a datos de las tablas.
- Le permiten crear lógica que pueda rechazar partes de un lote y permitir que otras partes de un lote se realicen correctamente.
- Le permiten especificar una acción alternativa de base de datos para las situaciones que cumplan las condiciones especificadas.

IMPLEMENTACIÓN DE UN DESENCADENADOR INSTEAD OF

En el código siguiente se muestra la creación de un desencadenador INSTEAD OF denominado delEmployee en la tabla HumanResources.Employee de la base de datos AdventureWorks.

```
CREATE TRIGGER [delEmployee] ON [HumanResources].[Employee]
INSTEAD OF DELETE NOT FOR REPLICATION AS
BEGIN
SET NOCOUNT ON;
DECLARE @DeleteCount int;
SELECT @DeleteCount = COUNT(*) FROM deleted;
IF @DeleteCount > 0
BEGIN
RAISERROR
(N'Employees cannot be deleted. They can only be marked as not
current.', -- Message
10, -- Severity.
1); -- State.
-- Roll back any active or uncommittable transactions
IF @@TRANCOUNT > 0
BEGIN
ROLLBACK TRANSACTION;
END
END;
END;
```

Cómo funcionan los desencadenadores anidados

Cualquier desencadenador puede contener una instrucción UPDATE, INSERT o DELETE que afecte otra tabla. Los desencadenadores se anidan cuando un desencadenador realiza una acción que inicia otro desencadenador.

Puede controlar si se anidan desencadenadores utilizando la opción de configuración del servidor nested triggers. El anidamiento se habilita en la instalación y se establece en el nivel de servidor, pero puede deshabilitarlo y volverlo a habilitar utilizando el procedimiento almacenado del sistema sp_configure.

Es posible anidar desencadenadores hasta una profundidad de 32 niveles. Cuando un desencadenador de una cadena anidada provoca un bucle infinito, se excede la capacidad de anidamiento. Entonces, el desencadenador termina y deshace la transacción. Puede utilizar desencadenadores anidados para realizar funciones, como hacer copia de seguridad de las filas afectadas por un desencadenador anterior.

Tenga en cuenta lo siguiente cuando utilice desencadenadores anidados:

- De forma predeterminada, la opción de configuración nested triggers está activada.
- Un desencadenador anidado no se activará dos veces en la misma transacción de desencadenador; un desencadenador no se llamará a sí mismo como respuesta a una segunda actualización de la misma tabla dentro del

desencadenador. Sin embargo, si un desencadenador modifica una tabla que provoca la activación de otro desencadenador y el segundo desencadenador modifica la tabla original, el desencadenador original se activará de forma recursiva. Para impedir la recursión indirecta de este tipo, desactive la opción nested triggers.

- Puesto que un desencadenador es una transacción, un error en cualquier nivel de un conjunto de desencadenadores anidados cancela toda la transacción y se deshacen todas las modificaciones de datos. Por tanto, debe incluir instrucciones PRINT cuando pruebe desencadenadores para poder averiguar dónde se produjo el error.

NIVELES DE ANIDAMIENTO

El nivel de anidamiento aumenta cada vez que se activa el desencadenador anidado. SQL Server acepta hasta 32 niveles de anidamiento, pero quizás desee limitar los niveles para no superar el nivel máximo de anidamiento. Puede utilizar la función @@NESTLEVEL para ver los niveles actuales de anidamiento.

DESHABILITAR DESENCADENADORES ANIDADOS

El anidamiento es una característica eficaz que puede utilizar para mantener la integridad de los datos en toda una base de datos. Sin embargo, en ocasiones quizás desee deshabilitar el anidamiento. Si el anidamiento está deshabilitado, un desencadenador no puede realizar una acción en cascada (una acción que inicie otro desencadenador que a su vez inicie otro desencadenador y así sucesivamente).

Puede decidir deshabilitar el anidamiento por diversos motivos:

- Los desencadenadores anidados requieren un diseño complejo y bien pensado. Los cambios en cascada pueden conducir a modificaciones de datos no previstas.
- Una modificación de datos en cualquier punto de una serie de desencadenadores anidados provoca la serie de desencadenadores. Si bien esto ofrece una protección eficaz de los datos, puede resultar un problema si las tablas deben actualizarse en un orden concreto.

Utilice la instrucción siguiente para deshabilitar el anidamiento.

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

Uso de XML

El trabajo con el lenguaje de marcado extensible (XML) es un requisito frecuente de muchas aplicaciones actuales. Hay muchos casos en los que un desarrollador de aplicaciones debe transformar datos entre los formatos XML y relacional, e incluso

almacenar datos XML y manipular datos de forma nativa en una base de datos relacional.

La cláusula FOR XML es fundamental para la recuperación de datos XML en SQL Server 2014. Esta cláusula indica al motor de consulta de SQL Server que devuelva los datos como una secuencia XML en lugar de como un conjunto de filas.

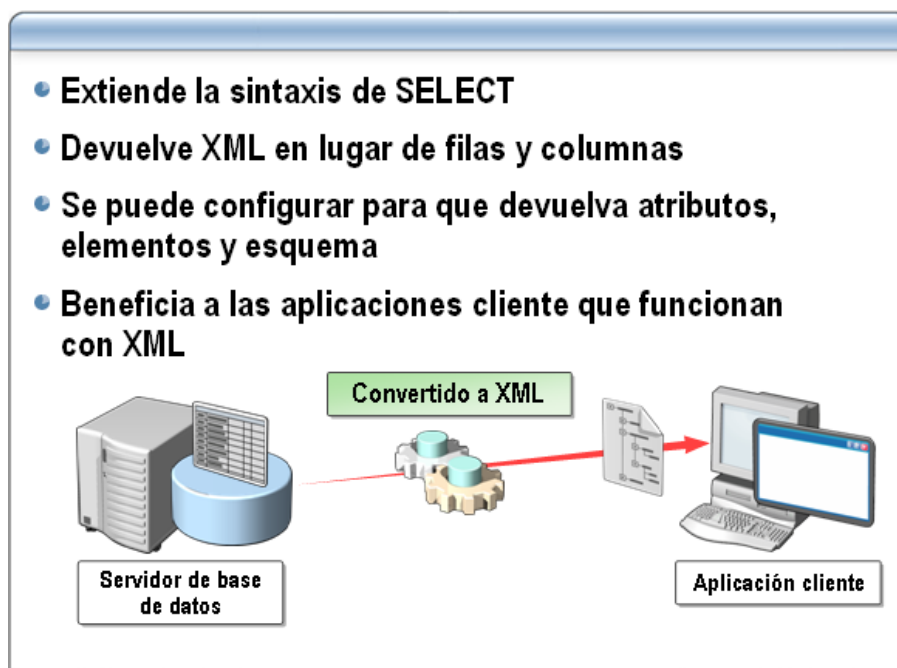
Los desarrolladores de aplicaciones pueden generar soluciones que extraen documentos empresariales XML como pedidos, facturas o catálogos directamente desde la base de datos.

- **Introducción a la cláusula FOR XML**
- **Qué son las consultas en modo RAW**
- **Qué son las consultas en modo AUTO**
- **Qué son las consultas en modo EXPLICIT**
- **Qué son las consultas en modo PATH**
- **Sintaxis para recuperar datos XML anidados**
- **Ejercicio: Uso de FOR XML**

Recuperación de XML mediante FOR XML.

Puede utilizar la cláusula FOR XML en una instrucción SELECT de Transact-SQL para recuperar datos como XML en lugar de como filas y columnas. Puede controlar el formato de los datos XML especificando uno de cuatro modos posibles: RAW, AUTO, EXPLICIT o PATH. Además, puede especificar diversas opciones para controlar el resultado.

La sintaxis de FOR XML La cláusula FOR XML se anexa a la instrucción SELECT, como se muestra en la sintaxis siguiente.



FOR XML

```
{
{ RAW [ ( 'NombreDeElemento' ) ] | AUTO }
[
<DirectivasComunes>
[ , { XMLDATA | XMLSCHEMA [ ( 'URIEspacioDeNombresDeDestino' ) ] } ]
[ , ELEMENTS [ XSINIL | ABSENT ]
]
| EXPLICIT
[
<DirectivasComunes>
[ , XMLDATA ]
]
| PATH [ ( 'NombreDeElemento' ) ]
[
<DirectivasComunes>
[ , ELEMENTS [ XSINIL | ABSENT ] ]
]
}
<DirectivasComunes> ::=
[ , BINARY BASE64 ]
[ , TYPE ]
[ , ROOT [ ( 'NombreRaíz' ) ] ]
```

En la tabla siguiente se describen los modos y las opciones de FOR XML más utilizados.

Modo/opción	Descripción
Modo RAW	Transforma cada fila del conjunto de resultados en un elemento XML que tiene una fila de identificador genérico como etiqueta de elemento. Puede especificar opcionalmente un nombre para el elemento de fila cuando utilice esta directiva.
Modo AUTO	Devuelve el resultado de una consulta como un árbol XML sencillo anidado. Cada tabla de la cláusula FROM para la cual se muestra al menos una columna en la cláusula SELECT se representa como un elemento XML. Las columnas mostradas en la cláusula SELECT se asignan a los atributos de elemento apropiados.
Modo EXPLICIT	Un formato personalizado especificado en la consulta da formato a los datos XML resultantes.
Modo PATH	Proporciona una forma más sencilla de combinar elementos y atributos, y de introducir niveles de anidamiento adicionales para representar propiedades complejas.
Opción ELEMENTS	Devuelve columnas como subelementos en lugar de como atributos para los modos RAW, AUTO y PATH.
Opción BINARY BASE64	Devuelve campos de datos binarios, como imágenes, como datos binarios codificados en base 64.
Opción ROOT	Agrega un elemento de nivel superior a los datos XML resultantes. Cuando una consulta devuelve varias filas como datos XML, los resultados no aparecen de manera predeterminada en un único elemento raíz. Los datos XML sin un único elemento raíz se denominan fragmento. Sin embargo, si necesita devolver un documento XML bien formado con un único elemento raíz, debe especificar esta opción. Puede especificar opcionalmente un nombre para este elemento raíz.
Opción TYPE	Devuelve los resultados de la consulta como el tipo de datos xml.
Opción XMLDATA	Devuelve un esquema XML con datos reducidos (XDR) incorporado.
Opción XMLSCHEMA	Devuelve un esquema XML (XSD) de World Wide Web Consortium (W3C) incorporado.

EJEMPLOS DE USO DE FOR XML

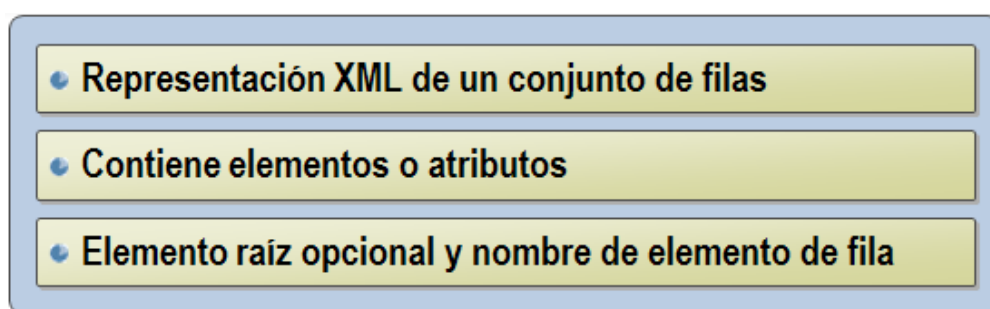
Hay algunas situaciones en las que quizás desee recuperar datos en formato XML en lugar de como un conjunto de filas. Por ejemplo, imagine las siguientes situaciones de acceso a datos:

- Recuperación de datos para su publicación en un sitio web Si recupera los datos como XML, puede aplicar una hoja de estilos del Lenguaje de transformación basado en hojas de estilo (XSLT) para representar los datos como el Lenguaje de marcado de hipertexto (HTML). También puede aplicar distintas hojas de estilos a los mismos datos XML para generar formatos de presentación alternativos para diferentes dispositivos cliente sin necesidad de volver a escribir la lógica de acceso a datos.

- Recuperación de datos para intercambiarlos con un asociado comercial XML es un formato natural para los datos que desea enviar a un asociado comercial. Si recupera datos empresariales en el formato XML, puede integrar fácilmente sus sistemas con organizaciones externas, independientemente de las tecnologías de datos que utilicen internamente.

Qué son las consultas en modo RAW

Las consultas en modo RAW se utilizan para recuperar una representación XML de un conjunto de filas. Las aplicaciones pueden procesar XML en el formato genérico o pueden aplicar una hoja de estilos XSLT para transformar XML en el formato de documento empresarial o en la representación de la interfaz de usuario necesarios.



Tenga en cuenta las siguientes características del modo RAW:

- Un elemento representa cada fila del conjunto de resultados que devuelve la consulta.
- Un atributo con el mismo nombre que el nombre o el alias de columna utilizado en la consulta representa cada columna del conjunto de resultados, a menos que se especifique la opción ELEMENTS, en cuyo caso cada columna se asigna a un sub elemento del elemento de fila.
- Las consultas en modo RAW pueden incluir columnas agregadas y cláusulas GROUP BY.

RECUPERACIÓN DE DATOS EN ELEMENTOS DE FILA GENÉRICOS

En el ejemplo siguiente se muestra cómo recuperar un fragmento XML que contiene datos de pedidos utilizando una consulta FOR XML en modo RAW.

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

Esta consulta produce un fragmento XML en el formato que contiene elementos <row> genéricos, como se muestra en el ejemplo siguiente.

```
<row CustID="1" CustomerType="S" SalesOrderID="43860"/>
<row CustID="1" CustomerType="S" SalesOrderID="44501"/>
```

Observe que la columna CustomerID utiliza el alias CustID, que determina el nombre de atributo.

RECUPERACIÓN DE DATOS COMO ELEMENTOS

En el ejemplo siguiente se muestra cómo recuperar los mismos datos como elementos, en lugar de como atributos, especificando la opción ELEMENTS.

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

Esta consulta produce un fragmento XML en el formato que se muestra en el ejemplo siguiente.

```
<row>
<CustID>1</CustID>
<CustomerType>S</CustomerType>
<SalesOrderID>43860</SalesOrderID>
</row>
<row>
<CustID>1</CustID>
<CustomerType>S</CustomerType>
<SalesOrderID>44501</SalesOrderID>
</row>
```

RECUPERACIÓN DE DATOS UTILIZANDO UN ELEMENTO RAÍZ Y UN NOMBRE DE ELEMENTO DE FILA PERSONALIZADO

En el ejemplo siguiente se muestra cómo recuperar los mismos datos utilizando un elemento raíz especificado con la opción ROOT y cómo modificar el nombre del elemento de fila utilizando el argumento opcional del modo RAW.

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

Esta consulta produce un documento XML bien formado el formato que se muestra en el ejemplo siguiente.

```
<Orders>
<Order CustID="1" CustomerType="S" SalesOrderID="43860"/>
```

```
<Order CustID="1" CustomerType="S" SalesOrderID="44501"/>
</Orders>
```

Tenga en cuenta que si también se especifica la opción ELEMENTS, las filas resultantes contendrán elementos en lugar de atributos.

Qué son las consultas en modo AUTO

Las consultas en modo AUTO producen representaciones XML de entidades de datos. Tenga en cuenta las siguientes características del modo AUTO:

- Cada fila devuelta por la consulta se representa mediante un elemento XML que tiene el mismo nombre que la tabla desde la que se recuperó (o el alias empleado en la consulta).
- Cada instrucción JOIN de la consulta produce un elemento XML anidado, lo que reduce la duplicación de datos en el fragmento XML resultante. El orden de las instrucciones JOIN afecta el orden de los elementos anidados.
- Para asegurarse de que los elementos secundarios se intercalan correctamente con sus elementos primarios, utilice una cláusula ORDER BY para devolver los datos en el orden jerárquico correcto.
- Cada columna del conjunto de resultados está representada por un atributo, a menos que se especifique la opción ELEMENTS, en cuyo caso cada columna se representa mediante un elemento secundario.
- En las consultas en modo AUTO no pueden utilizarse columnas agregadas ni cláusulas GROUP BY (aunque puede utilizar una consulta en modo AUTO para recuperar datos agregados desde una vista que utilice una cláusula GROUP BY).

- **Representación XML de entidades de datos**
- **Anidan datos basándose en la precedencia de combinación**
- **Pueden utilizar opciones como ELEMENTS y ROOT**

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

```
<Cust CustID="1" CustomerType="S">
  <Order SalesOrderID="43860" />
  <Order SalesOrderID="44501" />
  ...
</Cust>
<Cust CustID="2" CustomerType="S">
  ...
```

RECUPERACIÓN DE DATOS ANIDADOS UTILIZANDO EL MODO AUTO

En el ejemplo siguiente se muestra cómo utilizar una consulta en modo AUTO para devolver un fragmento XML que contiene una lista de pedidos.

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

Esta consulta produce un fragmento XML en el formato que se muestra en el ejemplo siguiente.

```
<Cust CustID="1" CustomerType="S">
<Order SalesOrderID="43860"/>
<Order SalesOrderID="44501"/>
</Cust>
```

Observe que la columna CustomerID y las tablas Customer y SalesOrderHeader utilizan alias para determinar los nombres de atributo y de elemento.

RECUPERACIÓN DE DATOS COMO ELEMENTOS

En el ejemplo siguiente se muestra cómo recuperar los mismos datos como elementos, en lugar de como atributos, especificando la opción ELEMENTS.

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

Esta consulta produce un fragmento XML en el formato que se muestra en el ejemplo siguiente.

```
<Cust>
<CustID>1</CustID>
<CustomerType>S</CustomerType>
<Order>
<SalesOrderID>43860</SalesOrderID>
</Order>
<Order>
<SalesOrderID>44501</SalesOrderID>
</Order>
</Cust>
```

...

Como ocurre en el modo RAW, también puede utilizar la opción ROOT, como se muestra en el ejemplo siguiente.

```
SELECT Cust.CustomerID CustID, CustomerType, SalesOrderID
FROM Sales.Customer Cust JOIN Sales.SalesOrderHeader [Order]
```

```
ON Cust.CustomerID = [Order].CustomerID  
ORDER BY Cust.CustomerID  
FOR XML AUTO, ELEMENTS, ROOT('Orders')
```

Esta consulta produce un documento XML en el formato que se muestra en el ejemplo siguiente.

```
<Orders>  
<Cust>  
  <CustID>1</CustID>  
  <CustomerType>S</CustomerType>  
  <Order>  
    <SalesOrderID>43860</SalesOrderID>  
  </Order>  
  <Order>  
    <SalesOrderID>44501</SalesOrderID>  
  </Order>  
</Cust>  
...  
</Orders>
```

Qué son las consultas en modo EXPLICIT

Algunas veces, los documentos empresariales que debe intercambiar con asociados comerciales requieren un formato XML que no puede recuperarse mediante consultas en modo RAW o AUTO. Cuando los datos de una tabla se asignan a un elemento XML, las columnas de la tabla pueden representarse como:

- El valor del elemento.
- Un atributo.
- Un elemento secundario.

Representaciones tabulares de documentos XML

Tag	Parent	Invoice!1!InvoiceNo	Invoice!1!Date!Element
1	NULL	43659	2001-07-01T00:00:00
1	NULL	43660	2001-07-02T00:00:00

Permiten un control total sobre el formato XML

```

SELECT 1 AS Tag, NULL AS Parent,
       SalesOrderID AS [Invoice!1!InvoiceNo],
       OrderDate AS [Invoice!1!Date!Element]
FROM    SalesOrderHeader
FOR XML EXPLICIT

```

```

<Invoice InvoiceNo="43659">
  <Date>2001-07-01T00:00:00</Date>
</Invoice>
<Invoice InvoiceNo="43660">...

```

TABLAS UNIVERSALES

La clave para entender la recuperación de documentos XML personalizados es el concepto de tabla universal. Una tabla universal es una representación tabular de un documento XML. Cada fila de una tabla universal representa datos que se representarán como un elemento en el documento XML resultante.

Las dos primeras columnas de una tabla universal definen la posición jerárquica en el documento XML resultante del elemento que contiene los datos para la fila.

Estas columnas son:

Tag. Un valor numérico que identifica de manera única la etiqueta para el elemento que contiene los datos de esta fila

Parent. Un valor numérico que identifica la etiqueta primaria inmediata para este elemento

Cada etiqueta XML diferente del documento resultante que se asigna a una tabla o una vista en la base de datos debe representarse mediante un valor Tag diferente en la tabla universal. El valor Parent determina la posición jerárquica de la etiqueta en el documento resultante. Las etiquetas situadas en el nivel superior del fragmento XML (y, por tanto, que no tienen ningún elemento primario inmediato) tienen un valor primario de NULL. Por ejemplo, el documento XML descrito anteriormente contiene un elemento Invoice

sin ningún elemento primario y un elemento Lineltem, que es secundario del elemento Invoice. En la tabla universal, se asigna un número Tag arbitrario a las etiquetas de los dos elementos para identificarlas y la columna Parent se utiliza para definir cómo se anidan los elementos.

DEFINICIÓN DE ASIGNACIONES DE COLUMNA EN UNA TABLA UNIVERSAL

Las columnas restantes de una tabla universal contienen los datos que se representarán en el documento. El nombre de columna especifica si los datos se representarán como un valor de elemento, como un atributo o como un elemento secundario. Las columnas de datos de una tabla universal tienen un nombre que consta de cuatro partes como máximo, con el formato siguiente.

NombreDeElemento ! NúmeroDeEtiqueta ! NombreDeAtributo ! Directiva
En la tabla siguiente se describen las partes de un nombre de columna.

Qué son las consultas en modo PATH

El modo PATH produce datos XML personalizados utilizando una sintaxis denominada XPath para asignar valores a atributos, elementos, subelementos, nodos de texto y valores de datos. Esta posibilidad de asignar columnas relacionales a nodos XML permite generar datos XML complejos sin la complejidad de una consulta en modo EXPLICIT.

- **Utilice XPath para especificar formato XML**
- **Permiten la creación de datos anidados**
- **Más fácil de utilizar que el modo EXPLICIT**

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

```
<row EmpID="1">
  <EmpName>
    <First>Guy</First>
    <Last>Gilbert</Last>
  </EmpName>
</row>
...
```

Tenga en cuenta las siguientes características de la sintaxis XPath:

- Los nodos XML de un árbol se expresan como rutas de acceso, separados por barras diagonales (/).
- Los atributos se indican con un signo (@) como prefijo.
- Las rutas de acceso relativas pueden indicarse utilizando un único punto (.) para representar el nodo actual y dos puntos seguidos (..) para representar el primario del nodo actual.

RECUPERACIÓN DE DATOS UTILIZANDO EL MODO PATH

En el ejemplo siguiente se muestra cómo utilizar una consulta en modo PATH para devolver un fragmento XML que contiene una lista de empleados.

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

Esta consulta produce un fragmento XML en el formato que se muestra en el ejemplo siguiente.

```
<row EmpID="1">  
  <EmpName>  
    <First>Guy</First>  
    <Last>Gilbert</Last>  
  </EmpName>  
</row>  
<row EmpID="2">  
  <EmpName>  
    <First>Kevin</First>  
    <Last>Browne</Last>  
  </EmpName>  
</row>
```

Observe que la columna EmployeeID se asigna al atributo EmpID con un signo (@). Las columnas FirstName y LastName se asignan como subelementos del elemento EmpName con la barra diagonal (/).

MODIFICACIÓN DEL NOMBRE DEL ELEMENTO DE FILA

En el ejemplo siguiente se muestra cómo se puede utilizar el argumento opcional NombreDeElemento de la consulta en modo PATH para modificar el nombre del elemento de fila predeterminado.

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

Esta consulta produce un fragmento XML en el formato que se muestra en el ejemplo siguiente.

```
<Employee EmpID="1">  
  <EmpName>
```



```
<First>Guy</First>
<Last>Gilbert</Last>
</EmpName>
</Employee>
<Employee EmpID="2">
  <EmpName>
    <First>Kevin</First>
    <Last>Browne</Last>
  </EmpName>
</Employee>
```

Observe que la columna EmployeeID se asigna al atributo EmpID (@EmpID), y las columnas FirstName y LastName se asignan como subelementos del elemento EmpName (EmpName/First y EmpName/Last)

Funciones del SQL Server 2012 y 2014

El lenguaje de expresiones incluye un conjunto de funciones que pueden usarse en las expresiones.

Las expresiones pueden usar una sola función, pero generalmente utilizan varias funciones, combinándolas con operadores.

Las funciones pueden clasificarse en los grupos siguientes:

- Funciones matemáticas que realizan cálculos basados en valores numéricos de entrada que se proporcionan como parámetros a la función y devuelven valores numéricos.
- Funciones de cadenas de caracteres que realizan operaciones en valores de entrada de cadena o de tipo hexadecimal y que devuelven una cadena o un valor numérico.
- Funciones de fecha y hora que realizan operaciones en valores de fecha y hora, y devuelven valores de tipo cadena, numéricos o de fecha y hora.
- Funciones del sistema que devuelven información sobre una expresión.

El lenguaje de expresiones proporciona las siguientes funciones matemáticas.

Función	Descripción
ABS	Devuelve el valor absoluto (positivo) de una expresión numérica.
EXP	Devuelve el exponente de la base e de la expresión especificada.
CEILING	Devuelve el menor entero mayor o igual que una expresión numérica.
FLOOR	Devuelve el mayor entero que es menor o igual que una expresión numérica.
LN	Devuelve el logaritmo natural de una expresión numérica.
LOG	Devuelve el logaritmo en base 10 de una expresión numérica.

<u>POWER</u>	Devuelve el resultado de elevar una expresión numérica a una determinada potencia.
<u>ROUND</u>	Devuelve una expresión numérica, redondeada a la longitud o precisión especificada.
<u>SIGN</u>	Devuelve el signo positivo (+), cero (0) o negativo (-) de una expresión numérica.
<u>SQUARE</u>	Devuelve el cuadrado de una expresión numérica.
<u>SQRT</u>	Devuelve la raíz cuadrada de una expresión numérica.

El evaluador de expresiones proporciona las siguientes funciones para cadenas.

Función	Descripción
<u>CODEPOINT</u>	Devuelve el valor de código Unicode del carácter más a la izquierda de una expresión de caracteres.
<u>FINDSTRING</u>	Devuelve el índice (de base 1) de la repetición especificada de una cadena de caracteres dentro de una expresión.
<u>HEX</u>	Devuelve una cadena que representa el valor hexadecimal de un entero.
<u>LEN</u>	Devuelve el número de caracteres de una expresión de caracteres.
<u>LEFT</u>	Devuelve el número de caracteres especificado de la parte más a la izquierda de la expresión de caracteres dada.
<u>LOWER</u>	Devuelve una expresión de caracteres después de convertir los caracteres en mayúsculas a minúsculas.
<u>LTRIM</u>	Devuelve una expresión de caracteres tras quitar todos los espacios iniciales en blanco.
<u>REPLACE</u>	Devuelve una expresión de caracteres tras reemplazar una cadena dentro de la expresión por otra cadena diferente o por la cadena vacía.
<u>REPLICATE</u>	Devuelve una expresión de caracteres replicada un determinado número de veces.
<u>REVERSE</u>	Devuelve una expresión de caracteres en orden inverso.
<u>RIGHT</u>	Devuelve el número de caracteres especificado de la parte más a la derecha de la expresión de caracteres dada.
<u>RTRIM</u>	Devuelve una expresión de caracteres después de quitar los espacios finales.
<u>SUBSTRING</u>	Devuelve una parte de una expresión de caracteres.
<u>TRIM</u>	Devuelve una expresión de caracteres después de quitar los espacios iniciales y finales.
<u>UPPER</u>	Devuelve una expresión de caracteres tras convertir los caracteres en minúsculas a mayúsculas.

El evaluador de expresiones proporciona las siguientes funciones de fecha y hora.

Función	Descripción
DATEADD	Devuelve un nuevo valor de tipo DT_DBTIMESTAMP agregando una fecha o un intervalo de tiempo a una fecha indicada.
DATEDIFF	Devuelve el número de límites de fecha y hora entre dos fechas especificadas.
DATEPART	Devuelve un entero que representa una parte de una fecha.
DAY	Devuelve un entero que representa la parte del día de la fecha especificada.
GETDATE	Devuelve la fecha actual del sistema.
GETUTCDATE	Devuelve el valor de fecha y hora que representa la hora UTC actual (Hora universal coordinada u Hora media de Greenwich).
MONTH	Devuelve un entero que representa la parte del mes de la fecha especificada.
YEAR	Devuelve un entero que representa la parte del año de la fecha especificada.

El evaluador de expresiones proporciona las siguientes funciones para valores NULL.

Función	Descripción
ISNULL	Devuelve un resultado booleano en función de si una expresión es NULL.
NULL	Devuelve un valor NULL asociado al tipo de datos solicitado.

Los nombres de expresión se muestran en mayúsculas, pero no se distinguen mayúsculas de minúsculas.

Por ejemplo, "null" es equivalente a "NULL".

.

Nuevas Clausulas del SQL Server 2012 y 2014

El lenguaje de expresiones incluye un conjunto de funciones que pueden usarse en las expresiones.

OVER (cláusula de Transact-SQL)

Determina las particiones y el orden de un conjunto de filas antes de que se aplique la función de ventana asociada. Es decir, la cláusula OVER define una ventana o un conjunto de filas definido por el usuario en un conjunto de resultados de la consulta. Una función de ventana calcula entonces un valor para cada fila de la ventana. Puede utilizar la cláusula OVER con funciones para calcular valores agregados tales como medias móviles, agregados acumulados, totales acumulados o N elementos superiores por resultados del grupo.

Se aplica a:

- Funciones de categoría
- Funciones de agregado

- Funciones analíticas
- Función NEXT VALUE FOR

Ejemplos

1. Utilizando la cláusula OVER con la función ROW_NUMBER

En el ejemplo siguiente se muestra cómo usar la cláusula OVER con la función ROW_NUMBER para mostrar un número de fila para cada fila de una partición. La cláusula ORDER BY especificada en la cláusula OVER ordena las filas de cada partición por la columna SalesYTD. La cláusula ORDER BY en la instrucción SELECT determina el orden en que se devuelve el conjunto completo de resultados de la consulta.

```
USE AdventureWorks2012;
GO
SELECT ROW_NUMBER() OVER(PARTITION BY PostalCode ORDER BY SalesYTD
DESC) AS "Row Number",
    p.LastName, s.SalesYTD, a.PostalCode
FROM Sales.SalesPerson AS s
    INNER JOIN Person.Person AS p
        ON s.BusinessEntityID = p.BusinessEntityID
    INNER JOIN Person.Address AS a
        ON a.AddressID = p.BusinessEntityID
WHERE TerritoryID IS NOT NULL
    AND SalesYTD <> 0
ORDER BY PostalCode;
GO
```

El conjunto de resultados es el siguiente.

Row Number	LastName	SalesYTD	PostalCode
1	Mitchell	4251368.5497	98027
2	Blythe	3763178.1787	98027
3	Carson	3189418.3662	98027
4	Reiter	2315185.611	98027
5	Vargas	1453719.4653	98027

2. Utilizar la cláusula OVER con funciones de agregado

En el ejemplo siguiente se utiliza la cláusula OVER con funciones de agregado en todas las filas devueltas por la consulta. En este ejemplo, el uso de OVER es más eficaz que usar subconsultas para obtener los valores agregados.

```
USE AdventureWorks2012;
GO
SELECT SalesOrderID, ProductID, OrderQty
    ,SUM(OrderQty) OVER(PARTITION BY SalesOrderID) AS Total
    ,AVG(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Avg"
    ,COUNT(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Count"
    ,MIN(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Min"
    ,MAX(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Max"
FROM Sales.SalesOrderDetail
WHERE SalesOrderID IN(43659,43664);
```

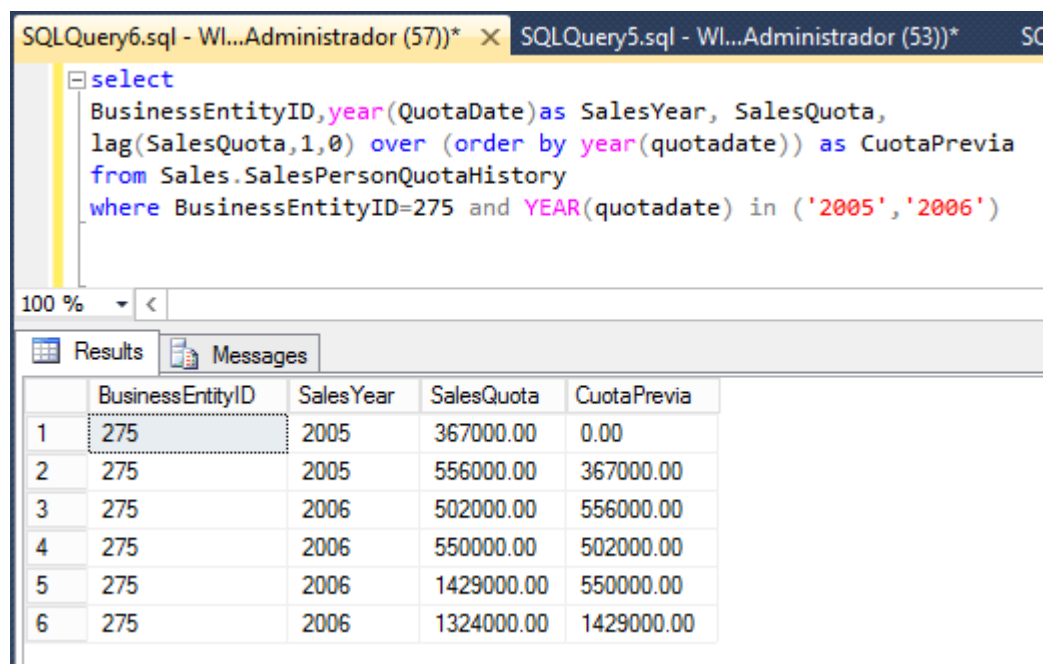
GO

El conjunto de resultados es el siguiente.

SalesOrderID	ProductID	OrderQty	Total	Avg	Count	Min	Max
43659	776	1	26	2	12	1	6
43659	777	3	26	2	12	1	6
43659	778	1	26	2	12	1	6
43659	771	1	26	2	12	1	6
43659	772	1	26	2	12	1	6
43659	773	2	26	2	12	1	6
43659	774	1	26	2	12	1	6
43659	714	3	26	2	12	1	6
43659	716	1	26	2	12	1	6
43659	709	6	26	2	12	1	6
43659	712	2	26	2	12	1	6
43659	711	4	26	2	12	1	6

LAG (Transact-SQL)

Accede a los datos de una fila anterior en el mismo conjunto de resultados sin el uso de un auto-join en SQL Server 2016. GAL proporciona acceso a una fila a la física dada desplazamiento que viene antes de la fila actual. Utilice esta función analítica en una sentencia SELECT para comparar los valores de la fila actual con los valores en una fila anterior.



The screenshot shows a SQL query window with the following query:

```
select
BusinessEntityID, year(QuotaDate) as SalesYear, SalesQuota,
lag(SalesQuota, 1, 0) over (order by year(quotadate)) as CuotaPrevia
from Sales.SalesPersonQuotaHistory
where BusinessEntityID=275 and YEAR(quotadate) in ('2005', '2006')
```

The results pane shows the following data:

	BusinessEntityID	SalesYear	SalesQuota	CuotaPrevia
1	275	2005	367000.00	0.00
2	275	2005	556000.00	367000.00
3	275	2006	502000.00	556000.00
4	275	2006	550000.00	502000.00
5	275	2006	1429000.00	550000.00
6	275	2006	1324000.00	1429000.00

LEAD (Transact-SQL)

Accede a los datos de una fila posterior en el mismo conjunto de resultados sin el uso de un auto-join en SQL Server 2016. LEAD proporciona acceso a una fila a la física dada desplazamiento que sigue la fila actual. Utilice esta función analítica en una sentencia SELECT para comparar los valores de la fila actual con los valores en una fila siguiente.

SQLQuery6.sql - WL...Administrador (57))* X SQLQuery5.sql - WL...Administrador (53))* SQLC

```

select
BusinessEntityID, year(QuotaDate) as SalesYear, SalesQuota,
lead(SalesQuota, 1, 0) over (order by year(quotadate)) as ProximaCuota
from Sales.SalesPersonQuotaHistory
where BusinessEntityID=275 and YEAR(quotadate) in ('2005', '2006')

```

100 % <

Results Messages

	BusinessEntityID	SalesYear	SalesQuota	ProximaCuota
1	275	2005	367000.00	556000.00
2	275	2005	556000.00	502000.00
3	275	2006	502000.00	550000.00
4	275	2006	550000.00	1429000.00
5	275	2006	1429000.00	1324000.00
6	275	2006	1324000.00	0.00

FIRST_VALUE (Transact-SQL)

Devuelve el primer valor en un conjunto ordenado de valores en SQL Server 2016.

SQLQuery6.sql - WL...Administrador (57))* X SQLQuery5.sql - WL...Administrador (53))*

```

select
Name, ListPrice,
FIRST_VALUE(Name) over (order by listprice asc) as LeastExpensive
from Production.Product
where ProductSubcategoryID=37

```

100 % <

Results Messages

	Name	ListPrice	LeastExpensive
1	Patch Kit/8 Patches	2.29	Patch Kit/8 Patches
2	Road Tire Tube	3.99	Patch Kit/8 Patches
3	Touring Tire Tube	4.99	Patch Kit/8 Patches
4	Mountain Tire Tube	4.99	Patch Kit/8 Patches
5	LL Road Tire	21.49	Patch Kit/8 Patches
6	ML Road Tire	24.99	Patch Kit/8 Patches
7	LL Mountain Tire	24.99	Patch Kit/8 Patches
8	Touring Tire	28.99	Patch Kit/8 Patches



Daniel Ramos Castañeda

danielramos@dba.pe



Certificaciones:

MCP, MCTS, MAP, MCPS, MCNPS

Especializaciones:

MCPS 2.0 - AOS: Business Intelligence Competency 1
MCPS 2.0 - AOS: Competency- Integrated E-Business Solutions
AOS: Application Integration Competency
MCPS 2.0 - AOS: Data Platform Competency
MCTS: SQL Server® 2005 y 2008
AOS: Business Intelligence Competency 2
AOS : BUSINESS INTELLIGENCE COMPETENCY - BUSINESS INTELLIGENCE PLATFORM
AOS: Data Management Solutions Competency-Database Management for SQL Sever
AOS : Microsoft Dynamics AX - SQL server
AOS : ERP Competency - Microsoft Dynamics GP - SQL server
AOS : Microsoft Dynamics NAV - SQL server
AOS : Microsoft Dynamics SL - SQL Server
AOS : Microsoft Dynamics Point of Sale - SQL Server



ÍNDICE

Capítulo 1	1
Concepto de base de datos relacional.	2
Sistemas administradores de bases de datos.	2
¿Cómo guarda los datos SQL Server 2014?	2
Creación de bases de datos.	3
Consideraciones a la hora de planear una base de datos	4
Componentes lógicos: registro de datos y registro de transacciones.	5
Proceso de registro	6
Orígenes de información de las bases de datos	7
SQL Server Management Studio	8
Vistas de catálogo	8
Funciones de metadatos	9
Procedimientos almacenados del sistema.....	10
Componentes físicos: archivos y grupos de archivos de bases de datos.....	10
Cuándo crear grupos de archivos	12
Creación de esquemas.....	14
Esquemas como espacios de nombres	14
Cómo funciona la resolución de nombres de objetos	15
Cómo funciona la resolución de nombres.....	16
Creación de instantáneas de base de datos	17
Restricciones para crear instantáneas.....	18
Cómo funcionan las instantáneas de base de datos	18
Modificación de datos.....	19
Recuperación de datos.....	19
Capítulo 2	21
Creación de tipos de datos y tablas.....	22
Los tipos de datos del sistema que ofrece SQL Server 2014.....	22
Tipos de datos suministrados por el sistema.....	22
Tipos de datos numéricos exactos y numéricos aproximados	24
Tipos de datos fijos y variables	24
Valores de datos grandes	24
Qué son los tipos de datos de alias	25
Cuándo crear tipos de datos de alias.....	25
Ejemplo de creación de un tipo de datos de alias	25
Las tablas, relación y aplicación a las tablas de sus restricciones.....	26
Cómo organiza SQL Server los datos en filas.....	26
El encabezado de fila	26
La parte de datos.....	27
Cómo organiza SQL Server los valores de datos grandes	27
Datos de objetos grandes (LOB).....	27
El especificador max	28
Consideraciones para la creación de tablas.....	29
Intercalación de columnas.....	30
Capacidad de aceptar valores NULL	31
Tipos de columna especiales	31
Ejemplo de creación de una tabla	31
Modificación y eliminación de tablas.....	32
Ejemplo de creación de una tabla ancha	32

Tipos de integridad de datos	33
Integridad de dominio.....	33
Integridad de entidad	33
Integridad referencial	33
Creación de restricciones	34
Restricciones DEFAULT	36
Restricciones CHECK.....	37
Restricciones PRIMARY KEY	37
Restricciones UNIQUE	38
Restricciones FOREIGN KEY	38
Deshabilitación de restricciones.....	39
Deshabilitación de la comprobación de las restricciones en los datos existentes	39
Deshabilitación de la comprobación de las restricciones al cargar datos nuevos	40
Capítulo 3.....	43
Creación de Índices	44
Diseño de Índices.....	44
¿Cómo tiene acceso SQL Server a los datos?	44
Qué es un índice agrupado.....	45
Qué es un índice no agrupado	47
Creación de índices.....	49
Qué son los índices únicos.....	50
Consideraciones para crear índices con varias columnas	51
Uso de procedimientos almacenados del sistema para obtener información de índice.....	53
Capítulo 4.....	55
Consultas básicas.....	56
La sentencia SELECT	56
La cláusula WHERE.....	57
operadores de comparación	58
Uso de comparaciones de cadenas	59
Uso de operadores lógicos	59
Obtención de un intervalo de valores	60
Uso de una lista de valores como criterio de búsqueda	61
Obtención de valores desconocidos.....	62
Ordenación de los datos	62
Eliminación de filas duplicadas.....	64
Cambio del nombre de las columnas	65
Uso de literales.....	66
Mantenimiento de datos: Insert, Update y Delete directo y desde otras tablas.....	67
Inserción de una fila de datos mediante valores.....	67
Uso de la instrucción INSERT...SELECT	67
Creación de una tabla mediante la instrucción SELECT INTO	68
Uso de la instrucción DELETE	69
Uso de la instrucción TRUNCATE TABLE	69
Actualización de filas basada en datos de la tabla	70
Actualización de filas basada en otras tablas	70
Capítulo 5.....	73
Agrupamiento de datos.....	74

Presentación de los primeros n valores.....	74
Uso de funciones de agregado	75
Uso de la cláusula GROUP BY.....	75
Uso de la cláusula GROUP BY con la cláusula HAVING	76
Sub consultas.....	77
Por qué utilizar subconsultas	77
Cómo utilizar subconsultas	77
Uso de una subconsulta como una expresión.....	79
Uniones de Tablas (Join)	79
Uso de herramientas para creación de vistas.....	80
Capítulo 6	82
Procedimientos almacenados.	83
Ventajas de los procedimientos almacenados	83
Ejecución (por primera vez o recompilación)	84
Optimización	84
Compilación	85
Creación de procedimientos almacenados	85
Uso de CREATE PROCEDURE	85
Sintaxis parcial para crear un procedimiento almacenado	86
Ejemplo de creación de un procedimiento almacenado sencillo.....	86
Ejemplo de llamada a un procedimiento almacenado	86
Recomendaciones para la creación de procedimientos almacenados.....	86
Sintaxis para modificar y quitar procedimientos almacenados	87
Ejemplo de modificación de un procedimiento almacenado	88
Quitar procedimientos almacenados	88
Creación de procedimientos almacenados parametrizados.....	88
Directrices para utilizar parámetros de entrada.....	88
Ejemplo de uso de parámetros de entrada.....	89
Llamada a procedimientos almacenados parametrizados.....	89
Uso de los valores predeterminados de un parámetro	90
Parámetros de salida y valores devueltos.....	90
Características de los parámetros de salida	90
Ejemplo de uso de parámetros de salida.....	91
USO DE FUNCIONES	92
Función Escalar	93
Invocación de funciones escalares	93
Función con valores de tabla en línea.....	94
Ejemplo de una función con valores de tabla en línea	95
Invocación de funciones con valores de tabla en línea	95
Función con valores de tabla de varias instrucciones	95
Ejemplo de una función con valores de tabla de varias instrucciones.....	96
Invocación de funciones con valores de tabla de varias instrucciones.....	96
Control de errores , Uso del Try..Catch.....	96
Sintaxis para el control de errores estructurado.....	97
Ejemplo de TRY...CATCH	97
Directrices para controlar errores	98
Creación del bloque CATCH.....	98
Deshacer transacciones con error	98
Uso de XACT_ABORT y XACT_STATE	99
Captura de información sobre el error si es necesario	100
Capítulo 7	102

Implementación de Desencadenadores (Triggers).....	103
Creación de desencadenadores.....	103
Tipos de desencadenadores.....	104
Desencadenadores frente a restricciones.....	104
Cómo funciona un desencadenador INSERT.....	105
Implementación de un desencadenador INSERT.....	105
Cómo funciona un desencadenador DELETE.....	105
Implementación de un desencadenador DELETE.....	106
Cómo funciona un desencadenador UPDATE.....	106
Implementación de un desencadenador UPDATE.....	107
Cómo funciona un desencadenador INSTEAD OF.....	107
Implementación de un desencadenador INSTEAD OF.....	108
Cómo funcionan los desencadenadores anidados.....	108
Niveles de anidamiento.....	109
Deshabilitar desencadenadores anidados.....	109
Uso de XML.....	109
Recuperación de XML mediante FOR XML.....	110
Ejemplos de uso de FOR XML.....	112
Qué son las consultas en modo RAW.....	113
Recuperación de datos en elementos de fila genéricos.....	113
Recuperación de datos como elementos.....	114
Recuperación de datos utilizando un elemento raíz y un nombre de elemento de fila personalizado.....	114
Qué son las consultas en modo AUTO.....	115
Recuperación de datos anidados utilizando el modo AUTO.....	115
Recuperación de datos como elementos.....	116
Qué son las consultas en modo EXPLICIT.....	118
Qué son las consultas en modo PATH.....	119
Recuperación de datos utilizando el modo PATH.....	120
Modificación del nombre del elemento de fila.....	120
Funciones del SQL Server 2012 y 2014.....	121
Nuevas Clausulas del SQL Server 2012 y 2014.....	123
OVER (cláusula de Transact-SQL).....	123
LAG (Transact-SQL).....	125
LEAD (Transact-SQL).....	126
Índice	128