IES CHAN DO MONTE
C.S. de Desarrollo de Aplicaciones Multiplataforma
Módulo Base de datos

UNIDAD 6: COMO REALIZAR MODIFICACIONES

Índice

1. ***Introducción*** ____ **2**
2. ***Inserción de datos. La sentencia INSERT*** ____ **2**
   - 2.1 **Insertar una fila mediante INSERT INTO …. VALUES** ____ **2**
   - 2.2 **Insertar varias filas mediante INSERT INTO …. SELECT** ____ **6**
   - 2.3 **Insertar datos en una tabla a través de un procedimiento** ____ **7**
   - 2.4 **Inserción creando una nueva tabla. La sentencia SELECT INTO.** ____ **7**
3. ***Eliminar datos. La sentencia DELETE y TRUNCATE*** ____ **9**
   - 3.1 **Sentencia DELETE** ____ **9**
   - 3.2 **DELETE con varias tablas** ____ **9**
   - 3.3 **Eliminar masivamente: TRUNCATE TABLE** ____ **11**
4. ***Actualizar datos. La sentencia UPDATE*** ____ **12**
   - 4.1 **sentencia UPDATE** ____ **12**
   - 4.2 **UPDATE con varias tablas** ____ **13**
5. ***Cláusula OUTPUT*** ____ **13**

1. Introducción

Dentro de este tema se van a estudiar las instrucciones del lenguaje DML (Lenguaje de manipulación de datos) de Transact-SQL que se emplean para **modificar los datos de una tabla** existente.

Se estudiarán las siguientes instrucciones:

- **INSERT: Insertar una o varias filas** en una tabla existente
- **SELECT ..INTO**: **Crear una nueva tabla** a partir de datos existentes
- **UPDATE**: Para realizar **cambios en una o varias columnas** de una o varias filas de una tabla
- **DELETE**: Para **eliminar una o varias filas** de una tabla
- **TRUNCATE**: Para **borrar todas las filas** de una tabla

2. Inserción de datos. La sentencia INSERT

Hay dos maneras de especificar los valores de datos:

- Utilizar una cláusula **VALUES** para especificar los **valores de datos para una fila**:

```
INSERT [INTO] tabla_o_vista  [(listaColumnas)]
VALUES (lista de valores_de_datos)
```

- Utilizar una **subconsulta SELECT** para especificar los **valores de datos para una o más filas.**

```
INSERT [INTO] tabla_o_vista
SELECT columnas
FROM tablas
```

2.1 Insertar una fila mediante INSERT INTO …. VALUES

La palabra clave **VALUES especifica los valores de una fila de una tabla.**

```
INSERT [INTO] tabla  [(listaColumnas)]
VALUES ( { DEFAULT | NULL | valor}[,...n] )
```

Dónde:

- ***tabla***. Es la tabla a la que queremos añadir registros
- ***lista de columna.*** son los campos que queremos rellenar para ese registro. Debe estar **encerrada entre paréntesis** y delimitada con *comas*.
  Si **una columna no está en listaColumnas,** se **debe poder proporcionar un valor basado en la definición de la columna**; en caso contrario, no se puede cargar la fila.
- ***Values:*** Son los valores que queremos dar a los distintos campos del registro. Pueden ser :
  - ✓ valor: Es una **constante, variable o expresión**.
    La expresión **no puede contener** una instrucción EXECUTE
  - ✓ **DEFAULT:** Se inserta **el valor predeterminado definido para una columna**.

Si **no hay un valor predeterminado** para la columna y **ésta permite el valor NULL**, se inserta **NULL**.
DEFAULT no es válido para una columna de identidad.

- ✓ **NULL:** inserta el valor NULL. La columna debe permitir valores nulos, sino daría error.

- Los **valores se especifican como una lista separada por comas,** y deben **ser del mismo tipo** que la columna correspondiente de la lista de columnas.
- La lista de valores debe estar **encerrada entre paréntesis.**
- Debe **haber un valor de datos para cada columna en listaColumnas** (si se especificó) o en la tabla.
- Si los valores de la lista **VALUES no están en el mismo orden que las columnas de la tabla** o no tienen un valor para cada columna de la tabla, **debe utilizarse listaColumnas** para especificar explícitamente la columna que almacena cada valor de entrada.
- **Se puede omitir la lista de columnas de la sentencia INSERT**, en cuyo caso se genera automáticamente una lista formada por todas **las columnas de la tabla en secuencia de izquierda a derecha.**
- Lógicamente, **cuando se omite la lista de columnas**, la palabra clave **NULL o DEFAULT debe ser utilizada** y la secuencia de valores de datos debe corresponder exactamente con la secuencia **de columnas de la tabla**.
- Se **proporciona automáticamente un valor para la columna,** si ésta tiene:
  - Tiene una **propiedad IDENTITY.** Se utiliza el siguiente valor de identidad incremental.
  - **Acepta valores NULL**. Se utiliza un valor NULL.
  - Tiene un **valor predeterminado**. Se utiliza el valor predeterminado de la columna.
  - Es **una columna calculada.** Se utiliza el valor calculado.

➢ *EJEMPLO DE INSERCIÓN INDIVIDUAL DE FILAS*

En el siguiente ejemplo se inserta una fila en la tabla *Departamento*. Las columnas de esta tabla son: *Num_departamento, Nombre_departamento, NSS_dirige y Fecha_direccion*.

Dado que los valores para **todas las columnas se suministran e incluyen en el mismo orden** que las columnas de la tabla, no es necesario especificar los nombres de columna en la lista de columnas.

```
INSERT INTO DEPARTAMENTO
 (Num_departamento,Nombre_departamento, NSS_dirige,Fecha_direccion)
 VALUES (7,'INVESTIGACION','8888889','7-7-2010')
--O también se puede omitir la lista de columnas
INSERT DEPARTAMENTO
 VALUES (7,'INVESTIGACION','8888889','7-7-2010')
```

➢ *EJEMPLO DE INSERTAR DATOS QUE NO ESTÁN EN EL MISMO ORDEN QUE LAS COLUMNAS DE LA TABLA*

En el siguiente ejemplo se debe usar una **lista de columnas para especificar de forma explícita** los valores insertados en cada columna. las **columnas no se incluyen en dicho orden en la lista de columnas**.

```
INSERT INTO DEPARTAMENTO  --Obligatorio incluir lista de columnas
(Nombre_departamento,Num_departamento,Fecha_direccion,NSS_dirige)
 VALUES ('MARKETING',8,GETDATE(),'SELECT NSS FROM EMPLEADOS WHERE NOMBRE='Pedro' and
 APELLIDO1='Smith' and APELLIDO2='Xil')  --la subconsulta solo debe dar una resultado,sino error.
```

➢ *EJEMPLO INSERTAR DATOS EN UNA TABLA CON COLUMNAS QUE TIENEN VALORES PREDETERMINADOS*

El ejemplo siguiente muestra cómo insertar filas en una tabla con columnas que generan automáticamente

un valor o tienen un valor predeterminado. La definición de la tabla es la siguiente:

```
CREATE TABLE TABLA1
(  id int IDENTITY CONSTRAINT PK_T1 PRIMARY KEY ,
   col1 AS 'Columna ' + col2, --Campo Calculado
   col2 varchar(30)
      CONSTRAINT DF_COL2 DEFAULT 'Valor defecto',
   col3 uniqueidentifier CONSTRAINT DF_COL3 DEFAULT NEWID(),
   col4 varchar(40) NULL
);
```

Ejemplo: **Las siguientes instrucciones INSERT son correctas**: insertan filas que contienen valores para algunas de las columnas, pero no para todas. En la última instrucción INSERT, no se especifica ninguna columna y solamente se insertan los valores predeterminados con la cláusula DEFAULT VALUES.

```
INSERT INTO Tabla1 (col4) VALUES ('Dato1');
INSERT INTO Tabla1 (col2, col4)  VALUES ('dato2', 'dato3');
INSERT INTO Tabla1 (col2)  VALUES ('dato4');
INSERT INTO Tabla1 --(col2,col3,col4) se pueden omitir ya que el resto de columnas se generan
                   --automáticamente
 VALUES ('dato5','A594CF2C-CF68-48BA-8CF1-828849488206','dato6');
INSERT INTO Tabla1 DEFAULT VALUES;
select * from tabla1
```

Resultados:

id	col1	col2	col3	col4
1	Columna Valor defecto	Valor defecto	25510DE4-09AD-4189-91CB-4D87431ABA15	**Dato1**
2	Columna dato2	**dato2**	AFAFDB46-8B4A-49A7-88C4-6ABCE83E7162	**dato3**
3	Columna dato4	**dato4**	53AA9E24-0A3A-414D-8DA7-69DDBD994D1B	NULL
4	Columna dato5	**dato5**	**A594CF2C-CF68-48BA-8CF1-828849488206**	**dato6**
5	Columna Valor defecto	Valor defecto	40C44CA2-C53B-458B-8269-1B1FED36EB71	NULL

Ejemplo: **Las siguientes instrucciones generan un error**, ya que no se **puede insertar valores de forma expícita en una columna calculada o de identidad** (cuando IDENTITY_INSERT es OFF). En el ejemplo anterior no se puede insertar explícitamente valores en las columnas Col1 ni en Col2

```
INSERT INTO Tabla1 (Col1,col2) VALUES ('dato7','dato8');--ERROR
INSERT INTO Tabla1 (id,col2)VALUES (6,'dato8'); --ERROR
```

➢ *EJEMPLO DE INSERTAR DATOS EN UNA TABLA CON UNA COLUMNA DE IDENTIDAD.*

De forma predeterminada, **no se pueden insertar datos directamente ni actualizarlos en una columna de identidad.**

Si se eliminó una fila y se quiere volver a insertar de nuevo esta fila que fué eliminada y se quiere conservar el valor de identidad original de esa fila, se puede omitir la configuración predeterminada mediante el uso de la sentencia:

```
SET IDENTITY_INSERT[ nombreBasedeDAtos].[Esquema ].nombreTabla {ON | OFF }
```

- **On**: se permite insertar una fila y asignar el valor de la columna el valor que se desee
- **OFF**: Es conveniente, tras acabar de insertar una fila, **de prohibir las inserciones en la columna identidad**, ejecutando el anterior comando con la opción OFF. SQL tomará **el valor mayor de identidad** para la columna en ese momento y lo usará como **el valor inicial**.

En el siguiente ejemplo se muestran los distintos métodos para insertar datos en una columna de identidad. La definición de la tabla es:

```
CREATE TABLE Animal(
  IDAnimal int IDENTITY NOT NULL PRIMARY KEY,
  Nombre VARCHAR(40) NOT NULL)
```

```
INSERT INTO Animal VALUES ('Vaca')
INSERT INTO Animal(Nombre) VALUES ('Gato')
/*la siguiente instrucción da error por que no se puede insertar un valor
  explícito SET IDENTITY_INSERT esta en OFF que es el valor predeterminado*/
INSERT INTO Animal(IDAnimal,Nombre) VALUES (3,'Loro')--ERROR

--Ahora si que se permite insertar un valor explicito
SET IDENTITY_INSERT Animal ON
INSERT INTO Animal(IDAnimal,Nombre) VALUES (4,'Loro')
/*Al desactivar la insercción explícita en una columna de identidad, el valor mayor insertado se toma como valor inicial para el incremento*/
SET IDENTITY_INSERT Animal OFF
INSERT INTO Animal VALUES ('Perro')
```

Resultado:

IDAnimal	Nombre
1	Vaca
2	Gato
4	Loro
5	Perro

Las siguientes funciones del sistema **devuelven información acerca de la definición de una columna de identidad**:

- **IDENT_SEED** (devuelve el valor inicial)
- **IDENT_INCR** (devuelve el valor incrementado)
- **@@IDENTITY o IDENT_CURRENT** para **recuperar el valor de identidad de la última fila insertada** durante una sesión.

Ejemplo:

```
SELECT IDENT_SEED('Animal') AS ValorInicial_Identidad,
    IDENT_INCR('Animal') AS Incremento_identidad,
    @@IDENTITY as ultimo_valor,
    IDENT_CURRENT ('Animal') AS ultimoValor
```

Resultado:

ValorInicial_Identidad	Incremento_identidad	ultimo_valor	ultimoValor
1	1	5	5

Se **puede utilizar** la palabra clave **$IDENTITY** en lugar del **nombre de la columna de identidad** cuando se consulta cualquier tabla que tenga una columna de identidad.

```
select $IDENTITY from Animal
--es equivalente
Select IdAnimal from Animal
```

2.2 Insertar varias filas mediante INSERT INTO .... SELECT

También es posible **insertar en una tabla el resultado de una consulta SELECT**. De este modo se insertarán tantas filas como haya devuelto la consulta SELECT.

La **sintaxis** es la siguiente:

```
INSERT INTO tabla[(lista de campos)]
SELECT tabla_derivada ;
```

Donde:

- ***tabla y lista de campos***. funcionan igual que en el caso anterior.
- ***tabla_derivada***. Es cualquier instrucción **SELECT** válida que devuelva filas con los datos que se van a cargar en la tabla.

  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 reglas correspondientes a cada uno de los campos:
  - La **lista de selección de la subconsulta debe coincidir** con la **lista de columnas de la instrucción INSERT**.
  - Si **no se especifica ninguna lista de columnas**, la **lista de selección debe coincidir** con las columnas de la **tabla o vista en la que se inserta**.

➢ *EJEMPLOS*

Ejemplo: Suponiendo que tenemos definida una tabla CopiaDept con la misma estructura que Departamentos, se inserta todos los registros de departamento en COPIADEPT

```
INSERT CopiaDept SELECT * FROM Departamentos
```

Ejemplo: Suponiendo que tenemos definida una tabla BuenosDept con la misma estructura que Departamentos. Insertar los departamentos con más de 3 empleados:

```
INSERT BuenosDept
SELECT D.*
FROM Departamentos D INNER JOIN Empleados E ON  D.NumDep= E.NumDep
GROUP BY D.NumDep, D.NombreDep, D.NumEmpdep, D.NSSGerente,
      D.FechaInicGerente
HAVING COUNT(*) >=3
```

Ejemplo, imaginemos que creamos una tabla para los clientes preferentes, donde vamos a almacenar los clientes que llevan con nosotros desde el año 2000:

```
/* Creamos la tabla */
CREATE TABLE  Clientes_Preferentes
          (fechaAlta   smalldatetime,
           nombre   varchar(20) NOT NULL,
           apellido1 varchar(25) NOT NULL,
          apellido2  varchar(25));
/* Insertamos los clientes preferentes*/
INSERT INTO  Clientes_Preferentes
SELECT fechaAlta, nombre, apellido1, apellido2
FROM  Clientes
WHERE YEAR(fechaAlta)>1999
```

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 hubiésemos escrito:

```
/* Insertamos los clientes preferentes*/
INSERT INTO  Clientes_Preferentes
SELECT nombre, apellido1, apellido2, fechaAlta
FROM  Clientes
WHERE YEAR(fechaAlta)>1999
```

Hubiese dado error porque la primera columna es de tipo fecha y el valor a asignar es texto (el nombre del cliente).

2.3 Insertar datos en una tabla a través de un procedimiento

Podemos **utilizar un procedimiento que nos devuelve un conjunto de filas para insertar datos**.

```
INSERT [INTO] tabla  [(listaColumnas)]
EXECUTE  nombreprocedimiento [parámetros]
```

- Un **procedimiento** es un **programa dentro de la base de datos** que **ejecuta una acción o conjunto de acciones específicas**.

  Un procedimiento tiene un **nombre, un conjunto de parámetros (opcional) y un bloque de código**.

  Los procedimientos almacenados pueden **devolver valores (numérico entero) o conjuntos de resultados**.

Para crear un procedimiento almacenado debemos emplear la sentencia **CREATE PROCEDURE**.

```
CREATE PROCEDURE <nombre_procedure> [@param1 <tipo>, ...]
AS
-- Sentencias del procedure
```

EJEMPLOS

Ejemplo: Tenemos una tabla departamento2 con la misma estructura que departamento.

```
CREATE PROCEDURE PROC_INSERTAR @supervisor varchar(15)
AS
SELECT * FROM departamento2 where NSS_dirige=@supervisor
```

Podemos utilizar el procedimiento para insertar las filas seleccionadas que cumplan la condición según el parámetro de entrada en la tabla departamento

```
INSERT DEPARTAMENTO EXECUTE PROC_INSERTAR '7777777'
```

También podemos utilizer **INSERT…..EXECUTE ('STRING')** donde string es una consulta.

```
INSERT DEPARTAMENTO EXECUTE ('SELECT * FROM departamento2
where NSS_dirige=''1111111''')
```

2.4 Inserción creando una nueva tabla. La sentencia SELECT INTO.

Se puede colocar **el conjunto de resultados de cualquier consulta en una nueva tabla** utilizando la instrucción **SELECT INTO**.

Una forma de insertar datos es crear una tabla que incluya los datos de otra.
Esta es la sentencia **SELECT... INTO.**

```
SELECT lista de campos
INTO Nueva_Tabla
FROM ...;
```

Donde**:**

- ***lista de campos***. es la lista de campos a incluir en la nueva tabla.
- ***Nueva_Tabla.*** es el nombre de la tabla que se va a crear, (si en la base de datos ya hay una tabla con ese nombre se 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.
- **No se transfiere ninguna otra propiedad del campo o de la tabla** como por ejemplo las claves e índices.
- Hay que asegurarnos de que el **nombre de la tabla** especificado en la instrucción SELECT INTO es **único**. Si existe una tabla con el mismo nombre, la instrucción SELECT INTO falla.
- Se deben **crear alias de columna para las columnas calculadas** en la **lista de selección**.

EJEMPLO

Se crea la tabla y se inserción la información utilizando una única instrucción.

```
SELECT fechaAlta, nombre, apellido1, apellido2
INTO Clientes_Preferentes
FROM  Clientes
WHERE YEAR(fechaAlta)>1999;
```

- Se puede **crear una tabla temporal local o global.**

Las tablas temporales son similares a las permanentes, salvo por el hecho de que las tablas temporales se almacenan en **tempdb** y se **eliminan automáticamente cuando ya no se utilizan**.

Hay **dos tipos de tablas temporales: locales y globales**

- Para crear una **tabla temporal local** situando antes del nombre de la tabla el signo almohadilla **(#)** o cree una **tabla temporal global** situando antes del nombre de la tabla un signo de doble almohadilla **(##)**.
- Una **tabla temporal local** sólo está disponible para **la conexión que la ha creado**, mientras que una **tabla temporal global** está disponible para **todas las conexiones:**
  - Una tabla temporal local se elimina cuando el usuario cierra la conexión.
  - Una tabla temporal global se elimina cuando la tabla ya no está siendo utilizada por ninguna conexión.

EJEMPLO

Ejemplo: Este ejemplo crea una tabla temporal local en una consulta realizada en la tabla Empleados

```
SELECT NSS, Nombre, Salario, Comision, FechaIngreso
INTO #EmpleadosSep1
FROM Empleados
WHERE NumDep = 1
```

Para **ver los metadatos de esta tabla temporal** podemos consultar la vista de catálogo **sys.tables** en la base de datos **Tempdb**

```
USE TEMPDB
SELECT * FROM SYS.TABLES
```

name	object_id	principal_id	schema_id	type	type_desc
#0D44F85C	222623836	NULL	1	U	USER_TABLE
#EmpleadosSep1_____ _____00000000006D	**334624235**	**NULL**	**1**	**U**	**USER_TABLE**

3. Eliminar datos. La sentencia DELETE y TRUNCATE

3.1 Sentencia DELETE

La sentencia **DELETE quita una o varias filas de una tabla o vista.** Una forma simplificada de la sintaxis de DELETE es:

```
DELETE [TOP ( expresion ) [ PERCENT ] ]
[FROM] tabla_o_vista
WHERE condición
```

- ***WHERE***: **Especifica las condiciones utilizadas para limitar el número de filas que se van a eliminar.**

  Si no se proporciona una cláusula WHERE, se borran **todas las filas de la tabla**.

  La palabra **FROM** es **opcional** (originalmente era obligatorio) y no añade funcionalidad

Observaciones

- **La instrucción DELETE puede fallar si infringe un desencadenador o se intenta quitar una fila a la que hacen referencia datos de otra tabla con una restricción FOREIGN KEY**. Si ocurre un error**, se cancela la instrucción, se devuelve un error y no se elimina ninguna fila.**
- **DELETE quita físicamente las filas de una en una y graba cada fila eliminada en el registro de transacciones.**

➢ *EJEMPLOS*

```
/* Eliminar todos los registros de la tabla FAMILIAR */
DELETE FROM  Familiar;
/* o bién */
DELETE Familiar;
/* Borrar el departamento de COMPRAS */
DELETE Departamento
WHERE nombre='COMPRAS';
/* Borrar el 10% de los Departamentos */
DELETE TOP (10) PERCENT
FROM Departamento;
```

3.2 DELETE con varias tablas

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 (donde podemos especificar DELETE FROM o DELETE solamente) y**
- **la otra el origen que utilizamos para eliminar**.

Ejemplo: Borrar los empleados de la tabla empleados que estén en la tabla TablaBorrar utilizando una extensión de una tabla.

```
Delete Empleados
from TablaBorrar –origen de datos otra tabla
where  TablaBorrar.NSS=Empleados.NSS –-importante enlazar las 2 tablas

Nota:--Si no lo enlazamos borraría todos los empleados
Delete Empleados
from TablaBorrar

--tb se podría hacer con subconsultas
Delete Empleados
Where nss in (select nss from TablaBorrar)
```

EJEMPLOS

-Borrar los empleados del departamento "Investigación" utilizando una extensión de combinación

```
DELETE Empleados    /* Podemos omitir el FROM de la 1ª fila (DELETE FROM) */
FROM Empleados INNER JOIN Departamentos
ON Empleados.NumDep = Departamentos.NumDep --Enlazar
WHERE NombreDep = 'Investigación'

-- o también

DELETE Empleados
FROM Departamentos
WHERE NombreDep = 'Investigación' and Empleados.NumDep = Departamentos.NumDep
                                                    --Enlazar
```

-Borrar los empleados que tenga un salario entre los cinco más altos.

```
--Utilizando una subconsulta para seleccionar los empleados a borrar
DELETE EMPLEADOS
WHERE SALARIO IN (SELECT DISTINCT TOP(5) SALARIO FROM EMPLEADOS
                        ORDER BY SALARIO DESC)

 --Utilizando una extension de combinación
 DELETE EMPLEADOS
FROM  EMPLEADOS E INNER JOIN
              (SELECT DISTINCT TOP(5) SALARIO FROM EMPLEADOS
               ORDER BY SALARIO DESC) AS S
  ON E.SALARIO=S.SALARIO  --IMPORTANTE

 --utilizando una extensión de tabla derivada
DELETE EMPLEADOS
FROM (SELECT DISTINCT TOP(5) SALARIO FROM EMPLEADOS
         ORDER BY SALARIO DESC) AS S
WHERE EMPLEADOS.SALARIO=S.SALARIO  --IMPORTANTE ENLAZAR
```

3.3 Eliminar masivamente: TRUNCATE TABLE

Si queremos **eliminar todos los registros de una tabla** podemos utilizar también la instrucción **TRUNCATE TABLE.**

Sintaxis:

```
TRUNCATE TABLE nombre_tabla
```

Esta sentencia **quita todas las filas de una tabla sin registrar las eliminaciones individuales de filas en el registro de transacciones**.

Desde un punto de vista funcional, **TRUNCATE TABLE** es equivalente a la instrucción **DELETE** sin una cláusula **WHERE**

TRUNCATE TABLE es más rápida y utiliza menos recursos de registros de transacciones y de sistema.

Si la tabla **contiene una columna de identidad**, el contador para d**icha columna se restablece al valor de inicialización** definido para ella. Para **conservar el contador de identidad,** se utiliza **DELETE**.

Cuando trabajamos con **TRUNCATE TABLE** debemos tener en cuenta las siguientes consideraciones:

- **TRUNCATE TABLE** no admite la cláusula **WHERE**.
- No podemos ejecutar **TRUNCATE TABLE sobre tablas que sean "padres" en foreign keys**.

```
/* Eliminar todos los registros de la tabla FAMILIAR */
TRUNCATE TABLE  Familiar;
```

DIFERENCIAS ENTRE TRUNCATE Y DELETE

TRUNCATE TABLE	DELETE FROM
Es una operación **DDL**.	Es una operación **DML**.
No permite el borrado selectivo. TRUNCATE TABLE elimina todo el contenido de la tabla.	**Permite el borrado selectivo**, mediante la cláusula WHERE.
No se puede ejecutar, si la tabla tiene asociadas, aun si no existiesen registros en la tabla que contiene la FK.	Se puede ejecutar si hay FK asociadas a la tabla, pero siempre y cuando no tenga registros asociados o la FK este deshabilitada.
Es la forma **más rápida** de eliminar el contenido de una tabla.	**Es más lenta**.
No se activa ningún trigger (desencadenador)	Puede activarse el **trigger de ON DELETE** y poder determinar que registros están siendo eliminados
En caso de que la tabla tuviese un campo **Identity, se resetea el valor a 1** (o al valor base determinado en el campo).	**No reinicia el valor del campo Identity**, en caso de que la tabla tuviese uno.
No se registra en el fichero log de transacciones, con lo cual no se puede recuperar las filas si ocurriera algún error	**Registra cada operación en el fichero de log** sobre los registros afectados.

4. Actualizar datos. La sentencia UPDATE

4.1 sentencia UPDATE

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

Sintaxis:

```
UPDATE tabla
SET  campo1 = valor1 | DEFAULT | NULL {[,campo2 = valor2,...
       ,campoN = <valorN]}
[WHERE condicion];
```

- ***tabla***: Es el nombre de la tabla que se va a actualizar.

  Las modificaciones realizadas por la instrucción UPDATE no pueden **afectar a más de una de las tablas base** a las que se hace referencia en la cláusula FROM de la vista.

- ***SET***: Especifica la lista de nombres de columnas o variables que se van a actualizar.
- ***Campo1,campo2,...***: columna o columna que contiene los datos que se van a cambiar.
  - **No se pueden actualizar las columnas de identidad**.
- ***Valor***: los valores que queremos dar a los distintos campos pueden ser :
  - valor: Es una **constante, variable o expresión**.
    La expresión **no puede contener** una instrucción EXECUTE
  - DEFAULT: Se modifica por **el valor predeterminado definido para la columna**.
  - NULL: se modifica por el valor NULL. La columna debe permitir valores nulos, sino daría error.
- ***WHERE***: **Especifica las condiciones utilizadas para limitar el número de filas que se van a modificar**

  Si no se proporciona una cláusula WHERE, se modifican **todas las filas de la tabla**.

Observaciones

- Si la **actualización de una fila infringe una restricción o una regla**, **infringe la configuración de valores NULL para la columna o si el nuevo valor es de un tipo de datos incompatible**, se **cancela la instrucción, se devuelve un error y no se actualiza ningún registro**.
- Cuando una instrucción **UPDATE encuentra un error aritmético** (desbordamiento, división por cero o un error 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.
- **La instrucción UPDATE se registra en el registro de transacciones (log).**

➢ *EJEMPLOS*

Al empleado cuyo número es 44444 se le cambia el departamento a 2 y el salario a 1500.

```
UPDATE Empleados SET NumDep=2, Salario=1500
WHERE NSS=44444
```

4.2 UPDATE con varias tablas

Cuando para la condición de la cláusula WHERE **necesitamos un dato de otra tabla** podemos utilizar una subconsulta o una combinación de tablas.

Ejemplo: A los empleados del departamento de investigación, aumentar el salario un 15%

```
/* CON COMBINACIÓN DE TABLAS*/
UPDATE Empleados
SET Salario = Salario * 1.15
FROM Empleados INNER JOIN Departamentos
   ON Empleados.NumDep = Departamentos.NumDep
WHERE NombreDep =  'Investigación'

/* CON SUBCONSULTAS*/
UPDATE Empleados
SET Salario = Salario * 1.15
FROM Empleados and NumDep=
    (SELECT NumDep FROM DEPARTAMENTO WHERE NombreDep = 'Investigación'
```

Ejemplo: Cuando el **campo de la otra tabla se utiliza para la cláusula SET**, entonces debemos utilizar la cláusula FROM con combinación de tablas. Por ejemplo:

```
UPDATE  Pedidos
SET importe=cantidad*precio
FROM Pedidos INNER JOIN productos ON cod_producto=producto.codigo;
```

5. Cláusula OUTPUT

A partir de la versión de SQL Server 2005 disponemos de la cláusula **OUTPUT para recuperar los valores que hemos insertado.**

Al igual que en un trigger disponemos de las **tablas lógicas INSERTED y DELETED**._

- Las columnas con prefijo **DELETED** reflejan el **valor antes de que se complete la instrucción UPDATE o DELETE**. Es decir, son **una copia de los datos "antes" del cambio**. **DELETED no se puede utilizar con la cláusula OUTPUT en la instrucción INSERT**.
- Las columnas con **prefijo INSERTED reflejan el valor después de que se complete la instrucción UPDATE o INSERT**, pero antes de que se ejecuten los desencadenadores. Es decir, son una copia de los datos "despues" del cambio. **INSERTED no se puede utilizar con la cláusula OUTPUT en la instrucción DELETE**.

Vamos a ver su funcionamiento con un ejemplo, cuando insertamos datos en una tabla:

```
DECLARE @FILAS_INSERTADAS TABLE
(nss char(9),
 nombre varchar(25),
 apellido1 varchar(25),
 apellido2 varchar(25),
 fechaAlta  smalldatetime,
 localidad varchar(40)
);
```

```
INSERT INTO Clientes (nss, nombre, apellido1, apellido2, fechaAlta,
                localidad)
OUTPUT INSERTED.* INTO @FILAS_INSERTADAS
VALUES ('44344644','Sonia','Vila','Dominguez',DEFAULT, 'Lugo');

SELECT * FROM  @FILAS_INSERTADAS;
--Y ahora, vamos a ver su funcionamiento cuando actualizamos datos en una tabla:
DECLARE @FILAS_ACTUALIZADAS TABLE
(nss char(9),
 nombre varchar(25)
);

UPDATE Clientes
SET nombre='Juan José'
OUTPUT DELETED .* INTO @FILAS_ACTUALIZADAS
WHERE nss='44111644';

SELECT * FROM  @FILAS_ ACTUALIZADAS;
```