**Actividad Práctica 6: Nubes de puntos. Análisis de densidades y visualización**

**INTRODUCCIÓN**.  
  
En este ejercicio vamos a utilizar datos libres procedentes de [OpenStreetMap](http://www.openstreetmap.org/)y [Flickr](https://www.flickr.com/) para detectar cuáles son las zonas más fotografiadas de la ciudad de Barcelona. La representación final de los datos se va a llevar a cabo utilizando el SIG de escritorio OpenJUMP.  
  
La delimitación de la ciudad de Barcelona la vamos a obtener de la cartografía disponible en OpenStreetMap, mientras que la información relativa a las fotografías de FLICKR ya se encuentra disponible en el fichero [flickr.zip](http://www.sigte.udg.edu/formasig/moodle/pluginfile.php/3324/mod_page/content/17/flickr.zip)que ha sido generado, previamente, a partir de la API de Flickr.   
  
Para saber más sobre la API de flickr podéis dirigiros a este enlace <https://www.flickr.com/services/api/>  
  
El fichero flickr.txt, contiene para cada fotografía, los siguientes atributos:  
  
**Usuario**: Identificador del usuario que ha tomado la fotografía  
**Foto\_id**: Identificador único de la foto  
**Latitud**: Latitud de la localización en la que fue tomada la fotografía  
**Longitud**: Longitud de la localización en la que fue tomada la fotografía  
**Fecha**: Fecha en la que fue tomada la fotografía.  
  
  
**EJERCICIO**  
  
1.- **Base de datos con capacidad Espacial**.

En primer lugar debemos generar una nueva base de datos con capacidad espacial a la que vamos a llamar **fotos\_barcelona**. En la guía de usuario de pgAdmin se dan todos los detalles necesarios para generar una base de datos espacial.  
  
2.- **Importación datos tabulados**.

Una vez creada la base de datos, vamos a crear una tabla llamada **flickr** donde posteriormente importaremos los datos contenidos en el fichero[**flickr.zip**](http://www.sigte.udg.edu/formasig/moodle/pluginfile.php/3324/mod_page/content/17/flickr.zip)  
  
CREATE TABLE flickr(  
usuario varchar(15),  
foto\_id varchar(15),  
latitud float,  
longitud float,  
fecha timestamp);  
  
Donde los campos *usuario*, *foto\_id* se definen como columnas de tipo VARCHAR y una longitud de 15 caracteres. Las columna *latitud* y *longitud* se definen como valores numéricos de tipo float (números reales que pueden incluir decimales). Finalmente la columna *fecha* se define como un tipo de datos timestamp para permitir almacenar valores de tiempo que incluyen tanto la fecha como la hora.  
  
Para la importación de los datos contenidos en el fichero **flickr.txt** vamos a utilizar el comando COPY. En nuestro caso, el fichero flickr.txt se localiza en la carpeta C:\datos\flickr.txt  
  
COPY flickr FROM 'C:\datos\flickr.txt';  
  
y para comprobar que la tabla flickr contiene esos datos, ejecutamos el comando:  
  
SELECT \* FROM flickr;  
  
3.-**Importación de la delimitación geográfica de la ciudad de Barcelona**.

En primer lugar vamos a crear la tabla **barcelona** donde almacenar la geometría de la ciudad.  
  
CREATE TABLE barcelona(  
geom geometry);  
  
Como podemos apreciar, la tabla solo contiene una columna llamada *geom* de tipo geometry.  
  
Para obtener la delimitación geográfica de la ciudad vamos a acceder al sitio web de OpenStreetMap [http://www.openstreetmap.org](http://www.openstreetmap.org/) desde donde vamos a buscar el topónimo *Barcelona*.  
  
  
  
La imagen anterior muestra el resultado de la búsqueda de 'Barcelona'. Entre los distintos resultados que aparecen, vamos a pinchar sobre el primero ya que es el que hace referencia a la ciudad. Otros resultados hacen referencia al límite de provincia o a otras ciudades del mismo nombre.  
  
Tras pinchar sobre el resultado accedemos a otra pantalla como la que se aprecia a continuación:

En la parte izquierda de la imagen aparecen algunas de las propiedades del objeto 'Barcelona'. Para la realización de este ejercicio, solo es necesario apuntar el valor numérico de la Relación Barcelona. En este caso**347950**.   
  
Este valor es el identificador (único) que tiene la relación 'Barcelona' dentro de la base de datos de OpenStreetMap y es neceario para poder obtener la geometría de la ciudad en formato Well Known Text (WKT). Este formato es directament importable a una base de datos espacial PostgreSQL/PostGIS.  
  
Una vez conocemos el valor de la relación 'Barcelona', accedemos al sitio <http://polygons.openstreetmap.fr/> . Llegados a este punto, es el momento de indicar el valor de la relación cuya geometría queremos obtener en formato WKT y pulsar sobre 'Submit Query'.

Ya casi lo tenemos. La siguiente pantalla nos ofrece varias posibilidades en cuanto al formato que queremos obtener sobre la geometría (relación) indicada.   
  
  
Seleccionamos el formato WKT y, ahora sí, nos aparece finalmente la geometría de la ciudad de Barcelona.   
  
  
  
¡Pero todavía queda la importación a la base de datos!  
  
Para importar la geometría vamos a utilizar la función St\_GeomFromEWKT(EWKT) que nos permite generar una geometría a partir de sus coordenadas en formato Extended WKT. Este formato extendido permite indicar también el sistema de referencia o SRID.  
  
  
Desde pgAdmin ejecutamos el comando:  
  
INSERT INTO barcelona (geom) VALUES (  
St\_GeomFromEWKT(  
'  
srid=4326;MULTIPOLYGON(((-5.698922 39.5269383,-5.6984672 39.5272101,-5.6978908 39.527646 ........................)))  
', 4326);  
  
Por razones de espacio se han simplificado las coordenadas de la geometría.  
  
  
Tal y como se aprecia en la siguiente imagen, es probable (según la versión de pgAdmin que estemos utilizando) que las coordenadas de la geometría no se muestren en pantalla apareciendo simplemente como un espacio en blanco. 

En favor de pgAdmin, hay que tener presente que este software no fue diseñado para trabajar con objetos geométricos por lo que en ocasiones las geometrías no se visualizan en pantalla a pesar de estar “ahí”. Se trata sin duda de un 'bug' de pgAdmin.   
  
A pesar de que las coordenadas no se muestren, podemos ejecutar el comando sin problemas. Eso sí, debemos asegurarnos, incluso sin verlo en pantalla, que el comando SQL sea correcto.  
  
4.- **Visualizar el contenido de la tabla 'barcelona' en OpenJump**.

En el material del curso se explica cómo hacerlo.

5.- **Visualizar el contenido de la tabla flickr**.  
  
Llegados a este punto, es necesario crear una nueva columna en la tabla **flickr** de tipo geometry ya que, hasta el momento, esta tabla no contiene ninguna columna que se pueda mostrar gráficamente en un SIG de escritorio como OpenJUMP . En esta nueva columna vamos a almacenar un objeto espacial que represente la localización en que fue tomada cada una de las fotos. Esta geometría de tipo POINT se creará a partir de los valores almacenados en las columnas latitud y longitud.  
  
Para crear una nueva columna de tipo geometry vamos a utilizar el siguiente comando:  
  
ALTER TABLE flickr ADD COLUMN geom geometry;  
  
Podemos comprobar que se ha añadido una nueva columna mediante el comando   
  
SELECT \* FROM flickr;

Para crear la geometría de tipo POINT vamos a utilizar el comando UPDATE para moficiar el contenido de la columna que acabamos de generar y la función espacial St\_MakePoint(x,y) para crear la geometría de tipo POINT a partir de las columnas latitud y longitud  
  
UPDATE flickr SET geom=St\_MakePoint(longitud, latitud);  
  
Donde la longitud se corresponde con el eje de las X y la latitud con el eje de las Y.  
  
Llegados a este punto, acabamos de generar unas geometrías de tipo POINT pero todavía no hemos indicado cuál es el sistema de referencia (SRID) de esas geometrías. Por lo tanto vamos a ejecutar de nuevo el comando UPDATE con la función St\_SetSRID(geometria, SRID) para indicar el sistema de referencia de las geometrías que acabamos de crear.  
  
UPDATE flickr SET geom=St\_SetSRID(geom, 4326);  
  
Lógicamente podríamos ejecutar las dos sentencias de una sola vez mediante el comando:  
  
UPDATE flickr SET geom=St\_SetSRID(St\_MakePoint(longitud, latitud), 4326);  
  
Ahora sí, podemos visualizar el contenido de la tabla flickr desde OpenJUMP  
  
  
  
6.- **Fotografías tomadas en Barcelona**.  
  
En esta ocasión vamos a hacer un **JOIN espacial** entre la geometría de tipo POLYGON de la tabla Barcelona y las geometrías de tipo POINT de la tabla flickr. De este modo vamos a seleccionar únicamente las fotografías tomadas dentro de la ciudad. Tenemos, pues, distintas posibilidades espaciales:   
  
St\_Intersects  
  
SELECT flickr.\* FROM flickr , barcelona WHERE St\_Intersects(flickr.geom, barcelona.geom);  
  
St\_WithIn  
  
SELECT flickr.\* FROM flickr , barcelona WHERE St\_WithIn(flickr.geom, barcelona.geom);  
  
St\_Contains  
  
SELECT flickr.\* FROM flickr , barcelona WHERE St\_Contains(barcelona.geom, flickr.geom);  
  
Donde St\_WithIn i St\_Contains son equivalentes con el orden de los parámetros invertido. A está dentro de B, si B contiene A. St\_Within(A,B)=St\_Contains(B,A)  
  
  
Con el objetivo de simplificar los comandos SQL que vendrán a continuación, vamos a crear una nueva tabla **fotos\_barcelona**.  
  
CREATE TABLE fotos\_barcelona AS   
SELECT flickr.\* FROM flickr , barcelona WHERE St\_Contains(barcelona.geom, flickr.geom);  
  
7.- **Cálculo de densidades de fotografías tomadas en Barcelona**.

Para el cálculo de densidades vamos a simular una malla sobre el territorio de Barcelona y vamos a contar cuántas fotografías han sido tomadas en cada una de las celdas que definen la malla. El siguiente paso será colorear cada celda de la malla, en función del número de fotografías que se hayan tomado dentro de ella. Se trata pues de dos pasos bien diferenciados. Para el primero, nos viene muy bien la función St\_SnapToGrid. Esta función nos permite agrupar geometrías dentro una malla determinada.

Veamos la función St\_SnapToGrid en detalle.  
  
St\_SnapToGrid(geom, origenX, origenY, tamañoX, tamañoY).  
  
Donde:  
  
**geom** es la geometría, cuyas coordenadas, se van a ajustar a una malla determinada.  
**origenX**: Coordenada de origen de la malla para el eje de las X  
**origenY**: Coordenada de origen de la malla para el eje de las Y  
**tamañoX**:Tamaño, sobre el eje de las X, de las celdas que definen la malla  
**tamañoY**:Tamaño, sobre el eje de las Y, de las celdas que definen la malla  
  
Gráficamente:

Como se aprecia en la imagen anterior, la función St\_SnapToGrid ajusta las coordenadas de una geometría dada (en la imagen geometrías de tipo POINT) a una malla definida por un punto de origen (0, 0) y un tamaño de celda concretos (X,Y).  
  
Aplicando la función St\_SnapToGrid a las fotografías de la tabla fotos\_barcelona obtenemos la siguiente sentencia SQL:  
  
SELECT St\_SnapToGrid(geom,0,0, 0.003,0.003) FROM fotos\_barcelona;

Dado que estamos utilizando coordenadas geográficas para latitudes alrededor de los 41 grados, un tamaño de celda de 0.003 grados se corresponde con una distancia del orden de 100 metros.   
  
Puedes comprobar el efecto de la función St\_SnapToGrid para tamaños de celda distintos.  
  
8.-**Generar las geometrías de las celdas**.

Antes de colorear las celdas de la malla, hay que crearlas. La función St\_SnapToGrid, utiliza una malla generada (y destruida) al vuelo, por lo que después de utilizar la función St\_SnapToGrid, la malla utilizada no está disponible. El próximo paso consiste, pues, en generar las geometrías de cada celda.  
  
La siguiente imagen muestra el resultado obtenido con la función St\_SnapToGrid. Como se aprecia en la imagen las geometrías de tipo POINT (puntos azules) se ajustan perfectametne a una malla. Todos los puntos que se encuentran dentro de cada celda de la malla (representada sobre fondo gris), serán ajustados a la misma coordenada de tipo POINT localizada en el centro de la celda  
  
  
  
Para generar todas las geometrías de las celdas, vamos primero a utilizar la función *distinct*para seleccionar las geometrías, sin repeticiones, retornadas por la función St\_SnapToGrid. Si obtenemos las geometrías sin repeticiones podremos luego obtener las geometrías de celda, también sin repeticiones.

Utilizando el siguiente comando, vamos a crear una nueva tabla **puntos\_snap** (sin repeticiones).

CREATE TABLE puntos\_snap AS  
SELECT distinct St\_SnapToGrid(geom,0,0,0.003,0.003) AS geom FROM fotos\_Barcelona ;  
  
Con la geometrías obtenidas anteriormente, vamos a utilizar la función St\_Expand para expandir esas geometrías. De modo gráfico y volviendo a la imagen anterior, vamos a expandir las geometrías de los puntos azules para obtener la superficie de influencia representada en fondo de color gris.  
  
Igual que hemos hecho con anterioridad, y para mayor comodidad, vamos a crear una nueva tabla **puntos\_expand**.  
  
CREATE TABLE puntos\_expand AS  
SELECT St\_Expand(geom, 0.0015) AS geom FROM puntos\_snap;  
  
El tamaño de la expansión (0.0015) se corresponde con la mitad del tamaño de la malla (0.003) ya que dicha extensión se lleva a cabo en todas direcciones (Norte, Sur, Este y Oeste).

9.- **Recuento de las fotografías tomadas en cada celda de la malla**.  
  
Veamos primero el comando necesario:  
  
SELECT count(\*) AS n, puntos\_expand.geom   
FROM fotos\_barcelona, puntos\_expand  
WHERE St\_Within(st\_snaptogrid(fotos\_Barcelona.geom,0,0,0.003, 0.003), puntos\_expand.geom)  
GROUP BY puntos\_Expand.geom   
ORDER BY count(\*) DESC;  
  
En la cláusula WHERE aparece un JOIN espacial entre dos geométricas mediante la función St\_WithIn.  
  
La primera de esas geometrías es el resultado de la función St\_SnapToGrid. Como hemos comentado anteriormente, primero ajustamos las fotografías de barcelona a una malla determinada, y con esas geometrías y las geometrías de las celdas (tabla puntos\_expand) efectuamos un JOIN espacial. Este JOIN espacial nos permitirá, junto con la función de agregado *count* y con la cláusula GROUP BY, proceder al recuento de cuántas fotos hay en cada celda.   
  
Lo último será representar el resultado de la consulta asignando, desde OpenJUMP, colores únicos para cada valor del recuento tal y como se aprecia en la siguiente imagen:  
  
  
  
**Opcional**.  
  
La consulta anterior devuelve 57 valores distintos para la columna 'n'. Esto significa que debemos asignar una paleta de 57 colores para poder representar gráficamente el resultado de la consulta. Como te puedes imaginar no se trata de una tarea fácil por lo que vamos a ver cómo podemos simplificar la representación agrupando los valores de 'n'. Es decir, vamos a establecer rangos de valores (0-5, 5-10, 10-30, 30-60, etc) para poder utilizar una paleta mucho más sencilla.  
  
En esta ocasión vamos a utilizar cláusula CASE que incluye PostgreSQL y cuya sintaxis es:  
  
CASE WHEN condicion THEN resultado  
[WHEN condicion THEN resultado]  
...  
[ELSE resultado]  
END  
  
Esta cláusula nos permite asignar distintos valores a una misma columna en función de expresiones condicionales. El valor asignado será el que cumpla la primera condición.  
  
Para el caso que nos ocupa vamos a crear una nueva columna 'rango' dependiente de los valores de la columna 'count(\*)'. Concretamente vamos a definir los siguientes rangos de valores: 0-5, 6-10, 11-40, 41-80, 81-120, 121-Infinito  
  
CASE   
WHEN count(\*)<6 THEN 1  
WHEN count(\*)<11 THEN 2  
WHEN count(\*)<41 THEN 3  
WHEN count(\*)<81 THEN 4  
WHEN count(\*)<121 THEN 5  
ELSE 6  
END AS rango  
  
Introduciendo el código anterior en nuestra consulta, obtenemos:  
  
SELECT CASE   
WHEN count(\*)<6 THEN 1   
WHEN count(\*)<11 THEN 2   
WHEN count(\*)<41 THEN 3   
WHEN count(\*)<81 THEN 4   
WHEN count(\*)<121 THEN 5   
ELSE 6   
END AS rango, count(\*) AS n, puntos\_expand.geom   
FROM fotos\_barcelona, puntos\_expand   
WHERE St\_Within(st\_snaptogrid(fotos\_Barcelona.geom,0,0,0.003, 0.003), puntos\_expand.geom)  
GROUP BY puntos\_Expand.geom   
ORDER BY count(\*) DESC;  
  
Ahora sí podemos representar el resultado de la consulta de manera más sencilla y legible.

Última modificación: jueves, 11 de diciembre de 2014, 08:51