Tema 3. Las consultas multitabla (I)

Introducción

En este tema vamos a estudiar las **consultas multitabla** llamadas así porque están **basadas** en **más de una tabla**.

El SQL de Microsoft Jet 4.x soporta dos grupos de consultas multitabla:

- la **unión** de tablas

- la **composición** de tablas

La unión de tablas

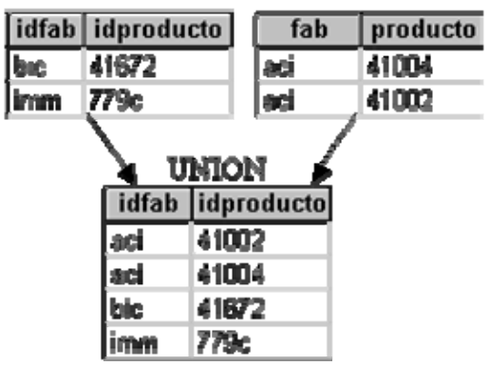
Esta operación se utiliza cuando tenemos **dos tablas** con las **mismas columnas** y queremos obtener una **nueva tabla** con las **filas de la primera y las filas de la segunda**. En este caso la tabla resultante tiene las mismas columnas que la primera tabla (que son las mismas que las de la segunda tabla).

Por ejemplo tenemos una tabla de libros nuevos y una tabla de libros antiguos y queremos una lista con todos los libros que tenemos. En este caso las dos tablas tienen las mismas columnas, lo único que varía son las filas, además queremos obtener una lista de libros (las columnas de una de las tablas) con las filas que están tanto en libros nuevos como las que están en libros antiguos, en este caso utilizaremos este tipo de operación.

Cuando hablamos de tablas pueden ser **tablas reales** almacenadas en la base de datos **o tablas lógicas** (resultados de una consulta), esto nos permite utilizar la operación con más frecuencia ya que pocas veces tenemos en una base de datos tablas idénticas en cuanto a columnas. El **resultado** es siempre una **tabla lógica**.

Por ejemplo queremos en un sólo listado los productos cuyas existencias sean iguales a cero y también los productos que aparecen en pedidos del año 90. En este caso tenemos unos productos en la tabla de productos y los otros en la tabla de pedidos, las tablas no tienen las mismas columnas no se puede hacer una union de ellas pero lo que interesa realmente es el identificador del producto (idfab,idproducto), luego por una parte sacamos los códigos de los productos con existencias cero (con una consulta), por otra parte los códigos de los productos que aparecen en pedidos del año 90 (con otra consulta), y luego unimos estas dos tablas lógicas.

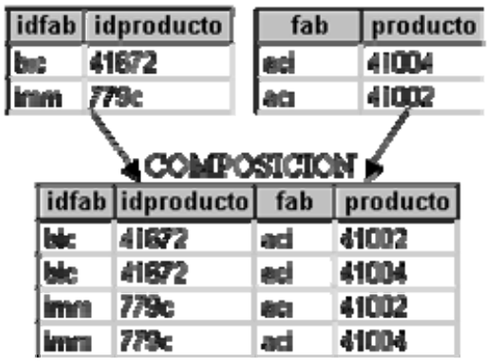
El operador que permite realizar esta operación es el operador **UNION**.



La composición de tablas

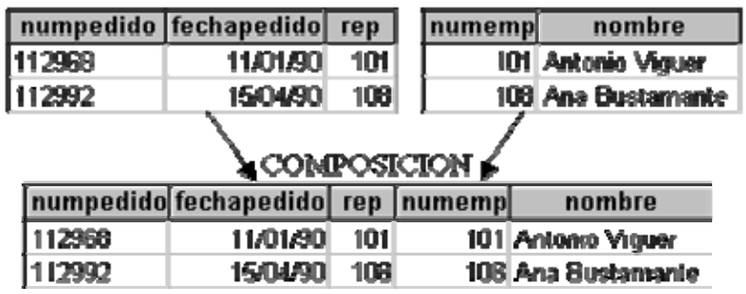
La composición de tablas consiste en **concatenar** filas de una tabla con filas de otra. En este caso obtenemos una tabla con las **columnas** de la **primera tabla unidas** a las **columnas** de la **segunda tabla**, y las filas de la tabla resultante son **concatenaciones** de **filas** de la **primera tabla con** filas de la **segunda tabla**.

El ejemplo anterior quedaría de la siguiente forma con la composición:



A diferencia de la unión la composición permite obtener una fila con datos de las dos tablas, esto es muy útil cuando queremos visualizar filas cuyos datos se encuentran en dos tablas.

Por ejemplo queremos listar los pedidos con el nombre del representante que ha hecho el pedido, pues los datos del pedido los tenemos en la tabla de pedidos pero el nombre del representante está en la tabla de empleados y además queremos que aparezcan en la misma línea; en este caso necesitamos componer las dos tablas (Nota: en el ejemplo expuesto a continuación, hemos seleccionado las filas que nos interesan).



Existen distintos tipos de composición, aprenderemos a utilizarlos todos y a elegir el tipo más apropiado a cada caso.

Los **tipos de composición** de tablas son:

. El **producto cartesiano**

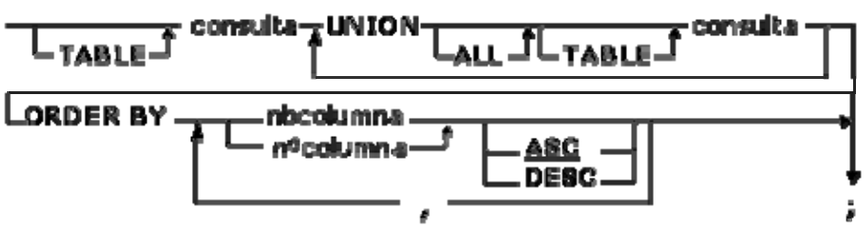
. El **INNER JOIN**

. El **LEFT / RIGHT JOIN**

El operador UNION

Como ya hemos visto en la página anterior, el operador **UNION** sirve para obtener a partir de dos **tablas** con las **mismas columnas**, una nueva tabla con las **filas** de la **primera y** las **filas** de la **segunda**.

La sintaxis es la siguiente:



Consulta puede ser un **nombre de tabla**, un **nombre de consulta** (en estos dos casos el nombre debe estar precedido de la palabra **TABLE**), o una **sentencia SELECT** completa (en este caso no se puede poner **TABLE**). La sentencia **SELECT** puede ser cualquier sentencia **SELECT** con

la única restricción de que no puede contener la cláusula **ORDER BY**.

Después de la primera consulta viene la palabra **UNION** y a continuación la segunda consulta. La segunda consulta sigue las mismas reglas que la primera consulta.

- Las dos consultas deben tener el **mismo número** de **columnas** pero las columnas pueden **llamarse** de **diferente** forma y ser de **tipos** de datos **distintos**.

- Las **columnas** del **resultado** se **llaman** como las de la **primera consulta**.

- Por **defecto** la unión **no incluye filas repetidas**, si alguna fila está en las dos tablas, sólo aparece una vez en el resultado.

Si **queremos** que aparezcan todas las filas incluso las **repeticiones** de **filas**, incluimos la palabra **ALL** (*todo* en inglés).

El empleo de **ALL** tienen una **ventaja**, la consulta **se ejecutará más rápidamente**. Puede que la diferencia no se note con tablas pequeñas, pero si tenemos tablas con muchos registros (filas) la diferencia puede ser notable.

- Se puede **unir más** de **dos** tablas, para ello después de la segunda consulta **repetimos** la palabra **UNION** ... y así sucesivamente.

- También podemos indicar que queremos el **resultado ordenado** por algún criterio, en este caso se incluye la cláusula **ORDER BY** que ya vimos en el tema anterior. La cláusula **ORDER BY** se escribe **después** de la **última consulta**, al final de la sentencia; para indicar las **columnas de ordenación** podemos utilizar su **número de orden** o el **nombre de la columna**, en este último caso se deben de utilizar los nombres de columna de la **primera consulta** ya que son los que se van a utilizar para nombrar las columnas del resultado.

Para ilustrar la operación vamos a realizar el ejercicio visto en la página anterior, vamos a obtener los códigos de los productos que tienen existencias iguales a cero o que aparezcan en pedidos del año 90.

```
SELECT idfab,idproducto
FROM productos
WHERE existencias = 0
UNION ALL
SELECT fab,producto
FROM pedidos
WHERE year(fechapedido) = 1990
ORDER BY idproducto
```

o bien

```
TABLE [existencias cero]
UNION ALL
TABLE [pedidos 90]
ORDER BY idproducto
```

Se ha incluido la cláusula **ALL** porque no nos importa que salgan filas repetidas.

Se ha incluido **ORDER BY** para que el resultado salga ordenado por idproducto, observar que hemos utilizado el nombre de la columna de la primera **SELECT**, también podíamos haber puesto **ORDER BY 2** pero no **ORDER BY producto** (es el nombre de la columna de la segunda tabla).

Para el 2º caso hemos creado una consulta llamada *existencias cero* con la primera **SELECT**, y una consulta llamada *pedidos 90* con la segunda **SELECT**. Observar que los nombres de las consultas están entre corchetes porque contienen espacios en blanco, y que en este caso hay que utilizar **TABLE**.

El producto cartesiano

El producto cartesiano es un tipo de composición de tablas, aplicando el producto cartesiano a dos tablas se obtiene una tabla con las **columnas** de la **primera tabla unidas** a las **columnas** de la **segunda tabla**, y las filas de la tabla resultante son **todas las posibles concatenaciones** de **filas** de la **primera tabla** con filas de la **segunda tabla**.
La sintaxis es la siguiente:

- El **producto cartesiano** se indica **poniendo** en la **FROM** las **tablas** que queremos componer

separadas por comas, podemos obtener así el producto cartesiano de dos, tres, o más tablas.

nbtabla puede ser un **nombre de tabla** o un **nombre de consulta**. Si todas las tablas están en una base de datos externa, añadiremos la cláusula **IN basedatosexterna después** de la **última tabla**. Pero para mejorar el rendimiento y facilitar el uso, se recomienda utilizar una tabla vinculada en lugar de la cláusula IN.

Hay que tener en cuenta que el producto cartesiano **obtiene** todas las **posibles combinaciones** de filas por lo tanto si tenemos dos tablas de 100 registros cada una, el resultado tendrá 100x100 filas, si el producto lo hacemos de estas dos tablas con una tercera de 20 filas, el resultado tendrá 200.000 filas (100x100x20) y estamos hablando de tablas pequeñas. Se ve claramente que el producto cartesiano es una **operación costosa** sobre todo si operamos con más de dos tablas o con tablas voluminosas.

Se puede **componer** una **tabla consigo misma**, en este caso es **obligatorio** utilizar un nombre de **alias** por lo menos para una de las dos.
Por ejemplo: **SELECT * FROM empleados, empleados emp**

En este ejemplo obtenemos el producto cartesiano de la tabla de empleados con ella misma. Todas las posibles combinaciones de empleados con empleados.

Para ver cómo funciona el producto cartesiano cogemos las consultas [existencias cero] y [pedidos 90] creadas en la página anterior, y creamos una consulta que halle el producto cartesiano de las dos.

```
SELECT *
FROM [existencias cero],[pedidos 90]
```

obtenemos la siguiente tabla:

idfab	idproducto	fab	producto
bic	41672	aci	41002
bic	41672	aci	41004
imm	779c	aci	41002
imm	779c	aci	41004

Se observa que tenemos las dos filas de la primera consulta combinadas con las dos filas de la segunda.

Esta operación **no** es de las **más utilizadas**, normalmente cuando queremos componer dos tablas es para añadir a las filas de una tabla, una fila de la otra tabla, por ejemplo añadir a los pedidos los datos del cliente correspondiente, o los datos del representante, esto equivaldría a un producto cartesiano con una selección de filas:

```
SELECT *
FROM pedidos,clientes
WHERE pedidos.clie=clientes.numclie
```

Combinamos todos los pedidos con todos los clientes pero luego seleccionamos los que cumplan que el código de cliente de la tabla de pedidos sea igual al código de cliente de la tabla de clientes, por lo tanto nos quedamos con los pedidos combinados con los datos del cliente correspondiente.

Las columnas que aparecen en la cláusula WHERE de nuestra consulta anterior se denominan **columnas de emparejamiento** ya que permiten emparejar las filas de las dos tablas. Las columnas de emparejamiento no tienen por qué estar incluidas en la lista de selección.

Normalmente emparejamos tablas que están relacionadas entre sí y una de las columnas de emparejamiento es clave principal, pues en este caso, cuando **una** de las **columnas de emparejamiento** tienen un **índice** definido es más eficiente utilizar otro tipo de composición, el **INNER JOIN**.

El INNER JOIN

El **INNER JOIN** es otro tipo de composición de tablas, permite **emparejar filas** de distintas tablas de forma **más eficiente** que con el producto cartesiano **cuando** una de las **columnas de emparejamiento** está **indexada**. Ya que en vez de hacer el producto cartesiano completo y luego seleccionar la filas que cumplen la condición de emparejamiento, para cada fila de una de las tablas **busca directamente** en la otra tabla **las filas que** cumplen la condición, con lo cual **se emparejan** sólo las filas que luego aparecen en el resultado.

La sintaxis es la siguiente:

FROM – tabla1 – INNER JOIN – tabla2 – ON – tabla1.col1 – comp – tabla2.col2

Ejemplo:

```
SELECT *
FROM pedidos INNER JOIN clientes ON pedidos.clie = clientes.numclie
```

tabla1 y *tabla2* son **especificaciones de tabla** (nombre de tabla con alias o no, nombre de consulta guardada), de las tablas cuyos registros se van a combinar.

Pueden ser las dos la **misma tabla**, en este caso es **obligatorio** definir al menos un **alias** de tabla.

col1, col2 son las **columnas de emparejamiento**.

Observar que dentro de la cláusula **ON** los nombres de **columna** deben ser **nombres cualificados** (llevan delante el nombre de la tabla y un punto).

Las **columnas** de **emparejamiento** deben contener la **misma clase de datos**, las dos de tipo texto, de tipo fecha etc... los campos numéricos deben ser de tipos similares. Por ejemplo, se puede combinar campos Auto numérico y Long puesto que son tipos similares, sin embargo, no se puede combinar campos de tipo Simple y Doble. Además las columnas no pueden ser de tipo Memo ni OLE.

comp representa cualquier operador de **comparación** (=, <, >, <=, >=, o <>) y se utiliza para establecer la condición de emparejamiento.

Se pueden definir **varias condiciones** de emparejamiento **unidas** por los operadores **AND** y **OR** poniendo cada condición entre **paréntesis**. Ejemplo:

```
SELECT *
FROM pedidos INNER JOIN productos ON (pedidos.fab = productos.idfab) AND
(pedidos.producto = productos.idproducto)
```

Se pueden **combinar más** de **dos tablas**
En este caso hay que **sustituir** en la sintaxis una **tabla** por un **INNER JOIN completo**.

Por ejemplo:

```
SELECT *
FROM (pedidos INNER JOIN clientes ON pedidos.clie = clientes.numclie) INNER JOIN
empleados ON pedidos.rep = empleados.numemp
```

En vez de *tabla1* hemos escrito un INNER JOIN completo, también podemos escribir:

```
SELECT *
FROM clientes INNER JOIN (pedidos INNER JOIN empleados ON pedidos.rep =
empleados.numemp) ON pedidos.clie = clientes.numclie
```

En este caso hemos sustituido *tabla2* por un INNER JOIN completo.

El LEFT JOIN y RIGHT JOIN

El **LEFT JOIN** y **RIGHT JOIN** son otro tipo de composición de tablas, también denominada **composición externa**. Son una extensión del **INNER JOIN**.

Las composiciones vistas hasta ahora (el **producto cartesiano** y el **INNER JOIN**) son **composiciones internas** ya que todos los valores de las filas del resultado son valores que están en las tablas que se combinan.

Con una composición interna sólo se obtienen las filas que tienen al menos una fila de la otra tabla que cumpla la condición, veamos un ejemplo:

Queremos combinar los empleados con las oficinas para saber la ciudad de la oficina donde trabaja cada empleado, si utilizamos un producto cartesiano tenemos:

```
SELECT empleados.*,ciudad
FROM empleados, oficinas
```

WHERE empleados.oficina = oficinas.oficina

Observar que hemos cualificado el nombre de columna oficina ya que ese nombre aparece en las dos tablas de la FROM.

Con esta sentencia los **empleados** que **no tienen** una **oficina** asignada (un valor nulo en el campo oficina de la tabla empleados) **no aparecen en el resultado** ya que la condición **empleados.oficina = oficinas.oficina** será siempre nula para esos empleados.

Si utilizamos el **INNER JOIN**:

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

Nos pasa lo mismo, el empleado 110 tiene un valor nulo en el campo oficina y no aparecerá en el resultado.

Pues en los casos en que **queremos** que **también aparezcan** las **filas que no tienen una fila coincidente** en la otra tabla, **utilizaremos** el **LEFT** o **RIGHT JOIN**.

- La sintaxis del **LEFT JOIN** es la siguiente:

```
FROM – tabla1 – LEFT JOIN – tabla2 – ON – tabla1.col1 – comp – tabla2.col2
```

La descripción de la sintaxis es la **misma** que la del **INNER JOIN** (ver página anterior), lo único que cambia es la palabra **INNER** por **LEFT** (**izquierda** en inglés).

Esta operación consiste en **añadir al resultado** del **INNER JOIN** las **filas** de la **tabla** de la **izquierda** que **no tienen correspondencia** en la otra tabla, y **rellenar** en esas filas los **campos** de la **tabla** de la **derecha** con **valores nulos**.

Ejemplo:

```
SELECT *
FROM empleados LEFT JOIN oficinas ON empleados.oficina = oficinas.oficina
```

Con el ejemplo anterior obtenemos una lista de los empleados con los datos de su oficina, y el empleado 110 que no tiene oficina aparece con sus datos normales y los datos de su oficina a nulos.

- La sintaxis del **RIGHT JOIN** es la siguiente:

```
FROM – tabla1 – RIGHT JOIN – tabla2 – ON – tabla1.col1 – comp – tabla2.col2
```

La sintaxis es la misma que la del **INNER JOIN** (ver página anterior), lo único que cambia es la palabra **INNER** por **RIGHT** (**derecha** en inglés).

Esta operación consiste en **añadir al resultado** del **INNER JOIN** las **filas** de la **tabla** de la **derecha** que **no tienen correspondencia** en la otra tabla, y **rellenar** en esas filas los **campos** de la **tabla** de la **izquierda** con **valores nulos**.

Ejemplo:

```
SELECT *
FROM empleados RIGHT JOIN oficinas ON empleados.oficina = oficinas.oficina
```

Con el ejemplo anterior obtenemos una lista de los empleados con los datos de su oficina, y además aparece una fila por cada oficina que no está asignada a ningún empleado con los datos del empleado a nulos.

- Una operación **LEFT** JOIN o **RIGHT JOIN se puede anidar** dentro de una operación **INNER** JOIN, pero una operación **INNER** JOIN **no** se puede **anidar dentro** de **LEFT** JOIN o **RIGHT JOIN**. Los anidamientos de **JOIN** de distinta naturaleza no funcionan siempre, a veces depende del orden en que colocamos las tablas, en estos casos lo mejor es probar y si no permite el anudamiento, cambiar el orden de las tablas (y por tanto de los **JOIN**s) dentro de la cláusula **FROM**.

Por ejemplo podemos tener:

```
SELECT *
FROM clientes INNER JOIN (empleados LEFT JOIN oficinas ON empleados.oficina =
oficinas.oficina) ON clientes.repclie = empleados.numclie
```

Combinamos empleados con oficinas para obtener los datos de la oficina de cada empleado, y luego añadimos los clientes de cada representante, así obtenemos los clientes que tienen un representante asignado y los datos de la oficina del representante asignado.

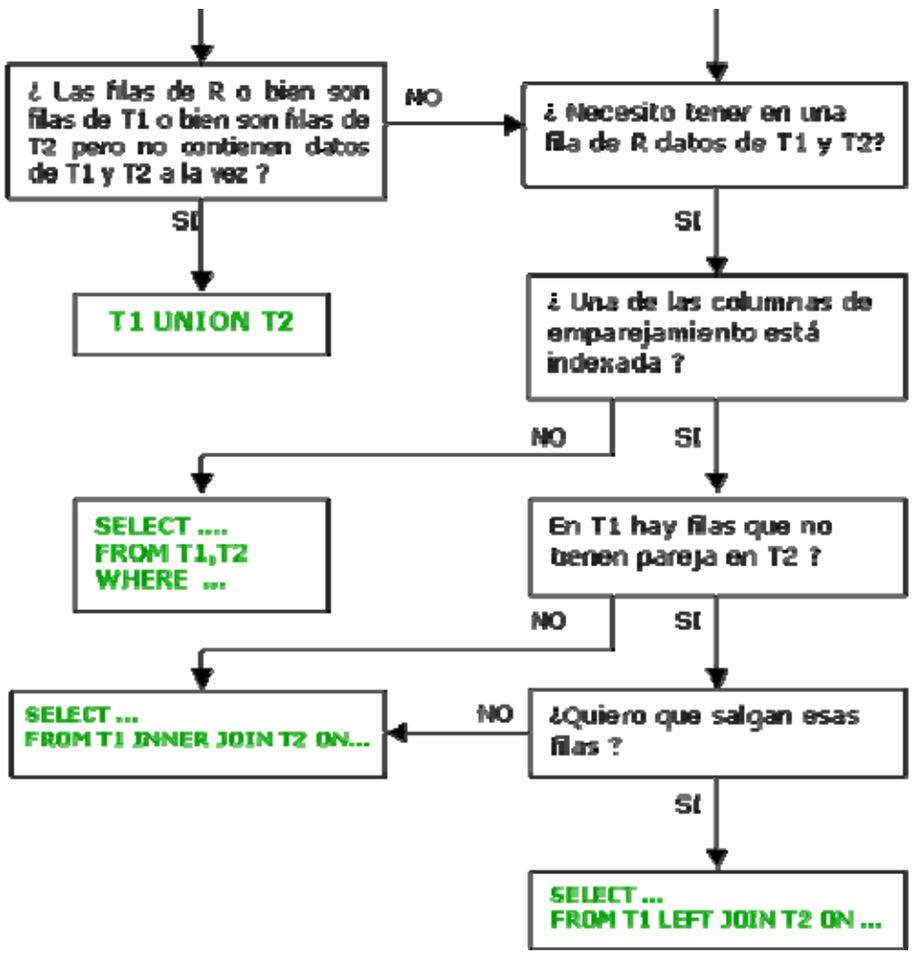
Si hubiéramos puesto **INNER** en vez de **LEFT** no saldrían los clientes que tienen el empleado 110 (porque no tiene oficina y por tanto no aparece en el resultado del **LEFT JOIN** y por tanto no entrará en el cálculo del **INNER JOIN** con clientes).

Resumen de cuándo utilizar cada operación.

Para saber en cada caso qué tipo de operación se debe utilizar, a continuación tienes un gráfico que indica qué preguntas se tienen que hacer y según la respuesta, qué operación utilizar.

Para resumir hemos llamado T1 y T2 las tablas de las que queremos sacar los datos y R la tabla lógica que representa el resultado de consulta. T1 y T2 podrían ser tablas guardadas o consultas.

En la última parte cuando se pregunta "En T1 hay filas que no tienen pareja en T2", la pregunta se debe de interpretar como "en alguna de las tablas hay filas que no tienen pareja".



Ejercicios tema 3. Las consultas multitabla

1 Listar las oficinas del este indicando para cada una de ellas su número, ciudad, números y nombres de sus empleados. Hacer una versión en la que aparecen sólo las que tienen empleados, y hacer otra en las que aparezcan las oficinas del este que no tienen empleados.

2 Listar los pedidos mostrando su número, importe, nombre del cliente, y el límite de crédito del

cliente correspondiente (todos los pedidos tienen cliente y representante).

3 Listar los datos de cada uno de los empleados, la ciudad y región en donde trabaja.

4 Listar las oficinas con objetivo superior a 600.000 pts indicando para cada una de ellas el nombre de su director.

5 Listar los pedidos superiores a 25.000 pts, incluyendo el nombre del empleado que tomó el pedido y el nombre del cliente que lo solicitó.

6 Hallar los empleados que realizaron su primer pedido el mismo día en que fueron contratados.

7 Listar los empleados con una cuota superior a la de su jefe; para cada empleado sacar sus datos y el número, nombre y cuota de su jefe.

8 Listar los códigos de los empleados que tienen una línea de pedido superior a 10.000 ptas o que tengan una cuota inferior a 10.000 Pts.

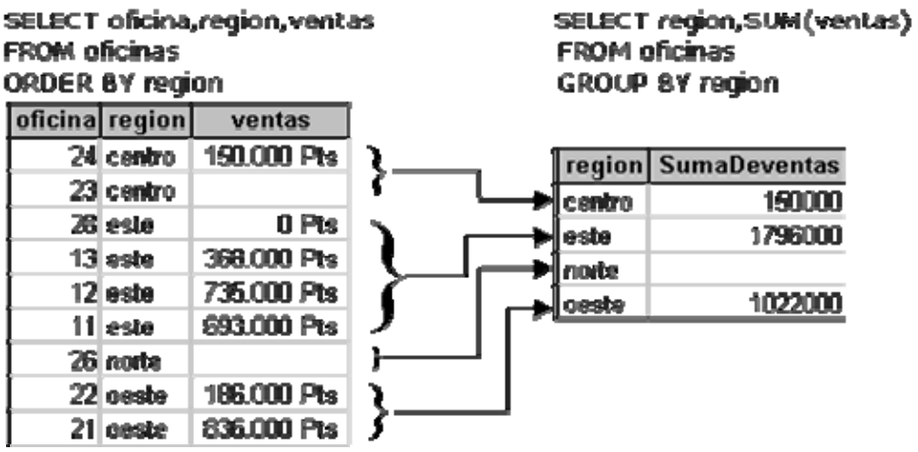
4. Las consultas de resumen (I)

Introducción

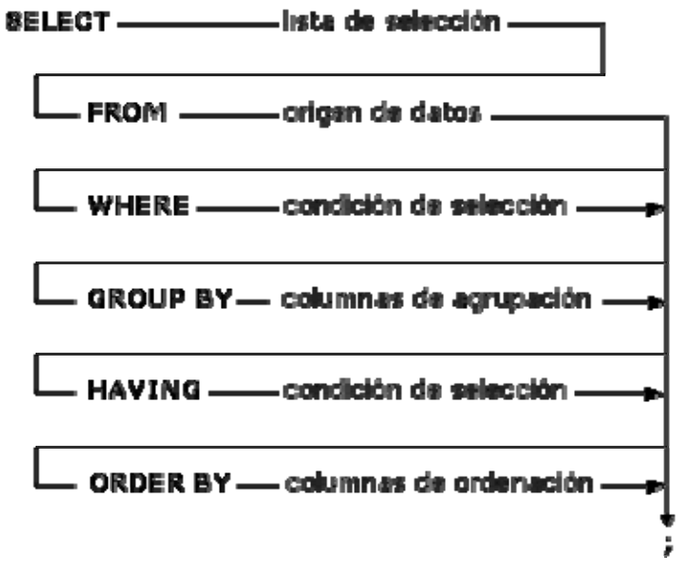
En SQL de Microsoft Jet 4.x y de la mayoría de los motores de bases de datos relacionales, podemos definir un **tipo de consultas** cuyas filas resultantes **son un resumen de las filas de la tabla origen**, por eso las denominamos **consultas de resumen**, también se conocen como consultas sumarias.

Es importante entender que las filas del resultado de una consulta de resumen tienen una **naturaleza distinta** a las filas de las demás tablas resultantes de consultas, ya que corresponden a varias filas de la tabla orgen. Para simplificar, veamos el caso de una consulta basada en una sola tabla, una fila de una consulta 'no resumen' corresponde a una fila de la tabla origen, contiene datos que se encuentran en una sola fila del origen, mientras que **una fila de una consulta de resumen corresponde** a un **resumen de varias filas** de la **tabla origen**, esta diferencia es lo que va a originar una serie de restricciones que sufren las consultas de resumen y que veremos a lo largo del tema.

En el ejemplo que viene a continuación tienes un ejemplo de consulta normal en la que se visualizan las filas de la tabla oficinas ordenadas por región, en este caso cada fila del resultado se corresponde con una sola fila de la tabla oficinas, mientras que la segunda consulta es una consulta resumen, cada fila del resultado se corresponde con una o varias filas de la tabla oficinas.



Las consultas de resumen introducen dos **nuevas cláusulas** a la sentencia SELECT, la **cláusula GROUP BY** y la **cláusula HAVING**, son cláusulas que **sólo** se pueden utilizar en una consulta de resumen, se tienen que escribir **entre** la cláusula **WHERE** y la cláusula **ORDER BY** y tienen la siguiente sintaxis:



Las detallaremos en la página siguiente del tema, primero vamos a introducir otro concepto

relacionado con las consultas de resumen, las funciones de columna.

Funciones de columna

En la lista de selección de una consulta de resumen aparecen **funciones de columna** también denominadas funciones de dominio agregadas. Una función de columna **se aplica a una columna** y obtiene un **valor que resume el contenido de la columna**.

Tenemos las siguientes funciones de columna:

SUM (expresión)	MIN (expresión)
AVG (expresión)	MAX (expresión)
STDEV (expresión)	COUNT (nbcolumna)
STDEVP (expresión)	COUNT (*)

El **argumento** de la función indica **con qué valores** se tiene que operar, por eso ***expresión*** suele ser un **nombre de columna**, columna que contiene los valores a resumir, pero también puede ser cualquier expresión válida que devuelva una lista de valores.

- La función **SUM()** calcula la **suma** de los valores indicados en el argumento. Los datos que se suman deben ser de **tipo numérico** (entero, decimal, coma flotante o monetario...). El resultado será del mismo tipo aunque puede tener una precisión mayor.

Ejemplo:

```
SELECT SUM(ventas)
FROM oficinas
```

Obtiene una sola fila con el resultado de sumar todos los valores de la columna ventas de la tabla oficinas.

- La función **AVG()** calcula el **promedio** (la media aritmética) de los valores indicados en el argumento, también se aplica a **datos numéricos**, y en este caso el tipo de dato del resultado puede cambiar según las necesidades del sistema para representar el valor del resultado.

- **StDev()** y **StDevP()** calculan la **desviación estándar** de una población o de una muestra de la población representada por los valores contenidos en la columna indicada en el argumento. Si la consulta base (el origen) tiene menos de dos registros, el resultado es nulo.

Es interesante destacar que el **valor nulo no equivale al valor 0**, las **funciones de columna no consideran** los **valores nulos** mientras que consideran el valor 0 como un valor, por lo tanto en las funciones AVG(), STDEV(), STDEVP() los resultados no serán los mismos con valores 0 que con valores nulos. Veámoslo con un ejemplo:

Si tenemos esta tabla:

Tabla1 : Tabla

col1
10
5
0
3
6
0

La consulta

```
SELECT AVG(col1) AS
media
FROM tabla1
```

devuelve:

media
4

En este caso los ceros entran en la media por lo que sale igual a 4
(10+5+0+3+6+0)/6 = 4

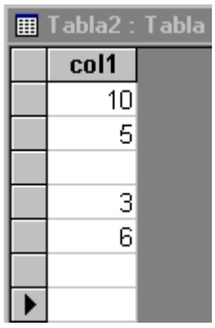
Con esta otra tabla:

```
SELECT AVG(col1) AS
media
FROM tabla2
```

devuelve:

media
6

En este caso los ceros se han sustituido por valores nulos y no entran en el cálculo por lo que la media sale igual a 6
(10+5+3+6)/4 = 4



- Las funciones **MIN()** y **MAX()** determinan los **valores menores** y **mayores** respectivamente. Los valores de la columna pueden ser de **tipo numérico**, **texto** o **fecha**. El resultado de la función tendrá el mismo tipo de dato que la columna. Si la columna es de **tipo numérico MIN()** devuelve el **valor menor** contenido en la columna, si la columna es de **tipo texto MIN()** devuelve el **primer valor** en **orden alfabético**, y si la columna es de **tipo fecha**, **MIN()** devuelve la **fecha más antigua** y **MAX()** la **fecha más reciente**.

- La función **COUNT(nb columna)** cuenta el **número de valores** que hay **en la columna**, los datos de la columna pueden ser de **cualquier tipo**, y la función siempre devuelve un número entero. Si la columna contiene **valores nulos** esos valores **no se cuentan**, si en la columna aparece un **valor repetido**, lo **cuenta varias veces**.

- **COUNT(*)** permite **contar filas** en vez de valores. Si la columna no contiene ningún valor nulo, **COUNT**(nbcolumna) y **COUNT(*)** devuelven el mismo resultado, mientras que si hay valores nulos en la columna, **COUNT(*)** cuenta también esos valores mientras que **COUNT**(nb columna) no los cuenta.

Ejemplo:
¿Cuántos empleados tenemos?

```
SELECT COUNT(numemp)
FROM empleados
```

o bien

```
SELECT COUNT(*)
FROM empleados
```

En este caso las dos sentencias devuelen el mismo resultado ya que la columna numemp no contiene valores nulos (es la clave principal de la tabla empleados).

¿Cuántos empleados tienen una oficina asignada?

```
SELECT COUNT(oficina)
FROM empleados
```

Esta sentencia por el contrario, nos devuelve el número de valores no nulos que se encuentran en la columna oficina de la tabla empleados, por lo tanto nos dice cuántos empleados tienen una oficina asignada.

- Se pueden combinar varias funciones de columna en una expresión pero no se pueden anidar funciones de columna, es decir:

```
SELECT (AVG(ventas) * 3) + SUM(cuota)
FROM ...
```
es correcto

```
SELECT AVG(SUM(ventas))
FROM ...
```
NO es correcto, no se puede incluir una función de columna dentro de una función de columna

Selección en el origen de datos.

Si queremos **eliminar** del origen de datos algunas **filas**, basta incluir la cláusula **WHERE** que ya conocemos después de la cláusula **FROM**.

Ejemplo: Queremos saber el acumulado de ventas de los empleados de la oficina 12.

```
SELECT SUM(ventas)
FROM empleados
```

WHERE oficina = 12

Origen múltiple.

Si los **datos** que necesitamos utilizar para obtener nuestro resumen **se encuentran** en **varias tablas**, formamos el origen de datos adecuado en la cláusula **FROM** como si fuera una consulta **multitabla** normal.

Ejemplo: Queremos obtener el importe total de ventas de todos los empleados y el mayor objetivo de las oficinas asignadas a los empleados:

```
SELECT SUM(empleados.ventas), MAX(objetivo)
FROM empleados LEFT JOIN oficinas ON empleados.oficina=oficinas.oficina
```

NOTA: combinamos empleados con oficinas por un **LEFT JOIN** para que aparezcan en el origen de datos todos los empleados incluso los que no tengan una oficina asignada, así el origen de datos estará formado por una tabla con tantas filas como empleados hayan en la tabla empleados, con los datos de cada empleado y de la oficina a la que está asignado. De esta tabla sacamos la suma del campo ventas (importe total de ventas de todos los empleados) y el objetivo máximo. Observar que el origen de datos no incluye las oficinas que no tienen empleados asignados, por lo que esas oficinas no entran a la hora de calcular el valor máximo del objetivo.

La cláusula GROUP BY

Hasta ahora las consultas de resumen que hemos visto utilizan todas las filas de la tabla y producen una única fila resultado.

- Se pueden obtener **subtotales** con la cláusula **GROUP BY**. Una consulta con una cláusula **GROUP BY** se denomina **consulta agrupada** ya que agrupa los datos de la tabla origen y **produce una única fila resumen por cada grupo formado**. Las columnas indicadas en el **GROUP BY** se llaman **columnas de agrupación**.

Ejemplo:

```
SELECT SUM(ventas)
FROM repventas
```

Obtiene la suma de las ventas de todos los empleados.

```
SELECT SUM(ventas)
FROM repventas
GROUP BY oficina
```

Se forma un grupo para cada oficina, con las filas de la oficina, y la suma se calcula sobre las filas de cada grupo. El ejemplo anterior obtiene una lista con la suma de las ventas de los empleados de cada oficina.

La consulta quedaría mejor incluyendo en la lista de selección la oficina para saber a qué oficina corresponde la suma de ventas:

```
SELECT oficina,SUM(ventas)
FROM repventas
GROUP BY oficina
```

- Un columna de agrupación no puede ser de tipo memo u OLE.
- La **columna de agrupación** se puede indicar mediante un **nombre de columna** o cualquier expresión válida basada en una columna pero no se pueden utilizar los alias de campo.

Ejemplo:

```
SELECT importe/cant , SUM(importe)
FROM pedidos
GROUP BY importe/cant
```

Está permitido, equivaldría a agrupar las líneas de pedido por precio unitario y sacar de cada precio unitario el importe total vendido.

```
SELECT importe/cant AS precio,
```

No está permitido, no se puede utilizar un alias

SUM(importe) campo.
FROM pedidos
GROUP BY precio

- **En la lista de selección sólo pueden aparecer** :

valores constantes
funciones de columna
columnas de agrupación (columnas que aparecen en la cláusula **GROUP BY**)
o cualquier expresión basada en las anteriores.

SELECT SUM(importe),rep*10 **FROM pedidos** **GROUP BY rep*10**	**SELECT SUM(importe),rep** **FROM pedidos** **GROUP BY rep*10**
Está permitido	No está permitido, rep es una columna simple que no está encerrada en una función de columna, ni está en la lista de columnas de agrupación.

- Se **pueden agrupar** las filas **por varias columnas**, en este caso se indican las columnas separadas por una coma y en el **orden de mayor a menor agrupación**. Se permite incluir en la lista de agrupación hasta 10 columnas.

Ejemplo: Queremos obtener la suma de las ventas de las oficinas agrupadas por región y ciudad:

```
SELECT SUM(ventas)
FROM oficinas
GROUP BY region,ciudad
```

Se agrupa primero por región, y dentro de cada región por ciudad.

- **Todas las filas** que tienen **valor nulo** en el campo de agrupación, pasan a formar **un único grupo**. Es decir, considera el valor nulo como un valor cualquiera a efectos de agrupación.

Ejemplo:

```
SELECT oficina,SUM(ventas) AS ventas_totales
FROM repventas
GROUP BY oficina
```

En el resultado aparece una fila con el campo oficina sin valor y a continuación una cantidad en el campo ventas_totales, esta cantidad corresponde a la suma de las ventas de los empleados que no tienen oficina asignada (campo *oficina* igual a nulo).

La cláusula HAVING

- La cláusula **HAVING** nos permite **seleccionar filas** de la tabla resultante **de una consulta de resumen**.

- Para la condición de selección se pueden utilizar los mismos tests de comparación descritos en la cláusula **WHERE**, también se pueden escribir condiciones compuestas (unidas por los operadores **OR**, **AND**, **NOT**), pero existe una restricción.

- **En la condición de selección sólo pueden aparecer** :

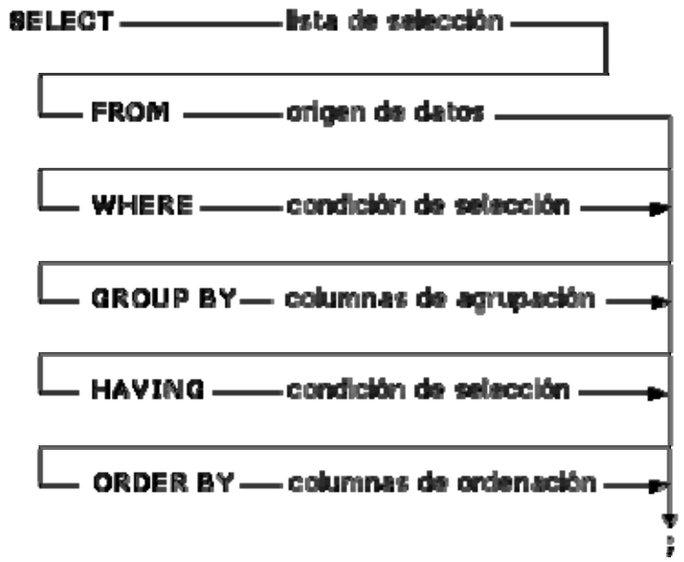
valores constantes
funciones de columna
columnas de agrupación (columnas que aparecen en la cláusula GROUP BY)
o cualquier expresión basada en las anteriores.

Ejemplo: Queremos saber las oficinas con un promedio de ventas de sus empleados mayor que 500.000 ptas.

```
SELECT oficina
FROM empleados
GROUP BY oficina
HAVING AVG(ventas) > 500000
```

NOTA: Para obtener lo que se pide hay que calcular el promedio de ventas de los empleados de cada oficina, por lo que hay que utilizar la tabla empleados. Tenemos que agrupar los empleados por oficina y calcular el promedio para cada oficina, por último nos queda seleccionar del resultado las filas que tengan un promedio superior a 500.000 ptas.

Resumen del tema



- ¿Cómo se ejecuta internamente una consulta de resumen?

- Primero se forma la tabla origen de datos según la cláusula **FROM**,
- se seleccionan del origen de datos las filas según la cláusula **WHERE**,
- se forman los grupos de filas según la cláusula **GROUP BY**,
- por cada grupo se obtiene una fila en la tabla resultante con los valores que aparecen en las cláusulas **GROUP BY, HAVING** y en la lista de selección,
- se seleccionan de la tabla resultante las filas según la cláusula **HAVING**,
- se eliminan de la tabla resultante las columnas que no aparecen en la lista de selección,
- se ordenan las filas de la tabla resultante según la cláusula **ORDER BY**

- Una consulta se convierte en consulta de resumen en cuanto aparece **GROUP BY**, **HAVING** o una función de columna.

- En una consulta de resumen, la lista de selección y la cláusula **HAVING** sólo pueden contener:
valores constantes
funciones de columna
columnas de agrupación (columnas que aparecen en la cláusula **GROUP BY**)
o cualquier expresión basada en las anteriores.

Ejercicios tema 4. Las consultas de resumen

1 ¿Cuál es la cuota media y las ventas medias de todos los empleados?

2 Hallar el importe medio de pedidos, el importe total de pedidos y el precio medio de venta (el precio de venta es el precio unitario en cada pedido).

3 Hallar el precio medio de los productos del fabricante ACI.

4 ¿Cuál es el importe total de los pedidos realizados por el empleado Vicente Pantalla?

5 Hallar en qué fecha se realizó el primer pedido (suponiendo que en la tabla de pedidos tenemos todos los pedidos realizados hasta la fecha).

6 Hallar cuántos pedidos hay de más de 25000 ptas.

7 Listar cuántos empleados están asignados a cada oficina, indicar el número de oficina y cuántos hay asignados.

8 Para cada empleado, obtener su número, nombre, e importe vendido por ese empleado a cada cliente indicando el número de cliente.

9 Para cada empleado cuyos pedidos suman más de 30.000 ptas, hallar su importe medio de pedidos. En el resultado indicar el número de empleado y su importe medio de pedidos.

10 Listar de cada producto, su descripción, precio y cantidad total pedida, incluyendo sólo los productos cuya cantidad total pedida sea superior al 75% del stock; y ordenado por cantidad total pedida.

11 Saber cuántas oficinas tienen empleados con ventas superiores a su cuota, no queremos saber cuales sino cuántas hay.