Curso Introducción a SQL Espacial sobre PostGIS

Structured Query Language (SQL)

Actividad práctica 3: Ejecución de comandos SQL

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

Actividad práctica 3: Ejecución de comandos SQL

Creando la base de datos

Desde el software pgAdmin creamos una nueva base de datos llamada practica4. Para ello puedes reproducir el video *Creando una base de datos desde pgAdmin*.

El siguiente paso consiste en ejecutar el fichero *crear_tablas.sql*. Para ello vamos a cargar el fichero desde una consola SQL dentro de pgAdmin siguiendo los pasos que se detallan en la Guía de Usuario de pgAdmin, al final del apartado *Ejecutar Expresiones SQL*.

Esto habrá creado la estructura (vacía) de nuestra base de datos. Lo siguiente consistirá en insertar datos en las tablas que acabamos de crear. Con este fin utilizaremos los comandos *COP Y* e *INSERT INTO*.

Insertando datos

Antes de ejecutar los siguientes comandos, asegúrate que el usuario postgres tiene permisos para acceder a la carpeta donde se encuentran los datos (c:\curso_postgis).

```
COPY viticultor FROM 'c:\curso_postgis\viticultor.dat';
COPY parcela FROM 'c:\curso_postgis\parcela.dat';
COPY variedad FROM 'c:\curso_postgis\variedad.dat';
COPY poblacion FROM 'c:\curso_postgis\poblacion.dat';
```

Puedes ejecutar los comandos todos de una sola vez (importante el carácter ';' que marca el fin de cada sentencia) o uno por uno.

Con esto tendremos pobladas nuestras tablas. También podemos añadir nuevos datos con el siguiente comando INSERT INTO.

```
INSERT INTO viticultor (id, nombre, apellidos, telefono, fax, poblacion, codigopostal)
VALUES (11111, 'Pedro','Martín Lozano', '972 555 555',null,'Sant Cugat', '08173 ');
```

En cambio si intentamos introducir un viticultor con un valor nulo en su atributo id, por ejemplo con el comando:

```
INSERT INTO viticultor (id, nombre, apellidos, telefono, fax, poblacion, codigopostal)
VALUES (null, 'Pedro','Martín Lozano', '972555555',null,'Sant Cugat', '08173');
```

se producirá el siguiente error.

ERROR: el valor null para la columna «id» viola la restricción not null

Este error está indicando que el atributo 'id' no acepta valores nulos. Si repasamos el comando SQL de creación de la tabla viticultor (archivo crear_tablas.sql) veremos que el atributo id está definido como NOT NULL.

```
CREATE TABLE viticultor
(
  id integer NOT NULL PRIMARY KEY,
  nombre character varying(50),
  apellidos character varying(50),
  telefono character(15),
  fax character varying(50),
  poblacion character varying(255),
  codigopostal character varying(50)
);
```

Tampoco será posible ejecutar este comando:

```
INSERT INTO viticultor (id, nombre, apellidos, telefono, fax, poblacion, codigopostal)
VALUES (11111, 'Ana', 'García Ferrer', '934555555',null,'Girona', '17001');
```

Pues el atributo id está definido además como clave primaria y ello conlleva que no puede haber valores repetidos para este atributo. En esta ocasión se producirá el siguiente error.

ERROR: llave duplicada viola restricción de unicidad «viticultor_pkey»
DETAIL: Ya existe la llave (id)=(11111).

Modificando datos

Podemos modificar cualquier atributo de nuestras tablas siempre que se mantengan las restricciones preestablecidas.

El siguiente comando modifica el atributo id de la tabla viticultor para el viticultor con id=11111 asignándolo el nuevo valor 22222.

UPDATE viticultor **SET** id=22222 **WHERE** id=11111;

El comando UPDATE puede operar sobre un conjunto de filas e incluso sobre todas las filas de una tabla. El comando que mostramos a continuación modifica todas las filas de la tabla *variedad.* Concretamente modifica el atributo *variedad* convirtiéndolo en minúsculas gracias a la función *lower*.

Antes de modificar la tabla podemos observar su contenido con el comando

SELECT * **FROM** variedad;

id integer	variedad character varying(255)
1	MACABEO
2	XAREL.LO
3	MERLOT
4	CHARDONNAY
5	PINOT NOIR
6	SYRAH
7	SUMOLL
8	ALBARIÑO
9	PARELLADA
10	TEMPRANILLO
11	CABERNET SAUVIGNON
12	MOSCATEL
13	GARNACHA TINTA

Si ahora ejecutamos el comando UPDATE de modificación

UPDATE variedad **SET** variedad = lower(variedad);

y finalmente volvemos a visualizar la tabla variedad, veremos como el atributo variedad de todas las filas se ha convertido a minúsculas.

id integer	variedad character varying(255)
1	macabeo
2	xarel.lo
3	merlot
4	chardonnay
5	pinot noir
6	syrah
7	sumoll
8	albariño
9	parellada
10	tempranillo
11	cabernet sauvignon
12	moscatel
13	garnacha tinta

Consultas

Vamos a ver ahora la parte más apetecible de esta actividad: La consultas.

Consultas básicas

Veremos ahora algunos ejemplos de uso del comando **SELECT**. Ya hemos visto como visualizar el contenido de una tabla:

SELECT * **FROM** viticultor ;

Este comando debe leerse como "selecciona todos los campos de la tabla viticultor".

Si deseamos que en el listado únicamente aparezcan algunos atributos, podemos indicar esos atributos del siguiente modo.

SELECT nombre, apellidos, telefono **FROM** viticultor;

Si además queremos ordenar el resultado de la consulta, utilizaremos la cláusula *ORDER BY*

SELECT nombre, apellidos, telefono **FROM** viticultor **ORDER BY** apellidos;

Consultas de filtrado

En pgAdmin podemos mostrar el contenido de una tabla sin necesidad de teclear ningún comando SQL, motivo por el cual, los comandos de selección vistos anteriormente son poco utilizados. Más comunes que los anteriores, son los comandos que extraen únicamente las filas que cumplen ciertas condiciones. Veamos ahora algunos ejemplos.

1.- Para obtener un listado con las parcelas plantadas antes del año 1950:

```
SELECT * FROM parcela WHERE plantado_en <1950;
```

Traduciendo el comando anterior al español:

"*selecciona todas las columnas (*) de parcela donde la columna plantado_en tenga un valor inferior a 1950"*

***2.*- Parcelas plantadas en la década de los 50. Tenemos dos opciones según qué operadores vayamos a utilizar.**

```
SELECT * FROM parcela WHERE plantado_en >= 1950 AND plantado_en <= 1960;
SELECT * FROM parcela WHERE plantado_en between 1950 AND 1960;
```

Fíjate que el operador *between* incluye las parcelas plantadas en el año 1950 y las plantadas en el año 1960.

Uso del operador like

3.- Viticultores con el primer apellido García.

```
SELECT * FROM viticultor WHERE apellidos ILIKE 'garcía%';
```

4.- Viticultores con el segundo apellido García.

```
SELECT * FROM viticultor WHERE apellidos ILIKE '%garcía';
```

5.-Viticultores con el algún apellido 'García'.

```
SELECT * FROM viticultor WHERE apellidos ILIKE '%garcía%';
```

En estos casos hemos utilizado el operador *ilike* en lugar de *like* para que se incluyan en el resultado final tanto las filas que contengan mayúsculas (GARCÍA, García, etc.) como las que contengan minúsuculas (garcía).

Fíjate en el uso y posición del carácter % en cada caso.

'garcía%': Indica que la columna apellidos debe empezar por 'garcía' sin importar para nada los caracteres que aparezcan (si los hay) a la derecha.

'%garcía': Indica que la columna apellidos debe terminar con la palabra 'garcía' sin importar para nada los caracteres que aparezcan (si los hay) a la izquierda.

'%garcía%': Indica que la columna apellidos debe contener en alguna posición la palabra 'garcía' sin importar para nada los caracteres que aparezcan (si los hay) a la derecha o a la izquierda.

6.- Listado de parcelas de tipo A plantadas hace menos de 20 años.

```
SELECT * FROM parcela WHERE tipo_parcela='Tipo A' AND plantado_en > 1992;
```

7.- Listado de parcelas ordenadas por el tipo_parcela y en segundo lugar por la superficie que ocupan. De mayor superficie a menor.

```
SELECT * FROM parcela ORDER BY tipo_parcela asc, superficie DESC;
```

funciones de agregado Las funciones de agregado, como se explica en las lecturas de esta unidad, operan sobre un conjunto de filas. Entre las funciones de agregado más importantes tenemos SUM, AVG, MIN y MAX. Veamos algunos ejemplos de uso.

8.-Para conocer el número de parcelas de la tabla parcela.

```
SELECT count(*) FROM parcela;
```

Cuenta todas (*) las filas de parcela, incluyendo las filas que contienen valores nulos en algunas de sus columnas.

9.-Para conocer cuantas parcelas tienen un valor conocido (no nulo) en la columna plantado_en .

```
SELECT count(plantado_en) FROM parcela;
```

10.- Valor medio del año de plantación de todas las cepas. Considerando únicamente las cepas de las cuales se conoce el año de plantación.

```
SELECT avg(plantado_en) FROM parcela;
```

11.- Año de plantación de las cepas más antiguas y de las cepas más modernas.

```
SELECT  max(plantado_en) as cepas_antiguas, min(plantado_en) as cepas_modernas FROM parcela;
```

Para hacer más entendedora la selección hemos renombrado (al vuelo) las columnas con los valores correspondientes de cepas_antiguas y cepas_modernas.

Si añadimos los criterios de selección a las funciones de agregado podemos dar respuesta a cuestiones como las que vienen a continuación.

12.-¿Cuántas parcelas tienen cepas plantadas en al década de los 50?.

```
SELECT  count(*) FROM parcela WHERE plantado_en >=1950 AND plantado_en < 1960;
```

13.-¿Cual es la superficie total de cepas de tipo A plantadas antes de 1980?

Imagina que no recuerdas el formato exacto de la columna tipo_superficie. No sabes si los valores son 'TIPO A', 'Tipo A', 'Tipo a', etc. En este caso nos viene muy bien el operador ilike que ya hemos visto con anterioridad.

```
SELECT  sum(superficie) FROM parcela WHERE tipo_parcela ilike 'tipo a' AND plantado_en<1980;
```

14.- Entre las cepas de la variedad con variedad_id=1, ¿De qué año son las cepas más antiguas?

```
SELECT  min(plantado_en) FROM parcela WHERE variedad_id=1;
```

Cláusula GROUP BY

GROUP BY permite definir agrupaciones de filas que comparten un mismo valor para la columna indicada. Si combinamos las funciones de agregado con la cláusula GROUP BY se nos abre un nuevo abanico de posibilidades.

Veamos algunas aplicaciones prácticas que nos ayudarán entender mejor este concepto.

15.-Número de parcelas que tiene cada viticultor.

```
SELECT  count(*) FROM parcela GROUP BY viticultor_id;
```

Es decir, "selecciona el conteo de todas las filas (*) de la tabla parcela agrupando las filas que tengan el mismo valor en la columna viticultor_id".

El resultado obtenido por el comando anterior será algo parecido (aunque con más filas) a :

count bigint
2
2
12

...

Como se puede apreciar este listado es poco práctico pues no relaciona el número de parcelas con el id de cada viticultor. Sabemos que hay, entre otros, un viticultor con 12 parcelas pero no sabemos de qué viticultor se trata. Para corregir esta situación solo es necesario añadir la columna viticultor_id al resultado de la selección. El comando final es:

```
SELECT  count(*), viticultor_id FROM parcela GROUP BY viticultor_id;
```

count bigint	viticultor_id integer
2	83805
2	83390
12	84036
9	80236
1	89030

...

Ahora sí, sabemos que el viticultor con id=84036 tiene 12 parcelas. Si queremos saber el resto de atributos de este viticultor ejecutaremos en un nuevo comando:

```
SELECT * FROM viticultor WHERE id=84036;
```

16.- Obtener un listado con la relación del número de parcelas que tienen cepas de un mismo año.

```
SELECT plantado_en , count(plantado_en) as Num_parcelas FROM parcela
GROUP BY plantado_en;
```

17.- Calcular la suma de las superficies de todas las parcelas del viticultor anterior.

```
SELECT sum(superficie) FROM parcela WHERE viticultor_id=84036;
```

sum double precision
94616.5063476

En caso de existir parcelas con superficie desconocida (valor nulo), dichas parcelas quedarían excluidas del sumatorio.

18.- Averiguar cual es la superficie media de todas las parcelas. Calcular también cual es la superficie media de las parcelas de un mismo viticultor.

Para todas las parcelas:

```
SELECT avg(superficie) FROM parcela;
```

avg double precision
12013.8223842

Para cada viticultor:

```
SELECT  avg(superficie), viticultor_id
FROM parcela
GROUP BY viticultor_id ;
```

avg double precision	viticultor_id integer
14592.1447753	83805
12303.1079711	83390
7884.70886230	84036
15349.7288682	80236
3043.25952148	89030
8894.42097981	86310
9007.32839965	87739
7522.59720865	80365

Observa como hemos añadido la columna viticultor_id para que aparezca en el resultado de la consulta.

19.- ¿Cuántas parcelas de tipo A tiene cada viticultor?

```
SELECT  count(*) as numero_parcelas, viticultor_id FROM parcela
WHERE tipo_parcela='Tipo A'
GROUP BY viticultor_id
ORDER BY numero_parcelas DESC;
```

Además hemos ordenado el resultado de manera descendiente para ver con más facilidad los viticultores que tienen más parcelas de Tipo A.

consultas cruzadas

Las consultas cruzadas son las consultas en las que intervienen más de una tabla. Un ejemplo sencillo es obtener un listado de parcelas donde aparezca, además de la información de las

parcelas, toda la información del viticultor al que pertenece cada parcela. Como vemos se trata de obtener información de dos tablas distintas: parcela y viticultor.

```
SELECT  parcela.*, viticultor.* FROM parcela, viticultor WHERE parcela.viticultor_id = viticultor.id;
```

En las consultas cruzadas, como se menciona en la lectura del comando SQL de manipulación, es importante poder vincular las tablas a partir de atributos comunes. En este caso el atributo viticultor_id de la tabla parcela debe corresponderse con el atributo id de la tabla viticultor. Estas columnas pueden tener nombres distintos pero deben almacenar los mismos valores. Más concretamente, cualquier valor de parcela.viticultor_id debe aparecer, forzosamente, en viticultor.id. Puede darse el caso que algún valor viticultor.id no aparezca en la tabla parcela.viticultor_id. En este caso estaremos delante de un viticultor que no tienen ninguna parcela.

Prueba a ejecutar el comando anterior sin la condición WHERE. Verás que el número de filas de la consulta se incremente considerablemente.

Podemos vincular tantas tablas como sea necesario. El comando que veremos a continuación obtiene una relación de las parcelas donde aparece, además del nombre de la parcela, el nombre y apellidos de su viticultor, el nombre de la variedad de uva que contiene la parcela y el nombre de la población donde se ubica cada parcela. Además el resultado está ordenado descendientemete por el tipo de variedad de uva.

```
SELECT  parcela.nombre as Nombre_Parcela, viticultor.nombre, apellidos, variedad, poblacion.poblacion
FROM parcela, viticultor, variedad, poblacion
WHERE
parcela.viticultor_id=viticultor.id AND parcela.poblacion_id=poblacion.id AND parcela.variedad_id=variedad.id
ORDER BY variedad DESC;
```

Debido a que, tanto la tabla parcela como la tabla viticultor, contienen una columna llamada 'nombre', las apariciones de dichas columnas en el comando SQL deben ir acompañadas del nombre de la tabla a la que pertenecen. Así evitamos la ambigüedad. Además hemos renombrado la columna nombre de la tabla parcela por 'Nombre_Parcela'.

La siguiente imagen muestra una pequeña parte del resultado de la consulta anterior.

nombre_parcela character varying(255)	nombre character varying(50)	apellidos character varying(50)	variedad character varying(255)	poblacion character varying(255)
FONTANALA	Joan	Benedicto Martínez	xarel.lo	SUBIRATS
1LA SALA	Santiago	Casas Uribeondo	xarel.lo	VILOBI PENEDES
SOTA EL BOSC III	Jaume	Tugues	xarel.lo	CASTELLET I GORNAL
MEST. SOBRE VIA	Salvador	Rodrigo Revilla	xarel.lo	GELIDA
MEST. SOTA C.	Julio	Martí Mejía	xarel.lo	GELIDA
LA SERRA - XAREL.LO	Roi	Pascual Rosales	xarel.lo	PIERA
MAS PARDAL	Maria	Céspedes Rodríguez	xarel.lo	VILAFRANCA PENEDES

...

20.- Obtener un listado de los viticultores (con nombre y apellidos) que tienen alguna parcela en la población de GELIDA.

En este caso, las tablas implicadas son viticultor, parcela y población. Vinculando las tablas con los atributos correspondientes en cada caso obtenemos el comando:

```
SELECT  viticultor.nombre, viticultor.apellidos
FROM viticultor, parcela, poblacion
WHERE parcela.viticultor_id=viticultor.id AND parcela.poblaciOn_id=poblacion.id AND
poblacion.poblacion='GELIDA';
```

Con este comando un viticultor que tenga N parcelas en GELIDA, aparecerá N veces en el listado.

Si no deseamos que aparezca ningún viticultor repetido, bastará con añadir la palabra reservada DISTINCT

```
SELECT  DISTINCT viticultor.nombre, viticultor.apellidos
FROM viticultor, parcela, poblacion
WHERE parcela.viticultor_id=viticultor.id AND parcela.poblaciOn_id=poblacion.id AND
poblacion.poblacion='GELIDA';
```

21.- Obtener un listado con el número de parcelas de cada viticultor donde aparezca el nombre y apellidos de los viticultores.

```
SELECT  count(*), viticultor.nombre, apellidos
FROM parcela, viticultor
WHERE viticultor_id=viticultor.id
GROUP BY viticultor.nombre, apellidos ORDER BY count(*) DESC;
```

En esta ocasión hemos agrupado (GROUP BY) utilizando las columnas 'nombre' y 'apellidos' de la tabla viticultor. Podríamos haber ordenado únicamente por el nombre pero entonces no podríamos incluir la columna apellidos en la selección. Si intentamos ejecutar el siguiente comando:

```
SELECT  count(*), viticultor.nombre, apellidos
FROM parcela, viticultor
WHERE viticultor_id=viticultor.id
GROUP BY viticultor.nombre
ORDER BY count(*) DESC;
```

obtendremos el error

ERROR: la columna «viticultor.apellidos» debe aparecer en la cláusula GROUP BY o ser usada en una función de agregación

Si agrupamos las filas únicamente por el nombre del viticultor entonces no podemos solicitar también el apellido de cada viticultor. No podemos, por un lado, indicar que se unan (en una misma fila) todos los 'Juanes' y después pedir que para cada Juan se muestre también el apellido. Sencillamente no es posible.

Subconsultas o consultas anidadas

Imaginad que necesitamos un listado con todas las parcelas, de todos los viticultores, plantadas en años posteriores al año de plantación de la cepa más antigua del viticultor con id igual a 84211.

Como es habitual en el lenguaje SQL existen varias soluciones para dar respuesta a nuestras necesidades. El caso más sencillo consiste en ejecutar un comando para obtener el año de plantación de la cepa más antigua para el viticultor con id igual a 84211. Utilizando la función de agregado MIN construimos el siguiente comando:

```
SELECT  min(plantado_en) FROM parcela WHERE viticultor_id=84211;
```

Que nos devuelve el mínimo año (el año más antiguo) de las parcelas del viticultor correspondiente.

min integer
1973

Ahora que conocemos el año de la plantación más antigua podemos construir un nuevo comando para obtener el resultado deseado

```
SELECT  * FROM parcela WHERE plantado_en > 1973;
```

Sin embargo podemos obtener el mismo resultado con una sola consulta que incluya las dos consultas anteriores. Únicamente debemos reemplazar el año (1973) por la consulta que hemos utilizado para obtener ese año.

```
SELECT  *
FROM parcela
WHERE plantado_en > (
                        SELECT  min(plantado_en) FROM parcela
                        WHERE viticultor_id=84211
                    );
```

Es importante que la expresión subordinada (la que contiene la función de agregado MIN) esté indicada entre paréntesis y devuelva un único valor (una única columna y una única fila). En caso contrario la expresión global devolverá un error.

22.- ¿Qué variedades de uva tienen las cepas más antiguas?

```
SELECT  variedad FROM variedad, parcela
WHERE parcela.variedad_id = variedad.id AND plantado_en = (
                                        SELECT  min(plantado_en) FROM parcela
                                        );
```

Si ejecutamos únicamente la consulta subordinada obtendremos el año de las cepas más antiguas (1939). Lógicamente las cepas de ese año pueden ser de variedades distintas. Concretamente hay dos parcelas de la variedad xarel·lo y una de la variedad parellada.

Si queremos que no aparezcan variedades repetidas, utilizaremos una vez más, la palabra reservada DISTINCT.

```
SELECT  distinct variedad
FROM variedad, parcela
WHERE parcela.variedad_id = variedad.id AND plantado_en = (
                                SELECT  min(plantado_en) FROM parcela
                                );
```

23.- Parcelas con una superficie mayor que la parcela más antigua de la variedad parellada.

```
SELECT  * FROM parcela WHERE superficie >
        (
                SELECT  superficie
                FROM parcela, variedad
                WHERE variedad_id=variedad.id AND variedad='parellada' AND plantado_en =
                                (
                                SELECT  min(plantado_en)
                                FROM parcela, variedad
                                WHERE variedad_id=variedad.id AND variedad='parellada'
                                )
                );
```

En esta ocasión tenemos dos consultas subordinadas. La consulta más interna (la primera que se ejecuta) obtiene el año más antiguo (min) de la parcela con variedad parellada. Esta consulta vincula dos tablas: parcela y variedad pues deseamos saber el año de la cepa más antigua para la variedad parellada.

La consulta inmediatamente superior también vincula las tablas parcela y variedad ya que deseamos obtener la superficie de la parcela con la variedad parellada plantada en el año devuelto por la primera subconsulta. Si no vinculáramos las dos tablas obtendríamos las superficies de todas las parcelas (independientemente de cual fuera su variedad) plantadas en el mismo año que la cepa más antigua de la variedad parellada.

Finalmente solo necesitamos una consulta sencilla para obtener las parcelas cuya superficie sea mayor a la superficie devuelta por la segunda consulta subordinada.

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