DML. Consultas multitabla
DML. Multi-table Queries

Índice

1. Introducción

Hasta ahora hemos visto consultas que obtienen los datos de una sola tabla, en este tema veremos cómo obtener datos de diferentes tablas.

En esta parte ampliaremos la cláusula FROM y descubriremos nuevas palabras reservadas (UNION, EXCEPT, INTERSECT, etc.) que corresponden a operaciones relacionales.

Para obtener datos de varias tablas tenemos que combinar estas tablas mediante alguna operación basada en el álgebra relacional *RELATIONAL ALGEBRA*.

El álgebra relacional define una serie de operaciones cuyos operandos son tablas y cuyo resultado es también una tabla.

Las operaciones de álgebra relacional implementadas en Transact-Sql son:

- La unión UNION
- La diferencia EXCEPT
- La intersección INTERSECT
- El producto cartesiano CROSS JOIN
- La composición interna INNER JOIN
- La composición externa LEFT JOIN, RIGHT JOIN Y FULL JOIN

En todo el tema cuando hablemos de tablas nos referiremos tanto a las tablas que físicamente están almacenadas en la base de datos como a las tablas temporales y a las resultantes de una consulta o vistas.

2. La unión de tablas UNION

La unión de tablas consiste en coger dos tablas y obtener una tabla con las filas de las dos tablas, en el resultado aparecerán las filas de una tabla y, a continuación, las filas de la otra tabla.

Para poder realizar la operación, las dos tablas tienen que tener el mismo esquema (mismo número de columnas y tipos compatibles) y la tabla resultante hereda los encabezados de la primera tabla.

La sintaxis es la siguiente:

```
{< consulta >|(< consulta >)}
   UNION [ALL]
 {< consulta >|(< consulta >)}
 [{UNION [ALL] {< consulta >|(< consulta >)}[ ...n ] ]
 [ORDER BY {expression_columna|posicion_columna [ASC|DESC]}
            [ ,...n ]]
```

< consulta > representa la especificación de la consulta que nos devolverá la tabla a combinar.

Puede ser cualquier especificación de consulta con la limitación de que no admite la cláusula ORDER BY, los alias de campo se pueden definir pero sólo tienen efecto cuando se indican en la primera consulta ya que el resultado toma los encabezados de esta.

Ejemplo: Suponemos que tenemos una tabla *Valencia* con las nuevas oficinas de Valencia y otra tabla *Madrid* con las nuevas oficinas de Madrid y queremos obtener una tabla con las nuevas oficinas de las dos ciudades:

```
SELECT oficina as OFI, ciudad FROM Valencia
UNION ALL
SELECT oficina, ciudad FROM Madrid;
```

El resultado sería:

OFI	ciudad
11	Valencia
28	Valencia
23	Madrid

El resultado coge los nombres de columna de la primera consulta y aparecen primero las filas de la primera consulta y después las de la segunda.

Si queremos que el resultado aparezca ordenado podemos incluir la cláusula ORDER BY, pero después de la última especificación de consulta, y expresion_columna será cualquier columna válida de la primera consulta.

```
SELECT oficina as OFI, ciudad FROM Valencia
UNION
SELECT oficina, ciudad FROM Madrid
ORDER BY ofi;
```

OFI	ciudad
11	Valencia
23	Madrid
28	Valencia

Ahora las filas aparecen ordenadas por el número de oficina y hemos utilizado el nombre de columna de la primera consulta.

Cuando aparezcan en el resultado varias filas iguales, el sistema por defecto elimina las repeticiones.

Si se especifica ALL, el sistema devuelve todas las filas resultante de la unión incluidas las repetidas

El empleo de ALL hace que la consulta se ejecute más rápidamente ya que el sistema no tiene que eliminar las repeticiones por lo que habrá que utilizarlo siempre que sea posible (cuando no pueda darse el caso de filas repetidas, cuando queremos que aparezcan las repticiones o cuando nos da igual que salgan o no repeticiones).

Se pueden combinar varias tablas con el operador UNION. Por ejemplo supongamos que tenemos otra tabla Pamplona con las oficinas nuevas de Pamplona:

```
SELECT oficina, ciudad FROM Valencia
UNION
SELECT oficina, ciudad FROM Madrid
UNION
SELECT oficina, ciudad FROM Pamplona;
```

Combinamos las tres tablas.

Otro ejemplo:

Obtener todos los productos cuyo precio exceda de 20 € o que se haya vendido más de 300 euros del producto en algún pedido.

```
 SELECT idfab, idproducto
 FROM  productos
 WHERE precio > 20
 UNION
 SELECT fab, producto
 FROM pedidos
 WHERE importe > 300;
```

Read more: http://technet.microsoft.com/en-us/library/ms180026(SQL.90).aspx

3. La diferencia EXCEPT

Aparecen en la tabla resultante las filas de la primera consulta que no aparecen en la segunda.

Las condiciones son las mismas que las de la unión.

```
   {<consulta>|(<consulta>)}
   EXCEPT
 {<consulta>|(<consulta>)}
 [{EXCEPT {<consulta>|(<consulta>)}}[ ...n ] ]
 [ORDER BY {expression_columna|posicion_columna [ASC|DESC]}
            [ ,...n ]]
```

Por ejemplo tenemos las tablas T1 y T2

T1: Cod	T2: Codigo
1	2
2	3
4	4
5	5
6	

```
SELECT cod FROM T1
EXCEPT
SELECT codigo FROM T2;
```

Devuelve:

Cod
1
6

Ejemplo:

Listar los productos que no aparezcan en ningún pedido.

```
 SELECT idfab, idproducto
 FROM  productos
 EXCEPT
 SELECT DISTINCT fab, producto
 FROM pedidos;
```

4. La intersección INTERSECT

Tiene una sintaxis parecida a las anteriores pero en el resultado de la intersección aparecen las filas que están simultáneamente en las dos consultas.

Las condiciones son las mismas que las de la unión.

```
   { <consulta>|(<consulta>)}
   INTERSECT
 {<especificacion_consulta>|(<especificacion_consulta>)}
    [{INTERSECT {<consulta>|(<consulta>)}} [ ...n ] ]
 [ORDER BY {expression_columna|posicion_columna [ASC|DESC]}
            [ ,...n ]]
```

Retomando el ejemplo anterior:

```
SELECT cod FROM T1
INTERSECT
SELECT cod FROM T2;
```

Devuelve:

Cod
2
4
5

Ejemplo:

Obtener todos los productos que valen más de 20 euros y que además se haya vendido en un pedido más de 300 euros de ese producto.

```
SELECT idfab, idproducto
FROM  productos
WHERE precio > 20
INTERSECT
SELECT fab, producto
FROM pedidos
WHERE importe > 300;
```

Read more: http://technet.microsoft.com/en-us/library/ms188055(SQL.90).aspx

5. La composición de tablas

Hasta ahora hemos operado con tablas que tenían el mismo esquema, pero muchas veces lo que necesitamos es obtener una tabla que tenga en una misma fila datos de varias tablas, por ejemplo, obtener las facturas y que en la misma fila de factura aparezca el nombre y dirección del cliente. Pues en lo que queda del tema estudiaremos este tipo de consultas basadas en la composición de tablas. La composición de tablas consiste en obtener a partir de dos tablas cualesquiera una nueva tabla fusionando las filas de una con las filas de la otra, concatenando los esquemas de ambas tablas. Consiste en formar parejas de filas.

La sentencia SELECT permite realizar esta composición, incluyendo dos o más tablas en la cláusula FROM.

Es hora de ampliar la cláusula FROM que vimos en el tema anterior.

Empezaremos por estudiar la operación a partir de la cual están definidas las demás operaciones de composición de tabla, el producto cartesiano.

5.1. El producto cartesiano CROSS JOIN

El producto cartesiano *CARTESIAN PRODUCT* obtiene todas las posibles concatenaciones de filas de la primera tabla con filas de la segunda tabla.

Se indica escribiendo en la cláusula FROM los nombres de las tablas separados por una coma o utilizando el operador CROSS JOIN.

```
FROM {<tabla_origen>} [ ,...n ]
   |<tabla_origen> CROSS JOIN <tabla_origen>
```

Tabla_origen puede ser un nombre de tabla o de vista o una tabla derivada *DERIVED TABLE* (resultado de una SELECT), en este último caso la SELECT tiene que aparecer entre paréntesis y la tabla derivada debe llevar asociado obligatoriamente un alias de tabla. También puede ser una composición de tablas.

Se pueden utilizar hasta 256 orígenes de tabla en una instrucción, aunque el límite varía en función de la memoria disponible y de la complejidad del resto de las expresiones de la consulta. También se puede especificar una variable **table** como un origen de tabla.

Ejemplo:

```
SELECT *
FROM empleados, oficinas;
```

Si ejecutamos esta consulta veremos que las filas del resultado están formadas por las columnas de empleados y las columnas de oficinas. En las filas aparece cada empleado combinado con la primera oficina, luego los mismos empleados combinados con la segunda oficina y así hasta combinar todos los empleados con todas las oficinas.

Si ejecutamos:

```
SELECT *
FROM empleados CROSS JOIN oficinas;
```

Obtenemos lo mismo.

Este tipo de operación no es la que se utiliza más a menudo, lo más frecuente sería combinar cada empleado con los datos de SU oficina. Lo podríamos obtener añadiendo a la consulta un WHERE para filtrar los registros correctos:

```
SELECT *
FROM empleados, oficinas
WHERE empleados.oficina=oficinas.oficina;
```

Aquí nos ha aparecido la necesidad de cualificar los campos ya que el nombre *oficina* es un campo de empleados y de oficinas por lo que si no lo cualificamos, el sistema nos da error.

Hemos utilizado en la lista de selección *, esto nos recupera todas las columnas de las dos tablas.

```
SELECT empleados.*,ciudad, region
FROM empleados, oficinas
WHERE empleados.oficina=oficinas.oficina;
```

Recupera todas las columnas de empleados y las columnas ciudad y región de sus oficinas.

También podemos combinar una tabla consigo misma (consulta reflexiva), pero en este caso hay que definir un alias de tabla, en al menos una, sino el sistema da error ya que no puede nombrar los campos.

```
SELECT *
FROM oficinas, oficinas as ofi2;
```

Volveremos sobre este tipo de consultas al final del tema.

No insistiremos más sobre el producto cartesiano porque no es la operación más utilizada, ya que normalmente cuando queramos componer dos tablas lo haremos con una condición de selección basada en campos de combinación y para este caso es más eficiente el JOIN que veremos a continuación.

5.2. La composición interna INNER JOIN

Una composición interna es aquella en la que los valores de las columnas que se están combinando se comparan mediante un operador de comparación.

Es otra forma, mejor (más eficiente), de expresar un producto cartesiano con una condición.

Es la operación que más emplearemos ya que lo más frecuente es querer juntar los registros de una tabla relacionada con los registros correspondientes en la tabla de referencia (añadir a cada factura los datos de su cliente, añadir a cada línea de pedido los datos de su producto, etc..,).

```
FROM
<tabla_origen> INNER JOIN <tabla_origen> ON <condicion_combi>
```

tabla_origen tiene el mismo significado que en el producto cartesiano.

condicion_combi es cualquier condición que permite seleccionar las parejas de filas que aparecen en el resultado. Normalmente será una condición de igualdad.

```
SELECT *
FROM empleados INNER JOIN oficinas
       ON empleados.oficina=oficinas.oficina;
```

Obtiene los empleados combinados con los datos de su oficina.

```
SELECT *
FROM pedidos INNER JOIN productos
 ON producto = idproducto AND fab = idfab;
```

Obtiene los pedidos combinados con los productos correspondientes.

Normalmente la condición de combinación será una igualdad pero se puede utilizar cualquier operador de comparación estándar (<>, >…).

Llegados a este punto podemos hacer un paréntesis para indicar que algunos sistemas como por ejemplo Oracle y MySQL tienen implementado el NATURAL JOIN (producto natural del álgebra relacional) que permite realizar un INNER JOIN sin indicar la condición de combinación porque el sistema por defecto interpreta como condición de combinación la igualdad de todas las columnas que se llaman igual en las dos tablas. Pero esto que puede parece una comodidad a veces puede provocar problemas si en las tablas hay campos con el mismo nombre pero que no se utilizan para relacionar las tablas, por ejemplo:

```
SELECT *
FROM empleados AS e NATURAL JOIN oficinas AS o;
```

Equivaldría a:

```
SELECT *
FROM empleados AS e INNER JOIN oficinas AS o
       ON e.oficina=o.oficina AND e.ventas=o.ventas;
```

La comparación del campo ventas provoca un efecto no deseado, estaremos recuperando los empleados con los datos de su oficina pero únicamente los que tengan ventas iguales a las ventas de su oficina.

Como NATURAL JOIN no existe en Transact-SQL no volveremos a nombrar esta instrucción.

Es fácil ver la utilidad de la instrucción INNER JOIN y de hecho se utilizará muy a menudo, pero hay algún caso que no resuelve. En las consultas anteriores, no aparecen las filas que no tienen fila correspondiente en la otra tabla.

```
SELECT numemp,nombre,empleados.oficina, ciudad
FROM empleados INNER JOIN oficinas
       ON empleados.oficina=oficinas.oficina;
```

Numemp	nombre	oficina	ciudad
101	Antonio Viguer	12	Alicante
102	Alvaro Jaumes	21	Badajoz
103	Juan Rovira	12	Alicante
104	José González	12	Alicante
105	Vicente Pantalla	13	Castellon
106	Luis Antonio	11	Valencia
107	Jorge Gutiérrez	22	A Coruña
108	Ana Bustamante	21	Badajoz
109	María Sunta	11	Valencia

No aparecen los empleados que no tienen oficina, ni las oficinas que no tienen empleados, porque para que salga la fila, debe de existir una fila de la otra tabla que cumpla la condición.

Para resolver este problema debemos utilizar otro tipo de composición, la composición externa que veremos más adelante.

5.3. Consultas reflexivas

Como ya hemos comentado, se puede combinar una tabla consigo misma, a este tipo de consulta se le llama consulta reflexiva. Podemos tener una consulta reflexiva con cualquier tipo de combinación, con el producto cartesiano, con el INNER JOIN o con el OUTER JOIN. Funciona igual que una consulta normal pero en este caso nos tenemos que imaginar que tenemos dos copias de la tabla y cada copia es una tabla.

Es evidente que como trabajamos con dos tablas que se llaman igual y cuyos campos también se llaman igual hace falta definir un alias de tabla en al menos una de las tablas para poder diferenciar los campos. Si no definimos un alias de tabla en al menos una tabla, el sistema nos devuelve un error. También podemos definir un alias en las dos tablas si lo preferimos.

Veamos un ejemplo:

Queremos saber el codigo, nombre de cada empleado así como el nombre de su jefe. En la tabla Empleados tenemos un registro por cada empleado y en el campo jefe tenemos el código del jefe del empleado, pero no tenemos el nombre del jefe, habría que buscar en esa misma tabla el empleado con ese código para saber cómo se llama, pues tenemos que combinar empleados con empleados, una copia de la tabla actuará como tabla de empleados y la otra copia actuará como tabla de jefes:

```
SELECT *
FROM Empleados INNER JOIN Empleados AS Jefes ON Empleados.jefe=Jefes.numemp;
```

En este tipo de consultas la condición de combinación (ON) puede resultar liosa. Un truco que recomiendo es imaginarse las dos tablas, a la izquierda los empleados, a la derecha los jefes y cogemos un empleado cualquiera (de la tabla de la izquierda), para saber cómo se llama su jefe ¿qué hacemos? Mirar el número que hay en la columna jefe de ese empleado y buscar ese número en la columna numemp de la tabla de jefes, pues ya tenemos la condición de combinación, la igualdad entre los dos campos que hemos consultado (jefe de Empleados escrito en T-SQL Empleados.jefe) y (numemp de Jefes escrito en T-SQL Jefes.numemp).

Este truco sirve para obtener la condición de combinación de cualquier tipo de consulta tanto las reflexivas como las normales entre tablas distintas.

5.4. La Composición externa OUTER JOIN (LEFT/ RIGHT/ FULL)

La composición externa se escribe de manera similar al INNER JOIN indicando una condición de combinación pero el resultado incluirá filas que no cumplen la condición de combinación.

Sintaxis

```
 FROM
<tabla_origen> {LEFT|RIGHT|FULL} [OUTER] JOIN <tabla_origen>
 ON <condicion_combi>
```

La palabra OUTER es opcional y no añade ninguna función.

Las palabras LEFT, RIGHT y FULL indican la tabla de la cual se van a añadir las filas sin correspondencia. Por ejemplo:

```
SELECT numemp,nombre,empleados.oficina, ciudad
FROM empleados LEFT JOIN oficinas
       ON empleados.oficina=oficinas.oficina;
```

Recupera:

Numemp	nombre	oficina	ciudad
101	Antonio Viguer	12	Alicante
102	Alvaro Jaumes	21	Badajoz
103	Juan Rovira	12	Alicante
104	José González	12	Alicante
105	Vicente Pantalla	13	Castellon
106	Luis Antonio	11	Valencia
107	Jorge Gutiérrez	22	A Coruña
108	Ana Bustamante	21	Badajoz
109	María Sunta	11	Valencia
110	Juan Victor	NULL	NULL

Ahora sí aparece el empleado 110 que no tiene oficina

Obtiene los empleados con su oficina y los empleados (tabla a la **izquierda** *LEFT* del JOIN) que no tienen oficina. Estos empleados aparecen también en el resultado pero con los campos de la tabla *oficinas* rellenados a NULL ya que no tienen oficina.

```
SELECT numemp,nombre,empleados.oficina, ciudad, oficinas.oficina
FROM empleados RIGHT JOIN oficinas
       ON empleados.oficina=oficinas.oficina;
```

Numemp	nombre	oficina	ciudad	oficina
106	Luis Antonio	11	Valencia	11
109	María Sunta	11	Valencia	11
101	Antonio Viguer	12	Alicante	12
103	Juan Rovira	12	Alicante	12
104	José González	12	Alicante	12
105	Vicente Pantalla	13	Castellon	13
102	Alvaro Jaumes	21	Badajoz	21
108	Ana Bustamante	21	Badajoz	21
107	Jorge Gutiérrez	22	A Coruña	22
NULL	NULL	NULL	Madrid	23
NULL	NULL	NULL	Aranjuez	24
NULL	NULL	NULL	Pamplona	26
NULL	NULL	NULL	Valencia	28

Las oficinas 23,24,26 y 28 no tienen empleados.

Obtiene los empleados con su oficina y las oficinas (tabla a la **derecha** (RIGHT) del JOIN) que no tienen empleados. Estas oficinas aparecen también en el resultado con los campos de la tabla *empleados* rellenados a NULL.

```
SELECT numemp,nombre,empleados.oficina, ciudad, oficinas.oficina
FROM empleados FULL JOIN oficinas
       ON empleados.oficina=oficinas.oficina;
```

Numemp	nombre	oficina	ciudad	oficina
101	Antonio Viguer	12	Alicante	12
102	Alvaro Jaumes	21	Badajoz	21
103	Juan Rovira	12	Alicante	12
104	José González	12	Alicante	12
105	Vicente Pantalla	13	Castellon	13
106	Luis Antonio	11	Valencia	11
107	Jorge Gutiérrez	22	A Coruña	22
108	Ana Bustamante	21	Badajoz	21
109	María Sunta	11	Valencia	11
110	Juan Victor	NULL	NULL	NULL
NULL	NULL	NULL	Madrid	23
NULL	NULL	NULL	Aranjuez	24
NULL	NULL	NULL	Pamplona	26
NULL	NULL	NULL	Valencia	28

Aparecen tanto los empleados sin oficina como las oficinas sin empleados.

```
SELECT numemp,nombre,empleados.oficina, ciudad, oficinas.oficina
FROM empleados FULL OUTER JOIN oficinas
       ON empleados.oficina=oficinas.oficina;
```

Es equivalente, la palabra OUTER como hemos dicho no añade ninguna funcionalidad y se utiliza si se quiere por cuestiones de estilo.

NOTA: Cuando necesitamos obtener filas con datos de dos tablas con una condición de combinación utilizaremos un JOIN, os aconsejo empezar por escribir el JOIN con la condición que sea necesaria para combinar las filas, y luego plantearos si la composición debe de ser interna o externa. Para este segundo paso ésta sería la norma a seguir:

Empezamos por poner INNER JOIN.
Si pueden haber filas de la primera tabla que no estén relacionadas con filas de la segunda tabla **y** nos interesa que salgan en el resultado, entonces cambiamos a LEFT JOIN.
Si pueden haber filas de la segunda tabla que no estén relacionadas con filas de la primera tabla **y** nos interesa que salgan en el resultado, entonces cambiamos a RIGHT JOIN.
Si necesitamos LEFT y RIGHT entonces utilizamos FULL JOIN.

Siguiendo el ejemplo anterior nos preguntaríamos:

¿Pueden haber empleados que no tengan oficina y nos interesan?, si es que sí, necesitamos un LEFT JOIN.
Seguiríamos preguntando:
¿Pueden haber oficinas que no tengan empleados y nos interesan?, si es que sí, necesitamos un RIGHT JOIN.
Si al final necesitamos LEFT y también RIGHT entonces utilizamos FULL JOIN.

6. Tablas derivadas

En la FROM también podemos incluir tablas derivadas *DERIVED TABLES*, una tabla derivada es una instrucción SELECT completa que está escrita entre paréntesis y que actúa como una tabla o vista dentro de otra SELECT. Cuando se utiliza una **tabla derivada** es **obligatorio** definir para esa tabla un **alias** de tabla.
Ejemplo:

```
SELECT numemp,nombre,empleados.oficina, ciudad
FROM empleados INNER JOIN oficinas
       ON empleados.oficina=oficinas.oficina
WHERE region='Este';
```

Esta SELECT recupera el código, nombre, nº de oficina y ciudad de la oficina de los empleados que trabajan en oficinas del Este.
Esta SELECT Se podría implementar con una tabla derivada de la siguiente forma:

```
SELECT numemp,nombre,empleados.oficina, ciudad
FROM empleados INNER JOIN (SELECT * FROM oficinas WHERE region='Este') AS Ofi
       ON empleados.oficina=Ofi.oficina;
```

Las dos instrucciones obtienen el mismo resultado, pero de diferente forma:
La primera junta cada empleado con su oficina y luego selecciona las filas cuya región sea Este, mientras que la segunda obtiene primero una tabla derivada con las oficinas del Este y junta esta tabla con los empleados, como se utiliza un INNER JOIN sólo aparecen en el resultado los empleados relacionados con la tabla derivada, es decir los empleados que tienen una oficina del Este.

Aunque parezca mejor utilizar la segunda forma (el JOIN se efectúa con menos filas), hasta que no domines los diferentes modos de obtener datos de varias tablas, nos olvidaremos de las tablas derivadas.

7. Combinar varias operaciones

Podemos combinar dos tablas, pero también ponemos combinar tres o más tablas, en estos casos solemos hablar de Join múltiple.

En las operaciones anteriores *tabla_origen* puede ser a su vez una composición de tablas, en este caso aunque sólo sea obligatorio cuando queramos cambiar el orden de ejecución de las composiciones, es recomendable utilizar paréntesis para delimitar las composiciones.

Por ejemplo:

```
SELECT numemp,nombre,empleados.oficina, ciudad, oficinas.oficina, pedidos.*
FROM (oficinas RIGHT JOIN empleados
       ON empleados.oficina=oficinas.oficina)
        INNER JOIN pedidos on rep=numemp;
```

O bien:

```
SELECT numemp,nombre,empleados.oficina, ciudad, oficinas.oficina, pedidos.*
FROM oficinas RIGHT JOIN (empleados INNER JOIN pedidos on rep=numemp)
       ON empleados.oficina=oficinas.oficina;
```

Recomendación para cuando tenemos que redactar una FROM con más de dos tablas:

- Utilizar paréntesis
- Dentro del paréntesis tenemos que tener un JOIN completo con sus dos "tablas" y ON correspondiente.
- El paréntesis se tiene que ver como una "tabla"
- Se calculan primero los paréntesis más internos.

En el ejemplo anterior:

```
SELECT numemp,nombre,empleados.oficina, ciudad, oficinas.oficina, pedidos.*
FROM oficinas RIGHT JOIN (empleados INNER JOIN pedidos on rep=numemp)
       ON empleados.oficina=oficinas.oficina;
```

En el paréntesis:

```
(empleados INNER JOIN pedidos on rep=numemp)
```

Tenemos el JOIN completo para juntar los empleados con sus pedidos, y la tabla resultante de este JOIN se combina con la tabla oficinas para añadir a esas filas los datos de la oficina del empleado. Si este paréntesis lo ves como un nombre de tabla tienes la sintaxis del JOIN:

```
FROM oficinas RIGHT JOIN tabla
       ON empleados.oficina=oficinas.oficina;
```