EL DDL, LENGUAJE DE DEFINICIÓN DE DATOS

SQL SERVER 2005

Manual de Referencia para usuarios

Salomón Ccance
CCANCE WEBSITE

EL DDL, LENGUAJE DE DEFINICIÓN DE DATOS

El DDL (Data Definition Language, o Data Description Language según autores), es la parte del SQL dedicada a la definición de la base de datos, consta de sentencias para definir la estructura de la base de datos, permite definir gran parte del nivel interno de la base de datos. Por este motivo estas sentencias serán utilizadas normalmente por el administrador de la base de datos.

La definición de la estructura de la base de datos incluye tanto la creación inicial de los diferentes objetos que formarán la base de datos, como el mantenimiento de esa estructura. Las sentencias del DDL utilizan unos verbos que se repiten para los distintos objetos. Por ejemplo para crear un objeto nuevo el verbo será CREATE y a continuación el tipo de objeto a crear. CREATE DATABASE es la sentencia para crear una base de datos, CREATE TABLE nos permite crear una nueva tabla, CREATE INDEX crear un nuevo índice… Para eliminar un objeto utilizaremos el verbo DROP (DROP TABLE, DROP INDEX…) y para modificar algo de la definición de un objeto ya creado utilizamos el verbo ALTER (ALTER TABLE, ALTER INDEX…).

Los objetos que veremos en este tema son:

- Bases de datos
- Tablas
- Vistas
- Índices

Como ya hemos comentado, las sentencias DDL están más orientadas al administrador de la base de datos, es el que más las va a utilizar, el programador tiene que conocer cuestiones relativas a la estructura interna de una base de datos, pero no tiene que ser experto en ello por lo que el estudio del tema se centrará en las sentencias y sobre todo en las cláusulas que pensamos pueden ser útiles a un programador y no entraremos en mucho detalle en cuanto a la estructura física de la base de datos y en la administración de la misma.

DEFINIR UNA BASE DE DATOS (CREATE DATABASE)

Estructura interna

Como ya vimos en el primer tema, las bases de datos de SQL Server 2005 utilizan tres tipos de archivos:

- Archivos de datos principales (.mdf): Archivo requerido.
- Archivos de datos secundarios (.ndf): Archivo opcional.
- Archivos de registro (.ldf): Archivo requerido.

Y se deben situar en sistemas de archivos FAT o NTFS. Se recomienda NTFS.

CREATE DATABASE

```
CREATE DATABASE nbBasedeDatos
          [ ON
                    [ PRIMARY ] [ <esp_fichero> [ ,...n ]
                    [ , <grupo> [ ,...n ] ]
          [ LOG ON {  < esp_fichero > [ ,...n ] } ]
          ]
          [ COLLATE nbintercalacion]
          [ WITH <external_access_option> ]
      ]
      [;]
```

Como vemos la instrucción mínima es:

```
CREATE DATABASE nbBasedeDatos
```

nbBasedeDatos: Es el nombre de la nueva base de datos. Los nombres de base de datos deben ser únicos en una instancia de SQL Server y cumplir las reglas de los identificadores. Puede tener 128 caracteres como máximo, excepto en un caso que veremos más adelante.

Con la cláusula ON especificamos los ficheros utilizados para almacenar los archivos de datos.

```
        [ ON
              [ PRIMARY ] [  <esp_fichero> [ ,...n ]
              [ , <grupo> [ ,...n ] ]
    <esp_fichero> ::=
        (
          NAME = nbfichero_logico ,
          FILENAME = 'nbfichero_fisico'
           [ , SIZE = tamaño [ KB | MB | GB | TB ] ]
           [ , MAXSIZE = { max_size [ KB | MB | GB | TB ] | UNLIMITED }
]
           [ , FILEGROWTH = incremento_crecimiento [ KB | MB | GB | TB
| % ] ]
       )
```

nbfichero_logico es el nombre que se utiliza para hacer referencia al archivo en todas las instrucciones Transact-SQL. El nombre de archivo lógico tiene que cumplir las reglas de los identificadores de SQL Server y tiene que ser único entre los nombres de archivos lógicos de la base de datos.

nbfichero_fisico es el nombre del archivo físico que incluye la ruta de acceso al directorio. Debe seguir las reglas para nombres de archivos del sistema operativo.

Con la cláusula SIZE indicamos el tamaño original del archivo.

Los archivos de SQL Server 2005 pueden crecer automáticamente a partir del tamaño originalmente especificado. Con FILEGROWTH se puede especificar un incremento de crecimiento y cada vez que se llena el archivo, el tamaño aumentará en la cantidad especificada.

Cada archivo también puede tener un tamaño máximo especificado con MAXSIZE. Si no se especifica un tamaño máximo, el archivo puede crecer hasta utilizar todo el espacio disponible en el disco. Esta característica es especialmente útil cuando SQL Server se utiliza como una base de datos incrustada en una aplicación para la que el usuario no dispone fácilmente de acceso a un administrador del sistema. El usuario puede dejar que los archivos crezcan automáticamente cuando sea necesario y evitar así las tareas administrativas de supervisar la cantidad de espacio libre en la base de datos y asignar más espacio manualmente.

Si no se especifica un nombre de archivo de datos, SQL Server utiliza el nombre de la base de datos como nbfichero_logico y nbfichero_fisico.

Si queremos definir varios archivos de datos, después de la palabra ON escribiremos las definiciones de cada archivo separadas por comas.

PRIMARY especifica que la lista de archivos asociada define el grupo de archivos principal, y el primer archivo especificado se convierte en el archivo de datos principal.
Si no se especifica PRIMARY, el primer archivo enumerado en la instrucción CREATE DATABASE se convierte en el archivo principal.
Detrás de la lista de archivos del grupo de archivos principal, con <grupo> se puede colocar una lista opcional de elementos separados por comas que definan los grupos de archivos de usuario y sus

archivos.

```
<grupo> ::=
  {
  FILEGROUP nbgrupo [ DEFAULT ]
      <esp_fichero>  [ ,...n ]
  }
```

Nbgrupo es el nombre del grupo y a continuación indicamos los archivos de datos que pertenecen a ese grupo, los archivos pertenecientes al grupo se indican con los del grupo principal.

DEFAULT
Cambia el grupo de archivos predeterminado de la base de datos a Nbgrupo. Sólo un grupo de archivos de la base de datos puede ser el grupo de archivos predeterminado.

Con la cláusula LOG ON definiremos los archivos utilizados para almacenar el registro de la base de datos (los archivos de registro).

```
[ LOG ON { <esp_fichero> [ ,...n ] } ]
```

Si no se especifica LOG ON, se crea automáticamente un archivo de registro cuyo tamaño es el 25 por ciento de la suma de los tamaños de todos los archivos de datos de la base de datos, o 512 KB (lo que sea mayor), también limitará el nombre de la base de datos a 123 caracteres ya que el sistema generará el nombre del archivo de registro añadiendo al nombre de la base de datos un sufijo.

Con la cláusula COLLATE podemos cambiar la intercalación predeterminada.
La intercalación define:

- El alfabeto o lenguaje cuyas reglas de ordenación se aplican si se especifica la ordenación de diccionario.
- La página de códigos usada para almacenar datos de caracteres que no son Unicode.
- Las reglas de comportamiento frente a mayúsculas y minúsculas y caracteres acentuados.

La sintaxis es la siguiente:

```
COLLATE <nbintercalacion>
< nbintercalacion >:: =
        nbinterWindows_  CaseSensitivity_AccentSensitivity
```

nbinterWindows: Es un nombre de intercalación de Windows.

CaseSensitivity: Especifica que sí se distingue entre mayúsculas y minúsculas (CS), o no (CI). SQL Server Mobile sólo admite esta opción.

AccentSensitivity: Especifica si se distinguen los caracteres acentuados (AS), o no (AI). SQL Server Mobile sólo admite la opción AS.

Si no se especifica, se asigna a la base de datos la intercalación predeterminada de la instancia de SQL Server. No se puede especificar un nombre de intercalación en una instantánea de base de datos, ni tampoco con las cláusulas FOR ATTACH o FOR.

Existen otras cláusulas a nivel de administración de la base de datos que no detallaremos aquí como son:


```
  [ WITH <external_access_option> ]
<external_access_option> ::=
  {
    DB_CHAINING { ON | OFF }
```

```
| TRUSTWORTHY { ON | OFF }
}
```

Para controlar el acceso externo a la base de datos y desde ésta.

```
CREATE DATABASE nbBasedeDatos
      ON <esp_fichero>  [ ,...n ]
       FOR { ATTACH [ WITH <service_broker_option> ]
          | ATTACH_REBUILD_LOG }
  [;]
  <service_broker_option> ::=
  {
      ENABLE_BROKER
    | NEW_BROKER
    | ERROR_BROKER_CONVERSATIONS
  }
```

Para adjuntar una base de datos.

```
CREATE DATABASE nbInstantanea_basedatos
      ON
          (
          NAME = nbfichero_logico ,
      FILENAME = 'nbfichero_fisico'
          ) [ ,...n ]
      AS  SNAPSHOT OF nbBaseDatos_origen
  [;]
```

Para crear una instantánea de base de datos (copia de sólo lectura de una base de datos).

```
CREATE DATABASE database_name
      [DATABASEPASSWORD 'database_password'
        [ENCRYPTION {ON|OFF}]
     ]
     [COLLATE collation_name comparison_style]
  database password ::= identifier
```

Para crear una base de datos protegida mediante contraseña, opción disponible para SQL Server Mobile (Microsoft SQL Server 2005 Mobile Edition (SQL Server Mobile), antes denominado Microsoft SQL Server 2000 Windows CE 2.0 (SQL Server CE), es una base de datos compacta y con una gran variedad de funciones diseñada para admitir una lista ampliada de dispositivos inteligentes y Tablet PC. Entre los dispositivos inteligentes están todos los dispositivos que ejecuten Microsoft Windows CE 5.0, Microsoft Mobile Pocket PC 2003, Microsoft Mobile Version 5.0 Pocket PC o Microsoft Mobile Version 5.0 Smart Phone. Esta compatibilidad adicional con dispositivos permite a los programadores usar la misma funcionalidad de base de datos en un gran número de dispositivos.)

ELIMINAR UNA BASE DE DATOS (DROP DATABASE)

Para eliminar una base de datos tenemos la instrucción DROP DATABASE.

```
DROP DATABASE { nbBasedeDatos } [ ,...n ] [;]
```

La base de datos puede ser una base de datos normal o una instantánea de base de datos.

Para poder ejecutar la sentencia el usuario debe tener permiso de CONTROL y se debe de ejecutar en un contexto diferente del de la base de datos a eliminar, por ejemplo:

```
use  b1
  DROP DATABASE b1
```

Falla porque con el use estamos en el contexto de la base de datos a eliminar.

Como se ve en la sintaxis podemos eliminar varias bases de datos con una sóla sentencia DROP DATABASE.

Por ejemplo:

```
DROP DATABASE b1,b2
```

Elimina las bases de datos b1 y b2.

MODIFICAR LAS PROPIEDADES DE UNA BD ALTER DATABASE

Si después de crear la base de datos queremos cambiar algo de su definición podríamos eliminarla (con DROP DATABASE) y luego crearla otra vez (con CREATE DATABASE), pero si ya la hemos rellenado con tablas u otros objetos, esta solución no sería muy práctica. Para cambiar la definición de la base de datos una vez creada tenemos que utilizar la sentencia ALTER DATABASE.

Para poder ejecutar esta sentencia se debe de tener el permiso ALTER en la base de datos. Esta sentencia se debe ejecutar en el modo de confirmación automática (modo de administración de transacciones predeterminado) y no se permite en una transacción explícita o implícita.

Las instantáneas de bases de datos no se pueden modificar, y para modificar las opciones de base de datos asociadas a la réplica, se utiliza el procedimiento almacenado del sistema sp_replicationdboption.

Sintaxis:

```
ALTER DATABASE nbBasedeDatos
  {
      <cambiar_ficheros>
    | <cambiar_grupos>
    | <opciones>
    | MODIFY NAME = nuevo_nbBasedeDatos
    | COLLATE nbintercalación
  }
  [;]
```

Con esta sentencia resumida vemos que nos permite cambiar la definición de la base de datos, nos va a permitir cambiar la definición de los ficheros que conforman la base de datos, también nos permite cambiar la definición de los grupos, la definición de varias opciones, el tipo de intercalación e incluso cambiar el nombre de la base de datos (con la cláusula MODIFY NAME).

Por ejemplo:

```
ALTER DATABASE Cliente
  MODIFY NAME = Clientes;
```

La base de datos Cliente pasa a llamarse Clientes.

Como con la instrucción CREATE DATABASE veremos aquí un resumen de lo que más le puede interesar a un programador, sin entrar en demasiados detalles de administración.

Como muchas de las palabras ya las hemos explicado con la sentencia CREATE DATABASE, sólo insistiremos en lo nuevo.

```
<cambiar_ficheros>::=
  {
      ADD FILE < esp_fichero > [ ,...n ]
          [ TO FILEGROUP { nbgrupo |  DEFAULT } ]
    | ADD LOG FILE < esp_fichero > [ ,...n ]
    | REMOVE FILE nbfichero
    | MODIFY FILE < esp_fichero  >
  }
```

Con este grupo de opciones podemos cambiar la definición de los archivos de datos de la base de datos.

ADD FILE permite añadir un nuevo archivo de datos (o varios) si no se añade nada (o TO FILEGROUP DEFAULT), el archivo se añadirá al grupo principal, si añadimos TO FILEGROUP nbgrupo, se añadirá el archivo al grupo indicado.

Con ADD LOG FILE podemos añadir un nuevo archivo de registro.

Con REMOVE FILE nbfichero eliminamos el archivo con nombre nbfichero.

Se elimina la definición del archivo lógico (nbfichero) y elimina el archivo físico asociado. El archivo no se puede quitar a menos que esté vacío.

Con MODIFY FILE <esp_fichero> podemos cambiar las especificaciones de alguno de los archivos de la base de datos.

```
<esp_fichero>::=
  (
      NAME = nbarchivo
      [ , NEWNAME = nuevo_nbarchivo ]
      [ , FILENAME  = 'nbarchivo_fisico' ]
      [ , SIZE = tamaño [ KB | MB | GB | TB ] ]
      [ , MAXSIZE  = { tamaño_máximo [ KB | MB | GB | TB ] |
UNLIMITED } ]
      [ , FILEGROWTH = incremento [  KB | MB | GB | TB| % ] ]
      [ , OFFLINE ]
  )
```

Sólo se puede cambiar una propiedad <especificaciónDeArchivo> cada vez.

NAME se debe especificar siempre para identificar el archivo que se va a modificar.

Si se especifica SIZE, el nuevo tamaño debe ser mayor que el tamaño actual del archivo. Para reducir el tamaño de una base de datos, se utiliza el comando de consola SHRINKDATABASE de Transact_SQL.

Para modificar el nombre lógico de un archivo de datos o de un archivo de registro, especificamos el nombre lógico nuevo para el archivo en la cláusula NEWNAME. Por ejemplo:

```
ALTER DATABASE MiBasedeDatos
  MODIFY FILE ( NAME = bd1,  NEWNAME = base1)
```

Con esta sentencia, el archivo bd1 ahora se llama base1.

Con FILENAME podemos cambiar el nombre del fichero físico, esto nos permite también cambiar la ubicación del archivo físico.

Por ejemplo:

```
ALTER DATABASE MiBasedeDatos
  MODIFY FILE(NAME = nbficherológico,
   FILENAME = 'nueva_ruta/nb_fichero_físico')
```

La cláusula OFFLINE establece el archivo sin conexión e impide el acceso a todos los objetos del grupo de archivos. ¡Muy importante!, esta opción sólo se debe de utilizar si el archivo está dañado y si se puede restaurar. Un archivo establecido en OFFLINE sólo se puede restablecer con conexión mediante la restauración del archivo a partir de una copia de seguridad. Para obtener más información acerca de cómo restaurar un solo archivo, consultar en la ayuda la sentencia RESTORE (Transact-SQL).

UNLIMITED especifica que el tamaño del archivo aumenta hasta que el disco esté lleno. En SQL Server 2005, un archivo de registro especificado con un aumento ilimitado tiene un tamaño máximo de 2 TB y un archivo de datos tiene un tamaño máximo de 16 TB.

En cuanto a la modificación de la definición de grupos, tenemos esta sintaxis:

```
<cambiar_grupos>::=
  {
       | ADD  FILEGROUP nbgrupo
       | REMOVE  FILEGROUP nbgrupo
       | MODIFY FILEGROUP nbgrupo
          {
               {  READONLY | READWRITE }
               |  { READ_ONLY | READ_WRITE }
    }
          | DEFAULT
          | NAME = nuevo_ nbgrupo
          }
  }
```

Con ADD FILEGROUP nbgrupo añadimos un nuevo grupo de archivos de la base de datos.

Con REMOVE FILEGROUP nbgrupo eliminamos el grupo de la base de datos.

El grupo de archivos no se puede quitar a menos que esté vacío. Para quitar todos los archivos del grupo, primero se mueven los archivos a otro grupo, si los archivos están vacíos, se pueden eliminar directamente.

Con MODIFY FILEGROUP nbgrupo, modificamos el grupo de archivos.

DEFAULT Cambia el grupo de archivos predeterminado de la base de datos a nbgrupo.

NAME = nuevo_nbgrupo Cambia el nombre del grupo a nuevo_nbgrupo.

READ_ONLY | READONLY Especifica que el grupo de archivos es de sólo lectura. En este caso no se permitirá la actualización de los objetos del mismo. Una base de datos de sólo lectura no permite realizar modificaciones en los datos por lo que:

- Se omite la recuperación automática cuando se inicia el sistema.
- No es posible reducir la base de datos.
- No se produce ningún bloqueo en las bases de datos de sólo lectura. Esto puede acelerar el rendimiento de las consultas.

Para cambiar este estado, se debe tener acceso exclusivo a la base de datos.

El grupo de archivos principal no puede ser de sólo lectura.

Se puede utilizar indistintamente la palabra clave READONLY o READ_ONLY, pero READONLY se quitará

en una versión futura de Microsoft SQL Server por lo que se recomienda READ_ONLY.

READ_WRITE | READWRITE Especifica que el grupo es de lectura y escritura por lo que pueden realizarse actualizaciones en los objetos del grupo de archivos.

Para cambiar este estado se debe tener acceso exclusivo a la base de datos.

Se puede utilizar indistintamente la palabra clave READWRITE o READ_ WRITE, pero READWRITE se quitará en una versión futura de Microsoft SQL Server por lo que se recomienda READ_WRITE.

Nota. El estado de estas opciones se puede determinar mediante el examen de la columna is_read_only de la vista de catálogo sys.databases o la propiedad Updateability de la función DATABASEPROPERTYEX.

Para finalizar tenemos el apartado <opciones> que nos permite definir y/o cambiar muchas opciones de la base de datos. La lista de opciones es muy larga, como podemos observar a continuación, y no entraremos en detalles.

```
    <opciones>::=
      SET
      {
          { <optionspec> [ ,...n ] [ WITH  <termination> ] }
          | ALLOW_SNAPSHOT_ISOLATION {ON | OFF }
          | READ_COMMITTED_SNAPSHOT {ON | OFF } [ WITH  <termination>
]
      }
      <optionspec>::=
      {
          <db_state_option>
        | <db_user_access_option>
        | <db_update_option>   |  <external_access_option>
        | <cursor_option>
        | <auto_option>
        | <sql_option>
        | <recovery_option>
        | <database_mirroring_option>
        | <supplemental_logging_option>
        | <service_broker_option>
        | <date_correlation_optimization_option>
        |  <parameterization_option>
      }
      <db_state_option> ::=
          { ONLINE | OFFLINE | EMERGENCY }
    <db_user_access_option> ::=
          { SINGLE_USER | RESTRICTED_USER |  MULTI_USER }
    <db_update_option> ::=
          { READ_ONLY | READ_WRITE }
    <external_access_option> ::=
          DB_CHAINING { ON | OFF }
        | TRUSTWORTHY { ON | OFF }
    }
      <cursor_option> ::=
      {   CURSOR_CLOSE_ON_COMMIT { ON | OFF }
        | CURSOR_DEFAULT { LOCAL | GLOBAL }
      }
    <auto_option> ::=
      {
          AUTO_CLOSE { ON | OFF }
        | AUTO_CREATE_STATISTICS { ON | OFF }
        | AUTO_SHRINK { ON | OFF }
        | AUTO_UPDATE_STATISTICS { ON | OFF }
```

```
        | AUTO_UPDATE_STATISTICS_ASYNC { ON | OFF }
      }
      <sql_option> ::=
      {
          ANSI_NULL_DEFAULT { ON | OFF }
        | ANSI_NULLS { ON | OFF }
        | ANSI_PADDING { ON | OFF }
        | ANSI_WARNINGS { ON | OFF }
        | ARITHABORT { ON | OFF }
        | CONCAT_NULL_YIELDS_NULL { ON | OFF }
        | NUMERIC_ROUNDABORT { ON | OFF }
        | QUOTED_IDENTIFIER { ON | OFF }
        | RECURSIVE_TRIGGERS { ON | OFF }
      }
    <recovery_option> ::=
      {
          RECOVERY { FULL | BULK_LOGGED | SIMPLE }
        | TORN_PAGE_DETECTION { ON | OFF }
        | PAGE_VERIFY { CHECKSUM | TORN_PAGE_DETECTION | NONE }
      }
    < database_mirroring_option> ::=
      { <partner_option> | <witness_option> }
          <partner_option> ::=
          PARTNER { = 'partner_server'
                  |  FAILOVER
                  |  FORCE_SERVICE_ALLOW_DATA_LOSS
                  |  OFF
                  |  RESUME
                  |  SAFETY { FULL | OFF }
                  |  SUSPEND
                  |  REDO_QUEUE ( integer { KB | MB | GB } | UNLIMITED
)
                  |  TIMEOUT integer
                  }
      <witness_option> ::=
      WITNESS { = 'witness_server'
                  |  OFF
                  }
      <supplemental_logging_option> ::=
          SUPPLEMENTAL_LOGGING { ON | OFF }
    <service_broker_option> ::=
      {
          ENABLE_BROKER
        | DISABLE_BROKER
        | NEW_BROKER
        | ERROR_BROKER_CONVERSATIONS
      }
    <date_correlation_optimization_option> ::=
      {
          DATE_CORRELATION_OPTIMIZATION { ON | OFF }
      }
    <parameterization_option> ::=
      {
          PARAMETERIZATION { SIMPLE | FORCED }
      }
    <termination> ::=
      {
          ROLLBACK AFTER integer [ SECONDS ]
        | ROLLBACK IMMEDIATE
        | NO_WAIT
      }
```

Se pueden ver más detalles de estas opciones en la ayuda de SQL SERVER buscando ALTER DABATABASE.

CREAR UNA TABLA (CREATE TABLE)

Para crear una nueva tabla se emplea la sentencia CREATE TABLE.

Se necesita el permiso CREATE TABLE en la base de datos y el permiso ALTER en el esquema en que se crea la tabla.

Si las columnas de la instrucción CREATE TABLE se definen como un tipo definido por el usuario CLR, se necesita la propiedad del tipo o el permiso REFERENCES.

Si las columnas de la instrucción CREATE TABLE tienen una colección de esquemas XML asociada, se necesita la propiedad de la colección de esquemas XML o el permiso REFERENCES.

Empezaremos por una sintaxis reducida:

```
    CREATE TABLE
          [ nbBaseDatos.[nbEsquema].| nbEsquema.]nbTabla
              ( { <definicion_columna> | < definicion_colCalc > } [
,...n ]
              [ <restriccion_tabla> ] [ ,...n ] )
       [ ; ]
```

nbBaseDatos Es el nombre de la base de datos en la que se crea la tabla. Debe ser el nombre de una base de datos existente. Si no se especifica nbBaseDatos, se utiliza de manera predeterminada la base de datos actual. El inicio de sesión de la conexión actual debe estar asociado a un Id. de usuario existente en la base de datos especificada por nbBaseDatos, y ese Id. de usuario debe tener permisos CREATE TABLE.

nbEsquema Es el nombre del esquema al que pertenece la nueva tabla.

nbTabla Es el nombre de la nueva tabla. Los nombres de tablas deben seguir las reglas de los identificadores. nbTabla puede contener un máximo de 128 caracteres excepto para los nombres de tablas temporales locales (nombres precedidos por un único signo de número (#)) que no pueden superar los 116 caracteres.

Después de indicar el nombre de la tabla, entre paréntesis definimos, separadas por una coma, cada una de las columnas de la tabla y al final las restricciones a nivel de tabla si las hay.

Definir columnas físicas

```
    < definicion_columna > ::=
      nbCol <tipo_dato>
          [ COLLATE nbIntercalacion ]
          [ NULL | NOT NULL ]
          [
             [ CONSTRAINT nbRestriccion ] DEFAULT exp_constante ]
           |   [   IDENTITY   [(   semilla,   incremento   )]   [NOT   FOR
REPLICATION] ]
          ]
          [ ROWGUIDCOL ]
       [  <restriccion_columna> [ ...n ] ]
```


Como mínimo debemos indicar el nombre de la columna nbCol y su tipo de datos.

Los nombres de columnas deben seguir las reglas de los identificadores y deben ser únicos en la tabla. nbCol puede contener de 1 a 128 caracteres. nbCol se puede omitir en las columnas creadas con un tipo de datos timestamp, en tal caso, si no se especifica nbCol, el nombre de la columna timestamp será de manera predeterminada timestamp.

En cuanto al tipo de dato, esta es la sintaxis:

```
<tipo_dato> ::=
  [nbEsquema_tipo. ] nbtipo
      [   ( precision [ , escala ] | max |
           [   { CONTENT | DOCUMENT } ] xml_schema_collection ) ]
```

[nbEsquema_tipo.] nbtipo
nbtipo Especifica el tipo de datos de la columna y nbEsquema_tipo el esquema al que pertenece el tipo. El tipo de datos puede ser uno de los siguientes:

- Un tipo de datos del sistema de SQL Server 2005 como los que ya conocemos.
- Un tipo de alias basado en un tipo de datos del sistema de SQL Server. Los tipos de datos de alias se crean con la instrucción CREATE TYPE para poder utilizarlos en una definición de tabla. La asignación NULL o NOT NULL de un tipo de datos de alias puede anularse durante la instrucción CREATE TABLE. No obstante, la especificación de longitud no se puede cambiar; la longitud del tipo de datos de alias no se puede especificar en una instrucción CREATE TABLE.
- Un tipo definido por el usuario CLR. Los tipos definidos por el usuario CLR se crean con la instrucción CREATE TYPE para poder utilizarlos en una definición de tabla.
- Si no se especifica el parámetro nbEsquema_tipo, el SQL Server Database Engine (Motor de base de datos de SQL Server) hace referencia a nbtipo en el siguiente orden:

- El tipo de datos del sistema de SQL Server.
- El esquema predeterminado del usuario actual en la base de datos actual.
- El esquema de dbo de la base de datos actual.

Precision es la precisión del tipo de datos especificado.

Escala es la escala del tipo de datos especificado.

Max sólo se aplica a los tipos de datos varchar, nvarchar y varbinary para almacenar 231 bytes de datos de caracteres y binarios, y 230 bytes de datos Unicode.

También podemos definir tipos de datos XML.

Una vez indicado el tipo de datos de la columna podemos opcionalmente completar su definición con una serie de cláusulas.

```
[ COLLATE nbIntercalacion ]
```

Con la cláusula COLLATE podemos definir el tipo de intercalación que se utilizará para la columna (Ver CREATE TABLE).

```
[ NULL | NOT NULL ]
```

Determina si se permiten valores nulos (NULL) en la columna o no (NOT NULL). Realmente NULL no es estrictamente una restricción, si no indicamos nada la columna permitirá valores nulos, pero se puede especificar de la misma forma que NOT NULL. NOT NULL se puede especificar para las columnas calculadas sólo si se especifica también PERSISTED.

```
[  CONSTRAINT nbRestriccion ] DEFAULT exp_constante ]
```

Con la cláusula DEFAULT podemos especificar un valor por defecto, es decir el valor que tomará el campo cuando no se haya especificado explícitamente un valor durante la inserción. Las definiciones DEFAULT se pueden aplicar a cualquier columna excepto a las definidas como timestamp o a aquellas que tengan la propiedad IDENTITY. Si se especifica un valor por defecto a una columna de un tipo definido por el usuario, dicho tipo debe ser compatible con la conversión implícita de exp_constante en el tipo definido por el usuario. exp_constante sólo puede ser NULL o un valor constante (por ejemplo, una cadena de caracteres, una función escalar o una función del sistema, definida por el usuario o CLR).

Para mantener la compatibilidad con las versiones anteriores de SQL Server, se puede asignar un nombre de restricción a DEFAULT con [CONSTRAINT nbRestriccion]. Los nombres de restricción deben ser únicos en el esquema al que pertenece la tabla.

```
[ IDENTITY [( semilla, incremento )] [NOT FOR REPLICATION]  ]
```

IDENTITY indica que la nueva columna es una columna de identidad. Cuando se agrega una nueva fila a la tabla, el Motor de base de datos proporciona un valor incremental único para la columna. Las columnas de identidad se utilizan normalmente como claves principales.

La propiedad IDENTITY se puede asignar a las columnas tinyint, smallint, int, bigint, decimal(p,0) o numeric(p,0). Sólo se puede crear una columna de identidad para cada tabla. Las restricciones DEFAULT y los valores predeterminados enlazados no se pueden utilizar en las columnas de identidad. En este caso, deben especificarse el valor de inicialización y el incremento, o ninguno de esto valores. Si no se especifica ninguno, el valor predeterminado es (1,1).

semilla es el valor que se utiliza para la primera fila cargada en la tabla.

Incremento es el valor incremental que se agrega al valor de identidad de la fila cargada anterior.

De forma general, si se especifica la cláusula NOT FOR REPLICATION para una restricción, dicha restricción no se impone cuando los agentes de réplica realizan operaciones de inserción, actualización o eliminación. Si se especifica esta cláusula, junto con IDENTITY, los valores no se incrementan en las columnas de identidad cuando los agentes de réplica realizan inserciones.

```
[ ROWGUIDCOL ]
```

ROWGUIDCOL indica que la nueva columna es una columna de GUID de filas. Sólo se puede designar una columna uniqueidentifier por tabla como columna ROWGUIDCOL. La propiedad ROWGUIDCOL se puede asignar únicamente a una columna uniqueidentifier.

Las columnas de tipos de datos definidos por el usuario no se pueden designar con ROWGUIDCOL.

La propiedad ROWGUIDCOL no impone la unicidad de los valores almacenados en la columna.

ROWGUIDCOL tampoco genera automáticamente valores para nuevas filas insertadas en la tabla, por lo que se debe de utilizar la función NEWID en las instrucciones INSERT o utilizar la función NEWID como el valor predeterminado de la columna para generar valores únicos en cada fila.

Para ver más consideraciones sobre columnas IDENTITY y ROWGUIDCOL.

Por último nos quedan las restricciones de clave que aparecen en la sintaxis como:

```
[ <restriccion_columna> [ ...n ]  ]
< restriccion_columna > ::=
  [ CONSTRAINT nbRestriccion]
  {     { PRIMARY KEY | UNIQUE }[ CLUSTERED |  NONCLUSTERED ]
```

```
            [ WITH FILLFACTOR = factorRelleno
            | WITH ( <  opcion_indice > [ , ...n ] )
         ]
            [ ON { partition_scheme_name ( partition_column_name )
                | filegroup | "default" } ]
        | [ FOREIGN KEY ]
            REFERENCES [ nbEsquema.] nbTablaPadre [ ( col_padre )  ]
            [ ON DELETE { NO  ACTION | CASCADE | SET NULL | SET
DEFAULT } ]
       }
           [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET
DEFAULT } ]
            [ NOT FOR REPLICATION ]
        | CHECK [ NOT FOR REPLICATION ] (expresion_validacion)
```

nbRestriccion es el nombre de la restricción, como hemos visto antes, debe ser único en el esquema al que pertenece la tabla. Las restricciones se implementan internamente con índices por lo que a veces podemos utilizar el término índice o restricción indistintamente.

```
{PRIMARY KEY | UNIQUE} [ CLUSTERED | NONCLUSTERED ]
```

PRIMARY KEY indica que la columna es la clave principal de la tabla. Sólo se puede crear una restricción PRIMARY KEY para cada tabla. Si la clave primaria (principal) está compuesta por varias columnas entonces no podemos utilizar esta restricción, tendremos que utilizar una restricción de tabla que veremos más adelante.

CLUSTERED indica que el índice que se va a crear es un índice agrupado. Como sólo puede haber un índice agrupado por tabla, si todavía no hay ninguno definido, por defecto se creará con la clave primaria, si ya existe un índice agrupado, la clave principal se creará sin índice agrupado.

Una clave primaria no puede contener valores nulos, por lo que todas las columnas definidas en una restricción PRIMARY KEY se deben definir como NOT NULL. Si cuando definimos la columna, no se indica nada, la columna se establecerá a NOT NULL.

Si la clave principal se define en una columna de tipo definido por el usuario CLR, la implementación del tipo debe admitir el orden binario.

UNIQUE indica que la columna no admite valores duplicados, por lo que se crea un índice único. Una tabla puede tener varios índices únicos.

Si no se especifica CLUSTERED o NONCLUSTERED, de forma predeterminada se utiliza NONCLUSTERED.

Como cada restricción UNIQUE genera un índice. El número de restricciones UNIQUE no puede hacer que el número de índices de la tabla exceda de 249 índices no agrupados y 1 índice agrupado.

Si se define una restricción UNIQUE en una columna de tipo definido por el usuario CLR, la implementación del tipo debe admitir el orden binario o basado en el operador.

Las demás opciones de índices:

```
[ WITH FILLFACTOR = factorRelleno
   | WITH ( < opcion_indice > [ , ...n ] ) ]
< opcion_indice > ::=
  {
      PAD_INDEX = { ON | OFF }
    | FILLFACTOR = factorRelleno
    | IGNORE_DUP_KEY = { ON | OFF }
    | STATISTICS_NORECOMPUTE = { ON | OFF }
    | ALLOW_ROW_LOCKS = { ON | OFF}
```

```
        | ALLOW_PAGE_LOCKS ={ ON | OFF}
      }
```

Son cláusulas que nos permiten definir con más detalle el índice pero que no veremos aquí por entrar demasiado en cuestiones internas.

La cláusula CHECK.

```
CHECK [ NOT FOR REPLICATION ] ( expresion_validacion )
```

Con CHECK indicamos una regla de validación que deberán cumplir todas las filas de la tabla, es una restricción que exige la integridad del dominio al limitar los valores posibles que se pueden escribir en la columna.

expression es una expresión lógica que devuelve TRUE o FALSE.

Si queremos definir una restricción CHECK sobre una columna calculada, esta se deberá definir como PERSISTED.

Una columna puede tener cualquier número de restricciones CHECK y la condición puede incluir varias expresiones lógicas combinadas con AND y OR. Varias restricciones CHECK para una columna se validan en el orden en que se crean.

La condición de búsqueda debe evaluarse como una expresión booleana y no puede hacer referencia a otra tabla.

Una restricción CHECK en el nivel de columna sólo puede hacer referencia a la columna restringida y una restricción CHECK en el nivel de tabla sólo puede hacer referencia a columnas de la misma tabla.

Las restricciones CHECK no se pueden definir en las columnas text, ntext o image.

Por ejemplo queremos que la columna Precio de la tabla que estamos definiendo no pueda contener valores negativos:

```
…
  Precio CURRENCY CONSTRAINT precio_pos CHECK  (Precio > = 0)
  …
```

Por último a nivel de columna podemos definir una restricción de clave ajena:

```
[ FOREIGN KEY ]
            REFERENCES [ nbEsquema.] nbTablaPadre [ ( col_padre ) ]
            [ ON DELETE { NO   ACTION | CASCADE | SET NULL | SET
DEFAULT } ]
            [ ON UPDATE { NO   ACTION | CASCADE | SET NULL | SET
DEFAULT } ]
            [ NOT FOR  REPLICATION ]
```

Con esta cláusula definimos una regla de integridad referencial, los valores contenidos en la columna deberán apuntar a un registro en la tabla de referencia (la tabla padre).

La palabra FOREIGN KEY no es obligatoria cuando estamos a nivel de columna, el utilizarla o no tiene el mismo efecto.

Nbesquema es el nombre del esquema al que pertenece la tabla padre y nbTablaPadre es el nombre de la tabla padre.

Col_padre es la columna de la tabla padre que se relaciona con la columna que estamos definiendo. Esta columna debe tener el mismo tipo de datos que la columna en la que se define la restricción.

La col_padre debe tener en la tabla padre una restricción de PRIMARY KEY o UNIQUE, si no se indica una columna padre (columna de referencia) se utiliza la clave primaria de la tabla padre.

Las reglas de integridad referencial obligan a que si se especifica un valor distinto de NULL en una columna con una restricción FOREIGN KEY, ese valor debe existir en la columna referenciada de la tabla padre; de lo contrario, se devolverá un error de infracción de clave externa.

Las restricciones FOREIGN KEY sólo pueden hacer referencia a las tablas de la misma base de datos en el mismo servidor. La integridad referencial entre bases de datos debe implementarse a través de desencadenadores.

Las restricciones FOREIGN KEY pueden hacer referencia a otras columnas de la misma tabla. Esto se denomina autorreferencia.

```
create table personas (codigo integer primary Key,
  dni integer UNIQUE,
  madre integer references personas,
  padre integer references personas (col1),
  dniTutor integer references personas  (col4));
```

En esta tabla de personas, identificamos a cada persona por un código, también tenemos el dni de la persona y el código de su padre y de su madre, además nos guardamos el dni del tutor de la persona.

Las madres, padres y tutores son personas que deben de existir en la tabla Personas y en este caso utilizamos el código para referenciar padres y madres y el dni para referencial el tutor, esto último lo podemos hacer porque el campo dni tiene una restricción UNIQUE.

Por último nos queda completar las reglas de integridad referencial en cuanto a qué hacer cuando se eliminan o modifican valores que intervienen en una relación referencial.

```
ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }
```

Indica qué ocurre cuando se intenta eliminar un registro padre en la relación que estamos definiendo. El valor predeterminado es NO ACTION.

NO ACTION El Motor de base de datos genera un error y no es posible eliminar la fila de la tabla primaria (el padre).

CASCADE Si se borra una fila de la tabla primaria, se eliminan automáticamente todas las filas correspondientes de la tabla que estamos definiendo, en otras palabras, si se elimina un padre, se eliminan todos sus hijos.

SET NULL Si se borra una fila de la tabla primaria, todas las filas correspondientes de la tabla que estamos definiendo tomarán el valor NULL en el campo clave ajena. En otras palabras, si se elimina un padre, sus hijos se quedan sin padre.

Para ejecutar esta restricción, la columna clave ajena debe admitir valores NULL.

SET DEFAULT Es como la anterior pero en vez del valor NULL toman el valor que tienen predeterminado. Si no hay ningún valor predeterminado establecido de forma explícita, tomarán el valor NULL. Hay que tener en cuenta que el valor predeterminado debe de existir en la tabla primaria.

Veamos estas opciones con un ejemplo:

Tenemos una tabla Proveedores y una tabla artículos, en la tabla artículos nos guardamos el código del proveedor del artículo, por lo tanto definiremos la tabla artículos de la siguiente forma:

```
CREATE TABLE articulos (
  Codigo INTEGER PRIMARY KEY,
  Denominacion   VARCHAR(30),
  …..,
  Proveedor INTEGER REFERENCES Proveedores,
  …)
```

El campo Proveedor es clave ajena y hace referencia a un código de proveedor de la tabla Proveedores. Aquí no hemos añadido ninguna cláusula ON DELETE por lo que se toma NO ACTION.

```
…..,
  Proveedor INTEGER REFERENCES Proveedores ON  DELETE NO ACTION,
  …)
```

Estas dos sentencias son equivalentes e indican que si se intenta borrar de la tabla Proveedores un proveedor asignado a un artículo, el sistema da un error y no deja eliminar el proveedor.

```
…..,
  Proveedor INTEGER REFERENCES Proveedores ON  DELETE CASCADE,
  …)
```

Se eliminará el proveedor y todos los artículos asignados a él.

```
…..,
  Proveedor INTEGER REFERENCES Proveedores ON  DELETE SET NULL,
  …)
```

Se eliminará el proveedor de la tabla Proveedores y en la tabla Artículos, todas las filas que tenían ese número de proveedor pasarán a tener el valor nulo en el campo proveedor.

```
ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }
```

Indica qué ocurre cuando se intenta cambiar un valor del campo relacionado de la tabla padre en la relación que estamos definiendo. El valor predeterminado es NO ACTION.

Las opciones son las mismas que para ON DELETE.

NO ACTION El Motor de base de datos genera un error y no es posible modificar la fila de la tabla primaria (el padre).

CASCADE Si se modifica un valor de la columna padre en la tabla primaria, se modificarán automáticamente todas las filas correspondientes de la tabla que estamos definiendo, en otras palabras, si se modifica el identificativo de un padre, se actualizan todos sus hijos.

SET NULL Si se modifica un valor de la columna padre en la tabla primaria, todas las filas correspondientes de la tabla que estamos definiendo tomarán el valor NULL en el campo clave ajena. En otras palabras, si se modifica el identificativo de un padre, sus hijos se quedan sin padre.

Para ejecutar esta restricción, la columna clave ajena debe admitir valores NULL.

SET DEFAULT Es como la anterior pero en vez del valor NULL toman el valor que tienen predeterminado.

Si no hay ningún valor predeterminado establecido de forma explícita, tomarán el valor NULL. Hay que tener en cuenta que el valor predeterminado debe de existir en la tabla primaria.

Volviendo al ejemplo anterior:

```
….,
  Proveedor INTEGER REFERENCES Proveedores ON  UPDATE NO ACTION,
  …)
```

Si cambiamos el código del proveedor 3 a 3000 y hay artículos asignados al proveedor 3, el sistema da un error y no deja modificar el proveedor.

```
….,
  Proveedor INTEGER REFERENCES Proveedores ON  UPDATE CASCADE,
  …)
```

Se modifica el proveedor y todos los artículos asignados a él pasan a tener el valor 3000 en el campo Proveedor.

```
….,
  Proveedor INTEGER REFERENCES Proveedores   ON UPDATE SET NULL,
  …)
```

Se modifica el proveedor y todos los artículos asignados a él pasan a tener el valor NULL en el campo Proveedor.

Definir restricciones de tabla

Hasta el momento hemos aprendido a definir restricciones sobre una columna, mientras la estamos definiendo añadimos a su definición la restricción o restricciones que queramos.

También existen restricciones de tabla, son restricciones que se definen después de definir todas las columnas de la tabla y que pueden afectar a una o varias columnas de la tabla. Como veremos la sintaxis para definir una restricción de tabla es muy parecida a la sintaxis de la misma restricción de columna, lo que varía es que ahora tenemos que indicar las columnas afectadas por la restricción.

```
< restriccion_tabla > ::=
  [ CONSTRAINT nbRestriccion]
  {
     {PRIMARY KEY | UNIQUE } [ CLUSTERED | NONCLUSTERED ]
  (nbCol[ASC|DESC][  ,...n ] )
        [   WITH FILLFACTOR = factorRelleno
           |WITH ( < opcion_indice > [ , ...n ] )
        ]
        [ ON { partition_scheme_name (partition_column_name)
            | filegroup |  "default" } ]
   |
       FOREIGN  KEY(nbCol[,...n])REFERENCES nbTablaPadre
  [(col_padre)[,...n] )]
        [ ON DELETE { NO  ACTION | CASCADE | SET NULL | SET DEFAULT
} ]
        [ ON UPDATE { NO  ACTION | CASCADE | SET NULL | SET DEFAULT
} ]
        [ NOT FOR REPLICATION ]
   | CHECK [NOT FOR REPLICATION] (expresion_validacion )
  }
```

Como las cláusulas son las mismas que para las restricciones de columna, no repetiremos la explicación de cada cláusula, lo que veremos es un ejemplo y la explicación de ese ejemplo.

Volviendo al ejemplo anterior de la tabla de Personas, podemos definir la misma utilizando restricciones de tabla:

```
CREATE TABLE personas (
  codigo int,
  dni int,
  madre int,
  padre int,
  dniTutor int,
  PRIMARY KEY (codigo),
  UNIQUE (dni),
  FOREIGN KEY (madre) REFERENCES Personas,
  FOREIGN KEY (padre) REFERENCES Personas  (codigo),
  FOREIGN KEY (dniTutor) REFERENCES Personas  (dni));
```

También podíamos haber mezclado restricciones de columna con restricciones de tabla, lo que no podemos hacer es definir dos veces la misma restricción, o la definimos a nivel de columna o a nivel de tabla pero no dos veces!

Utilizando la restricciones de tabla parece que la definición queda más clara, por un lado tenemos la definición de cada columna, y luego las restricciones.

Las restricciones de tabla se hacen imprescindibles cuando la restricción afecta a una combinación de columnas. Por ejemplo imaginemos una tabla de productos en la que la clave principal está formada por el código de fabricante y código de producto (porque dos productos diferentes pueden tener el mismo código de producto con proveedores diferentes).

No podemos definir la clave de esta manera:

```
CREATE TABLE Productos (
  Codproducto int PRIMARY KEY,
  Codproveedor int PRIMARY KEY,
  …
```

El Motor de la base de datos entendería que queremos definir dos claves primarias y eso es imposible. En este caso habría que utilizar una restricción de tabla:

```
CREATE TABLE Productos (
  Codproducto int,
  Codproveedor int,
  … las demás columnas,
  PRIMARY KEY (Codproducto, Codproveedor));
```

Ocurre lo mismo con las demás restricciones. Imaginemos ahora una tabla de líneas de pedido en la que tenemos en una línea el producto pedido, y la cantidad pedida.

```
CREATE TABLE LineasPed (
  pedido INT,
  nlin INT,
  codprod INT,
  codprov INT,
  cantidad INT,
  PRIMARY KEY (pedido, nlin),
  FOREIGN KEY (codprod,codprov) REFERENCES  Productos,
  UNIQUE  (pedido,codprod,codprov));
```

La combinación (codprod,codprov) forma una clave ajena que hace referencia a la tabla Productos, en este caso como la clave principal de la tabla Productos está compuesta por los dos campos, la clave ajena tiene que tener el mismo número de campo y del mismo tipo.

Con la restricción UNIQUE indicamos que la combinación formada por un número de pedido un código de producto y código de proveedor no se puede repetir, esto hace que no se puedan insertar en un mismo pedido dos líneas del mismo producto. No se puede duplicar la combinación pero sí las columnas individualmente (pueden haber varias filas con el mismo número de pedido, varias filas con el mismo código de producto y varias filas con el mismo codigo de proveedor.

Aquí termina la explicación de la sintaxis básica de las instrucciones CREATE TABLE. Nos queda hablar de las columnas calculadas y de las tablas temporales, otras cláusulas de la instrucción se pueden consultar en la ayuda de la instrucción SQL CREATE TABLE.

Definir columnas calculadas.

Una columna calculada es una columna cuyo valor no se introduce, sino que se obtiene como resultado de un cálculo.

```
< definicion_columna > ::=
nbCol AS expresion
  [ PERSISTED
[NOT  NULL]
    [ <restriccion_columna> [...n ]  ]
  ]
```

expresion es la expresión que define el valor de una columna calculada y está basada en otras columnas de la tabla.

Por ejemplo, una columna calculada puede ser definida así:

```
importe AS precio * cantidad
```

La expresión puede ser un nombre de columna no calculada, una constante, una función, una variable o cualquier combinación de estos elementos conectados mediante uno o más operadores. La expresión no puede ser una subconsulta ni contener tipos de datos de alias.

Una columna calculada es, por defecto, una columna virtual no almacenada físicamente en la tabla. PERSISTED indica que la columna calculada se almacena en la tabla y automáticamente se actualizan los valores almacenados en ella cuando se actualizan las columnas de las que depende.

Para que la columna pueda ser definida con PERSISTED la expresión que la calcula debe ser determinista. Una expresión es determinista a menos que utilice una función no determinista.

Una columna calculada se puede utilizar en la lista de selección, cláusula WHERE, cláusula ORDER BY u otras ubicaciones en que se puedan utilizar expresiones regulares, con las siguientes excepciones:

- Una columna calculada no puede utilizarse como definición de restricción DEFAULT o FOREIGN KEY ni como NOT NULL.
- Una columna calculada no puede ser el destino de una instrucción INSERT o UPDATE. No tiene sentido querer asignar un determinado

Definir tablas temporales

Una tabla temporal es una tabla creada por un determinado proceso y desaparece cuando termina éste. Se pueden crear tablas temporales locales y globales. Las tablas temporales locales son visibles sólo en la sesión actual y las tablas temporales globales son visibles para todas las sesiones.

Para indicar que la tabla que queremos crear es temporal añadimos a su nombre el prefijo # (#nbTabla) para tablas temporales locales y el prefijo ## (##nbTabla) tablas temporales globales.

Por ejemplo:

```
CREATE TABLE #trabajo (col1 INT PRIMARY  KEY);
```

Esta instrucción crea una tabla temporal local llamada trabajo con una sola columna.

Las tablas temporales funcionan casi como las tablas normales con algunas diferencias.

No se pueden crear particiones en las tablas temporales.

nbTabla no puede tener más de 116 caracteres. Esto se debe a que si se crea una tabla temporal local en un procedimiento almacenado o una aplicación que varios usuarios pueden ejecutar al mismo tiempo, el Motor de base de datos tiene que ser capaz de distinguir las tablas creadas por los distintos usuarios, lo consigue añadiendo internamente un sufijo numérico a cada nombre de tabla temporal local. El nombre completo de una tabla temporal tal como se almacena en la tabla sysobjects de tempdb consta del nombre de la tabla especificado en la instrucción CREATE TABLE y el sufijo numérico generado por el sistema.

Las tablas temporales se quitan automáticamente cuando están fuera de ámbito, a menos que ya se hayan quitado explícitamente mediante DROP TABLE:

Una tabla temporal local creada en un procedimiento almacenado se quita automáticamente cuando se completa el procedimiento almacenado. Cualquiera de los procedimientos almacenados anidados ejecutados por el procedimiento almacenado que creó la tabla, puede hacer referencia a la tabla. El proceso que llamó al procedimiento almacenado que creó la tabla no puede hacer referencia a la tabla.
Las tablas temporales se quitan automáticamente al final de la sesión actual.

Las tablas temporales globales se quitan automáticamente cuando la sesión que creó la tabla finaliza y las tareas restantes han dejado de hacer referencia a ellas. La asociación entre una tarea y una tabla se mantiene sólo durante la vida de una única instrucción Transact-SQL. Esto significa que la tabla temporal global se quita al finalizar la última instrucción Transact-SQL que estuviera haciendo referencia activamente a la tabla cuando finalizó la sesión que la creó.

Una tabla temporal local creada en un procedimiento almacenado o desencadenador, puede tener el mismo nombre que una tabla temporal creada antes de que se llame al procedimiento almacenado o al desencadenador. No obstante, si una consulta hace referencia a una tabla temporal y hay dos tablas temporales con el mismo nombre, no está definido en cuál de las dos tablas debe resolverse la consulta.

Los procedimientos almacenados anidados pueden crear también tablas temporales con el mismo nombre que la tabla temporal creada por el procedimiento almacenado que la llamó. Sin embargo, en el caso de las modificaciones que se van a resolver en la tabla creada en el procedimiento anidado, la tabla debe tener la misma estructura, con los mismos nombres de columnas, que la tabla creada en el procedimiento que realiza la llamada. Esto se muestra en el ejemplo siguiente.

Con tablas temporales globales o locales, la sintaxis CREATE TABLE admite la definición de restricciones, excepto las restricciones FOREIGN KEY. Si se especifica una restricción FOREIGN KEY en una tabla temporal, la instrucción devuelve un mensaje de advertencia que indica que la restricción se ha omitido. La tabla se crea sin las restricciones FOREIGN KEY. En las restricciones FOREIGN KEY no se puede hacer referencia a tablas temporales.

Se recomienda utilizar variables de tabla en lugar de tablas temporales. Las tablas temporales son útiles cuando es necesario crear en ellas índices de forma explícita o bien cuando los valores de tabla deben ser visibles en varios procedimientos almacenados o funciones. En general, las variables de tabla contribuyen a que el procesamiento de las consultas sea más eficaz.

ELIMINAR UNA TABLA (DROP TABLE)

Para eliminar una tabla de una base de datos tenemos la sentencia DROP TABLE. Con ella quitamos una o varias definiciones de tabla y todos los datos, índices, desencadenadores, restricciones y especificaciones de permisos que tengan esas tablas.
Las vistas o procedimientos almacenados que hagan referencia a la tabla quitada se deben quitar explícitamente con DROP VIEW o DROP PROCEDURE.
Su sintaxis es:

```
    DROP TABLE [nbBaseDatos.[nbEsquema].|nbEsquema.]nbTabla[ ,...n ] [
; ]
```

Para que las reglas de integridad referencial se cumplan, no se puede eliminar una tabla señalada por una restricción FOREIGN KEY. Primero se debe quitar la restricción FOREIGN KEY o la tabla que tiene la clave ajena.

Se pueden quitar varias tablas de cualquier base de datos en una misma sentencia DROP TABLE. Se irán eliminando en el mismo orden en que aparecen en la lista por lo que podremos eliminar dos tablas relacionadas con una sola sentencia pero escribiendo la tabla que contiene la clave ajena primero y después la tabla principal.

Requiere el permiso CONTROL en la tabla o pertenecer a la función fija de base de datos db_ddladmin.

Ejemplo:

```
DROP  TABLE mitabla;
```

Elimina la tabla miTabla tanto su definición como los datos, índices definidos sobre ella y permisos.

MODIFICAR LA DEFINICIÓN DE UNA TABLA (ALTER TABLE)

```
ALTER TABLE [nbBaseDatos.[nbEsquema].|  nbEsquema.]nbTabla
  { ALTER COLUMN nbColumna
        {
          <tipo_dato>
          [ NULL | NOT NULL ]
         [ COLLATE nbIntercalacion ]
       | {ADD | DROP } { ROWGUIDCOL  | PERSISTED }
       }
  | [WITH{ CHECK | NOCHECK}] ADD
        {
          <definicion_columna>
       | <definicion_colCalc>
       | <restriccion_tabla>
         } [ ,...n ]
  | DROP
        {
         [CONSTRAINT] nbRestriccion
        |COLUMN nbColumna
        } [,...n ]
  | {CHECK|NOCHECK} CONSTRAINT {ALL|nbRestriccion[ ,...n ]}
  | {ENABLE|DISABLE} TRIGGER {ALL | nbTrigger [ ,...n ] }
  }
  [ ; ]
```


Aunque la sintaxis parece un poco complicada, realmente no lo es. La sentencia nos permite variar la

definición de una tabla ya creada, en qué consiste esta variación: modificar la definición de columnas ya existentes (ALTER COLUMN), añadir más columnas o restricciones (ADD), eliminar columnas y restricciones (DROP), habilitar/deshabilitar restricciones (CHECK CONSTRAINT) y habilitar/deshabilitar triggers.

Como muchas de las cláusulas las hemos estudiado con CREATE TABBLE, sólo incidiremos en lo nuevo.
Para modificar una columna escribiremos la cláusula ALTER COLUMN seguida del nombre de la columna que queremos modificar y la nueva definición, podemos cambiar su tipo de datos indicando uno nuevo, hacer que la columna acepte o no valores nulos (NULL|NOTNULL), cambiar la intercalación (COLLATE).
Con ADD ROWGUIDCOL hacemos que la columna sea GUID de filas y DROP ROWGUIDCOL hacemos que ya no lo sea.

Si la columna es una columna calculada podemos cambiar su condición de columna almacenada con ADD/DROP PERSISTED.

Ejemplo:

```
ALTER TABLE Clientes ALTER COLUMN direccion  VARCHAR(40);
```

Hace que la columna direccion de la tabla Clientes, ahora admita 40 caracteres alfanuméricos.

Cuando cambiamos el tipo de una columna hay que tener en cuenta que el nuevo tipo debe ser compatible con el antiguo para que no se pierdan los datos almacenados.

Además la columna no se puede modificar si es ROWGUIDCOL, calculada o si se utiliza en una columna calculada, si se utiliza en un índice (a menos que la columna sea del tipo de datos varchar, nvarchar o varbinary, el tipo de datos no se cambie y el nuevo tamaño sea igual al tamaño anterior o mayor que éste), si se utiliza en estadísticas, en una restricción PRIMARY KEY, FOREIGN KEY, CHECK o UNIQUE. Sin embargo, se permite el cambio de longitud de una columna de longitud variable en una restricción CHECK o UNIQUE.

Para añadir una nueva columna o restricción utilizamos la cláusula ADD seguida de la definición de lo que queremos añadir, para eso seguimos la misma sintaxis que para definir las columnas y restricciones de tabla del CREATE TABLE.
Por ejemplo:

```
ALTER TABLE Clientes ADD email varchar(50);
```

Añade una nueva columna email.

```
ALTER TABLE Clientes ADD CONSTRAINT Pk PRIMARY KEY (codcli);
```

Añade una restricción de clave primaria sobre la columna codcli que ya existe en la tabla.

La cláusula WITH CHECK|NOCHECK permite establecer si la restricción que estamos añadiendo se tiene que comprobar en las filas que ya existen en la tabla.

Con la cláusula DROP podemos eliminar de la tabla una restricción (DROP CONSTRAINT), o una columna (DROP COLUMN).

Con las siguientes limitaciones:

Una restricción PRIMARY KEY no puede quitarse si existe un índice XML en la tabla.

Una columna no puede quitarse si se utiliza en un índice, en una restricción CHECK, FOREIGN KEY, UNIQUE o PRIMARY KEY, DEFAULT.

Ejemplo:

```
ALTER TABLE Clientes DROP COLUMN email;
```

Elimina de la tabla Clientes la columna email que habíamos creado anteriormente.

```
ALTER TABLE Clientes DROP CONSTRAINT pk ;
```

Elimina de la tabla Clientes la restricción de PRIMARY KEY, la columna sigue en la tabla pero ya no es clave principal.

Para finalizar, las dos últimas cláusulas nos permiten indicar si se tienen que comprobar o no determinadas restricciones y habilitar y deshabilitar triggers.

CHECK CONSTRAINT ALL activa la comprobación de todas las restricciones definidas sobre la tabla.

DISABLE TRIGGER ALL deshabilita todos los triggers definidos sobre la tabla.

Cuando deshabilitamos un trigger, éste sigue definido, pero no entra en acción cuando se produce el evento que debería activarlo. En cualquier momento lo podremos habilitar con otra instrucción ALTER TABLE.

CREAR UNA VISTA (CREATE VIEW)

Una vista es una tabla virtual que representa los datos de una o más tablas de una forma alternativa. Para crear una nueva vista se emplea la sentencia CREATE VIEW, debe ser la primera instrucción en un lote de consultas.

Una vista sólo se puede crear en la base de datos actual.

Para ejecutar CREATE VIEW, se necesita, como mínimo, el permiso CREATE VIEW en la base de datos y el permiso ALTER en el esquema en el que se está creando la vista.

Sintaxis:

```
CREATE VIEW [nbEsquema.] nbVista
  [ (columna [ ,...n ] ) ]
      AS ( sentencia_select ) [ ; ]
```

nbEsquema Es el nombre del esquema al que pertenece la nueva tabla.

nbVista Es el nombre de la nueva vista. Los nombres de vistas deben seguir las reglas de los identificadores.

sentencia_select Es la instrucción SELECT que define la vista. Dicha instrucción puede utilizar más de una tabla y otras vistas. Se necesitan permisos adecuados para seleccionar los objetos a los que se hace referencia en la cláusula SELECT de la vista que se ha creado.

Una vista no tiene por qué ser un simple subconjunto de filas y de columnas de una tabla determinada. Es posible crear una vista que utilice más de una tabla u otras vistas mediante una cláusula SELECT de cualquier complejidad.

También se pueden utilizar funciones y varias instrucciones SELECT separadas por UNION o UNION ALL. Una vista puede tener como máximo 1.024 columnas.

Ejemplos:

```
CREATE VIEW oficinas_este
  AS SELECT * FROM oficinas WHERE region =  'Este';
```

Crea una vista con las oficinas del este.

```
CREATE VIEW oficinas_empleados
  AS
  SELECT oficinas.oficina AS ofi, ciudad, dir,  region, objetivo,
oficinas.ventas AS ventas_ofi, empleados.*
  FROM oficinas INNER JOIN empleados
          ON oficinas.oficina = empleados.oficina;
```

Crea una vista con los datos de todos los empleados y de sus oficinas.
En este caso hemos tenido que definir alias de campo porque en el origen de la sentencia SELECT existe duplicidad de nombres.

```
CREATE VIEW oficinas_EO
  AS
  SELECT * FROM oficinas WHERE region = 'Este';
  UNION ALL
  SELECT * FROM oficinas WHERE region = 'Oeste';
```

Por defecto las columnas de la vista heredan los nombres de las columnas de la sentencia SELECT asociada, pero podemos cambiar estos nombres indicando una lista de columnas después del nombre de la vista.

```
CREATE VIEW oficinas_este (Eoficina,  Eciudad, Eregion, Edir,
Eobjetivo,Eventas)
  AS SELECT * FROM oficinas WHERE region = 'Este';
```

Utilizando una lista de columnas ya no tenemos que definir alias de columna en la sentencia SELECT como pasaba en el caso de la vista oficinas_empleados.

Normalmente se utiliza la lista de columnas cuando una columna proviene de una expresión aritmética, una función o una constante; cuando dos o más columnas puedan tener el mismo nombre, normalmente debido a una combinación; o cuando una columna de una vista recibe un nombre distinto al de la columna de la que proviene.

En definitiva se puede optar por utilizar la lista de columnas o definir alias de campo en la sentencia SELECT.

Cuando utilizamos una vista en una operación de actualización (INSERT, UPDATE, DELETE), la vista debe ser actualizable, para ello debe seguir las siguientes reglas:

Cualquier modificación, incluida en las instrucciones UPDATE, INSERT y DELETE, debe hacer referencia a las columnas de una única tabla base.

Las columnas que se vayan a modificar en la vista deben hacer referencia directa a los datos subyacentes de las columnas de la tabla, es decir que las columnas no se pueden obtener de otra forma, como con una función de agregado: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR y VARP, o un cálculo.

Las columnas formadas mediante los operadores de conjunto UNION, UNION ALL, CROSSJOIN, EXCEPT e INTERSECT equivalen a un cálculo y tampoco son actualizables.

Las columnas que se van a modificar no se ven afectadas por las cláusulas GROUP BY, HAVING o

DISTINCT.

Las restricciones anteriores se aplican a cualquier subconsulta de la cláusula FROM de la vista, al igual que a la propia vista. Normalmente, el Database Engine (Motor de base de datos) debe poder realizar un seguimiento sin ambigüedades de las modificaciones de la definición de la vista a una tabla base.

ELIMINAR UNA VISTA (DROP VIEW)

Para eliminar una vista de una base de datos tenemos la sentencia DROP TABLE.
Sintaxis:

```
DROP VIEW [nbEsquema.]nbVista[ ,...n ] [ ; ]
```

Se eliminan las vista de la base de datos actual. Cuando eliminamos una vista eliminamos su definición y los permisos asociados a ella.

Se pueden quitar varias vistas en una misma sentencia DROP VIEW escribiendo los nombres de las vistas a eliminar separados por comas.

Para ejecutar DROP VIEW, como mínimo, se necesita el permiso ALTER en SCHEMA o el permiso CONTROL en OBJECT.

Ejemplo:

```
DROP VIEW oficinas_este, oficinas_EO;
```

Elimina las vistas oficinas_este y oficinas_EO.

Si eliminamos una tabla mediante DROP TABLE, se deben quitar explícitamente, con DROP VIEW, las vistas basadas en esta tabla ya que no se quitarán por sí solas.

DEFINICIÓN DE INDICE

Un índice es una estructura de datos definida sobre una columna de tabla (o varias) y que permite localizar de forma rápida las filas de la tabla en base a su contenido en la columna indexada además de permitir recuperar las filas de la tabla ordenadas por esa misma columna.

Funciona de forma parecida al índice de un libro donde tenemos el título del capítulo y la página donde empieza dicho capítulo, en un índice definido sobre una determinada columna tenemos el contenido de la columna y la posición de la fila que contiene dicho valor dentro de la tabla.

La definición de los índices de la base de datos es tarea del administrador de la base de datos, los administradores más experimentados pueden diseñar un buen conjunto de índices, pero esta tarea es muy compleja, consume mucho tiempo y está sujeta a errores, incluso con cargas de trabajo y bases de datos con un grado de complejidad no excesivo.

TIPOS DE INDICE

Índice simple y compuesto.

Un índice simple está definido sobre una sóla columna de la tabla mientras que un índice compuesto está formado por varias columnas de la misma tabla (tabla sobre la cual está definido el índice.

Cuando se define un índice sobre una columna, los registros que se recuperen utilizando el índice aparecerán ordenados por el campo indexado. Si se define un índice compuesto por las columnas col1 y col2, las filas que se recuperen utilizando dicho índice aparecerán ordenadas por los valores de col1 y todas las filas que tengan el mismo valor de col1 se ordenarán a su vez por los valores contenidos en col2, función igual que la cláusula ORDER BY vista en el tema de consultas simples.

Por ejemplo si definimos un índice compuesto basado en las columnas (provincia, localidad), las filas que se recuperen utilizando este índice aparecerán ordenadas por provincia y dentro de la misma provincia por localidad.

Índice agrupado y no agrupado,

El término índice agrupado no se debe confundir con índice compuesto, el significado es totalmente diferente.

Un índice agrupado (CLUSTERED) es un índice en el que el orden lógico de los valores de clave determina el orden físico de las filas correspondientes de la tabla. El nivel inferior, u hoja, de un índice agrupado contiene las filas de datos en sí de la tabla. Una tabla o vista permite un solo índice agrupado al mismo tiempo.

Los índices no agrupados existentes en las tablas se vuelven a generar al crear un índice agrupado, por lo que es conveniente crear el índice agrupado antes de crear los índices no agrupados.

Un índice no agrupado especifica la ordenación lógica de la tabla. Con un índice no agrupado, el orden físico de las filas de datos es independiente del orden indizado.

Índice único

Índice único es aquel en el que no se permite que dos filas tengan el mismo valor en la columna de clave del índice. Es decir que no permite valores duplicados.

VENTAJAS E INCOVENIENTES DE LOS INDICES

Ventajas

La utilización de índices puede mejorar el rendimiento de las consultas, ya que los datos necesarios para satisfacer las necesidades de la consulta existen en el propio índice. Es decir, sólo se necesitan las páginas de índice y no las páginas de datos de la tabla o el índice agrupado para recuperar los datos solicitados; por tanto, se reduce la E/S global en el disco. Por ejemplo, una consulta de las columnas a y b de una tabla que dispone de un índice compuesto creado en las columnas a, b y c puede recuperar los datos especificados del propio índice.

Los índices en vistas pueden mejorar de forma significativa el rendimiento si la vista contiene agregaciones, combinaciones de tabla o una mezcla de agregaciones y combinaciones.

Inconvenientes

Las tablas utilizadas para almacenar los índices ocupan espacio.

Los índices consumen recursos ya que cada vez que se realiza una operación de actualización, inserción o borrado en la tabla indexada, se tienen que actualizar todas las tablas de índice definidas sobre ella (en la actualización sólo es necesaria la actualización de los índices definidos sobre las columnas que se actualizan).

Por estos motivos no es buena idea definir índices indiscriminadamente.

Consideraciones a tener en cuenta

A la hora de definir índices se deben de tener en cuenta estas consideraciones:

- Hay que evitar crear demasiados índices en tablas que se actualizan con mucha frecuencia y procurar definirlos con el menor número de columnas posible.
- Es conveniente utilizar un número mayor de índices para mejorar el rendimiento de consultas en tablas con pocas necesidades de actualización, pero con grandes volúmenes de datos. Un gran número de índices contribuye a mejorar el rendimiento de las consultas que no modifican datos, como las instrucciones SELECT, ya que el optimizador de consultas dispone de más índices entre los que elegir para determinar el método de acceso más rápido.
- La indización de tablas pequeñas puede no ser una solución óptima, porque puede provocar que el optimizador de consultas tarde más tiempo en realizar la búsqueda de los datos a través del índice que en realizar un simple recorrido de la tabla. De este modo, es posible que los índices de tablas pequeñas no se utilicen nunca; sin embargo, sigue siendo necesario su mantenimiento a medida que cambian los datos de la tabla.
- Se recomienda utilizar una longitud corta en la clave de los índices agrupados. Los índices agrupados también mejoran si se crean en columnas únicas o que no admitan valores NULL.
- Un índice único en lugar de un índice no único con la misma combinación de columnas proporciona información adicional al optimizador de consultas y, por tanto, resulta más útil.
- Hay que tener en cuenta el orden de las columnas si el índice va a contener varias columnas. La columna que se utiliza en la cláusula WHERE en una condición de búsqueda igual a (=), mayor que (>), menor que (<) o BETWEEN, o que participa en una combinación, debe situarse en primer lugar. Las demás columnas deben ordenarse basándose en su nivel de diferenciación, es decir, de más distintas a menos distintas.

DEFINIR UN INDEX (CREATE INDEX)

```
    CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX
nombre_indice
          ON <objeto> (columna [ ASC | DESC ] [ ,...n ] )
        [ ; ]
    <objeto> ::=
      {
          [nbBaseDatos.[nbEsquema].| nbEsquema.]nbTablaVista
      }
```

Esta es la sintaxis simplicada de la instrucción CREATE INDEX que permite crear un índice en una tabla sobre una o varias columnas.

nbBaseDatos Es el nombre de la base de datos.

nbEsquema Es el nombre del esquema al que pertenece la tabla/vista.

nbTablaVista Es el nombre de la tabla o vista sobre la que se quiere crear el índice.

nombre_indice Es el nombre del índice que estamos creando.

Columna Es el nombre de la columna que forma parte del índice. Se pueden definir índices compuestos escribiendo entre paréntesis los nombres de las columnas separados por comas.
ASC los valores de la columna se ordenarán de forma ASCendente o DESCendente. Por defecto se asume ASC.

UNIQUE permite definir un índice único (no admite valores repetidos).

CLUSTERED el índice será agrupado.

NONCLUSTERED (valor por defecto) el índice será no agrupado.

Ejemplos:

```
CREATE INDEX I_clientes_nombre ON  Clientes (nombre)
```

Crea un índice no agrupado sobre la columna nombre de la tabla Clientes en la base de datos actual, las filas se ordenarán de forma ascendente.

```
CREATE INDEX I_clientes_ApeNom ON  Clientes (apellidos, nombre)
```

Crea un índice no agrupado sobre las columnas apellidos y nombre de la tabla Clientes en la base de datos actual, las filas se ordenarán de forma ascendente por apellido y dentro del mismo apellido por nombre.

```
CREATE INDEX I_clientes_EdadApe ON  Clientes (edad DESC,apellidos)
```

Crea un índice no agrupado sobre las columnas edad y apellidos de la tabla Clientes en la base de datos actual, las filas se ordenarán de forma descendente por edad y ascendente por apellido. Aparecerán los clientes de mayor a menor edad y los clientes de la misma edad se ordenarán por apellido (por orden alfabético).

```
CREATE CLUSTERED INDEX I_clientes_cod ON Clientes (codigo)
```

Crea un índice agrupado sobre la columna codigo de la tabla Clientes en la base de datos actual, las filas se ordenarán y almacenarán por orden de código.

```
CREATE UNIQUE INDEX U_clientes_col  ON Clientes (col)
```

Crea un índice único sobre la columna col de la tabla Clientes en la base de datos actual, la columna col no podrá contener valores duplicados.

ELIMINAR UN INDICE (DROP INDEX)

Para eliminar un índice tenemos la sentencia DROP INDEX.
La instrucción DROP INDEX no es aplicable a los índices creados mediante la definición de restricciones PRIMARY KEY y UNIQUE. Para quitar la restricción y el índice correspondiente, se tiene que ejecutar un ALTER TABLE con la cláusula DROP CONSTRAINT.

Sintaxis simplificada:

```
DROP INDEX <indice> [ ,...n  ] [ ; ]
  <indice>::=
  {
  nbindice ON [nbBaseDatos.[nbEsquema].|nbEsquema.]nbTablaVista
  }
```

nbBaseDatos Es el nombre de la base de datos.
nbEsquema Es el nombre del esquema al que pertenece la tabla/vista.
nbTablaVista Es el nombre de la tabla o vista de la que se quiere eliminar el índice.
nbindice Es el nombre del índice a eliminar.

Ejemplo:

```
DROP INDEX U_clientes_col ON  Clientes;
```

Elimina el índice U_clientes_col definido sobre la tabla Clientes.