Curso Introducción a SQL Espacial sobre PostGIS

Structured Query Language (SQL)

Comandos SQL de selección

formaSIG

La plataforma de aprendizaje en SIG Libre

sigte SERVEI DE SISTEMES D'INFORMACIÓ GEOGRÀFICA I TELEDETECCIÓ Universitat de Girona

UdGFormació FUNDACIÓ UNIVERSITAT DE GIRONA: INNOVACIÓ I FORMACIÓ

Edita: Servicio de SIG y Teledetección (SIGTE) de la Universitat de Girona

Año: 2014

Contenidos elaborados por: Toni Hernández Vallès

Comandos SQL de selección

A diferencia de los comandos SQL de estructura y manipulación vistos anteriormente, el comando SQL de selección permite extraer datos de nuestras bases de datos. En este curso vamos a centrarnos principalmente en este comando ya que permitirá analizar nuestros datos tanto des del punto de vista espacial como no espacial.

Gracias a la su gran versatilidad, este comando es el más potente y también el más utilizado. Mediante el comando SELECT podemos definir datos, extraer o seleccionar la información de la base de datos, definir filtros personalizados, dar formato a nuestros datos, ordenarlos, relacionarlos con otras tablas y aplicar gran variedad de funciones y transformaciones, las más importantes de las cuales, veremos en este curso.

SELECT

La sintaxis más sencilla (aunque poco utilizada) del comando SELECT es la siguiente.

```
SELECT <dato> ;
```

Donde 'dato' es un valor constante. Este uso del comando SELECT es poco habitual pero sirve para fines didácticos. En las lecciones sobre las funciones espaciales de PostGIS vamos a hacer uso del comando SELECT en este formato.

Otra sintaxis más habitual del comando es:

```
SELECT <lista_columnas>
FROM <lista_tablas>
[WHERE condiciones_de_la_seleccion] ;
```

Ahora sí, el comando SELECT extrae 'cierta' información de la base de datos y la presenta al usuario en forma de tabla con filas y columnas. Sin embargo a pesar del aspecto de 'tabla', el resultado del comando SELECT es transitorio y no se almacena en la base de datos (a menos que se especifique lo contrario).

Ejemplo de uso del comando SELECT:

```
SELECT * FROM vendedor;
```

Esta instrucción devuelve todas las columnas (el carácter * equivale a *todas*) de la tabla vendedor. Igual que hemos visto con otros comandos, el uso de condiciones es opcional. Al no

indicar ninguna condición de selección, la consulta devuelve *todas* las filas de la tabla vendedor.

Si deseamos seleccionar solo algunas columnas de la tabla vendedor, será necesario indicar esas columnas en el espacio reservado a ese fin. Por ejemplo.

```
SELECT nombre, primer_apellido FROM vendedor;
```

De nuevo al no indicar ningún criterio de selección, la consulta anterior devolverá un listado con el nombre y primer_apellido de *todos* los vendedores.

El siguiente paso será añadir algún criterio de selección. Por ejemplo para obtener un listado de los vendedores que se llaman 'Ana' utilizaremos el siguiente comando:

```
SELECT * FROM vendedor WHERE nombre='Ana';
```

Una de las grandes virtudes del comando SELECT viene dada precisamente por la cláusula WHERE. El uso de WHERE permite especificar una o más condiciones de selección. Podemos por ejemplo seleccionar todos los vendedores que cumplan varias condiciones: que se llaman o bien Ana o bien María y que su primer apellido sea García

Llegados a este punto, es el momento de presentar los operadores lógicos AND, OR y NOT.

Como es fácil imaginar, estos operadores se corresponden con los operadores 'Y', 'O' y 'NO' del álgebra relacional.

A continuación podemos ver el comportamiento de estos operadores lógicos AND, OR y NOT

a	***b***	***a* AND *b***	***a* OR *b***
Verdadero	Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso	Verdadero
Verdadero	NULL	NULL	Verdadero
Falso	Falso	Falso	Falso
Falso	NULL	Falso	NULL
NULL	NULL	NULL	NULL

a	**NOT *a***
Verdadero	Falso
Falso	Verdadero
NULL	NULL

Para la propuesta de consulta anterior, tenemos el siguiente sentencia SQL:

```
SELECT * FROM vendedor WHERE
(nombre='Ana' OR nombre='Maria') AND primer_apellido='García';
```

Para que una persona aparezca en el listado debe cumplir dos condiciones. La primera que se llame Ana o (OR) María y la segunda condición que su primer apellido sea García. Sin el uso adecuado de paréntesis el resultado obtenido puede no ser el esperado. Lógicamente no es lo mismo

*(nombre='Ana' **OR** nombre='Maria') **AND** primer_apellido='García'*

que

*nombre='Ana' **OR** (nombre='Maria' **AND** primer_apellido='García')*

Imaginemos ahora un ejemplo donde la cláusula WHERE presenta unas condiciones más complejas.

```
WHERE condicion1 AND (condicion2 OR condicion3 OR (condicion4 AND condicion5 AND
condicion6))
```

Las condiciones múltiples se resuelven de dentro hacia fuera. Es decir, primero las que están más anidadas y posteriormente las que se encuentran en los sucesivos niveles superiores. Para el ejemplo anterior, primero se resuelve la condición (condicion4 and condicion5 and condicion6). El resultado de esta evaluación (a partir de ahora condicion7) se evalúa con el resto de condiciones, quedando:

```
WHERE condicion1 AND (condicion2 OR condicion3 OR (condicion7))
```

El siguiente paso consiste en evaluar (condicion2 or condicion3 or condicion7). A esta nueva evaluación la llamaremos condicion8. Finalmente se resuelve el último paso de la condición inicial.

WHERE condicion1 **AND** (condicion8)

Otros operadores habituales en la cláusula WHERE son:

Operador	Ejemplo	Resultado
<	A < B	Devuelve Verdadero cuando el atributo A es menor que el atributo B
>	A > B	Devuelve Verdadero cuando el atributo A es mayor que el atributo B
<=	A <= B	Devuelve Verdadero cuando el atributo A es menor o igual que el atributo B
>=	A >= B	Devuelve Verdadero cuando el atributo A es mayor o igual que el atributo B
<>	A<>B	Devuelve Verdadero cuando A es distinto de B
!=	A != B	Devuelve Verdadero cuando A es distinto de B. Equivalente al anterior.
IN	A IN (valor1, valor2.....)	Devuelve cierto si el valor de A se encuentra dentro de los valores indicados entre paréntesis (valor1, valor2...)
NOT IN	A NOT IN (value1, value2...)	Devuelve cierto si el valor de A NO se encuentra dentro de los valores indicados entre paréntesis (valor1, valor2...)
BETWEEN .. AND	A BETWEEN B AND C	Devuelve Verdadero si el valor de A está comprendido entre los valores de B y C. Utilizado en tipos de datos numéricos y fechas.
NOT BETWEEN.. AND	A NOT BETWEEN B AND C	Devuelve Verdadero si el valor de A NO está comprendido entre los valores de B y C. Utilizado en tipos de datos numéricos y fechas.
IS NULL	A IS NULL	Devuelve Verdadero si el valor de A

		es nulo (no definido)
IS NOT NULL	A IS NOT NULL	Devuelve Verdadero si el valor de A NO es nulo
LIKE (distingue entre mayúsculas y minúsculas) ILIKE (no distingue entre mayúsculas y minúsculas)	A LIKE 'abc' A LIKE 'abc%' A LIKE '%abc%' A LIKE '%abc'	Verdadero si A='abc'. Debe haber coincidencia entre mayúsculas y minúsculas. Verdadero si A empieza por abc Verdadero si A contiene en cualquier parte la cadena de texto 'abc' Verdadero si A termina con abc

Para una referencia completa de los operadores de PostgreSQL, consultar http://www.postgresql.org/docs/9.3/static/functions.html.

Obteniendo selecciones de más de una tabla.

A diferencia de otros comandos SQL, el comando SELECT permite combinar más de una tabla en una misma expresión. Dadas las siguientes tablas: Provincia y Comarca

Provincia

id	Nombre
1	Cuenca
2	Girona

Comarca

id	nombre	Habitantes	Superficie(km2)
756	Alt Empordà	135.413	1357.53
917	Baix Empordà	130.738	701,69

Si ejecutamos el comando ***SELECT*** * ***FROM*** *Provincia,Comarca* ;

obtenemos:

id	Nombre	id	Nombre	Habitantes	Superficie(km2)
1	Cuenca	756	Alt Empordà	135.413	1357.53
1	Cuenca	917	Baix Empordà	130.738	701,69
2	Girona	756	Alt Empordà	135.413	1357.53

2	Girona	917	Baix Empordà	130.738	701,69

Como podemos ver cada fila de la tabla *Provincia* se combina con todas y cada una de las filas de la tabla *Comarca*. El número de filas del resultado obtenido es igual al numero de filas de la tabla *Provincia* multiplicado por el número de filas de la tabla *Comarca*. En este caso 2 x 2 = 4 filas. Si tuviéramos una tercera tabla 'C' entonces, el comando *SELECT * FROM Provincia, Comarca,C* devolvería cada una de las 4 filas (combinación entre provincias y comarcas) combinada con cada una de las filas de la tabla C.

Como podéis imaginar no tiene demasiado sentido combinar todas las filas de la tabla Provincia con todas las filas de la tabla Comarca. Lo lógico sería combinar las provincias únicamente con las comarcas que pertenecen a cada provincia. Para poder llevar a cabo esa combinación más lógica entre tablas, vamos a ver otra versión de las tablas Provincia y Comarca.

Provincia

Id	Nombre
...	...
16	Cuenca
17	Girona

Comarcas

Id	nombre	Habitantes	Superficie(km2)	idProvincia
...	...	...	...	...
756	Alt Empordà	135.413	1357.53	17
917	Baix Empordà	130.738	701,69	17

En este caso hemos añadido la columna *idProvincia* a la tabla *Comarca*. De este modo, siguiendo el modelo relacional de bases de datos (http://es.wikipedia.org/wiki/Modelo_relacional), estamos relacionando las entidades Provincia y Comarca. Ahora sí, podemos obtener un listado de todas las comarcas con el nombre de la provincia a la que pertenecen. Para ello aplicamos la siguiente expresión SQL.

```
SELECT * FROM provincia, comarca WHERE provincia.Id = comarca.idProvincia;
```

A diferencia de nuestro primer ejemplo (SELECT * FROM Provincia, Comarca), hemos añadido ahora la cláusula WHERE provincia.Id = Comarca.idProvicia. De este modo estamos indicando un criterio de combinación entre las distintas tablas implicadas en la expresión SQL . Concretamente se van a combinar las filas de la tabla Provincia en las que el atributo id sea igual al valor de la columna idProvincia de la tabla Comarca. De este modo evitamos la combinación exhaustiva entre ambas tablas.

Para poder llevar a cabo este tipo de consultas es imprescindible que existan columnas relacionadas (comarca.idProvincia está relacionada con Provincia.id) en las entidades que se

van a combinar. Estas columnas para relacionar distintas tablas son la columna principal sobre la que se sustentan las bases de datos relacionales. De ahí su nombre.

Podemos complicar un poco más el ejemplo anterior, añadiendo una tercera entidad *Comunidad_autonoma* y un nuevo atributo *idComunidad* a la tabla Provincia.

Comunidad_autonoma

Id	Nombre
…	…
07	Castilla-LA Mancha
08	Castilla y León
09	Cataluña.
…	…

Provincia

Id	Nombre	idComunidad
…	…	…
16	Cuenca	07
17	Girona	17
…	…	…

Podemos obtener un listado en el que aparezca, además, el nombre de la comunidad autónoma a la que pertenece la comarca. La expresión SQL que relaciona las (ahora) tres entidades es:

```
SELECT * FROM comarca, provincia, comunidad_autonoma WHERE provincia.Id =
comarca.idProvincia AND provincia.idComunidad = comunidad_autonoma.Id ;
```

En caso de existir nombres de atributos repetidos (los atributos 'id' y 'nombre' existen tanto en la tabla provincia como en la tabla comarca) deberemos indicar el nombre *completo* de esos atributos repetidos. Para ello basta especificar el nombre de la tabla seguido de un punto y el nombre del atributo tal y como se muestra en el ejemplo anterior.

Un error muy habitual en las expresiones SELECT, donde intervienen varias tablas, consiste en excluir alguna de las condiciones de la cláusula WHERE necesarias para evitar la combinación exhaustiva de filas. Cuantas más tablas intervengan mayor será la probabilidad de olvidar alguna de esas condiciones.

Ordenando las selecciones

Otra posibilidad del comando SELECT es la cláusula **ORDER BY** que permite ordenar el resultado de las selecciones.

En el ejemplo anterior podemos ordenar la selección por el atributo Comarca.nombre del siguiente modo:

```
SELECT * FROM comarca, municipio WHERE comarca.Id = municipio.idComarca
ORDER BY comarca.nombre ASC ;
```

Donde ASC significa que la ordenación es ascendente (A – Z). Para indicar un orden descendente la palabra reservada es DESC .

ORDER BY permite también ordenaciones para más de un atributo. La siguiente sentencia devuelve el listado anterior ordenado, primero por el nombre de la comarca, y cuando los nombres de comarcas son iguales, el listado se ordena por el nombre del municipio.

```
SELECT * FROM comarca, municipio WHERE comarca.Id = municipio.idComarca
ORDER BY comarca.nombre, municipio.nombre ASC ;
```

Incluso podemos ordenar ascendentemente por un atributo y descendentemente por otro.

```
SELECT * FROM comarca, municipio WHERE comarca.Id = municipio.idComarca
ORDER BY comarca.nombre ASC , municipio.nombre DESC ;
```

Eliminando las filas repetidas de una selección.

En ocasiones los resultados de una selección muestran información repetida. Si seleccionamos, por ejemplo, solo la columna *nombre* de una tabla *vendedor*, entonces aparecerá una fila con el valor 'Juan' para cada Juan de la tabla *vendedor*. Esto, como es lógico imaginar, puede resultar molesto e incluso confuso. Por este motivo SQL incorpora la función *DISTINCT* que permite agrupar las filas cuyas columnas tienen los mismos valores. Para obtener, por ejemplo, un listado con todos los nombres distintos de la tabla *vendedor* podemos utilizar la expresión:

```
SELECT DISTINCT(nombre) FROM vendedor;
```

En este caso solo aparecerá una fila con el valor Juan.

Para ver todos los vendedores con nombre y apellidos distintos aplicaremos el comando:

SELECT DISTINCT(nombre apellidos) **FROM** vendedor;

Limitando la selección a 'N' filas

En ocasiones solo deseamos conocer las 'N' primeras filas de una selección. Para ello podemos añadir a la setencia SQL la clásula **LIMIT** del siguiente modo:

SELECT * **FROM** comarca, municipio **WHERE** comarca.Id = municipio.idComarca
ORDER BY comarca.nombre **ASC** , municipio.nombre **DESC LIMIT** 5

El sistema gestor de bases de datos PostgreSQL incorpora además la cláusula **OFFSET**. OFFSET permite obviar las 'M' primeras filas eliminándolas de una selección .

SELECT * **FROM** comarca, municipio **WHERE** comarca.Id = municipio.idComarca
ORDER BY comarca.nombre ASC , municipio.nombre **DESC LIMIT** 5 **OFFSET** 5

La sentencia anterior devolverá solo 5 filas. Concretamente devolverá de la fila 6 hasta la fila 10 (ambas incluidas). Las cláusulas LIMIT y OFFSET deben ir siempre al final de la sentencia SQL.

Operadores.

El uso de operadores en SQL permite realizar operaciones con las columnas de nuestras tablas. Los operadores más utilizados son los operadores matemáticos +,-,* y /.

select Habitantes / 1000 from municipio; devolverá un listado de los habitantes de los municipios en millares.

Estos operadores son habituales para modificar las unidades de salida de ciertos atributos. Por ejemplo, para convertir una área en metros cuadrados a hectáreas bastará con dividir por 10.000

Lógicamente los operadores matemáticos solo se pueden aplicar sobre tipos de datos numéricos. No tendría ningún sentido aplicar una suma o una resta sobre el atributo NOMBRE de la tabla Comarca. Otros tipos de datos (character varying, date, etc) disponen de otros tipos de operadores. La siguiente lectura entra en más detalle en los tipos de datos y sus operadores.

Funciones

Más allá de los operadores, las funciones permiten una modificación más avanzada de los atributos.

La sintaxis de las funciones es distinta a la de los operadores. En este caso debemos escribir el nombre de la función seguida de paréntesis. Dentro de esos paréntesis deben aparecer los *parámetros* requeridos por la función. Estos parámetros pueden (o no) coincidir con las columnas (atributos) de la selección.

Sintaxis:

SELECT nombre_funcion(atributo) **FROM** …..

Pongamos un ejemplo sencillo. En PostgreSQL existe la función *upper()*. Esta función convierte cualquier cadena de texto (character varying) a mayúsculas.

El comando:

SELECT upper(comarca.nombre) **FROM** Comarca;

devolverá un listado del nombre de las comarcas en mayúsculas.

Cada parámetro de cada función debe ser de un tipo concreto. Si utilizamos la función upper con un parámetro numérico, p.ej SELECT upper(3), nos devolverá un error.

Otras funciones pueden necesitar más de un parámetro, como por ejemplo la función *concat()*. A partir de la versión de PostgreSQL 9.1, la función *concat* reemplaza a la función *textcat*.

Concat acepta un número variable de parámetros (siempre que sean de texto) y devuelve un solo atributo equivalente a la concatenación de esos 'n' parámetros.

concat('hola', ' mundo ', 'feliz') devolverá una sola columna con el texto 'hola mundo feliz'.

Si aplicamos concat sobre las tablas comarca y municipios.

SELECT concat(comarca.nombre, ' ', municipio.nombre) **FROM** comarcas, municipios **WHERE** comarca.id=municipios.idComarca;

obtendremos una sola columna con el nombre de la comarca y el nombre del municipio.

El segundo atributo (' ') de la función textcat sirve para evitar que el nombre de la comarca y el nombre del municipio aparezcan como una sola palabra (Alt Empordrà Agullana en lugar de Alt EmpordàAguallana).

Para un estudio más exhaustivo de las funciones de PostgreSQL podéis consultar la guía oficial en http://www.postgresql.org/docs/9.3/static/functions-string.html .

Funciones de agregado

A diferencia de las funciones vistas en el apartado anterior, las funciones de agregado se caracterizan por operar sobre un *conjunto de filas* en lugar de operar sobre cada una de las filas de la selección. En algunos textos, a las funciones de agregado también se las conoce como funciones de resumen.

Las principales funciones de agregado son:

COUNT(columnaA): Cuenta el número de filas que contienen valores no nulo para la columna indicada. Acepta el uso de '*' para contar todas las filas (incluyendo las que tienen valores nulos)

SUM(columnaA): Devuelve la suma de todas las filas para los valores de la columnaA .

MAX(columnaA): Devuelve el valor máximo de todas las filas para la columnaA.

MIN(columnaA): Devuelve el valor mínimo de todas las filas para la columnaA.

Veamos algunos ejemplos de las funciones de agregado aplicados a la siguiente tabla de libros.

Libros:

Titulo	**Autor**	**Páginas**
La luz del Yoga	B.K.S Iyengar	260
Luz sobre el pranayama	B.K.S Iyengar	180
Guía práctica de Yoga Iyengar	B.K.S Iyengar	nulo
Yoga Postures	James Hewitt	120

COUNT:

Ej: **SELECT** COUNT(paginas) **FROM** libros;

count
3

Existen tres filas que contienen valores no nulos para la columna *Páginas*.

Ej: **SELECT** COUNT(*) FROM libros;

count
4

Existen 4 libros independientemente de los valores de sus columnas.

Si deseamos establecer diferentes agrupaciones entre filas podemos utilizar las palabras reservadas *GROUP BY*. De este modo, antes de ejecutar la función de agregado, las filas serán agrupadas en función de cual sea su valor para los atributos indicados en GROUP BY. Después de definir las agrupaciones, se aplicará la función de agregado sobre cada una de esas agrupaciones.

Ej: select count(paginas) from libros group by autor;

count
2
1

En primer lugar se agrupa los libros que tiene cada autor y posteriormente se aplica la función *count.* Es decir, hay dos autores (la consulta devuelve dos filas) que tiene libros con un número de páginas conocido. Uno de los autores tiene dos libros y el otro autor tiene uno.

Ej: **SELECT** count(*) **FROM** libros **GROUP BY** autor;

Resultado:

count
3
1

Número de libros que tiene cada autor independientemente de si conocemos el número de páginas de cada libro. Tres libros para un autor y un libro para otro autor.

SUM:

Ej: **SELECT** SUM(paginas) **FROM** libros;

Resultado:

SUM
560

Los valores nulos quedan excluidos.

AVG:

Ej: **SELECT** AVG(paginas) **FROM** libros;

Resultado:

SUM
186.666

Valor promedio del número de páginas de todos los libros. Excluidos los valores nulos.

MAX:

Ej: select max(paginas) from libros;

Resultado:

MAX
260

Excluidos los valores nulos.

MIN:

Ej: **SELECT** MIN(paginas) **FROM** libros;

Resultado:

MIN
120

Excluidos los valores nulos.

Alias.

Otro uso muy habitual para la obtención de listados (o selecciones) consiste en renombrar “al vuelo” las columnas de dichos listados. En la tabla Municipios, por ejemplo, podemos calcular con el operador '/' la densidad de población a partir de las columnas 'Habitantes' y 'Superficie'. Para hacer más entendedor el resultado obtenido podemos renombrar la columna del siguiente modo.

SELECT (Habitantes/Superficie) **AS** Densidad **FROM** Municipios;

Donde AS indica al SGBD que debe renombrar el resultado de Habitantes/Superficie a 'Densidad'

Sentencia JOIN

Para la vinculación de tablas, además de la sintaxis que hemos visto anteriormente, podemos utilizar también la sentencia JOIN.

Veamos con algunos ejemplos como podemos utilizar JOIN y qué nuevas posibilidades nos ofrece.

Dadas las siguientes tablas *Empresa, Pedido y Cliente*

Tabla: Empresa

id	**Nombre**
1	Pedrega.SL
2	FiKSA
3	Corral

Tabla: Pedido

id	**Fecha**	**cliente_id**	**Precio**
1	3-2-2013	3	300
2	5-2-2013	4	1200
3	8-3-2013	5	500
4	10-5-2013	6	800
5	5-6-2013	7	150
6	15-7-2013	1	8500

Tabla: Cliente

Id	Nombre	C_postal	Empresa_id
1	Pere	17001	1
2	Ana	17003	2
3	David	17001	1
4	Laura	17001	1
5	Iolanda	17005	
6	Alex	17001	2
7	Silvia	17005	2

Si deseamos cruzar las tablas *Empresa* y *Cliente* para obtener un listado con los trabajadores y la empresa a la que pertenecen, podemos ejecutar el siguiente comando tal y como hemos venido haciendo en esta lectura.

```
SELECT * FROM empresa, cliente WHERE cliente.empresa_id = empresa.id;
```

El resultado que obtendremos será:

Id	Nombre	Id	nombre	c_postal	empresa_id
1	Pedrega.SL	4	Laura	17001	1
1	Pedrega.SL	3	David	17001	1
1	Pedrega.SL	1	Pere	17001	1
2	FiKSA	7	Sílvia	17005	2
2	FiKSA	6	Alex	17001	2
2	FiKSA	2	Ana	17003	2

O bien podemos obtener el mismo resultado utilizando la sentencia *JOIN,* y más concretamente la variedad *INNER JOIN*:

```
SELECT * FROM empresa INNER JOIN cliente ON cliente.empresa_id=empresa.id;
```

Si fuera necesario podríamos unir una tercera tabla añadiendo una nueva cláusula INNER JOIN al final de la sentencia anterior. Por ejemplo, podemos añadir a la información de cada pedido, el nombre del cliente y la empresa a la que pertenece con la siguiente sentencia SQL:

```
SELECT pedido.*, cliente.nombre, empresa.nombre FROM empresa INNER JOIN cliente ON cliente.empresa_id=empresa.id INNER JOIN pedido ON pedido.cliente_id=cliente.id;
```

El comando anterior devolverá el siguiente resultado:

id	fecha	cliente_id	precio	nombre	nombre
6	15-7-2013	1	8500	Pere	Pedrega.SL
1	3-2-2013	3	300	David	Pedrega.SL
2	5-2-2013	4	1200	Laura	Pedrega.SL
4	10-5-2013	6	800	Alex	FiKSA
5	5-6-2013	7	150	Sílvia	FiKSA

Existen distintas variedades de JOIN. Las más importantes son:

INNER JOIN: Devuelve las filas que cumplen la condición en las dos tablas implicadas. Este es el más utilizado.

Ejemplo: `SELECT * FROM empresa INNER JOIN cliente ON cliente.empresa_id=empresa.id;`

Resultado:

Id	Nombre	Id	nombre	c_postal	empresa_id
1	Pedrega.SL	4	Laura	17001	1
1	Pedrega.SL	3	David	17001	1
1	Pedrega.SL	1	Pere	17001	1
2	FiKSA	7	Sílvia	17005	2
2	FiKSA	6	Alex	17001	2
2	FiKSA	2	Ana	17003	2

LEFT JOIN: Devuelve todas las filas de la primera tabla (left) y las filas de la segunda tabla que cumplen la condición.

Ejemplo: **SELECT** * **FROM** empresa **LEFT JOIN** cliente **ON** cliente.empresa_id=empresa.id;

Resultado:

Id	Nombre	Id	nombre	c_postal	empresa_id
1	Pedrega.SL	4	Laura	17001	1
1	Pedrega.SL	3	David	17001	1
1	Pedrega.SL	1	Pere	17001	1
2	FiKSA	7	Sílvia	17005	2
2	FiKSA	6	Alex	17001	2
2	FiKSA	2	Ana	17003	2
3	Corral				

RIGHT JOIN: Devuelve todas las filas de la segunda tabla (right) y las filas de la primera (left) que cumplen la condición.

Ejemplo: **SELECT** * **FROM** empresa **RIGHT JOIN** cliente **ON** cliente.empresa_id=empresa.id;

Resultado:

Id	Nombre	Id	nombre	c_postal	empresa_id
1	Pedrega.SL	4	Laura	17001	1
1	Pedrega.SL	3	David	17001	1
1	Pedrega.SL	1	Pere	17001	1
2	FiKSA	7	Sílvia	17005	2
2	FiKSA	6	Alex	17001	2
2	FiKSA	2	Ana	17003	2
		5	Iolanda	17003	

FULL JOIN: Devuelve todas las filas siempre que se cumpla la condición en alguna de las dos tablas.

Ejemplo: **SELECT** * **FROM** empresa **FULL JOIN** cliente **ON** cliente.empresa_id=empresa.id;

Resultado:

Id	Nombre	Id	nombre	c_postal	empresa_id
1	Pedrega.SL	4	Laura	17001	1
1	Pedrega.SL	3	David	17001	1
1	Pedrega.SL	1	Pere	17001	1
2	FiKSA	7	Sílvia	17005	2
2	FiKSA	6	Alex	17001	2
2	FiKSA	2	Ana	17003	2
3	Corral				
		5	Iolanda	17003	

Consultas anidadas.

SELECT permite, además de todo lo visto hasta ahora, anidar distintas consultas. Podemos utilizar el resultado de una 'selección' dentro del espacio reservado a la clausula WHERE como si se tratara de una condición más. Incluso podemos utilizar el resultado de una 'selección' como si se tratara una tabla más (aunque sea transitoria). Veamos un ejemplo.

Dada la tabla Personas con los atributos: Nombre, fecha_nacimiento, salario. Vamos a obtener un listado de las personas que tienen un salario más alto que Juan.

Persona

Nombre	fecha_nacimiento	Salario (euros)
Juan	1/5/1984	1200
Isabel	1/1/1971	1350
Albert	22/7/1980	1100

Para obtener este listado existen dos posibilidades en función de si sabemos de antemano lo que gana Juan.

Si conocemos el salario de Juan (1200 euros) entonces podemos definir la siguiente selección y problema resuelto.

```
SELECT * FROM persona WHERE salario > 1200;
```

Pero, ¿y si no sabemos el salario de Juan?

Entonces, lógicamente, el primera paso consiste en saber cuánto gana Juan. Para ello definimos el siguiente comando

```
SELECT salario FROM persona WHERE persona='Juan';
```

El resultado del comando anterior es una sola fila (la de Juan) con una sola columna (salario). Por lo tanto ahora podemos fusionar, con el operador >, las dos selecciones en una sola.

```
SELECT * FROM persona
WHERE salario > (
SELECT salario FROM persona
WHERE persona='Juan'
);
```

Es imprescindible que la consulta más interior devuelva un solo valor (una fila con una sola columna).

Este uso del comando SELECT se conoce como consultas anidas o subconsultas.

En el ejemplo anterior, hemos utilizado una subconsulta para obtener un valor concreto (una fila y una sola columna) y lo hemos incluido en la cláusula WHERE de la consulta principal. Existe, sin embargo, otra posibilidad más avanzada que consiste en utilizar el resultado de la subconsulta como si de una tabla (más que de un simple valor) se tratara. La sintaxis queda entonces del siguiente modo:

```
SELECT atributoA, FROM tabla1, (select atributoB from tabla2) AS foo
WHERE tabla1.atributoX = foo.atributoY;
```

En este caso la subconsulta 'SELECT atributoB FROM tabla2' está definida dentro del espacio (de la consulta principal) reservado a la cláusula FROM.

En este segundo tipo de subconsultas debemos, obligatoriamente, indicar un alias (AS) a la subconsulta. En el ejemplo anterior hemos 'renombrado' la subconsulta a 'foo' y por lo tanto cualquier atributo, proveniente de la subconsulta, que utilicemos deberemos indicarlo con la nomenclatura nombrealias.nombreatributo.

Para el siguiente ejemplo vamos a utilizar de nuevo los datos de las siguientes tablas.

Provincia

id	Nombre
1	Cuenca
2	Girona
...	

Comarca

id	nombre	Habitantes	Superficie(km2)
756	Alt Empordà	135.413	1357.53
917	Baix Empordà	130.738	701,69

Vamos a construir una sentencia SQL para seleccionar todas las comarcas de Girona y de Cuenca que cuenten con más de 50.000 habitantes.

Como siempre existen varias vías distintas de llegar al mismo resultado, y lejos de ser la mejor opción vamos, en este caso y por motivos pedagógicos a mostrar una selección mediante el uso de una subconsulta.

```
SELECT * FROM comarca,
(
SELECT * FROM provincia
WHERE nombre = 'Cuenca' or nombre='Girona'
) as foo
WHERE comarca.habitantes>50.000 AND comarca.idProvincia=foo.idProvincia;
```

En este ejemplo la subconsulta devuelve todas las columnas pero solo de las provincias de Cuenca y Girona. Posteriormente se lleva a cabo otra consulta que combina los valores obtenidos anteriormente con los valores de la tabla comarca utilizando un atributo compartido (idProvincia). Finalmente se aplica el filtro del número de habitantes y se obtiene el resultado deseado.

Hasta aquí hemos visto las piezas más importantes que podemos utilizar en la construcción de sentencias SQL con el comando SELECT. A partir de ahora podremos combinar todos estos elementos de la manera que más se ajuste a nuestras necesidades. Cualquier consulta que seamos capaces de imaginar la podremos convertir en una consulta SQL. El límite de las consultas está, entonces, en nuestra imaginación.

sigte
SERVEI DE SISTEMES D'INFORMACIÓ GEOGRÀFICA I TELEDETECCIÓ
Universitat de Girona

UdGFormació
FUNDACIÓ UNIVERSITAT DE GIRONA: INNOVACIÓ I FORMACIÓ

www.sigte.udg.edu/formasig

formasig@sigte.org