Sobre Transact-SQL

Ejemplos de...

Previo

Ejecuta esto para tener unos datos mínimos sobre los que trabajar

```
USE master
GO
--IF DB_ID ('probando') IS NOT NULL
--DROP DATABASE probando;
--GO
DROP DATABASE IF EXISTS probando; -- por si acaso, elimina antes
CREATE DATABASE probando -- y vuelve a crear
GO
use probando
go
CREATE TABLE trantest (Col1 int primary key, Col2 int)

CREATE TABLE trantest2 (ColA int primary key, ColB int)

CREATE TABLE cuenta (id integer primary key, saldo decimal(6,2))
go

insert into cuenta values (1,6000.00);
insert into trantest values (1,0),(2,0),(3,2),(4,2);
```

Crear nuevas tablas a partir de consultas

SELECT: cláusula INTO

Si ejecutamos una consulta obtenemos resultados

select *columnas* from *tablas* where *condiciones*

Si queremos guardar el resultado en una nueva tabla

select *columnas* **into nuevatabla** from *tablas* where *condiciones*

La nueva tabla solo replica columnas y tipos de datos de las columnas, cualquier otra restricción habría que añadirla con alter table.

Crear una tabla temporal a partir de una consulta ("copiar" una tabla)

Se van a usar tablas temporales pero vale igualmente para tablas persistentes, "normales".

Usando todas las columnas

```
select *  into #ttemporal from trantest
select * from #ttemporal;
```

Results | Mes

	Col1	Col2
1	1	0
2	2	0
3	3	2
4	4	2

Usando algunas columnas, en otro orden, y filtrando las filas a copiar

```
select col2, col1 into #ttemporal2 from trantest where col2 < 1
select * from #ttemporal2;
```

Creando una columna IDENTITY (identificador automático)

La función INDENTITY(tipoDatos,semilla,incremento) genera un valor del tipo tipoDatos, secuencial desde "semilla" y a incrementos de "incremento". Solo se usa en una instrucción SELECT con una cláusula INTO table

```
select IDENTITY(int,100,1) orden,col2  into #ttemporal3 from trantest
select * from #ttemporal3;
```

Results | Messages

	orden	col2
1	100	0
2	101	0
3	102	2
4	103	2

Hemos copiado la columna col2 de trantest y hemos añadido una columna IDENTITY.

Copiando la tabla trantest entera, añadiendo un número de orden, y ordenando aleatoriamente las filas.

```
select IDENTITY(int,100,1) orden,col1,col2  into #ttemporal4 from trantest
order by newid()
select * from #ttemporal4;
```

Results | Messages

	orden	col1	col2
1	100	1	0
2	101	4	2
3	102	2	0
4	103	3	2

La función newid() genera un número único con el tipo de datos uniqueidentifier. Simplemente nos "asegura" que no habrá duplicados, pero no necesariamente sigue una secuencia incremental en su generación, puede resultar en "cualquier valor". Al utilizarlo en order by, la ordenación de las filas se convierte en aleatoria.

Fíjate que IDENTITY() sigue el orden natural, es como si se ejecutara primero la consulta y después se añadiera la nueva columna.

Numerando filas con row_number()

La función row_number() hace eso, numera las filas, solo que lo hace según el criterio de ordenación que nosotros le digamos.

Numera filas después de ordenar por col1

```
select ROW_NUMBER() over (order by col1) orden,col1,col2 from trantest
```

Results | Messages

	orden	col1	col2
1	1	1	0
2	2	2	0
3	3	3	2
4	4	4	2

Numera filas después de ordenar por col2, y para valores duplicados de col2, entonces por col1.

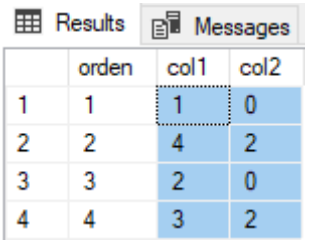
```
select ROW_NUMBER() over (order by col2,col1) orden,col1,col2 from trantest
```

Results | Messages

	orden	col1	col2
1	1	1	0
2	2	2	0
3	3	3	2
4	4	4	2

Numera filas después de ordenar las filas aleatoriamente.

```
select ROW_NUMBER() over (order by newid()) orden,col1,col2 from trantest
```



Results | Messages

	orden	col1	col2
1	1	1	0
2	2	4	2
3	3	2	0
4	4	3	2

La conclusión es que primero ordena por el criterio que le pasamos en la cláusula over, y después numera.

Insertar en una tabla el resultado de una consulta

Se trata de insertar filas en una tabla a partir de las filas resultado de una consulta. Es algo parecido a la creación de tabla anterior pero

1. la tabla destino ya está creada
2. la consulta debe devolver datos compatibles con las columnas destino
3. la tabla destino puede tener restricciones que imposibiliten la inserción.

Trantest y trantest2 son iguales estructuralmente ya que tienen 2 columnas cada una, y los tipos de datos de esas columnas son compatibles en el mismo orden:
columna 1: integer
columna 2: integer
Lo único que cambia es el nombre de las columnas respectivas.

Por lo tanto, es posible hacer una inserción múltiple como esta

```
insert into trantest2 (colA,colB) select col1,col2 from trantest
```

Se realiza la consulta sobre la tabla trantest (origen), se obtienen las filas, y se intentan insertar en la tabla trantest2 (destino).

La consulta puede ser todo lo compleja que queramos, solo debemos asegurarnos de las columnas que devolvemos y de sus tipos de datos.

Actualización de filas de más de una tabla

Preparamos los datos.

```
delete from trantest2
insert into trantest2 values (1,10),(3,20),(5,50);
```

La situación es algo así como la tabla trantest2 contiene varias modificaciones de filas de la tabla trantest, incluso hay una fila en la primera (5,50) que no existe en la segunda. Recuerda que trantest.col1 y trantest2.colA son claves primarias de sus respectivas tablas. Consultemos con un *full join* para tener una perspectiva clara de qué datos se van a modificar y por qué valores:

```
select * from trantest t1 full join trantest2 t2 on t1.col1=t2.colA
```

Results Messages

	Col1	Col2	ColA	ColB
1	1	0	1	10
2	2	0	NULL	NULL
3	3	2	3	20
4	4	2	NULL	NULL
5	NULL	NULL	5	50

Lo que pretendemos es, en una única instrucción, actualizar todas aquellas filas de trantest que lo necesiten. Serían las filas 1 y 3. La 5 no la tenemos en consideración ya que se trataría de una inserción.

Estas dos consultas obtienen el mismo resultado:

```
select col1,col2,colA,colB from trantest t1, trantest2 t2 where t1.col1=t2.colA
select col1,col2,colA,colB from trantest t1 join trantest2 t2 on (t1.col1=t2.colA)
```

Results Messages

	col1	col2	colA	colB
1	1	0	1	10
2	3	2	3	20

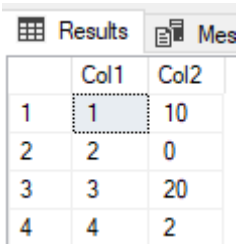
Update admite varias tablas sobre las que trabajar —con ciertas limitaciones—. Viene a ser algo así como que hacemos una consulta que involucra a varias tablas y un update sobre ese resultado. Una explicación para MySQL pero que explica lo que queremos hacer: "A vueltas con el update".

Esta orden se acepta en MySQL y otros servidores de bases de datos **pero no en SQL Server**:

```
update
trantest t1 join trantest2 t2 on (t1.col1=t2.colA)
set t1.col2=t2.colB
```

En Transact-SQL debemos reescribir la orden como

```
update trantest
set col2=t2.colB
from trantest t1 join trantest2 t2 on (t1.col1=t2.colA)
```



Results | Mes

	Col1	Col2
1	1	10
2	2	0
3	3	20
4	4	2

Fíjate en que debemos decirle a SQL Server qué tabla es la que va a recibir modificaciones, la columna que se modifica y el valor nuevo. Este valor nuevo hace referencia a una columna del *join* que se explicita a continuación.

Otra posibilidad es utilizar expresiones comunes de tabla.

Cursores

Se trata de refrescar nuestras habilidades con los cursores. Este tipo de objetos pertenecen al estándar de SQL y surgieron ante la necesidad del procesamiento de resultados fila a fila.

Los ejemplos siguientes extraen una parte del listado de unidades, concretamente las filas 2 y 3.

Base de datos inicial

```
USE ventas
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

if OBJECT_ID ( 'articulo','U' ) is not null
DROP table articulo;
go
if OBJECT_ID ( 'unidad','U' ) is not null
DROP table unidad;
go


CREATE TABLE unidad(
unidad nchar(1),
```

```
descripcion nvarchar(20),
CONSTRAINT PK_unidad PRIMARY KEY (unidad)
)
GO

insert into unidad values ('u','unidades')
insert into unidad values ('g','gramos')
insert into unidad values ('k','kilogramos')
insert into unidad values ('l','litros')
insert into unidad values ('d','decenas')
```

Ascendente

```
select * from unidad order by descripcion
go

declare @i integer
declare @launidad nchar(1)
declare @ladescripcion nvarchar(15)

DECLARE micursor CURSOR
FOR SELECT unidad, descripcion FROM unidad ORDER BY descripcion
OPEN micursor


-- Primera fila
set @i = 1
FETCH NEXT FROM micursor INTO @launidad, @ladescripcion

-- continuar
WHILE @@FETCH_STATUS = 0
BEGIN
if (@i >= 2 and @i <=3) print (@launidad+': '+@ladescripcion);
set @i = @i + 1
FETCH NEXT FROM micursor INTO @launidad, @ladescripcion
END

CLOSE micursor
DEALLOCATE micursor
GO
```

Descendente

Claramente, para obtener la segunda y tercera filas comenzando desde la última solo hay que redefinir el cursor en el lote anterior:

```
DECLARE micursor CURSOR
```

FOR SELECT unidad, descripcion FROM unidad ORDER BY descripcion **desc**

Resultados temporales (*common table expression*)

WITH common_table_expression

Es un resultado temporal que se puede utilizar en la siguiente instrucción. Viene a ser algo así como una preparación previa de datos para una instrucción posterior.

WITH cons_temp **AS** *consulta*
instrucción…;

Constemp es un nombre que actúa como si de un nombre de tabla se tratara. Simplificando, el resultado de consulta se mantiene para que lo aproveche instrucción —insertar, eliminar, modificar, consultar, etc.—. Se pueden definir tantos resultados temporales como se necesiten:

WITH cons_temp1 **AS** *consulta1,*
cons_temp2 AS *consulta2*,
...
instrucción…;

Preparamos los datos para la demostración.

delete from trantest
insert into trantest values (1,0),(2,0),(3,2),(4,2);
delete from trantest2
insert into trantest2 values (1,10),(3,20),(5,50);

El uso de este tipo de expresiones consiste en preparar una consulta previa a la que se va a hacer referencia inmediatamente después:

with tw
as (select col1,col2,colB
from **trantest t1, trantest2 t2 where** t1.col1=t2.colA)
update **tw** set **col2=colB**

Fíjate que no hay punto y coma detras del paréntesis de la consulta.

Es como si definiéramos una vista temporal de los datos que nos interesan y, después, realizáramos la actualización. Aquí lo importante es darse cuenta de que no hay ninguna ambigüedad: la columna col2 pertenece a trantest y colB a trantest2. Si se diera el caso de columnas que se llaman igual, habría que renombrar una de ellas. Supongamos este caso:

delete from trantest
insert into trantest values (1,0),(2,0),(3,2),(4,2);

```
CREATE TABLE trantest3 (Col1 int primary key, Col2 int)
insert into trantest3 values (1,100),(3,200),(5,500);

with tw
        as (select t1.col1,t1.col2,t3.col2 otra
                from trantest t1, trantest3 t3 where t1.col1=t3.col1)
update tw set col2=otra
```

Piensa que, una vez ejecutada la consulta, y antes del *update*, los nombres de columna ya no son ambiguos.

Results | Messages

	col1	col2	otra
1	1	0	100
2	3	2	200

SQL Parametrizado

A veces nos interesa ir construyendo poco a poco una orden para, después, ejecutarla. Generalmente son situaciones en las que dependemos de parámetros o variables, incluso de flujos condicionales que hacen que la orden sea diferente en cada momento.

Supongamos:

```
select *
from comkeyw ck join comentario c on ck.usuId=c.usuId and ck.numcom=c.numcom
where keyw = 'asequible'
order by cuando ASC;
```

	keyw	usuId	numcom	usuId	numcom	comenta	respondea	respondeanum	cuando
1	asequible	2	3	2	3	Esto es un comentario	NULL	NULL	2017-05-25 00:00:00.0000000
2	asequible	53	7	53	7	Esto es un comentario	NULL	NULL	2017-05-31 00:00:00.0000000

Sabemos que podemos parametrizar el valor de keyw, esto es, preparar la orden para que se ejecute con valores diferentes de una variable. Piensa que el objetivo es ejecutarla dentro de un procedimiento almacenado, de tal forma que podamos hacer algo así como:

```
execute miconsulta ‘asequible’
execute miconsulta ‘informática’
```

Sin declarar procedimientos, simplemente con variables, sabemos que podemos hacer esto:

```
declare @keyword nvarchar(25)

set @keyword='asequible'

select *
```

```
from comkeyw ck join comentario c on ck.usuId=c.usuId and ck.numcom=c.numcom
where keyw = @keyword
order by cuando ASC;
```

No hay problema, esto funciona porque lo que se espera es un valor escalar, en este caso una cadena de caracteres.

Pero, ¿y si quisiéramos parametrizar la columna por la que ordenamos y el criterio, ascendente o descendente? Con ello podríamos usar la misma consulta pero indicando que ordene los datos de otra forma.

```
declare @keyword nvarchar(25),
@columna nvarchar(30),
@orden char(4)

set @keyword='asequible'
set @columna='cuando'
set @orden='ASC'

select *
from comkeyw ck join comentario c on ck.usuId=c.usuId and ck.numcom=c.numcom
where keyw = @keyword
order by @columna @orden;
```

Verás que te salta un error de compilación, SQL Server no entiende la orden que le hemos enviado. En realidad, en "order by" espera una palabra clave, no una cadena de caracteres: *`order by 'cuando' 'ASC'`*.

Una solución es SQL dinámico o, en este caso, parametrizado. Vamos a utilizar un procedimiento del sistema, sp_executesql, que ofrece más ventajas y versatilidad que únicamente execute.

Necesitamos una variable de texto para ir construyendo la orden SQL. También necesitamos indicar qué va ser parámetro y qué no. Para no complicarlo mucho, vamos a pensar que solo parametrizamos la columna de ordenación.

```
DECLARE @sql nvarchar(1000),
    @queEsParametro nvarchar(255),
    @valorDelParametro nvarchar(25)
SET @queEsParametro = '@lacolum nvarchar(25)'
SET @valorDelParametro = 'cuando'

SET @sql = 'select * from comkeyw ck join comentario c on ck.usuId=c.usuId and
ck.numcom=c.numcom where keyw = \'asequible\' order by @lacolum ASC'

EXEC sp_executesql @sql, @queEsParametro, @valorDelParametro
```

El primer parámetro de sp_executesql es la propia orden sql "preparada". El segundo parámetro (`@queEsParametro`) le permite identificar dónde colocar el valor del parámetro

dentro de @sql. Finalmente, **@valorDelParametro**, es la columna que quiero utilizar para ordenar.

Habrás visto, además, que al construir la cadena de texto para @sql, hemos puesto

```
'select * ... where keyw = \'asequible\' ...ASC'.
```

Es común a todos los lenguajes de programación que, cuando tengo que utilizar un delimitador —como es la comilla simple en SQL— como un carácter más dentro de una cadena de texto, tengo que "escapar" ese carácter. En otras palabras, el compilador de SQL entiende que es un carácter más, no el final de la cadena de texto.

SQL dinámico

Lo cierto es que otra posibilidad es construir la orden tal cual, sin parámetros. Imagina que lo que pretendes es construir un procedimiento miconsulta @keyword @columna @criterio:

exec miconsulta `'cuando','ASC'`
exec miconsulta `'cuando','DESC'`
exec miconsulta `'numcom','DESC'`

En cada ejecución puedes cambiar la columna o el criterio de ordenación.

```
create procedure miconsulta
        @keyword nvarchar(25),
        @columna nvarchar(20),
        @criterio nchar(4)
as
DECLARE @sql nvarchar(1000)

SET @sql =
'select * from comkeyw ck join comentario c on ck.usuId=c.usuId and
ck.numcom=c.numcom where keyw = '
+ @kewword
+' order by ' + @columna + ' ' + @criterio

EXEC(@sql)
go
```

Además

- CASE...WHEN

- RIGHT, REPLICATE, LEN,
- WHILE
- Dos ejemplos muy simples de ~~SQL dinámico~~, y otro.
- Números aleatorios de 1 a 7; por ejemplo: `round(7*rand()+1,0,1)`