DML. Actualizar datos

Índice

1. Introducción

Hasta ahora hemos trabajado con tablas que tenían datos introducidos y cuando nos ha hecho falta hemos añadido nuevos datos en las mismas y hemos modificado algún dato directamente desde el entorno gráfico de MSSMS, en este tema veremos cómo hacerlo con instrucciones de Transact-SQL.

Seguimos en el DML porque las instrucciones que veremos actúan sobre los datos de la base de datos no sobre su definición y tenemos tres tipos de operaciones posibles:

- Insertar nuevas filas en una tabla.
- Modificar datos ya almacenados.
- Eliminar filas de una tabla.

2. Insertar en una tabla existente INSERT INTO

La inserción de nuevos datos en una tabla, se realiza añadiendo filas enteras a la tabla, la sentencia SQL que lo permite es la orden INSERT (o también denominada INSERT INTO).

De la sentencia INSERT completa, nosotros estudiaremos la sintaxis más utilizada y estándar:

```
INSERT [INTO] <destino>
  {
    [(lista_columnas)]
    {VALUES ({DEFAULT|NULL|expresion}[ ,...n ])
    |tabla_derivada
    }
  }
  |DEFAULT VALUES
[;]

<destino> ::=
{
  [nbBaseDatos.nbEsquema. | nbEsquema.]nbTablaVista
}
```

Con esta instrucción podemos insertar una fila de valores determinados o un conjunto de filas derivadas de otra consulta.

2.1. Insertar una fila de valores

Para insertar una fila de valores utilizamos la sintaxis:

```
INSERT [INTO] <destino> [(lista_columnas)]
    VALUES ({DEFAULT|NULL|expresion}[ ,...n ]) [;]
```

La palabra reservada INTO es opcional y no añade funcionalidad a la instrucción, originalmente era obligatoria y aconsejo utilizarla.

```
Por ejemplo en Access y Oracle INTO es obligatorio.
```

Después de indicar que queremos insertar, debemos indicar dónde, mediante <destino>.

<destino> es el nombre de la tabla donde queremos insertar, puede ser un nombre de tabla o un nombre de vista.

Si utilizamos una vista, y ésta tiene un origen basado en varias tablas, en su lista de selección deberán aparecer columnas de una sola tabla (no podemos insertar datos en varias tablas a la vez).

Con la cláusula VALUES indicamos entre paréntesis los valores a insertar, separados por comas.

Cada valor se puede indicar mediante:

- una expresión que normalmente será una constante,
- mediante la palabra reservada DEFAULT que indica 'valor por defecto' en este caso la columna se rellenará con el valor predeterminado de la columna, si la columna no tiene valor predeterminado se rellenará con el valor nulo NULL.
- Mediante la palabra reservada NULL valor nulo.

Delante de VALUES, de forma opcional, podemos indicar una lista de columnas entre paréntesis. Las columnas son columnas del destino.

Cuando indicamos nombres de columnas, esas columnas serán las que reciban los valores a insertar, la asignación de valores se realiza por posición, la primera columna recibe el primer valor, la segunda columna el segundo, y así sucesivamente.

En la lista, las columnas pueden estar en cualquier orden y también se pueden omitir algunas columnas del destino.

Una columna que no aparezca en la lista de columnas se rellenará de acuerdo a su definición:

- con su valor por defecto si está definida con la cláusula DEFAULT
- con el valor de identidad incremental siguiente si tiene la propiedad IDENTITY.
- con el valor calculado si es una columna calculada.
- con el valor NULL, en cualquier otro caso y si la columna lo admite.

Cuando no se indica una lista de columnas el sistema asume por defecto todas las columnas de la tabla y en el mismo orden que aparecen en la definición de la tabla, en este caso, los valores se tienen que especificar en el mismo orden que las columnas en la definición de la tabla, y se tiene que especificar un valor por cada columna ya que los valores se rellenan por posición, la primera columna recibe el primer valor, la segunda columna el segundo, y así sucesivamente.

En cualquier caso, el número de valores debe coincidir con el número de columnas y los tipos de dato de los valores deben ser compatibles con las columnas.

Aunque pueda parecer más engorroso escribir la lista de columnas, es un hábito recomendable, hace que la sentencia sea más fácil de leer y mantener (cuando leemos la sentencia sabemos en qué columnas asignamos los valores sin necesidad de consultar la definición de la tabla) y evita que se tenga que cambiar la sentencia si se modifica el esquema de la tabla (si el orden de las columnas dentro de la tabla cambia).

Los registros se agregan al final de la tabla.

Cuando se insertan nuevas filas en una tabla, el sistema comprobará que la nueva fila no infrinja ninguna regla de integridad, por ejemplo no podremos asignar a una columna PRIMARY KEY un valor nulo o que ya exista en la tabla, a una columna UNIQUE un valor que ya exista en la tabla, a una columna NOT NULL un valor NULL, a una clave ajena (FOREIGN KEY) un valor que no exista en la tabla de referencia.

De producirse alguna de las situaciones anterior, la instrucción genera un mensaje de error y la fila no se inserta.

Ejemplos.

```
INSERT INTO oficinas (oficina, ciudad) VALUES (26, 'Elx');
```

En este caso hemos indicado sólo dos columnas y dos valores, las demás columnas se rellenan con el valor por defecto si lo tiene (DEFAULT) o con NULL. Si alguna columna no nombrada no admite nulos ni tiene cláusula DEFAULT definida, la instrucción dará error.

```
INSERT INTO oficinas
      VALUES (27,'Móstoles','Centro',default ,null, default)
```

Aquí no hemos indicado una lista de columnas luego los valores se tienen que indicar en el mismo orden que las columnas dentro de la tabla, si nos equivocamos de orden, el valor se guardará en una columna errónea (si los tipos son compatibles) o generará un mensaje de error y la fila no se insertará (si los tipos no son compatibles).

En MySQL se pueden insertar varias filas de valores con un solo INSERT poniendo cada lista de valores entre paréntesis y separando los paréntesis por una coma:

```
INSERT INTO oficinas
      VALUES (27,'Móstoles','Centro',default ,null, default),
             (28, 'Madrid','Centro',default ,null, default)
```

2.2. Inserción de varias filas

Si los valores que queremos insertar los tenemos en otras tablas, podemos insertar varias filas a la vez indicando una consulta que genere las filas de valores a insertar. En este caso utilizamos la sintaxis:

```
INSERT [INTO] <destino>  [(lista_columnas)]
       tabla_derivada [;]
```

<destino> y *lista_columnas* funcionan igual que en el punto anterior.

Tabla_derivada es cualquier instrucción SELECT válida que devuelva filas con los datos que se van a cargar en el destino. No va entre paréntesis ni lleva punto y coma final.

Cada fila devuelta por la SELECT es una lista de valores que se intentará insertar como con la cláusula VALUES, por lo que las columnas devueltas por la SELECT deberán cumplir las mismas reglas que los valores de la lista de valores anteriores.

Ejemplo:

```
CREATE TABLE trabajo (col1 INT, col2 VARCHAR(20), col3 MONEY);
```

Creamos una tabla trabajo de 3 columnas

```
INSERT INTO trabajo SELECT oficina, ciudad, ventas
                        FROM oficinas
                        WHERE region = 'Centro';
```

Insertamos en *trabajo* el resultado de la SELECT (el número de oficina, ciudad y ventas de las oficinas del *Centro*).

En este caso no hemos incluido una lista de columnas, por lo que en la SELECT tenemos que generar los valores en el mismo orden que en *trabajo*.

Si hubiesemos escrito:

```
INSERT INTO trabajo SELECT ciudad, oficina, ventas
                        FROM oficinas
                        WHERE region = 'Centro';
```

Hubiese dado error porque la columna col1 es INT y el valor a asignar es texto (el nombre de la ciudad de la oficina).

```
INSERT INTO trabajo (col2, col1)
      SELECT ciudad, oficina
 FROM oficinas
 WHERE region = 'Este';
```

En este caso hemos incluido una lista de columnas, la SELECT debe generar los valores correspondientes, y col3 que no se rellena explícitamente se rellenará con NULL porque la columna col3 no está definida como columna calculada, ni con DEFAULT, ni IDENTITY y además admite nulos.

2.3. Insertar una fila de valores por defecto

TRANSACT-SQL nos permite insertar una fila de valores por defecto utilizando la sintaxis:

```
INSERT [INTO] <destino> DEFAULT VALUES
[;]
```

Hace que la nueva fila contenga los valores predeterminados definidos para cada columna.

Hay que tener en cuenta una serie de aspectos al utilizar esta instrucción:

Puede generar filas duplicadas en la tabla si los valores que se generan son siempre los mismos.

Si la tabla tiene una clave principal, esta tendrá que estar basada en una columna con la propiedad IDENTITY para que se generen valores diferentes automáticamente.

Si una columna está definida como NOT NULL tendrá que incluir un DEFAULT o ser una columna calculada con una expresión compatible.

3. Insertar creando una nueva tabla – SELECT… INTO

Existe otra forma de insertar datos en una tabla pero esta vez la inserción incluye la creación de la tabla, esta es la sentencia SELECT... INTO.

```
SELECT …
INTO nb_NuevaTabla
FROM …
```

nb_NuevaTabla es el nombre de la tabla que se va a crear, si en la base de datos ya hay una tabla con ese nombre, el sistema genera un error y no se ejecuta la sentencia.

En la nueva tabla las columnas tendrán el mismo tipo y tamaño que las columnas del resultado de la SELECT, se llamarán con el nombre de alias de la columna o en su defecto con el nombre de la columna, pero no se transfiere ninguna otra propiedad del campo o de la tabla como por ejemplo las claves e índices.

Si retomamos el ejemplo del punto anterior:

```
CREATE TABLE trabajo (col1 INT, col2 VARCHAR(20), col3 MONEY);
INSERT INTO trabajo SELECT oficina, ciudad, ventas
                        FROM oficinas
                        WHERE region = 'Centro';
```

Se podría obtener el mismo resultado con una sola instrucción:

```
SELECT oficina AS col1, ciudad AS col2, ventas AS col3
INTO trabajo
FROM oficinas
WHERE region = 'Centro';
```

Si se tiene poca experiencia en esta instrucción, lo mejor es primero escribir la SELECT que permite obtener las filas a insertar y una vez la tenemos añadir la cláusula INTO destino delante de FROM.

See Adding rows by using INSERT and SELECT INTO:
http://msdn.microsoft.com/en-US/library/ms188263(v=SQL.90).aspx

4. Modificar datos almacenados - UPDATE

La sentencia UPDATE modifica los valores de una o más columnas en las filas seleccionadas de una única tabla.

Para modificar los datos de una tabla es necesario disponer del privilegio UPDATE sobre dicha tabla.

```
UPDATE
   [ TOP ( expression ) [ PERCENT ] ]
   <destino>
   SET { nbcolumna = { expresion | DEFAULT | NULL }
       } [ ,...n ]
   [ FROM{ <origen> }]
   [ WHERE <condicion> ]
 [;]

<destino> ::=
{
  [nbBaseDatos.[nbEsquema.]| nbEsquema.]nbTablaVista
}
```

Con <destino> indicamos la tabla que se va a actualizar.

La cláusula SET especifica qué columnas van a modificarse y con qué valor, el valor se puede expresar mediante una expresión, la palabra DEFAULT que equivale al valor predeterminado de la columna, o el valor nulo NULL.

Las columnas de identidad no se pueden actualizar.

Expresión en cada asignación debe generar un valor del tipo de dato apropiado para la columna indicada. La expresión debe ser calculable basada en los valores de la fila actualmente en actualización. Si para el cálculo se utiliza una columna que también se modifica, el valor que se utilizará es el de antes de la modificación, lo mismo para la condición del WHERE.

Expresión también puede ser una subconsulta escalar (siempre que devuelva un único valor que cumpla las condiciones anteriormente expuestas).

Por ejemplo:

```
UPDATE oficinas SET ventas = 0;
```

Actualiza todas las filas de la tabla oficinas dejando el campo ventas con el valor cero.

Si el campo ventas está definido con un valor predeterminado 0, la sentencia anterior equivale a:

```
UPDATE oficinas SET ventas = DEFAULT;
```

Si lo que queremos es dejar el campo a nulo:

```
UPDATE oficinas SET ventas = NULL;
```

En una misma sentencia podemos actualizar varias columnas, sólo tenemos que indicar las distintas asignaciones separadas por comas:

```
UPDATE oficinas SET ventas = 0, objetivo = 0;
```

Los nombres de columna pueden especificarse en cualquier orden.

Si no queremos actualizar todas las filas de la tabla sino unas cuantas, utilizaremos la cláusula TOP, o unas determinadas, utilizaremos la cláusula WHERE.

```
 TOP ( expresion ) [ PERCENT ]
```

Especifica el número o porcentaje de filas aleatorias que se van a modificar.
expression debe generar un valor numérico e indica el número de filas a modificar empezando por el principio. Como en la SELECT, si añadimos la palabra PERCENT, el número representado por expresión se refiere al porcentaje de filas a modificar sobre el total. La cláusula TOP funciona casi igual que en la SELECT pero en este caso, las filas no se ordenan, y la expresión debe ir entre paréntesis.

Por ejemplo:

```
UPDATE TOP (10) PERCENT oficinas
SET ventas = 0;
```

Actualiza el 10% de filas de la tabla *oficinas*.

```
   [ WHERE <condicion> ]
```

Utilizamos la cláusula WHERE para filtrar las filas a actualizar. Se actualizarán todas las filas que cumplan la condición. Por ejemplo si queremos actualizar sólo las oficinas del Este:

```
UPDATE oficinas
SET ventas = 0
WHERE region = 'Este';
```

Cuando para la condición de la cláusula WHERE necesitamos un dato de otra tabla podemos utilizar una subconsulta:

```
UPDATE empleados SET ventas = 0
WHERE oficina IN (SELECT oficina
                  FROM oficinas
                  WHERE region = 'Este');
```

Cuando el campo de la otra tabla se utiliza para la cláusula SET, entonces debemos utilizar la cláusula FROM.
La cláusula FROM permite definir un origen de datos basado en varias tablas, y ese origen será el utilizado para realizar la actualización.

Por ejemplo queremos actualizar el importe de los pedidos con el precio de la tabla productos.

```
UPDATE pedidos SET importe = cant * precio
FROM pedidos INNER JOIN productos
ON fab = idfab AND producto = idproducto;
```

Modificamos la tabla *pedidos* dejando en la columna *importe* el resultado de multiplicar la *cantidad* del pedido por el *precio* del producto que se encuentra en la tabla *productos*.

Como nos aparece un campo de otra tabla tenemos que añadir una FROM que permita obtener una tabla con cada pedido junto con los datos del producto que aparece en el pedido, lo que hacemos con el INNER JOIN. De esta forma en el resultado de la FROM tenemos en una misma fila un pedido y el precio con el que se actualiza el importe y la cláusula SET se podrá ejecutar.

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

Cuando una instrucción UPDATE encuentra un error aritmético (error de desbordamiento, división por cero o de dominio) durante la evaluación de la expresión, la actualización no se lleva a cabo. El resto del lote no se ejecuta y se devuelve un mensaje de error.

Además del permiso de UPDATE, se requieren permisos SELECT para la tabla que se actualiza si la instrucción UPDATE contiene una cláusula WHERE o en el caso de que el argumento *expression* de la cláusula SET utilice una columna de la tabla, y permisos SELECT para la tabla del origen si utilizamos una cláusula FROM o un WHERE con subconsulta.

Para finalizar, os recomiendo seguir estos pasos para redactar una UPDATE:

- Escribe UPDATE y la tabla cuyas columnas se van a modificar (el destino)
- Escribe la cláusula SET con las asignaciones a realizar
- Comprueba que en la parte izquierda de cada asignación sólo hay columnas del destino
- Si en la parte derecha de alguna asignación aparece la columna de otra tabla hay que añadir una cláusula FROM (definir un origen). En este caso piensa la composición adecuada para tener en las filas de ese origen todos los campos que aparecen en la SET.
- ¿Hay que actualizar todas las filas del destino? Si la respuesta es NO, entonces añade un WHERE
    - Si para la condición del WHERE necesitas una columna que no está en el destino ni en el origen (o no hay origen - FROM) utiliza una subconsulta.

See: http://msdn.microsoft.com/en-us/library/ms188724(v=SQL.90).aspx

5. Eliminar filas - DELETE

La sentencia DELETE elimina filas de una tabla. Si se borran todas las filas, o se borra la única fila de una tabla, la definición de la tabla no desaparece, sólo que la tabla se queda vacía.

```
DELETE
    [ TOP ( expression ) [ PERCENT ] ] [ FROM ] <destino>
    [ FROM <origen> ]
    [ WHERE < condicion> ]
[; ]

<destino> ::=
{
  [nbBaseDatos. nbEsquema. | nbEsquema.]nbTablaVista
}
```

Con esta instrucción podemos eliminar una o varias filas de una tabla.

La palabra FROM (la primera) es opcional (originalmente era obligatorio y sigue obligatorio en muchos SGBDs) y no añade funcionalidad sólo sirve para introducir el destino.

<destino> es el nombre de la tabla de donde queremos eliminar las filas, puede ser un nombre de tabla o un nombre de vista (de momento basada en una sólo tabla).

La segunda cláusula FROM sirve para indicar un origen que permita una condición de WHERE sobre una tabla diferente de destino. No se encuentra en muchos SGBDs por lo que no recomiendo su utilización, para la sentencia DELETE aconsejo utilizar el WHERE con subconsulta.

La instrucción básica sería pues:

```
DELETE oficinas;
```

Equivalente a:

```
DELETE FROM oficinas;
```

Con esta instrucción eliminamos todas las filas de la tabla *oficinas*.

La cláusula WHERE permite eliminar determinadas filas, indica una condición que deben cumplir las filas que se eliminan.

Por ejemplo:

```
DELETE oficinas
WHERE region = 'Este';
```

Elimina las oficinas del Este.

```
TOP ( expresion ) [ PERCENT ]
```

Especifica el número o porcentaje de filas aleatorias que se van a eliminar.
expression debe generar un valor numérico e indica el número de filas a eliminar empezando por el principio. Como en la SELECT, si añadimos la palabra PERCENT, el número representado por expresión se refiere al porcentaje de filas a eliminar sobre el total. La cláusula TOP funciona casi igual que en la SELECT pero en este caso, las filas no se ordenan, y la expresión debe ir entre paréntesis.

Por ejemplo:

```
DELETE TOP (10) PERCENT
FROM oficinas;
```

Elimina el 10% de filas de la tabla *oficinas*.

Originalmente sólo se podía indicar una tabla en la cláusula FROM, pero ahora podemos indicar un origen basado en varias tablas.
Si utilizamos un origen basado en varias tablas, se debe de utilizar una extensión de TRANSACT-SQL que consiste en escribir dos cláusulas FROM, una indica la tabla de donde eliminamos las filas y la otra el origen que utilizamos para eliminar.
Este caso se produce cuando las filas a eliminar dependen de un valor que está en otra tabla. Por ejemplo queremos eliminar los empleados de las oficinas del Este. Como la región de la oficina no está en empleados, habría que añadir al origen la tabla oficinas para poder formular la condición del WHERE:

```
DELETE FROM empleados
FROM empleados INNER JOIN oficinas
      ON empleados.oficina = oficinas.oficina
WHERE region = 'Este';
```

En el origen tenemos las dos tablas y en la primera FROM indicamos de qué tabla queremos borrar.

Esto se podía haber resuelto, como toda la vida, y como os aconsejo, mediante una subconsulta:

```
DELETE FROM empleados
WHERE oficina IN (SELECT oficina
                  FROM oficinas
                  WHERE region = 'Este');
```

Para finalizar no debemos olvidar que para poder ejecutar un DELETE se requieren permisos DELETE en la tabla de donde vamos a eliminar, y también se requieren los permisos para utilizar SELECT si la instrucción contiene una cláusula WHERE.

Muy importante siempre que actualicemos datos en nuestras tablas, no debemos olvidar tampoco las reglas de integridad referencial. Si la tabla afectada interviene como tabla principal en una relación con otra tabla, no se podrán eliminar sus filas que estén relacionadas con registros de la otra tabla (no se pueden eliminar 'padres' que tengan 'hijos'). Si se van a eliminar varias filas y al menos una no se puede eliminar por infringir las reglas de integridad, entonces la operación abortará y no se eliminará ninguna fila.

En el ejemplo anterior, si un empleado asignado a una oficina del Este tiene pedidos, no se podrá eliminar y entonces no se eliminará ningún empleado.

6. Borrado masivo - TRUNCATE

Si queremos eliminar todas las filas de una tabla podemos utilizar también la instrucción TRUNCATE TABLE.

```
TRUNCATE TABLE
   [nbBaseDatos.[nbEsquema.]| nbEsquema.]nbTabla  [; ]
```

Esta sentencia quita todas las filas de una tabla sin registrar las eliminaciones individuales de filas. Desde un punto de vista funcional, TRUNCATE TABLE es equivalente a la instrucción DELETE sin una cláusula WHERE; no obstante, TRUNCATE TABLE es más rápida y utiliza menos recursos de registros de transacciones y de sistema.

En comparación con la instrucción DELETE, TRUNCATE TABLE ofrece las siguientes ventajas:

- Se utiliza menos espacio del registro de transacciones.

  La instrucción DELETE quita una a una las filas y graba una entrada en el registro de transacciones por cada fila eliminada.
  TRUNCATE TABLE quita los datos al cancelar la asignación de las páginas de datos utilizadas para almacenar los datos de la tabla y sólo graba en el registro de transacciones las cancelaciones de asignación de páginas.

- Por regla general, se utilizan menos bloqueos.

  Si se ejecuta la instrucción DELETE con un bloqueo de fila, se bloquea cada fila de la tabla para su eliminación. TRUNCATE TABLE siempre bloquea la tabla y la página, pero no cada fila.

- Las páginas cero se conservan en la tabla sin excepciones.

  Después de ejecutar una instrucción DELETE, la tabla puede seguir conteniendo páginas vacías. Por ejemplo, no se puede cancelar la asignación de las páginas vacías de un montón sin un bloqueo de tabla exclusivo como mínimo. Si en la operación de eliminación no se utiliza un bloqueo de tabla, la tabla contiene muchas páginas vacías. En el caso de los índices, la operación de eliminación puede dejar páginas vacías, aunque la asignación de estas páginas se puede cancelar rápidamente mediante un proceso de limpieza en segundo plano.

Si la tabla contiene una columna de identidad, el contador para dicha columna se restablece al valor de inicialización definido para ella. Si no se define ningún valor de inicialización, se utiliza el valor predeterminado 1. Para conservar el contador de identidad, se utiliza DELETE.

Pero no todo son ventajas, no se puede utilizar TRUNCATE TABLE en las siguientes tablas:

- Tablas a las que se hace referencia mediante una restricción FOREIGN KEY (las tablas que entran como principales en una relación).
- Tablas que participan en una vista indizada.

See: http://msdn.microsoft.com/en-us/library/ms175544(v=SQL.90).aspx