BASES DE DATOS

CONSULTAS

COMPLEJAS

1 Introducción

Muchas consultas útiles solicitan datos procedentes de dos o más tablas de la base de datos. Por ejemplo, estas peticiones de datos para la base de datos EMPLEADOS extraen los datos de dos, tres o cuatro tablas:

- Listar cada VEND edor junto con los datos de la oficina en la que trabaja (se necesita información de las tablas VENDEDORES y OFICINAS).
- Listar los pedidos remitidos la última semana, mostrando el importe del pedido, el nombre del cliente que lo ordenó y el nombre del producto solicitado (se necesita información de las tablas PEDIDOS, CLIENTES y VENDEDORES).

SQL permite recuperar datos procedentes de dos o más tablas haciendo operaciones de composición, combinación o "***join***" (términos equivalentes) entre ellas.

Otro ejemplo:

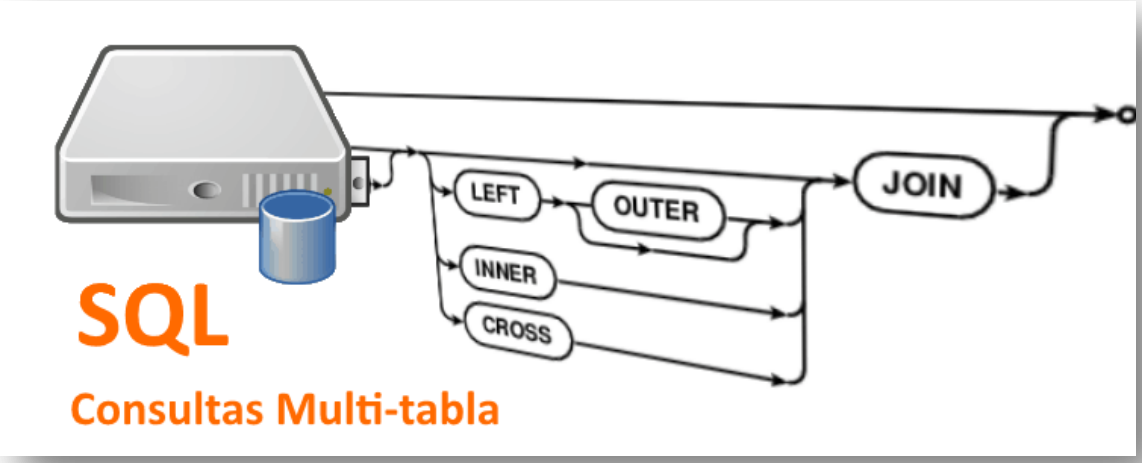
- Listar todos los pedidos, mostrando de cada uno el número de pedido y su importe, el nombre y el límite de crédito del cliente que los solicitó.

Los cuatro datos específicos solicitados están almacenados en dos tablas diferentes:

- La tabla PEDIDOS contiene el número de pedido y el importe de cada pedido, pero no tiene los nombres de cliente ni los límites de crédito.

- La tabla CLIENTES contiene los nombres de cliente y sus balances pero le falta la información referente a los pedidos.

Existe, sin embargo, un enlace entre estas dos tablas. En cada fila de la tabla PEDIDOS, la columna CLIE contiene el número del cliente que ordenó el pedido, el cual se corresponde con el valor en la columna NUM_CLIE de una de las filas de la tabla CLIENTES. Evidentemente, la sentencia SELECT que gestiona la petición debe utilizar de algún modo este enlace entre las tablas para generar sus resultados.



2 Tipos de consultas múltiples

2.1 Combinaciones simples

Se denomina combinación, composición o "**join**" de dos tablas a la operación que da como resultado otra tabla formada por parejas de filas (una fila de cada tabla) que cumplen una condición de combinación. Para cada par de filas que se puede formar entre las dos tablas iniciales, se comprueba si el contenido de una columna de una de las filas coincide con el contenido de otra columna de la otra fila. Cuando coinciden, ese par de filas pasa a formar una fila de la tabla resultante.

Las combinaciones son el fundamento del procesamiento de consultas multitabla en SQL. Todos los datos de una base de datos relacional están almacenados en sus columnas con valores explícitos, de modo que todas las relaciones posibles entre tablas pueden formarse comparando los contenidos de las columnas relacionadas.

Por ejemplo:

Listar todos los pedidos, mostrando el número de pedido y su importe, el nombre y el límite de crédito del cliente que los solicitó:

```
SELECT NUM_PEDIDO, IMPORTE, EMPRESA, LIMITE_CREDITO
FROM PEDIDOS, CLIENTES
WHERE PEDIDOS.CLIE = CLIENTES.NUM_CLIE
```

Esta consulta tiene dos novedades:

- la cláusula FROM tiene dos tablas en lugar de una sola.
- la condición de búsqueda: PEDIDOS.CLIE = CLIENTES.NUM_CLIE que compara columnas de dos tablas diferentes. Como todas las condiciones de búsqueda, ésta restringe las filas que aparecen en los resultados de la consulta. Puesto que ésta es una consulta de dos tablas, la condición de búsqueda restringe las parejas de filas que generan los resultados.

2.2 Consultas padre/hijo

Las consultas multitabla más comunes implican a dos tablas que tienen una relación natural padre/hijo. La consulta referente a pedidos y clientes de la sección precedente es un ejemplo de tal tipo de consulta. Cada pedido (hijo) tiene un cliente asociado (padre), y cada cliente (padre) puede tener muchos pedidos asociados (hijos). Los pares de filas que generan los resultados de la consulta son combinaciones de fila padre/hijo.

Las claves ajenas y las claves primarias crean relaciones padre/hijo en una base de datos relacional. La tabla que contiene la clave ajena es la tabla hijo en la relación; la tabla con la clave primaria es la tabla padre. Para utilizar la relación padre/hijo en una consulta debe especificarse una condición de búsqueda que compare la clave ajena y la clave primaria.

Por ejemplo:
Listar cada uno de los VENDEDORES y la ciudad y región en donde trabajan:

SELECT NOMBRE, CIUDAD, REGION
FROM VENDEDORES, OFICINAS
WHERE OFICINA_VEND = OFICINA

Tabla OFICINAS

OFICINA	CIUDAD	REGION	OBJETIVO	VENTAS
22	Denver	Oeste	$300,000.00	$186,042.00
11	New York	Este	$575,000.00	$692,637.00
12	Chicago	Este	$800,000.00	$735,042.00
13	Atlanta	Este	$350,000.00	$350,000.00
21	Los Angeles	Oeste	$725,000.00	$835,915.00

Tabla REPVENTAS

NUM_EMPL	NOMBRE	ED	OFIC_REP	TITULO
105	Bill Adams	37	13	Rep Ventas
109	Mary Jones	31	11	Rep Ventas
102	Sue Smith	48	21	Rep Ventas
106	Sam Clark	52	11	VP Ventas
104	Bob Smith	33	12	Dir Ventas
101	Dan Roberts	45	12	Rep Ventas
110	Tom Snyder	41	NULL	Rep Ventas
108	Larry Fitch	62	21	Dir Ventas
103	Paul Cruz	29	12	Rep Ventas
107	Nancy Angelli	49	22	Rep Ventas

Resultado de la consulta

NOMBRE	CIUDAD	REGION

Ahora vemos otra consulta que referencia las mismas dos tablas, pero con los papeles de padre e hijo intercambiados. Lista las oficinas con los nombres y títulos de sus directores:

SELECT CIUDAD, NOMBRE, TITULO
FROM OFICINAS, VENDEDORES
WHERE DIR = NUM_EMPL

2.3 Combinaciones con criterios de selección de filas

La condición de búsqueda o condición de combinación, que especifica las columnas de emparejamiento en una consulta multitabla, puede combinarse con otras condiciones de búsqueda para restringir aún más los contenidos de los resultados.

Por ejemplo:
Listar los nombres de las oficinas y los nombres y títulos de sus directores, de aquellas oficinas con un objetivo superior a $600.000:

SELECT CIUDAD, NOMBRE, TITULO
FROM OFICINAS, VENDEDORES
WHERE DIR = NUM_EMPL
AND OBJETIVO > 600,000.00

2.4 Múltiples columnas de emparejamiento

La tabla PEDIDOS y la tabla PRODUCTOS de la base de datos EMPLEADOS están relacionadas mediante un par de claves ajena/primaria. Las columnas FAB y PRODUCTO de la tabla PEDIDOS forman juntas una clave ajena para la tabla PRODUCTOS. Para combinar las tablas basándose en esta relación padre/hijo, deben especificarse ambos pares de columnas de emparejamiento.

Por ejemplo.

Listar los pedidos, mostrando los importes y las descripciones del producto.

```
SELECT NUM_PEDIDO, IMPORTE, DESCRIPCION
FROM PEDIDOS, PRODUCTOS
WHERE FAB = ID_FAB
    AND PRODUCTO = ID_PRODUCTO
```

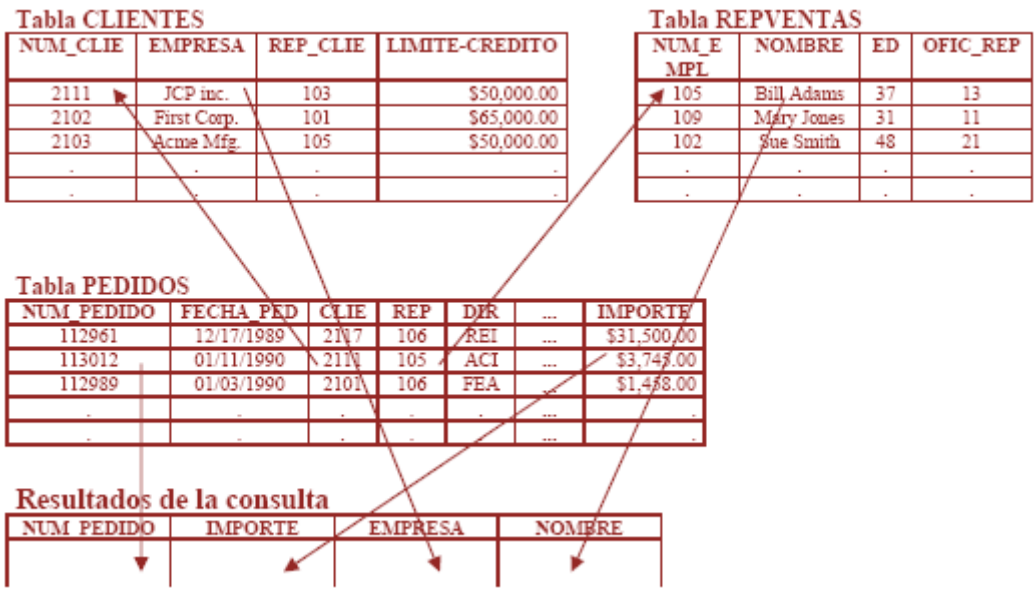
2.5 Consultas de tres o más tablas

SQL puede combinar datos de tres o más tablas utilizando las mismas técnicas básicas utilizadas para las consultas de dos tablas.

Por ejemplo:

Listar los pedidos superiores a 25.000, incluyendo el nombre del VEND edor que tomó el pedido y el nombre del cliente que lo solicitó.

```
SELECT NUM_PEDIDO, IMPORTE, EMPRESA, NOMBRE
FROM PEDIDOS, CLIENTES, VENDEDORES
WHERE CLIE = NUM_CLIE
    AND REP = NUM_EMPL
    AND IMPORTE > 25000
```



Esta consulta utiliza dos claves ajenas de la tabla PEDIDOS. La columna CLIE es una clave ajena para la tabla CLIENTES, que enlaza cada pedido con el cliente que lo remitió. La columna REP es una clave ajena para la tabla VENDEDORES, que liga cada pedido con el VEND edor que lo aceptó.

Es frecuente encontrar consultas de tres, cuatro o más tablas en aplicaciones SQL.

2.6 Otras composiciones

La mayoría de las consultas multitabla se basan en relaciones padre/hijo, pero SQL no exige que las columnas de emparejamiento estén relacionadas como clave primaria y clave ajena. Cualquier par de columnas de dos tablas pueden servir como columnas de emparejamiento, siempre que tengan tipos de datos comparables.

Por ejemplo:

Mostrar todos los pedidos recibidos en los días en que un nuevo VEND edor fue contratado:

```
SELECT NUM_PEDIDO, IMPORTE, FECHA_PEDIDO, NOMBRE
FROM PEDIDOS, VENDEDORES
WHERE FECHA_PEDIDO = CONTRATO
```

Los resultados de esta consulta provienen de los pares de filas de las tablas PEDIDOS y VENDEDORES, en donde el valor en la columna FECH_PEDIDO coincide con el valor en la columna CONTRATO para el VEND edor. *Ninguna de estas columnas es una clave ajena o una clave primaria*, y la relación entre los pares de filas es ciertamente una relación extraña, la única cosa que los pedidos y VENDEDORES correspondientes tienen en común es que resultan tener las mismas fechas.

Tabla REPVENTAS

NUM_EMPL	NOMBRE	...	CONTRATO
105	Bill Adams	...	02/12/1988
109	Mary Jones	...	10/12/1989
102	Sue Smith	...	12/10/1986
106	Sam Clark	...	06/14/1988
104	Bob Smith	...	05/19/1987
101	Dan Roberts	...	10/20/1986
110	Tom Snyder	...	01/13/1990
108	Larry Fitch	...	10/12/1989
103	Paul Cruz	...	03/01/1987
107	Nancy Angelli	...	11/14/1989

Tabla PEDIDOS

NUM_PEDIDO	FECHA_PEDIDO	CLIE
113051	02/10/1990	2118
112968	10/12/1989	2102
113036	01/30/1990	2107
113062	02/24/1990	2124
112979	10/12/1989	2114
113027	01/22/1990	2103
112992	11/04/1989	2118
112975	10/12/1989	2111
113055	02/15/1990	2108
.	.	.

Las columnas de emparejamiento como las de este ejemplo generan una **relación de muchos a muchos** entre las dos tablas.

2.7 Combinaciones basadas en la desigualdad

También la combinación de dos tablas puede realizarse con una comparación de desigualdad entre valores de una pareja de columnas de ambas tablas.

Por ejemplo:
Listar todas las combinaciones de VENDEDORES y oficinas en las que la cuota del VEND edor es superior al objetivo de la oficina:

```
SELECT NOMBRE, CUOTA, CIUDAD, OBJETIVO
FROM VENDEDORES, OFICINAS
WHERE CUOTA > OBJETIVO
```

3 Consideraciones SQL para consultas multitabla

Algunas consultas multitabla no pueden ser expresadas sin las características adicionales del lenguaje SQL descritas en las secciones siguientes. Concretamente:

- Los nombres de columna cualificados a veces son necesarios en consultas multitabla para eliminar referencias de columna ambiguas.
- Los alias de tablas pueden ser utilizadas en la cláusula FROM para simplificar nombres de columna cualificados y permitir referencias de columna no ambiguas en autocomposiciones.
- La selección de todas las columnas (SELECT *) tiene un significado especial para las consultas multitabla.
- Las autocombinaciones pueden ser utilizadas para crear una consulta multitabla que relaciona una tabla consigo misma.

3.1 Nombres de columna cualificados

La base de datos ***MiBase*** que utilizamos para los ejemplos incluye varias instancias en donde dos tablas contienen columnas con el mismo nombre.

La tabla OFICINAS y la tabla VENDEDORES, por ejemplo, tienen ambas una columna de nombre VENTAS. La columna de la tabla OFICINAS contiene las ventas anuales hasta la fecha para cada oficina; la de la tabla VENDEDORES contiene las ventas anuales hasta la fecha de cada VEND edor.

Normalmente, no hay confusión entre ambas columnas, ya que la cláusula FROM determina cuál de ellas es la adecuada en una consulta determinada, como en el siguiente ejemplo:

Mostrar las ciudades en las que las ventas superan al objetivo:

```
SELECT CIUDAD, VENTAS
FROM OFICINAS
WHERE VENTAS > OBJETIVO
```

Sin embargo, en la siguiente consulta los nombres duplicados provocan un problema:

Mostrar el nombre, las ventas y la oficina de cada VEND edor:

```
SELECT NOMBRE, VENTAS, CIUDAD
FROM VENDEDORES, OFICINAS
WHERE OFICINA_VEND = OFICINA
```

Error, pues hay un nombre de columna ambiguo "VENTAS"

Para eliminar la ambigüedad, debe utilizarse un nombre de columna cualificado para identificar la columna.

Un **nombre de columna cualificado** especifica el nombre de una columna y la tabla que contiene a la columna. Los nombres cualificados de las dos columnas VENTAS en la base de datos ejemplo son *OFICINAS.VENTAS y VENDEDORES.VENTAS*

Esta sería la versión correcta de la consulta anterior:

```
SELECT NOMBRE, VENDEDORES.VENTAS, CIUDAD
FROM VENDEDORES, OFICINAS
WHERE OFICINA_VEND = OFICINA
```

Utilizar nombres de columnas cualificados en una consulta multitabla es siempre una buena medida. La desventaja, naturalmente, es que hacen que el texto de la consulta sea mayor, pero esto deja de ser un inconveniente con el uso de alias de tablas.

3.2 Uso de alias de tablas

Si una consulta se refiere a la tabla de otro usuario, o si el nombre de una tabla es muy largo, puede ser tedioso escribir el nombre de la tabla como cualificador de una columna.

Por ejemplo, esta consulta, que referencia a la tabla CUMPLEANIOS propiedad del usuario SAM, lista los nombres, cuotas y cumpleaños de los VENDEDORES:

```
SELECT VENDEDORES.NOMBRE, CUOTA, SAM.CUMPLEANIOS.FECHA
FROM VENDEDORES, SAM.CUMPLEANIOS
WHERE VENDEDORES.NOMBRE = SAM.CUMPLEANIOS.NOMBRE
```

Resulta más fácil de leer y escribir cuando se utilizan los alias S y B para las tablas:

```
SELECT S.NOMBRE, S.CUOTA, B.FECHA
FROM VENDEDORES S, SAM.CUMPLEANIOS B
WHERE S.NOMBRE = B.NOMBRE
```

3.3 Selección de todas las columnas

"**SELECT ***" se utiliza para seleccionar todas las columnas de la tabla designada en la cláusula FROM. En una consulta multitabla, el asterisco selecciona todas las columnas de todas las tablas listadas en la cláusula FROM. La siguiente consulta, por ejemplo, produciría 15 columnas de resultados, las 9 columnas de la tabla VENDEDORES seguidas de las 6 columnas de la tabla OFICINAS.

Informar sobre todos los VENDEDORES y todas las oficinas en las que trabajan:

```
SELECT *
FROM VENDEDORES, OFICINAS
WHERE OFICINA_VEND = OFICINA
```

Obviamente, la forma SELECT * de una consulta resulta ser mucho menos práctica cuando hay dos, tres o más tablas en la cláusula FROM.

En la siguiente consulta, el ítem de selección VENDEDORES.* se aplica a una lista que contiene únicamente las columnas halladas en la tabla VENDEDORES:

```
SELECT VENDEDORES.*, CIUDAD, REGION
FROM VENDEDORES, OFICINAS
WHERE OFICINA_VEND = OFICINA
```

Ahora la consulta produciría once columnas de resultados, las nueve columnas de la Tabla VENDEDORES, seguidas de las dos columnas explícitamente solicitadas de la Tabla OFICINAS.

3.3 Autocombinaciones

Algunas consultas multitabla afectan a una relación que una tabla tiene consigo misma. Por ejemplo, supongamos que necesitamos obtener el nombre de cada VEND edor y, al lado, el nombre de su director.

Cada VEND edor aparece como una fila en la tabla VENDEDORES, y la columna DIRECTOR contiene el número de empleado del director del VEND edor. Parecería que la columna DIRECTOR debería ser una clave ajena para la tabla que contiene datos referentes a los VENDEDORES. En efecto, así es, se trata de una clave ajena para la propia tabla VENDEDORES.

Si se tratara de expresar esta consulta como cualquier otra consulta de dos tablas implicando una coincidencia clave ajena/clave primaria, aparecería tal como esta:

```
SELECT NOMBRE, NOMBRE
FROM VENDEDORES, VENDEDORES
WHERE DIRECTOR = NUM_EMPL
```

Esta sentencia SELECT es **ilegal** debido a la referencia duplicada a la tabla VENDEDORES en la cláusula FROM.

Para evitarlo se podría probar a eliminar la segunda referencia a la tabla VENDEDORES.

```
SELECT NOMBRE, NOMBRE
FROM VENDEDORES
WHERE DIRECTOR = NUM_EMPL
```

Esta consulta es legal, pero no hará lo que se desea que haga. Es una consulta monotabla, por lo que SQL recorre la tabla VENDEDORES fila a fila, aplicando la condición de búsqueda DIRECTOR = NUM_EMPL. En este caso, las filas que satisfacen esta condición son aquéllas en donde las dos columnas tienen el mismo valor, es decir, las filas en donde un VEND edor es su propio director. No existen tales filas, por lo que la consulta no produciría resultados.

Para comprender cómo resuelve SQL este problema, imaginemos que hubiera dos copias idénticas de la tabla VENDEDORES, una llamada EMPS, en la que nos vamos a fijar en los empleados, y otra llamada DIRS, en la que nos fijaremos en los directores. La columna DIRECTOR de la tabla EMPS sería entonces una clave ajena para la tabla DIRS, y la siguiente consulta funcionaría.

```
SELECT EMPS.NOMBRE, DIRS.NOMBRE
FROM EMPS, DIRS
WHERE EMPS.DIRECTOR = DIRS.NUM_EMPL
```

Bien, pues lo que haremos es que en lugar de duplicar realmente el contenido de la tabla, nos referiremos a ella utilizando nombres diferentes mediante ***alias***.

La solución correcta al ejemplo anterior es:

```
SELECT EMPS.NOMBRE, DIRS.NOMBRE
FROM VENDEDORES EMPS, VENDEDORES DIRS
WHERE EMPS.DIRECTOR = DIRS.NUM_EMPL
```

Ejemplo:
Listar los VENDEDORES con una cuota superior a la de sus directores:

```
SELECT EMPS.NOMBRE, EMPS.CUOTA, DIRS.CUOTA
FROM VENDEDORES EMPS, VENDEDORES DIRS
WHERE EMPS.DIRECTOR = DIRS.NUM_EMPL AND
EMPS.CUOTA > DIRS.CUOTA
```

4 Producto cartesiano de dos tablas

Una composición, combinación o **join** es un caso especial de una combinación más general de datos procedentes de dos tablas, conocida como el **producto cartesiano de dos tablas**.

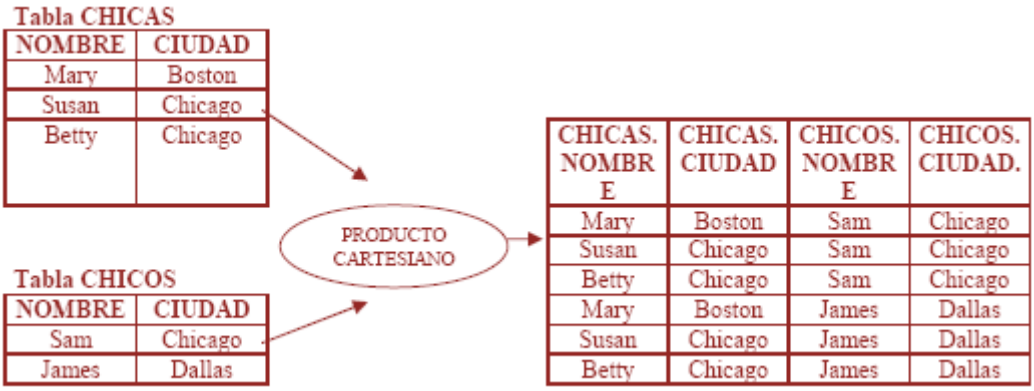
El producto de dos tablas es otra tabla (la tabla producto) que consta de todas las combinaciones posibles de pares de filas de las dos tablas (cada par de filas está formado por una fila de una tabla y otra fila de la otra tabla). Las columnas de la tabla producto son todas las columnas de la primera tabla, seguidas de todas las columnas de la segunda tabla.

Por ejemplo:

La tabla CHICAS contiene información de cinco chicas y las ciudades en donde viven. La tabla CHICOS, cinco chicos y las ciudades en donde viven. Para encontrar todas

las posibles parejas de chico/chica se podría utilizar esta consulta, que forma el producto cartesiano de las dos tablas:

SELECT CHICAS.NOMBRE, CHICAS.CIUDAD,
CHICOS.NOMBRE, CHICOS.CIUDAD
FROM CHICAS, CHICOS



Otro ejemplo:
Mostrar todas las combinaciones posibles de VENDEDORES con ciudades:

SELECT NOMBRE, CIUDAD
FROM VENDEDORES, OFICINAS

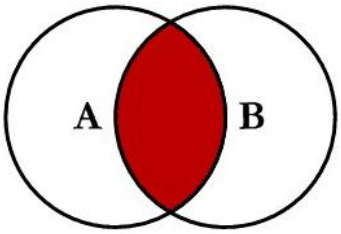
Produciría el producto cartesiano de las tablas VENDEDORES y OFICINAS, mostrando todos los pares posibles VEND edor/ciudad. Si hay 5 oficinas y tenemos 10 VENDEDORES, 5 * 10 = 50 combinaciones, luego habría 50 filas de resultados.

5 Combinación interna y combinación externa

5.1 Combinación interna

La combinación interna es la combinación estándar, la más utilizada. Si se habla de combinación sin más, se refiere a la interna.

Otra opción para utilizar una combinación interna es con la sintaxis **INNER JOIN**.




```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

Esta sintaxis es idéntica a la de una consulta SELECT habitual, con la particularidad de que en la cláusula FROM aparecen las tablas a combinar y la condición de combinación

SELECT tCoches.matricula, tMarcas.marca, tCoches.modelo, tCoches.color, tCoches.numero_kilometros, tCoches.num_plazas
FROM tCoches INNER JOIN tMarcas ON tCoches.marca = tMarcas.codigo

La sintaxis con INNER JOIN permite separar completamente las condiciones de combinación de otras condiciones. Cuando tenemos consultas que combinan nueve o diez tablas esto resulta más cómodo y comprensible.

La combinación interna es excluyente. Esto quiere decir que si una fila NO cumple la condición de combinación, entonces no se incluye en los resultados: si un coche no tiene grabada la marca, no aparece en el resultado de la consulta.

Las siguientes dos consultas son equivalentes, aunque utilicen distinta sintaxis:

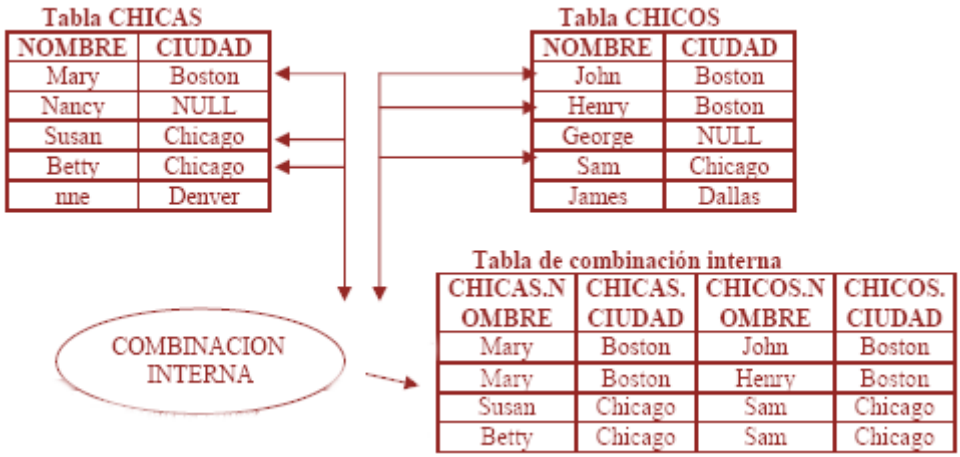
SELECT ventas., usuarios.nombre, usuarios.origen*
FROM ventas INNER JOIN usuarios ON ventas.idcliente=usuarios.id

SELECT ventas., usuarios.nombre, usuarios.origen*
FROM ventas, usuarios
WHERE ventas.idcliente=usuarios.id

Ejemplo:

Para encontrar las parejas chico/chica que viven en la misma ciudad se podría utilizar esta consulta, que forma la combinación interna de las dos tablas:

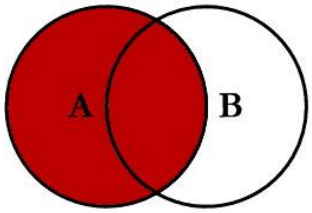
*SELECT **
FROM CHICAS, CHICOS
WHERE CHICAS.CIUDAD = CHICOS.CIUDAD



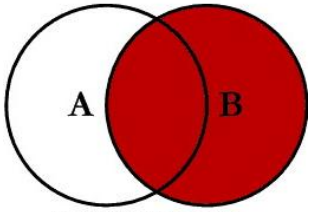
5.2 Combinación externa

La combinación externa puede ser de dos tipos: izquierda o LEFT OUTER JOIN y derecha o RIGHT OUTER JOIN.

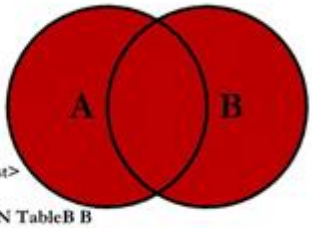
- Con **LEFT OUTER JOIN** obtenemos todas las filas (aunque no cumplan la condición de combinación) de la tabla que situemos a la izquierda en la clausula JOIN.
- Con **RIGHT OUTER JOIN** obtenemos todas las filas (aunque no cumplan la condición de combinación) de la tabla que situemos a la derecha en la clausula JOIN.
- Con **FULL OUTER JOIN** obtenemos todas las filas (aunque no cumplan la condición de combinación) de las dos tablas que intervienen en la clausula JOIN.



SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key



SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key



SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key

La sintaxis es muy parecida a la combinación interna, pero no es excluyente.

SELECT [ALL | DISTINCT]
<nombre_columna> [{,<nombre_columna>}]
FROM<nombre_tabla>
[{LEFT|RIGHT OUTER JOIN <nombre_tabla> ON <condicion_combinacion>}]
[WHERE <condicion> [{ AND|OR <condicion>}]]
[GROUP BY <nombre_columna> [{,<nombre_columna >}]]
[HAVING <condicion>[{ AND|OR <condicion>}]]
[ORDER BY <nombre_columna>|<indice_columna> [ASC | DESC]
[{,<nombre_columna>|<indice_columna> [ASC | DESC]}]]

La palabra reservada **OUTER** es opcional.

Con la sintaxis tradicional, que una combinación externa sea izquierda o derecha se indica situando un * a la izquierda o a la derecha del operador utilizado en la condición de combinación, dentro de la clausula WHERE. Es decir, **'*='** o **'=*'**.

Por ejemplo:
Listar las chicas y chicos de la misma ciudad y las chicas desemparejadas:

*SELECT **
FROM CHICAS, CHICOS
*WHERE CHICAS.CIUDAD * = CHICOS.CIUDAD*

CHICAS.NOMBRE	CHICAS.CIUDAD	CHICOS.NOMBRE	CHICOS.CIUDAD
Mary	Boston	John	Boston
Mary	Boston	Henry	Boston
Susan	Chicago	Sam	Chicago
Betty	Chicago	Sam	Chicago
Anne	Denver	NULL	NULL
Nancy	NULL	NULL	NULL

La consulta produce seis filas de resultados, mostrando los pares chico/chica emparejados y las chicas desemparejadas. Los chicos desemparejados no aparecen en el resultado. Este tipo de SELECT es una composición externa izquierda o "***left join***".

*SELECT **
FROM CHICAS LEFT OUTER JOIN CHICOS
ON CHICAS.CIUDAD = CHICOS.CIUDAD

Otro ejemplo:
Listar las chicas y chicos de la misma ciudad y los chicos desemparejados:

*SELECT **
FROM CHICAS, CHICOS
*WHERE CHICAS.CIUDAD = * CHICOS.CIUDAD*

Esta consulta también produce seis filas de resultados, mostrando los pares chico/chica coincidentes y los chicos desemparejados. Esta vez las chicas desemparejadas no aparecen en el resultado. Este tipo de SELECT es una combinación externa derecha o "***right join***".

*SELECT **
FROM CHICAS RIGHT OUTER JOIN CHICOS
ON CHICAS.CIUDAD = CHICOS.CIUDAD

Otro ejemplo más:
Listar los VENDEDORES y el nombre de las oficinas en las que trabajan:

SELECT NOMBRE, CIUDAD
FROM VENDEDORES, OFICINAS
*WHERE OFICINA_VEND * = OFICINA*

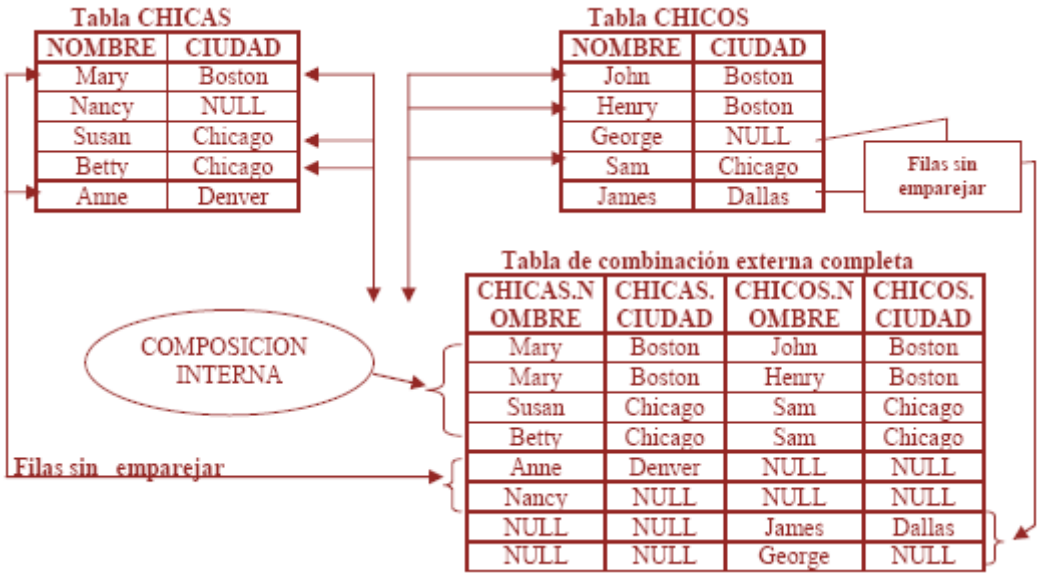
NOMBRE	CIUDAD
Tom Snyder	NULL
Mary Jones	New York
Sam Clark	New York
Bob Smith	Chicago
Paul Cruz	Chicago
Dan Roberts	Chicago
Bill Adams	Atlanta
Sue Smith	Los Angeles
Larry Fitch	Los Angeles
Nancy Angelli	Denver

Con la sintaxis tradicional, una **combinación externa completa** se indica situando un * a cada lado del operador utilizado en la condición de combinación, dentro de la clausula WHERE. Es decir, '***=***'.

Por ejemplo:

Mostrar las parejas de chicas y chicos que comparten las mismas ciudades, incluyendo las chicas y chicos desemparejados.

```
SELECT *
FROM CHICAS, CHICOS
WHERE CHICAS.CIUDAD *=* CHICOS.CIUDAD
```



Con la sintaxis moderna sería:

```
SELECT *
FROM CHICAS FULL OUTER JOIN CHICOS
ON CHICAS.CIUDAD = CHICOS.CIUDAD
```

En la práctica, las composiciones externa izquierda y derecha son más útiles que la composición externa total, especialmente en composiciones que afectan a emparejamientos *clave ajena/clave primaria*. En tal composición, la columna de clave

ajena puede contener valores NULL, produciendo filas no emparejadas de la tabla hijo (la tabla que contiene la clave ajena). Una composición externa derecha o izquierda incluirá estas filas hijo no emparejadas en los resultados de la consulta, sin incluir las filas padre no emparejadas.

6 Subconsultas

Una subconsulta es una sentencia SELECT que aparece dentro de otra sentencia SELECT que llamaremos consulta principal.

Se puede encontrar en la lista de selección, en la cláusula WHERE o en la cláusula HAVING de la consulta principal.

Una subconsulta tiene la misma sintaxis que una sentencia SELECT normal exceptuando que aparece encerrada entre paréntesis. No puede contener la cláusula ORDER BY, ni puede ser la UNION de varias sentencias SELECT, además tiene algunas restricciones en cuanto a número de columnas según el lugar donde aparece en la consulta principal. Estas restricciones las iremos describiendo en cada caso.

Cuando se ejecuta una consulta que contiene una subconsulta, la subconsulta se ejecuta por cada fila de la consulta principal.

Se aconseja no utilizar campos calculados en las subconsultas, ralentizan la consulta.

Las consultas que utilizan subconsultas suelen ser más fáciles de interpretar por el usuario.

6.1 Referencias externas

A menudo, es necesario, dentro del cuerpo de una subconsulta, hacer referencia al valor de una columna en la fila actual de la consulta principal, ese nombre de columna se denomina referencia externa.

Una **referencia externa** es un nombre de columna que estando en la subconsulta, no se refiere a ninguna columna de las tablas designadas en la FROM de la subconsulta sino a una columna de las tablas designadas en la FROM de la consulta principal. Como la subconsulta se ejecuta por cada fila de la consulta principal, el valor de la referencia externa irá cambiando.

Por ejemplo:

SELECT NUM_EMPL, NOMBRE,

(SELECT MIN(FECHAPEDIDO)FROM PEDIDOS WHERE REP = NUM_EMPL) AS PRIMER_PEDIDO

FROM VENDEDORES;

En este ejemplo la consulta principal es *SELECT... FROM VENDEDORES*.

La subconsulta es:

(*SELECT MIN(FECHAPEDIDO)FROM PEDIDOS WHERE REP = NUM_EMPL*).

En esta subconsulta tenemos una referencia externa (NUM_EMPL) es un campo de la tabla VENDEDORES (origen de la consulta principal).

¿Qué pasa cuando se ejecuta la consulta principal?

- se coge el primer empleado y se calcula la subconsulta sustituyendo NUM_EMPL por el valor que tiene en el primer empleado. La subconsulta obtiene la fecha más antigua en los pedidos del rep = 101,

- se coge el segundo empleado y se calcula la subconsulta con NUM_EMPL = 102 (NUM_EMPL del segundo empleado)... y así sucesivamente hasta llegar al último empleado.

Al final obtenemos una lista con el número, nombre y fecha del primer pedido de cada empleado.

Si quitamos la cláusula WHERE de la subconsulta obtenemos la fecha del primer pedido de todos los pedidos no del empleado correspondiente.

6.2 Anidar subconsultas

Las subconsultas pueden anidarse de forma que una subconsulta aparezca en la cláusula WHERE (por ejemplo) de otra subconsulta que a su vez forma parte de otra consulta principal. En la práctica, una consulta consume mucho más tiempo y memoria cuando se incrementa el número de niveles de anidamiento. La consulta resulta también más difícil de leer, comprender y mantener cuando contiene más de uno o dos niveles de subconsultas.

Por ejemplo:

```
SELECT NUM_EMPL, NOMBRE
FROM VENDEDORES
WHERE NUM_EMPL = (SELECT REP FROM PEDIDOS WHERE CLIE = (SELECT NUMCLIE FROM CLIENTES WHERE EMPRESA = 'First Corp.'));
```

En este ejemplo, por cada línea de pedido se calcula la subconsulta de clientes, y esto se repite por cada empleado, en el caso de tener 10 filas de VENDEDORES y 200 filas de pedidos (tablas realmente pequeñas), la subconsulta más interna se ejecutaría 2000 veces (10 x 200).

6.3 Subconsultas en la lista de selección

Cuando la subconsulta aparece en la lista de selección de la consulta principal, en este caso la subconsulta, no puede devolver varias filas ni varias columnas, de lo contrario se da un mensaje de error.

Muchos SQLs no permiten que una subconsulta aparezca en la lista de selección de la consulta principal pero eso no es ningún problema ya que normalmente se puede obtener lo mismo utilizando como origen de datos las dos tablas.

El ejemplo anterior se puede obtener de la siguiente forma:

```
SELECT NUM_EMPL, NOMBRE, MIN(FECHAPEDIDO)
FROM  VENDEDORES LEFT JOIN PEDIDOS
ON  VENDEDORES.NUM_EMPL = PEDIDOS.REP
GROUP BY NUM_EMPL, NOMBRE;
```

6.4 En la cláusula FROM

En la cláusula FROM se puede encontrar una sentencia SELECT encerrada entre paréntesis pero más que subconsulta sería una consulta ya que no se ejecuta para

cada fila de la tabla origen sino que se ejecuta una sola vez al principio, su resultado se combina con las filas de la otra tabla para formar las filas origen de la SELECT primera y no admite referencias externas.

En la cláusula FROM vimos que se podía poner un nombre de tabla o un nombre de consulta, pues en vez de poner un nombre de consulta se puede poner directamente la sentencia SELECT correspondiente a esa consulta encerrada entre paréntesis.

6.5 Subconsultas en las cláusulas WHERE y HAVING

Se suele utilizar subconsultas en las cláusulas WHERE o HAVING cuando los datos que queremos visualizar están en una tabla pero para seleccionar las filas de esa tabla necesitamos un dato que está en otra tabla.

Por ejemplo

```
SELECT NUM_EMPL, NOMBRE
FROM  VENDEDORES
WHERE CONTRATO = (SELECT MIN(FECHAPEDIDO) FROM PEDIDOS);
```

En este ejemplo listamos el número y nombre de los empleados cuya fecha de contrato sea igual a la primera fecha de todos los pedidos de la empresa.

En una cláusula WHERE / HAVING tenemos siempre una condición y la subconsulta actúa de operando dentro de esa condición.

En el ejemplo anterior se compara contrato con el resultado de la subconsulta.

Hasta ahora las condiciones estudiadas tenían como operandos valores simples (el valor contenido en una columna de una fila de la tabla, el resultado de una operación aritmética...) ahora la subconsulta puede devolver una columna entera por lo que es necesario definir otro tipo de condiciones especiales para cuando se utilizan con subconsultas. Estas nuevas condiciones se describen en el apartado siguiente.

6.6 Condiciones de selección con subconsultas

Las condiciones de selección son las condiciones que pueden aparecer en la cláusula WHERE o HAVING. La mayoría ya se han visto en el tema anterior pero ahora incluiremos las condiciones que utilizan una subconsulta como operando.

En SQL tenemos cuatro nuevas condiciones:

- test de comparación con subconsulta
- test de comparación cuantificada
- test de pertenencia a un conjunto
- test de existencia

En todos los tests estudiados a continuación ***expresión*** puede ser cualquier nombre de columna de la consulta principal o una expresión válida.

EL TEST DE COMPARACIÓN CON SUBCONSULTA.

Es el equivalente al test de comparación simple. Se utiliza para comparar un valor de la fila que se está examinado con un único valor producido por la subconsulta. La subconsulta debe devolver una única columna, sino se produce un error.

Si la subconsulta no produce ninguna fila o devuelve el valor nulo, el test devuelve el valor nulo, si la subconsulta produce varias filas, SQL devuelve una condición de error.

Por ejemplo: Lista las oficinas cuyo objetivo sea superior a la suma de las ventas de sus empleados.

SELECT OFICINA, CIUDAD

FROM OFICINAS

WHERE OBJETIVO > (SELECT SUM(VENTAS) FROM VENDEDORES WHERE OFICINA_VEND = OFICINA);

En este caso la subconsulta devuelve una única columna y una única fila (es un consulta de resumen sin GROUP BY)

EL TEST DE COMPARACIÓN CUANTIFICADA.

Este test es una extensión del test de comparación y del test de conjunto. Compara el valor de la expresión con cada uno de los valores producidos por la subconsulta. La subconsulta debe devolver una única columna sino se produce un error.

Tenemos el test **ANY** (algún, alguno en inglés) y el test **ALL** (todos en inglés).

EL TEST ANY.

La subconsulta debe devolver una única columna sino se produce un error.

Se evalúa la comparación con cada valor devuelto por la subconsulta.

Si alguna de las comparaciones individuales produce el resultado verdadero, el test ANY devuelve el resultado verdadero.

Si la subconsulta no devuelve ningún valor, el test ANY devuelve falso.

Si el test de comparación es falso para todos los valores de la columna, ANY devuelve falso.

Si el test de comparación no es verdadero para ningún valor de la columna, y es nulo para al menos alguno de los valores, ANY devuelve nulo.

Por ejemplo: Listar las oficinas cuyo objetivo sea superior a alguna de las sumas obtenidas.

SELECT OFICINA, CIUDAD

FROM OFICINAS

WHERE OBJETIVO > ANY (SELECT SUM(CUOTA) FROM VENDEDORES GROUP BY OFICINA_REP);

En este caso la subconsulta devuelve una única columna con las sumas de las cuotas de los empleados de cada oficina.

EL TEST ALL.

La subconsulta debe devolver una única columna sino se produce un error.

Se evalúa la comparación con cada valor devuelto por la subconsulta.

Si todas las comparaciones individuales, producen un resultado verdadero, el test devuelve el valor verdadero.

Si la subconsulta no devuelve ningún valor el test ALL devuelve el valor verdadero.

Si el test de comparación es falso para algún valor de la columna, el resultado es falso.

Si el test de comparación no es falso para ningún valor de la columna, pero es nulo para alguno de esos valores, el test ALL devuelve valor nulo.

Por ejemplo: Listar las oficinas cuyo objetivo sea superior a todas las sumas.

```
SELECT OFICINA, CIUDAD
FROM OFICINAS
WHERE OBJETIVO > ALL (SELECT SUM(CUOTA) FROM  VENDEDORES GROUP BY OFICINA_REP);
```

TEST DE PERTENENCIA A CONJUNTO (IN).

Examina si el valor de la expresión es uno de los valores incluidos en la lista de valores producida por la subconsulta.

La subconsulta debe generar una única columna y las filas que sean.

Si la subconsulta no produce ninguna fila, el test da falso.

Por ejemplo: listar los empleados de las oficinas del este.

```
SELECT NUM_EMPL, NOMBRE, OFICINA_REP
FROM  VENDEDORES
WHERE OFICINA_VEND IN (SELECT OFICINA FROM OFICINAS WHERE REGION = 'Este');
```

Con la subconsulta se obtiene la lista de los números de oficina del este y la consulta principal obtiene los empleados cuyo número de oficina sea uno de los números de oficina del este.

Hay que tener especial cuidado con los valores nulos cuando utilizamos el operador NOT IN porque el resultado obtenido no siempre será el deseado. Cuando la subconsulta devuelve algún nulo el resultado es falso o nulo pero nunca verdadero.

Obtener las oficinas que tienen algún empleado:

```
SELECT OFICINA, CIUDAD
FROM OFICINAS
WHERE OFICINA IN (SELECT OFICINA_VEND FROM  VENDEDORES);
```

Al contrario de lo que podríamos pensar la siguiente consulta NO devuelve las oficinas que no tienen ningún empleado porque la subconsulta devuelve algún NULL (por los empleados que no están asociados a ninguna oficina).

```
SELECT OFICINA, CIUDAD
FROM OFICINAS
WHERE OFICINA NOT IN (SELECT OFICINA_VEND FROM VENDEDORES);
```

Así sí es correcta:

SELECT OFICINA, CIUDAD
FROM OFICINAS
WHERE OFICINA NOT IN (SELECT OFICINA_VEND FROM VENDEDORES WHERE OFICINA_VEND IS NOT NULL);

EL TEST DE EXISTENCIA EXISTS.

Examina si la subconsulta produce alguna fila de resultados.

Si la subconsulta contiene filas, el test adopta el valor verdadero, si la subconsulta no contiene ninguna fila, el test toma el valor falso, nunca puede tomar el valor nulo.

Con este test la subconsulta puede tener varias columnas, no importa ya que el test se fija no en los valores devueltos sino en si hay o no fila en la tabla resultado de la subconsulta.

Cuando se utiliza el test de existencia en la mayoría de los casos habrá que utilizar una referencia externa. Si no se utiliza una referencia externa la subconsulta devuelta siempre será la misma para todas las filas de la consulta principal y en este caso se seleccionan todas las filas de la consulta principal (si la subconsulta genera filas) o ninguna (si la subconsulta no devuelve ninguna fila)

El siguiente ejemplo obtiene lo mismo que el ejemplo del test IN.

SELECT NUM_EMPL, NOMBRE, OFICINA_REP
FROM VENDEDORES
*WHERE EXISTS (SELECT * FROM OFICINAS WHERE REGION = 'Este' AND OFICINA_VEND = OFICINA);*

Observamos que delante de EXISTS no va ningún nombre de columna.

En la subconsulta se pueden poner las columnas que queramos en la lista de selección (hemos utilizado el *).

Hemos añadido una condición adicional al WHERE, la de la referencia externa para que la oficina que se compare sea la oficina del empleado.

Cuando se trabaja con tablas muy voluminosas el test **EXISTS** suele dar mejor rendimiento que el test **IN**.

Índice de contenidos

Fuentes

Consultas a la base de datos Joaquín Álvarez García/Pilar García López

IES Nº 1 de Gijón

aulaclic

Y otros.