CONSULTAS MULTITABLAS

SQL SERVER 2005

Manual de Referencia para usuarios

Salomón Ccance
CCANCE WEBSITE

CONSULTAS MULTITABLAS

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 e INTERSECT) que corresponden a operaciones relacionales.

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

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 vista.

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 nombres de columna 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 también hace que la consulta se ejecute más rápidamente ya que el sistema no tiene que eliminar las 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;
```

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
1
2
4
5
6

T2

Codigo
2
3
4
5

```
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;
```

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;
```

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;
```

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.

EL PRODUCTO CARTESIANO – CROSS JOIN

El producto cartesiano 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 (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 oficinas.

También podemos combinar una tabla consigo misma, 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;
```

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.

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, 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 (<>, >…).

Es fácil ver la utilidad de esta instrucción 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	Castellón
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.

LA COMPOSICIÓN EXTERNA LEFT, RIGHT Y FULL OUTER JOIN

La composición externa se escribe de manera similar al INNER JOIN indicando una condición de combinación pero en el resultado se añaden 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.

```
SELECT numemp,nombre,empleados.oficina,  ciudad
FROM empleados LEFT 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	Castellón
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 aparecerán también en el resultado con los campos de la tabla oficinas rellenados a NULL.

```
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	Castellón	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 aparecerán 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	Castellón	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 con 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.

COMBINAR VARIAS OPERACIONES

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);
```