TEMA 9 LAS CONSULTAS MULTITABLA

INTRODUCCIÓN

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

El MySQL 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 menores que 50 y también los productos que aparecen en *LineasPedidos* cuya cantidad este comprendida entre 50 y 60. En este caso tenemos unos productos en la tabla de Productos y los otros en la tabla de *LineasPedido*, las tablas no tienen las mismas columnas no se puede hacer una unión de ellas pero lo que interesa realmente es el identificador del producto (*IdFabricante, IdProducto*), luego por una parte sacamos los códigos de los productos con existencias menores que 50 (con una consulta), por otra parte los códigos de los productos que aparecen en *LineasPedido* cuya cantidad vendida esté entre 50 y 60 (con otra consulta), y luego unimos estas dos tablas lógicas.

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

Ejemplo: Las consultas por separado serían:

SELECT IdFabricante, IdProducto FROM Productos WHERE Existencias <50	SELECT Fabricante, Productod FROM LineasPedido WHERE Cantidad BETWEEN 50 AND 60

La consulta con la UNION será:

```
SELECT IdFabricante, IdProducto FROM Productos
WHERE Existencias <50
UNION
SELECT Fabricante, Producto FROM LineasPedido
WHERE Cantidad BETWEEN 50 AND 60
```

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:

COMPOSICIÓN

```
SELECT IdFabricante, IdProducto, Fabricante, Producto FROM Productos, LineasPedido
WHERE Existencias <50 AND Cantidad BETWEEN 50 AND 60
```

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 o más tablas.

Por ejemplo, queremos listar los pedidos de los 10 primeros días de mayo 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).

```
SELECT DISTINCT CodPedido, FechaPedido, CodRepresentante
FROM Pedidos
WHERE MONTH(FechaPedido)=5 AND DAY(FechaPedido) <= 10;
```

```
SELECT CodEmpleado, Nombre FROM Empleados
WHERE CodEmpleado IN("106", "109", "101", "108");
```

COMPOSICION

```
SELECT CodEmpleado, FechaPedido, CodRepresentante, Nombre
FROM Empleados, Pedidos
WHERE MONTH(FechaPedido)=5 AND DAY(FechaPedido) <= 10 AND CodEmpleado = CodRepresentante;
```

Existen distintos tipos de composición. Los **tipos de composición** de tablas son:

- Producto cartesiano
- INNER JOIN
- LEFT / RIGHT JOIN

EL OPERADOR UNION

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:

```
SELECT ...
UNION [ALL | DISTINCT]
SELECT ...
[UNION [ALL | DISTINCT]
……SELECT …….]
```

Dentro de cada sentencia **SELECT** se aplican todas las cláusulas, proyecciones y selecciones que se quiera, como en cualquier **SELECT** normal.

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**. 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.

Propiedades de UNION.

Los nombres resultantes de la UNION son los nombres de las columnas de la primera sentencia SELECT. La segunda y sucesivas sentencias SELECT deben seleccionar el mismo número de columnas, pero no tienen por qué llamarse igual y ser del mismo tipo (lo normal es que lo sean). Las columnas se comparan por posiciones, no por nombres.

El tipo de dato resultante lo determinan los valores seleccionados:

- Cadenas y fechas se convierten en cadenas.
- Enteros y fechas se convierte en cadenas.
- Cadenas y enteros se convierten en cadenas.

Ejemplo vamos a obtener los códigos de los productos que tienen existencias iguales menores que 50 y que aparezcan en los pedidos del 2 de mayo

```
SELECT IdFabricante, IdProducto
FROM Productos
WHERE Existencias <100
UNION ALL
SELECT Fabricante, Producto
FROM LineasPedido, Pedidos
WHERE MONTH(FechaPedido) = 5 AND DAY(FechaPedido) = 2
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).

Se puede utilizar LIMIT en una unión de forma similar a ORDER BY. Si aplicamos a los resultados globales se la añadimos al final.

```
SELECT IdFabricante, IdProducto
FROM Productos
WHERE Existencias <150
UNION ALL
SELECT Fabricante, Producto
FROM LineasPedido, Pedidos
WHERE MONTH(FechaPedido) = 5 AND LineasPedido.CodPedido =
Pedidos.CodPedido
ORDER BY IdProducto
LIMIT 5;
```

Si queremos utilizar LIMIT para cada SELECT encerramos entre paréntesis, LIMIT se aplicará solo al fragmento.

```
(SELECT IdFabricante, IdProducto
FROM Productos
WHERE Existencias <150
```

```
LIMIT 2)
UNION ALL
(SELECT Fabricante, Producto
FROM LineasPedido, Pedidos
WHERE MONTH(FechaPedido) = 5 AND LineasPedido.CodPedido = Pedidos.CodPedido
ORDER BY Producto
LIMIT 5)
```

EL PRODUCTO CARTESIANO

El producto cartesiano de dos tablas son todas las combinaciones de todas las filas de las dos tablas. Usando una sentencia SELECT se hace proyectando todos los atributos de ambas tablas. Los nombres de las tablas se indican en la cláusula FROM separados con comas:

La sintaxis es la siguiente:

```
FROM nbtabla AS aliastabla, nbtabla AS aliastabla …....
```

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 **el nombre cualificado para cada tabla**.
- 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
WHERE Emp.CodEmpleado = Empleados.CodEmpleado
ORDER BY Empleados.Oficina, Emp.CodEmpleado ;
```

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 vamos a crear las tablas existencias50 y pedidos0502 y creamos una consulta que halle el producto cartesiano de las dos.

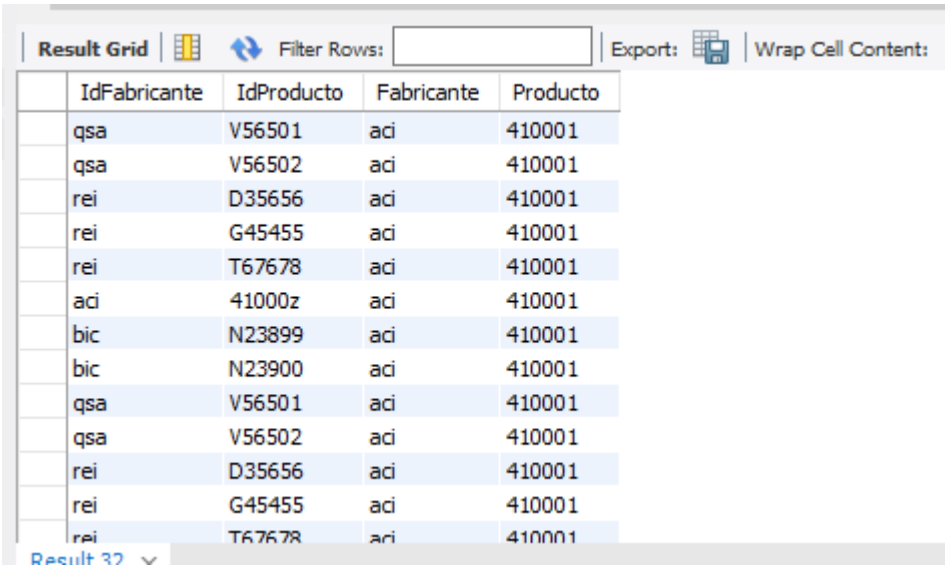
```
CREATE TABLE Existencias50 SELECT IdFabricante, IdProducto FROM Productos WHERE Existencias <100;
```

```
CREATE TABLE pedidos0502 SELECT Fabricante, Producto
FROM LineasPedido, Pedidos
WHERE MONTH(FechaPedido) = 5 AND Fabricante = 'aci' AND
LineasPedido.CodPedido = Pedidos.CodPedido;
```

Consulta del producto cartesiano será:

```
SELECT * FROM Existencias50, pedidos0502
```

obtenemos la siguiente tabla, con 96 filas (8 * 12) los registros de cada tabla



IdFabricante	IdProducto	Fabricante	Producto
qsa	V56501	aci	410001
qsa	V56502	aci	410001
rei	D35656	aci	410001
rei	G45455	aci	410001
rei	T67678	aci	410001
aci	41000z	aci	410001
bic	N23899	aci	410001
bic	N23900	aci	410001
qsa	V56501	aci	410001
qsa	V56502	aci	410001
rei	D35656	aci	410001
rei	G45455	aci	410001
rei	T67678	aci	410001

Result 32

Se observa que tenemos la fila de la primera consulta combinadas con las doce 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 las oficinas los datos de los empleados que trabajan en ellas, esto equivaldría a un producto cartesiano con una selección de filas:

```
SELECT *
FROM Oficinas, Empleados
WHERE Oficinas.CodOficina = Empleados.Oficina
ORDER BY Oficinas.CodOficina
```

Combinamos todas las oficinas con todos los empleados pero luego seleccionamos los que cumplan que el código de las oficinas de la tabla de oficinas sea igual al código de la oficina de la tabla de empleados, por lo tanto nos quedamos con las oficinas combinados con los datos del empleado 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** tiene un **índice** definido es más eficiente utilizar otro tipo de composición, el **INNER JOIN**.

COMPOSICIÓN (JOIN)

La composición 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.

Composiciones internas

Estas composiciones se denominan internas porque en la salida no aparece ninguna tupla que no esté presente en el producto cartesiano, es decir, la composición se hace en el interior del producto cartesiano de las tablas.

Las composiciones internas **INNER JOIN** usan estas sintaxis:

```
referencia_tabla, referencia_tabla
referencia_tabla [INNER | CROSS] JOIN referencia_tabla [condición]
```

La condición puede ser:

```
ON expresión_condicional | USING (lista_columnas)
```

La coma y JOIN son equivalentes, y las palabras INNER y CROSS son opcionales.

La condición en la cláusula ON puede ser cualquier expresión válida para una cláusula WHERE, de hecho, en la mayoría de los casos, son equivalentes.

La cláusula USING nos permite usar una lista de atributos que deben ser iguales en las dos tablas a componer.

referencia_tabla 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.

expresión_condicional son las **columnas de emparejamiento unidas por una comparación**. 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 AutoNumérico 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.

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

Ejemplos:

```
Obtener las oficinas con los nombre de sus empleados
```

```
SELECT Oficinas.*, Empleados.Nombre
FROM Oficinas INNER JOIN Empleados ON Oficinas.CodOficina = Empleados.Oficina
ORDER BY Oficinas.CodOficina
```

o bien (INNER es opcional)

```
SELECT Oficinas.*, Empleados.Nombre
FROM Oficinas JOIN Empleados ON Oficinas.CodOficina = Empleados.Oficina
ORDER BY Oficinas.CodOficina
```

o bien (CROSS es sinónimo de INNER)

```
SELECT Oficinas.*, Empleados.Nombre
FROM Oficinas CROSS JOIN Empleados ON Oficinas.CodOficina = Empleados.Oficina
ORDER BY Oficinas.CodOficina
```

o bien (USING(nbcol) se utiliza si las columnas de comparación tienen el mismo nombre)

```
Obtener las pedidos con los nombre de los clientes
SELECT Pedidos.*, Clientes.Nombre
FROM Pedidos CROSS JOIN Clientes USING(CodCliente)
ORDER BY Clientes.Nombre
```

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

Ejemplo:

```
SELECT CodPedido, Fabricante, Producto, Descripcion, Importe
FROM LineasPedido INNER JOIN Productos ON (LineasPedido.Fabricante = Productos.IdFabricante) AND (LineasPedido.Producto = Productos.IdProducto)
ORDER BY CodPedido
```

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

Ejemplo:

```
Obtener los pedidos numpedido, fecha, nombre del producto, nombre del cliente
SELECT CodPedido, FechaPedido, Clientes.Nombre AS CLIENTE, Empleados.Nombre AS Representante
FROM (Pedidos INNER JOIN Clientes ON Pedidos.CodCliente = Clientes.CodCliente) INNER JOIN Empleados ON Pedidos.CodRepresentante = Empleados.CodEmpleado
```

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

```
SELECT CodPedido, FechaPedido, Clientes.Nombre AS CLIENTE, Empleados.Nombre AS Representante
FROM Clientes INNER JOIN (Pedidos INNER JOIN Empleados ON Pedidos.CodRepresentante = Empleados.CodEmpleado) ON Pedidos.CodCliente = Clientes.CodCliente
```

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

Composiciones externas

Al contrario que con las composiciones internas, las externas no proceden de un producto cartesiano. Por lo tanto, en estas pueden aparecer filas que no aparecen en el producto cartesiano.

Para hacer una composición externa se toman las filas de una de las tablas una a una y se combinan con las filas de la otra.

Como norma general se usa un índice para localizar las filas de la segunda tabla que cumplen la condición, y para cada fila encontrada se añade una fila a la tabla de salida.

Si no existe ninguna fila en la segunda tabla que cumpla las condiciones, se combina la fila de la primera con una nula de la segunda.

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.

Las sintaxis para composiciones externas son:

```
referencia_tabla LEFT [OUTER] JOIN referencia_tabla [join_condition]
referencia_tabla NATURAL LEFT [OUTER] JOIN referencia_tabla
referencia_tabla RIGHT [OUTER] JOIN referencia_tabla [condición]
referencia_tabla NATURAL RIGHT [OUTER] JOIN referencia_tabla
```

La condición puede ser:

```
ON expresión_condicional | USING (lista_columnas)
```

La palabra OUTER es opcional.

Existen dos grupos de composiciones externas: izquierda y derecha, dependiendo de cuál de las tablas se lea en primer lugar.

Composición externa izquierda LEFT JOIN

En estas composiciones se recorre la tabla de la izquierda y se buscan filas en la de la derecha. Se crean usando la palabra LEFT (izquierda, en inglés).

Las sintaxis para la composición externa izquierda es:

```
referencia_tabla LEFT [OUTER] JOIN referencia_tabla [condición]
```

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.*, Oficinas.Nombre
FROM Empleados, Oficinas
WHERE Empleados.Oficina = Oficinas.CodOficina;
```

Observar que se ha 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.*, Oficinas.Nombre
FROM Empleados INNER JOIN Oficinas ON Empleados.Oficina = Oficinas.CodOficina;
```

Nos pasa lo mismo, los empleados 110 y 200 tienen 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**.

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 Empleados.*, Oficinas.Nombre
FROM Empleados LEFT JOIN Oficinas ON Empleados.Oficina = Oficinas.CodOficina;
```

Con el ejemplo anterior obtenemos una lista de los empleados con los datos de su oficina, y los empleado 110 y 200 que no tiene oficina aparece con sus datos normales y los datos de su oficina a nulos.

Composición externa derecha RIGHT JOIN

En este caso se recorre la tabla de la derecha y se buscan tuplas que cumplan la condición en la tabla izquierda.

La sintaxis es equivalente:

```
referencia_tabla RIGHT [OUTER] JOIN referencia_tabla [condición]
```

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 E.Nombre, O.CodOficina, Categoria, O.Nombre, Region
FROM Empleados AS E RIGHT JOIN Oficinas AS O ON E.Oficina = O.CodOficina;
```

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 anidamiento, cambiar el orden de las tablas (y por tanto de los **JOIN**s) dentro de la cláusula **FROM**.

Por ejemplo podemos tener:

```
SELECT O.CodOficina, E.Nombre AS Representante, C.Nombre AS Cliente
FROM Clientes AS C INNER JOIN (Empleados AS E LEFT JOIN Oficinas AS O
ON E.Oficina = O.CodOficina) ON C.CodRepCliente = E.CodEmpleado;
```

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).

Composición interna natural

Consiste en una proyección sobre un producto cartesiano restringido. Es decir, sólo elegimos determinadas columnas de ambas tablas, en lugar de seleccionar todas.

Podemos hacer esto a partir de una composición general, eligiendo todas las columnas menos las repetidas:

Sintaxis composiciones internas naturales:

```
referencia_tabla NATURAL JOIN referencia_tabla
```

Por ejemplo:

```
SELECT oficinas.CODoficina, empleados.nombre AS Empleado
FROM oficinas NATURAL JOIN empleados;
```

Composiciones naturales externas

Por supuesto, también podemos hacer composiciones externas naturales:

```
referencia_tabla NATURAL LEFT [OUTER] JOIN referencia_tabla
referencia_tabla NATURAL RIGHT [OUTER] JOIN referencia_tabla
```

NATURAL LEFT JOIN es similar a LEFT JOIN, realiza una unión LEFT JOIN haciendo coincidir todas las columnas que se llaman igual en ambas tablas, no se incluyen las cláusulas ON ni USING.

El problema es que si existen filas añadidas con respecto a la composición interna, no se eliminará ninguna columna.

```
SELECT C.Nombre AS Cliente, FechaPedido AS Fecha
FROM Clientes AS C NATURAL JOIN Pedidos AS P;
SELECT C.Nombre AS Cliente, FechaPedido AS Fecha
FROM Clientes AS C NATURAL LEFT JOIN Pedidos AS P;
```