

**CLASE
4**
Bases de datos espaciales

República Argentina
Ministerio de Educación
Programa Nacional Mapa Educativo

Curso de capacitación
Bases de datos espaciales

CLASE 4
Material de lectura. Versión 1

Temas de esta clase:

SQL espacial: análisis topológicos.
Índices espaciales.
Edición de información geográfica desde un SIG.
SQL: consultas de agregación.

Referencias:

A lo largo del documento encontraremos íconos y recuadros que requieren de una especial atención de los lectores:



ACTIVIDADES: son consignas de actividades para realizar la práctica con gvSIG y PGAdmin acompañando la lectura. En la presente clase hay 4 actividades para resolver.



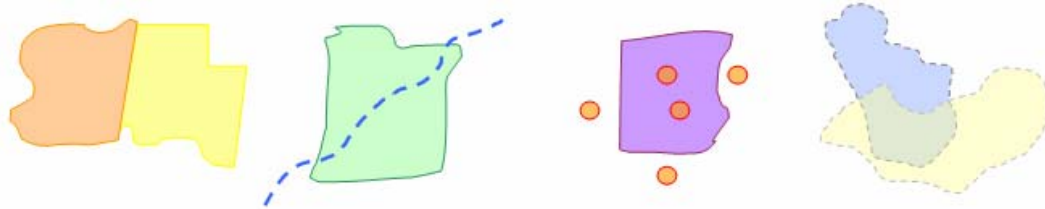
¡IMPORTANTE! Indica una actividad que no debe omitirse para poder desarrollar correctamente la práctica de la clase.

1 SQL espacial. Análisis topológicos

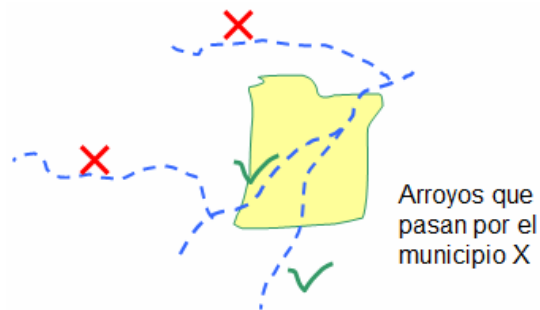
En esta clase veremos como emplear las herramientas que ofrece el lenguaje de consulta SQL del módulo PostGIS para realizar análisis basados en las relaciones topológicas entre los elementos de la información geográfica. Las relaciones topológicas entre los objetos espaciales son aquellas que no cambian cuando los datos son reproyectados, como la adyacencia entre elementos, la situación de estar adentro de, afuera de, tener su centro en, cruzarse con, intersectarse con, etc.. La topología se diferencia de otros atributos de los datos geográficos, como la forma o la longitud y la superficie que sí se modifican al cambiar de proyección.

Las consultas que veremos a continuación devuelven una respuesta del tipo verdadero o falso, ya que se efectúan a manera de pregunta en función de la relación topológica. Por ejemplo

- ¿Los arroyos se intersectan con los límites del municipio X?
- ¿Las parcelas tienen su centro en el área afectada por la crecida de tal río?
- ¿Los barrios son adyacentes al área central?



Las respuestas recibidas para estas consultas son listados de elementos con un campo booleano con valores “verdadero” y “falso” (TRUE/FALSE) que indican si se cumple o no la relación topológica consultada.



Una vez que obtenemos una respuesta de este tipo (TRUE/FALSE) seguramente vamos a necesitar graficar aquellos elementos que cumplen (o los que no cumplen) la condición establecida, y para esto veremos que es más útil realizar con SQL una pregunta del tipo:

Seleccionar los arroyos cuyas geometrías se intersectan con los límites del municipio X

Seleccionar las parcelas que tienen su centroide dentro del polígono del área afectada por la crecida del río.

Seleccionar los barrios que **no** son adyacentes al barrio central

Esta vez, las respuestas recibidas son los elementos geométricos que cumplen la condición especificada y como tales, pueden ser visualizados desde un SIG.



1.1 Touch. Está en contacto.

La primera relación que veremos es “estar con contacto con”. Esta función devuelve el valor “verdadero” cuando las geometrías tienen en común al menos un

vértice en los bordes de sus geometrías y el resto de las geometrías no se intersecta. Esta relación se establece entre cualquier tipo de geometría, menos entre dos puntos.



Por ejemplo, esta es la relación que existe entre el polígono de un barrio y la línea de un eje de calle que pasa por sus límites, o entre un camino rural y una parcela, o entre el límite de un país y los hitos fronterizos.

Si queremos conocer cuales son las calles que rodean al barrio del centro de la ciudad de Salta, la consulta será:

```
SELECT * FROM calles_salta where
ST_TOUCHES (the_geom, "BARRIO CENTRO") IS TRUE;
```



IMPORTANTE.

Hemos incorporado un nuevo esquema a la base de datos. El esquema "salta" contiene los datos referidos a la Ciudad de Salta. La forma de referirnos a las tablas que se encuentran en un esquema diferente a "public", es colocando el nombre del esquema antes del nombre de la tabla:

```
SELECT * FROM salto.calles;
SELECT SRID(the_geom) FROM salto.salud;
```

Estamos solicitando todos los campos de la tabla de calles para aquellos elementos en donde el análisis del contacto de su geometría con la del barrio Centro devuelva un valor verdadero. Ahora vamos a reemplazar "BARRIO CENTRO" por la consulta que nos devuelve la geometría del barrio:

```
SELECT * FROM salto.calles where
ST_TOUCHES (the_geom, (SELECT the_geom FROM salto.barrios WHERE
nom_barrio = 'CENTRO')) IS TRUE;
```

Ahora sí, la consulta está completa, y al correrla Postgres nos devuelve la tabla de calles filtrada para los elementos que cumplen la condición de estar en contacto con el barrio.

Editor SQL Constructor Gráfico de Consultas

Borrar Eliminar Todos

```
SELECT * FROM salta.calles where
ST_TOUCHES (the_geom, (SELECT the_geom FROM salta.barrios WHERE nom_barrio = 'CENTRO')) IS TRUE;
```

Panel de Salida

Salida de datos Comentar Mensajes Historial

	gid	fnode	tnode	lpoly	rpoly	length	double	precision	e1161	e1161	codigo	ancho	anchomed	double	precision	nombre	nom_nor	ladoi	ladod	desdei	desded	has
	integer	integer	integer	integer	integer				integer	integer	integer	integer	double	double		text	text	integer	integer	integer	integer	integer
1	4898	2548	2498	1537	1560	76.999132	0	0	8340	20	10					MITRE	MITRE	0	0	1051	1052	109
2	4921	2564	2532	1576	1472	63.031738	0	0	8905	30	15					AV URUGUAY	URUGUAY	0	0	1151	0	119
3	4922	2560	2564	1576	1591	119.854339	0	0	1730	20	10					PJE G BEE BEECH G P	0	0	1	2	99	
4	4930	2570	2548	1585	1560	43.815665	0	0	8340	20	10					MITRE	MITRE	0	0	1001	1002	104
5	4952	2587	2570	1600	1560	23.086793	0	0	8340	20	10					MITRE	MITRE	0	0	0	0	0
6	4960	2592	2506	1560	1562	145.773797	0	0	8365	20	10					ZUVIRIA	ZUVIRIA	0	0	1001	1002	109
7	4961	2592	2587	1609	1560	124.326184	0	0	1735	20	10					AMEGHIN	AMEGHINO	0	0	501	502	599
8	4969	2597	2553	1562	1563	71.344236	0	0	8390	20	10					DEAN FUJ	DEAN FUJ	0	0	1001	1002	104
9	4970	2597	2592	1614	1562	117.209215	0	0	1735	20	10					AMEGHIN	AMEGHINO	0	0	401	402	499
10	4978	2602	2514	1563	1565	144.779142	0	0	8420	20	10					PUEYRREI	PUEYRREDI	0	0	1001	1002	109
11	4979	2602	2597	1618	1563	123.146255	0	0	1735	20	10					AMEGHIN	AMEGHINO	0	0	301	302	399
12	4987	2608	2518	1565	1569	143.589693	0	0	8530	20	10					VICENTE I LOPEZ	VICI	0	0	1001	1002	109
13	4988	2608	2602	1622	1565	122.147452	0	0	1735	20	10					AMEGHIN	AMEGHINO	0	0	201	202	299

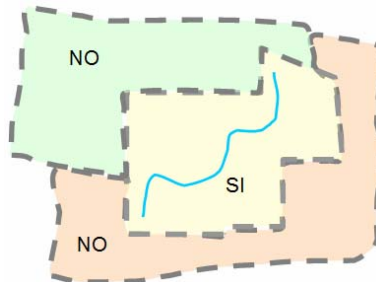
OK. Unix Lin 3 Col 1 Car 131 129 filas. 1681 ms

Cuando solicitamos que una condición sea verdadera, SQL nos permite omitir colocar "IS TRUE" en la consulta. Es por esto que la consulta que vimos más arriba, también se puede escribir sin escribir "IS TRUE" ya que SQL interpreta que queremos que la condición devuelva un valor verdadero:

```
SELECT * FROM salta.calles where
ST_TOUCHES (the_geom, (SELECT the_geom FROM salta.barrios WHERE
nom_barrio = 'CENTRO'));
```

1.2 Contiene

Esta función devuelve un valor verdadero si la geometría A contiene completamente a la geometría B. Esto implica que todos los puntos de B se encuentran dentro de A.



Ejemplos de esta relación son los que se establecen entre las zonas de supervisión y las escuelas, o un determinado uso del suelo y las parcelas catastrales, o una cuenca hidrográfica y sus cursos de agua.

Vamos a averiguar cuál es el barrio que contiene al Centro de Salud n° 3 de Salta.

```
SELECT nom_barrio FROM salta.barrios WHERE ST_CONTAINS
(the_geom, (SELECT the_geom FROM salta.salud WHERE centro = 3));
```

Editor SQL Constructor Gráfico de Consultas

```
SELECT nom_barrio FROM salta.barrios
WHERE ST_CONTAINS (the_geom, (SELECT the_geom FROM salta.salud WHERE centro = 3));
```

Panel de Salida

Salida de datos Comentar Mensajes Historial

	nom_barrio text
1	HERNANDO DE LERMA

En este caso, en “SELECT” solo colocamos el campo “nom_barrio”, y la respuesta nos devuelve el nombre del barrio que cumple con la condición.

1.3 Contiene propiamente

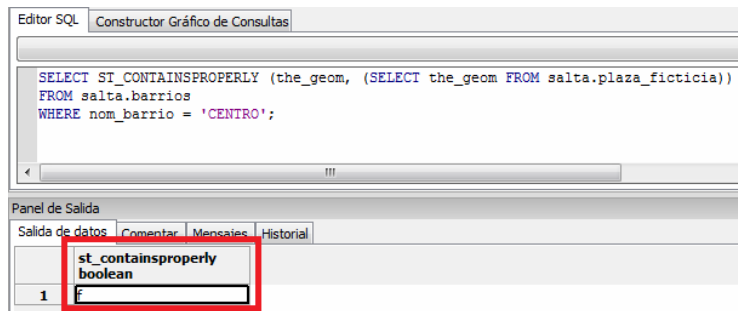
Una función similar a ST_CONTAINS es ST_CONTAINSPROPERLY, que significa “contiene propiamente”. Hace referencia a la relación de contención sin tener en cuenta el “boundary” o borde de la figura.



Por ejemplo, si existe un espacio verde que se encuentra dentro de un barrio, pero justo en su límite, la relación ST_CONTAINSPROPERLY entre el barrio y el espacio verde es falsa. De la misma manera, si tenemos un polígono A, este polígono se contiene a sí mismo (ST_CONTAINS es verdadero), pero no se contiene propiamente (ST_CONTAINSPROPERLY es falso).

Un ejemplo sería una plaza cuyo borde coincide con el límite de un barrio. Este barrio no contiene propiamente a dicha plaza.

```
SELECT ST_CONTAINSPROPERLY (the_geom, (SELECT the_geom FROM
salta.plaza_ficticia))
FROM salta.barrios
WHERE nom_barrio = 'CENTRO';
```



La respuesta obtenida ante la consulta es el valor “falso”.

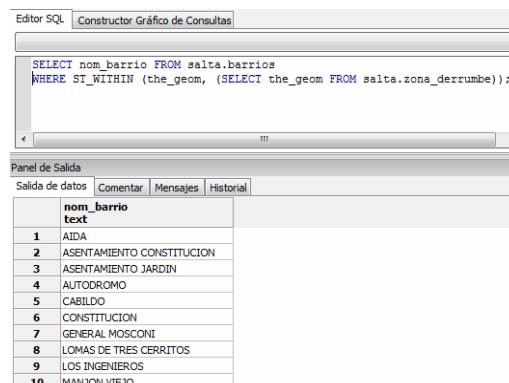
1.4 Within. Está contenido por.

Within es la función inversa a ST_CONTAINS, ya que analiza si A está contenido en B, y suele ser aplicada con mayor frecuencia. Podemos invertir los ejemplos anteriores para pensar la aplicación que se le puede dar a la función: cuáles son las escuelas que pertenecen a una zona de supervisión, o las viviendas ubicadas en zona industrial, los recorridos de transporte que no salen del centro de la ciudad, etc.



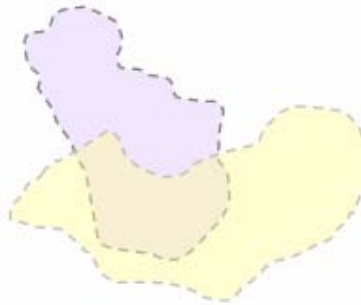
Para practicar, analicemos cuales son los barrios contenidos en las zonas con riesgo de derrumbe de la ciudad de Salta. Recordemos que solo estamos pidiendo aquellos barrios que están completamente contenidos por el polígono de la zona de riesgo. Más adelante veremos otras funciones para obtener los polígonos que están parcialmente afectados.

```
SELECT nom_barrio FROM salta.barrios WHERE ST_WITHIN (the_geom,
(SELECT the_geom FROM salta.zona_derrumbe));
```



1.5 Overlaps. Solapa.

Overlaps analiza si dos geometrías se solapan pero sin llegar a estar contenida una por otra. Solo se analiza el solapamiento entre elementos con el mismo tipo de geometría (punto con punto, línea con línea y polígono con polígono).



Volviendo al ejemplo del análisis de los barrios y las zonas inundadas, podemos realizar la consulta que nos devuelve los barrios cuya geometría se solapa con el polígono del área inundable.

```
SELECT nom_barrio FROM salta.barrios WHERE ST_OVERLAPS
(the_geom, (SELECT the_geom FROM salta.zona_derrumbe));
```

Editor SQL Constructor Gráfico de Consultas

```
SELECT nom_barrio FROM salta.barrios WHERE ST_OVERLAPS
(the_geom, (SELECT the_geom FROM salta.zona_derrumbe));
```

Panel de Salida

Salida de datos Comentar Mensajes Historial

	nom_barrio text
1	BELGRANO
2	CENTRO
3	COSMOPOL

Unix Lín 2 Col 1 Car 56 16 filas. 61 ms

1.6 Crosses. Se cruza.

Devuelve el valor verdadero si las geometrías se cruzan, es decir, si tiene al menos un punto en común, pero no todos. Esta relación solo se puede establecer entre líneas y polígonos.

Un claro ejemplo es saber si dos caminos se cruzan entre sí. En el caso de la relación entre líneas y polígonos, un ejemplo es analizar si un recorrido de transporte pasa por un distrito comercial. Si el recorrido de dicho transporte se desarrolla completamente dentro del distrito, el análisis de ST_TOUCHES dará el valor falso como resultado, ya que la relación implica que la geometría no puede tener todos los puntos en común.

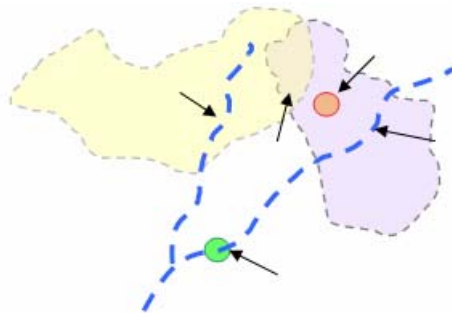
Veamos como ejemplo, cuales son las calles cruzadas por el canal "Esteco" de Salta.

```
SELECT * FROM salta.calles WHERE st_crosses (the_geom, (SELECT
the_geom FROM salta.hidrografia WHERE nombre = 'ESTECO'));
```



1.7 Intersects

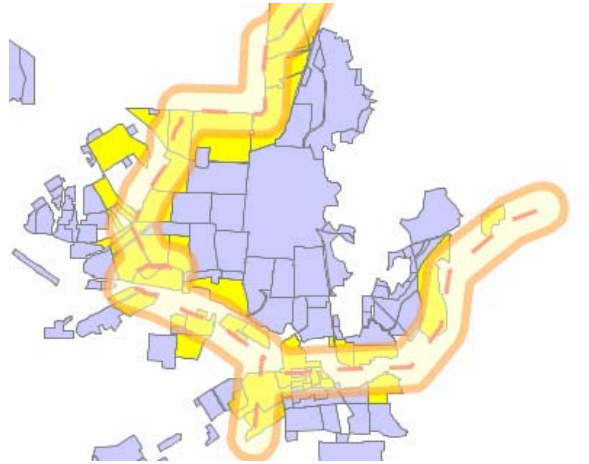
La relación Intersects está implicada en todas las funciones vistas hasta este momento. Esta relación se establece cuando dos geometrías comparten cualquier porción del espacio. Los tipos de geometrías que pueden intersectarse son líneas, puntos y polígonos.



Se puede analizar la relación de intersección entre cualquier clase de elementos del espacio, como caminos y zonas de peligro de incendio forestal, áreas de influencia de riego ambiental y asentamientos humanos, caminos y parajes, etc.

Usemos como ejemplo el análisis del área de influencia de 500 metros del tendido de alta tensión y los barrios de Salta.

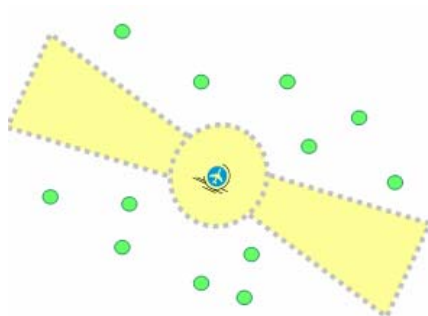
```
SELECT nom_barrio FROM salta.barrios WHERE ST_INTERSECTS
(the_geom, (SELECT BUFFER (the_geom, 500) FROM
salta.alta_tension));
```

En este ejemplo no nos interesa intersectar la geometría del tendido de alta tensión, sino el polígono de su área de influencia obtenido mediante la función “buffer”.

1.8 Disjoint

La función “ST_DISJOINT” devuelve el valor verdadero par las geometrías que no presenten ningún vértice en común.



Veamos, como ejemplo, el área de influencia de una línea de transporte que tiene buses con rampas y los domicilios de personas con discapacidad. Realizaremos el análisis con la función ST_DISJOINT y nos devolverá con valor verdadero aquellos domicilios que no se intersecten con el área servida por el transporte especial.

```
SELECT * FROM salta.pacientes WHERE ST_DISJOINT (the_geom,  
(SELECT buffer (the_geom,1000) FROM salta.transporte_especial));
```

Editor SQL Constructor Gráfico de Consultas

```
SELECT * FROM salta.pacientes WHERE ST_DISJOINT
(the_geom, (SELECT buffer (the_geom,1000) FROM salta.transporte_especial));
```

Panel de Salida

Salida de datos Comentar Mensajes Historial

	gid integer	the_geom geometry	paciente integer
1	1	0101000020A75600005A5A1AD43D1D4B41F0F0501696A95B41	1
2	2	0101000020A7560000D2D2923BC51B4B416969892302A95B41	2
3	3	0101000020A75600002D2D6DBA581D4B415A5A7A89A3A95B41	3
4	4	0101000020A75600001E1E5EB78C214B41D2D232E2D8AA5B41	4
5	5	0101000020A7560000F0F0B09DA7214B416969C914B0AB5B41	5
6	8	0101000020A7560000696929CAE62A4B41F0F050AD28AD5B41	8
7	9	0101000020A75600006969A915B02C4B41787858FAD7AC5B41	9
8	10	0101000020A75600006969A915B02C4B41B4B414AE1BAC5B41	10
9	11	0101000020A75600003C3C7CB0012B4B4178789854F3AB5B41	11
10	12	0101000020A75600003C3C7CB0012B4B419696762E95AB5B41	12
11	13	0101000020A75600007878B8AF0E2C4B413B3C1CFBCAAB5B41	13
12	14	0101000020A7560000E1E1A1C8002D4B413B3C1CFBCAAB5B41	14
13	15	0101000020A7560000A5A565C9F32B4B41A5A5456ED8AB5B41	15
14	16	0101000020A7560000FFFFBFFCBDB4B419696762E95AB5B41	16



ACTIVIDAD 1

- Restaurar el archivo “salta.backup”.
- Restaurar el archivo “tablas4.backup”
- Migrar la capa “areas_protegidas.shp” que tienen el sistema de referencia EPSG:22184, creando la tabla “areas_protegidas”.

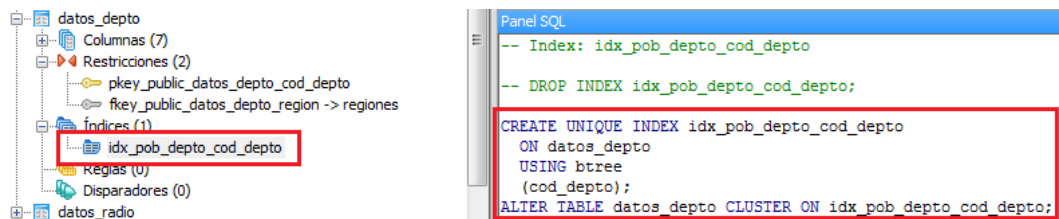
Escribir en el documento las consultas SQL que responden a las siguientes preguntas:

- ¿Cuáles son los Centros de Salud que se ubican en el barrio “CENTRO” de la ciudad de Salta?
- ¿Qué barrios atraviesa la línea de transporte especial?
- ¿La envolvente de 1000 metros de la línea de alta tensión se solapa con la zona de derrumbe?

2 Índices espaciales.

Los índices, en general, se utilizan en las bases de datos espaciales para lograr un acceso más rápido a los datos. Al igual que en cualquier sistema de recuperación de datos, se aplica el índice siguiendo el o los criterios de búsqueda más frecuentes. Por ejemplo, en una biblioteca el índice es probable que se encuentre organizado según criterio con el autor o el nombre del libro, y no por la editorial o la ciudad en que fue editado el libro. Si se trata de una tabla que contiene códigos que son muy utilizados para hacer “joins” o para realizar búsquedas, lo más conveniente es indexar la tabla utilizando el campo que contiene los códigos.

En el caso de la tabla de datos por departamento, si abrimos los objetos que dependen de esta tabla, podremos ver que tiene aplicado un índice llamado “idx_pob_depto_cod_depto”.



Si seleccionamos el índice en el árbol de objeto, veremos en el panel derecho la consulta SQL que lo crea:

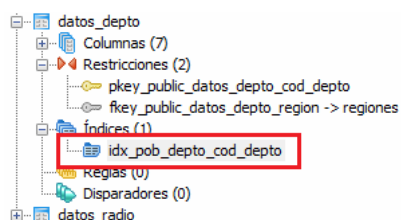
```
CREATE UNIQUE INDEX idx_pob_depto_cod_depto
ON datos_depto
USING btree (cod_depto);
```

Vemos que la consulta empieza con el comando “CREATE” y luego especifica que se trata de un nuevo índice. El comando “UNIQUE” es muy importante cuando los valores del campo indexado son únicos (no se repiten). Esto agiliza aún más las búsquedas.

Luego, la consulta de creación dice que el índice se aplica a la tabla “datos_depto”. Los comandos “USING btree” indican el método de acceso, que veremos a continuación y finalmente, entre paréntesis, señala el campo sobre el cual se aplicará el índice, en este caso “cod_depto”. Recordemos que cod_depto es el código único de departamento, que además de ser único, fue implementado por el INDEC para el uso de todo el Sistema Estadístico Nacional.

Es muy conveniente tener indexado este campo, ya que, como hemos visto en los ejercicios de las clases anteriores, el código es la forma más frecuente de acceso a estos datos y a todas las entidades geográficas normalizadas.

Para terminar vemos que agrega una segunda consulta en la definición del objeto índice (las consultas terminan al colocar un “punto y coma”):



```
Panel SQL
-- Index: idx_pob_depto_cod_depto
-- DROP INDEX idx_pob_depto_cod_depto;

CREATE UNIQUE INDEX idx_pob_depto_cod_depto
ON datos_depto
USING btree
(cod_depto);

ALTER TABLE datos_depto CLUSTER ON idx_pob_depto_cod_depto;
```

`ALTER TABLE datos_depto CLUSTER ON idx_pob_depto_cod_depto;`

En este caso se solicita que se altere la tabla “datos_depto” para que el almacenamiento físico en disco de los datos se adecue al índice creado anteriormente. Esto agrega aún más agilidad a la recuperación de los datos.

	nbi real	hogares integer	ipmh real	poblacion integer	cod_depto [PK] caracte	nom_depto text	region text
92	8	1317	30.6	40855	06469	LINCOLN	CENTRO
93	9	5411	30.6	16928	06476	LOBERIA	CENTRO
94	9.6	10010	35	32817	06483	LOBOS	CENTRO
95	14.2	164430	40.4	587795	06490	LOMAS DE ZAMORA	CENTRO
96	9.7	26178	34.2	90816	06497	LUJAN	CENTRO
97	12.1	4438	31.3	14798	06505	MAGDALENA	CENTRO
98	9.3	3196	33.1	10069	06511	MAIPU	CENTRO
99	19.8	72956	51.4	289798	06515	MALVINAS ARGENTINAS	CENTRO
100	10.6	5437	35.8	17728	06518	MAR CHIQUITA	CENTRO
101	19.4	10758	56.1	41651	06525	MARCOS PAZ	CENTRO
102	7.8	17543	28.8	58341	06532	MERCEDES	CENTRO
103	19.8	17543	28.8	58341	06539	MERLO	CENTRO
104	9.7	17543	28.8	58341	06547	MONTE	CENTRO
105	6	1791	22.3	5480	06553	MONTE HERMOSO	CENTRO
106	22	95538	59.6	379370	06560	MORENO	CENTRO
107	6.8	93980	22.9	305687	06568	MORON	CENTRO
108	9.3	4549	33.5	15711	06574	NAVARRO	CENTRO
109	8	27404	30	87664	06581	NECOCHEA	CENTRO
110	7.9	14657	27.7	45517	06588	9 DE JULIO	CENTRO
111	7	31583	29.3	101692	06595	OLAVARRIA	CENTRO
112	15.7	8457	37	27776	06602	PATAGONES	CENTRO
113	9.6	12383	34.7	38097	06609	PEHUAJO	CENTRO
114	6.9	1908	24.3	5977	06616	PIELERINT	CENTRO

clave primaria

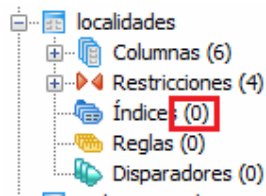
Los valores no se repiten

Todos los campos pueden ser indexados, pero siempre conviene mantener el criterio de indexar solo aquellos campos que son utilizados en distintas operaciones como la selección, joins, búsquedas, etc.

Cuando existe más de un índice en una tabla, Postgres analiza el pedido y en función de ello prioriza el uso de uno u otro índice existente.

Cuando se cuenta con grandes tablas de datos geométricos la respuesta cambia enormemente cuando se aplica el índice sobre el campo de la geometría. Por ejemplo, cuando visualizamos una tabla desde un SIG de escritorio o cualquier otro cliente, en realidad estamos haciendo permanentemente consultas de selección basándonos en la localización de los datos. Cuando nos movemos por una vista con datos de PostGIS, por detrás se ejecutan consultas de selección que nos devuelven los elementos que se intersectan con el área cubierta por la vista. Es por ello que es muy conveniente indexar las grandes tablas usando el campo geométrico.

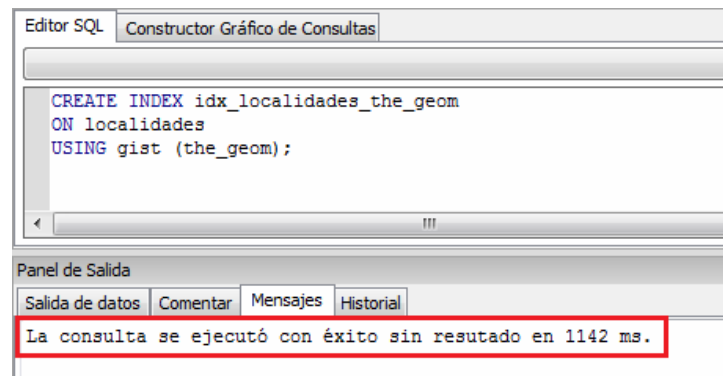
A modo de prueba podemos indexar el campo geométrico de la tabla de localidades. Primero comprobamos que no tiene índices creados.



Para crear el índice mediante lenguaje SQL, podemos correr la siguiente consulta:

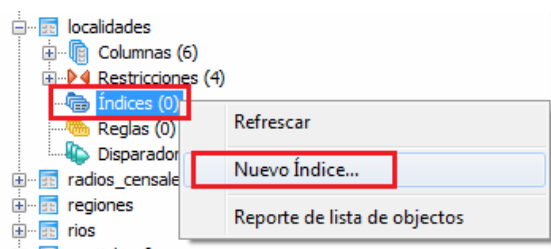
```
CREATE INDEX idx_localidades_the_geom
ON localidades
USING gist (the_geom);
```

En esta consulta ordenamos que se cree el índice llamado “idx_localidades_the_geom” en la tabla “localidades” sobre el campo “the_geom” usando el método de acceso “gist”.



El método de acceso a emplear en cada índice depende de la cantidad y tipo de datos a indexar, pero en líneas generales podemos decir que para los datos geométricos es conveniente utilizar el método de acceso “gist” ya que sirve para datos menos estructurados, mientras que para los otros datos, como los caracteres y los números, es conveniente usar “btree”.

Para crear un índice de utilizando las herramientas de PGAdmin, buscamos el objeto “Índices” en la tabla que queremos indexar, y luego hacemos clic con el botón derecho para luego seleccionar la opción “Nuevo índice” en el menú que se despliega.



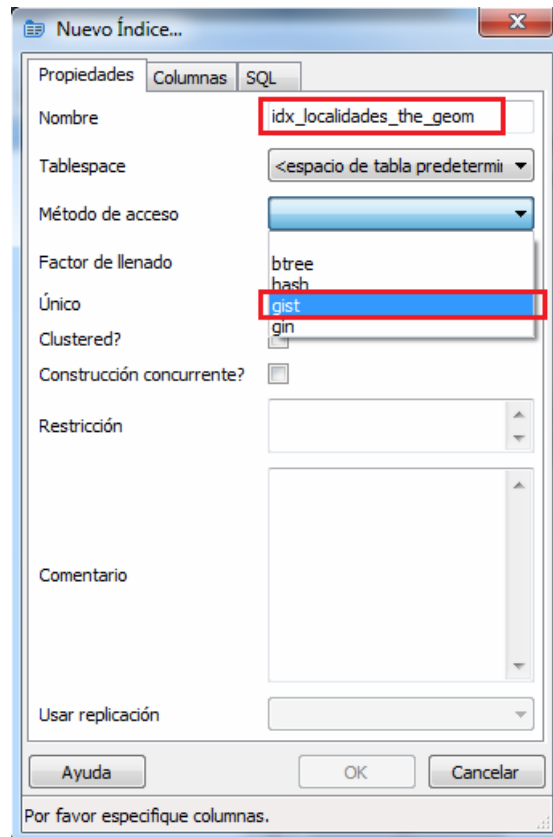
Allí, colocamos un nombre al índice, siguiendo la regla que se sugiere a continuación:

idx + tabla + campo

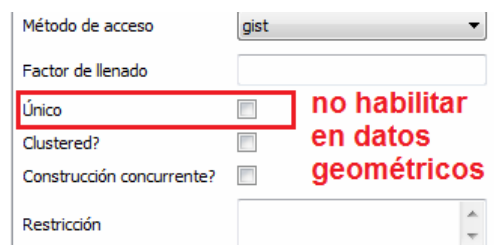
En este caso, como es el índice sobre el campo geométrico de la tabla de localidades, el nombre resultante será:

idx_localidades_the_geom

A continuación debemos seleccionar el método de acceso. Recordemos que si se trata de una columna geométrica utilizaremos el método “gist”.

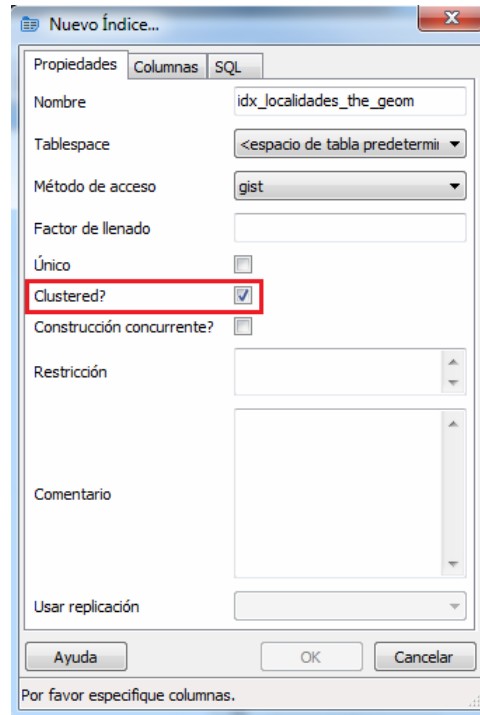


Luego, podemos activar la condición “único” para que el índice funcione sabiendo que cada dato aparece una única vez. Para poder utilizar esta condición es muy importante asegurarse que realmente nunca se almacenarán valores repetidos. En el caso de los campos geométricos no nos conviene habilitar esta opción, ya que se pueden dar casos en donde las geometrías se repiten. Pero cuando se trate de campos con un código único alfanumérico, es muy recomendable seleccionar esta opción, para asegurarnos de que no se colocarán valores incorrectos en el campo.

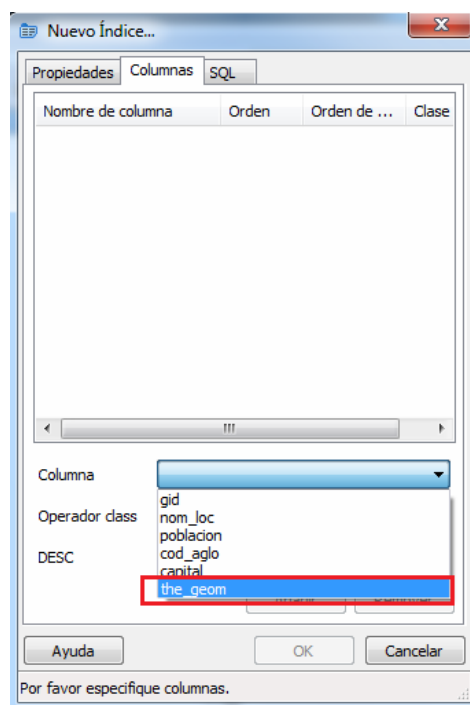


La opción “clustered” es fundamental para lograr un buen funcionamiento en tablas de gran cantidad de registros. Cuando habilitamos esta opción, el

almacenamiento físico de los datos en el disco se producirá acompañando la lógica del índice que estamos creando. Una tabla puede tener varios índices, pero solo se puede “clusterizar” la tabla en función de un solo índice. En el caso de las grandes tablas con información geométrica que se accederán desde visualizadores y SIG de escritorio, es recomendable clusterizar la tabla usando el índice sobre el campo geométrico.



Finalmente, en la solapa “Columnas”, seleccionamos la columna sobre la cual se aplicará el índice (the_geom). Y para terminar colocamos OK.



La diferencia entre las respuestas a una consulta según la existencia de índices se puede comprobar con el tiempo de respuesta que aparece en el sector inferior derecho de la ventana del constructor de consultas arbitrarias.

Editor SQL Constructor Gráfico de Consultas

Borrar Eliminar Todos

```
select * from localidades where intersects(the_geom ,
(select transform(the_geom,22183) from provincias where cod_prov = '82'))
```

Panel de Salida

Salida de datos Comentar Mensajes Historial

	gid integer	nom_loc text	poblacion integer	cod_aglo text	capital text	the_geom geometry
1	2147	LOS AMOR	1131	1335		0101000020
2	2148	CA-ADA	574	4102		0101000020
3	2149	GOLONDR	399	4120		0101000020
4	2150	TARTAGAL	1593	1053		0101000020
5	2151	INTIYACO	1071	1314		0101000020

OK. Unix Lin 3 Col 1 Car 129 380 filas. 118 ms

Editor SQL Constructor Gráfico de Consultas

Borrar Eliminar Todos

```
select * from localidades where intersects(the_geom ,
(select transform(the_geom,22183) from provincias where cod_prov = '82'))
```

Panel de Salida

Salida de datos Comentar Mensajes Historial

	gid integer	nom_loc text	poblacion integer	cod_aglo text	capital text	the_geom geometry
1	2147	LOS AMOR	1131	1335		0101000020
2	2148	CA-ADA	574	4102		0101000020
3	2149	GOLONDR	399	4120		0101000020
4	2150	TARTAGAL	1593	1053		0101000020
5	2151	INTIYACO	1071	1314		0101000020

OK. Unix Lin 4 Col 1 Car 130 380 filas. 116 ms

En esta pequeña tabla, la indexación mejoró levemente la respuesta ante una consulta con implicancia geométrica. En el caso de grandes tablas, la mejora es muy evidente.



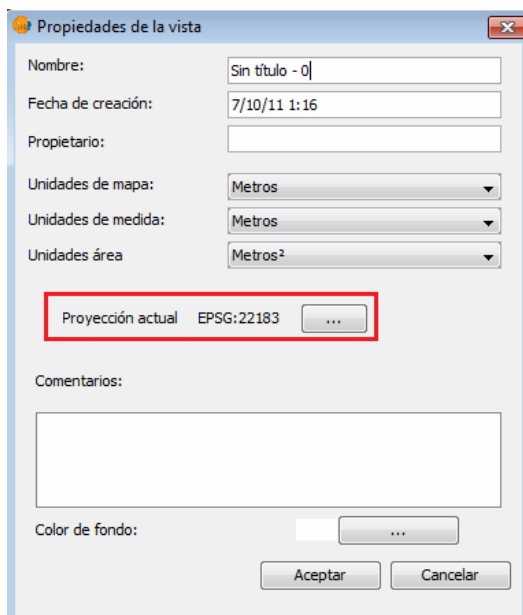
ACTIVIDAD 2

- Crear un índice UNICO para la tabla "datos_radio" sobre el campo "cod_rad". (Generarlo a través de la interfaz de PGAdmin o con SQL).
- Procurar que el almacenamiento físico de los datos se adecuen a este índice (CLUSTER)
- Pegar en el documento las consultas SQL que se obtienen al seleccionar el nuevo índice desde el árbol de objetos.

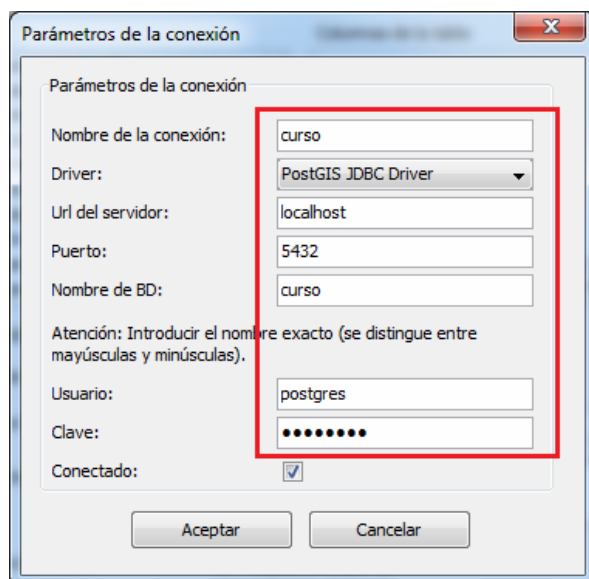
3 Edición de información geográfica desde un SIG

Ahora procederemos a realizar modificaciones de las tablas con datos geométricos, desde gvSIG.

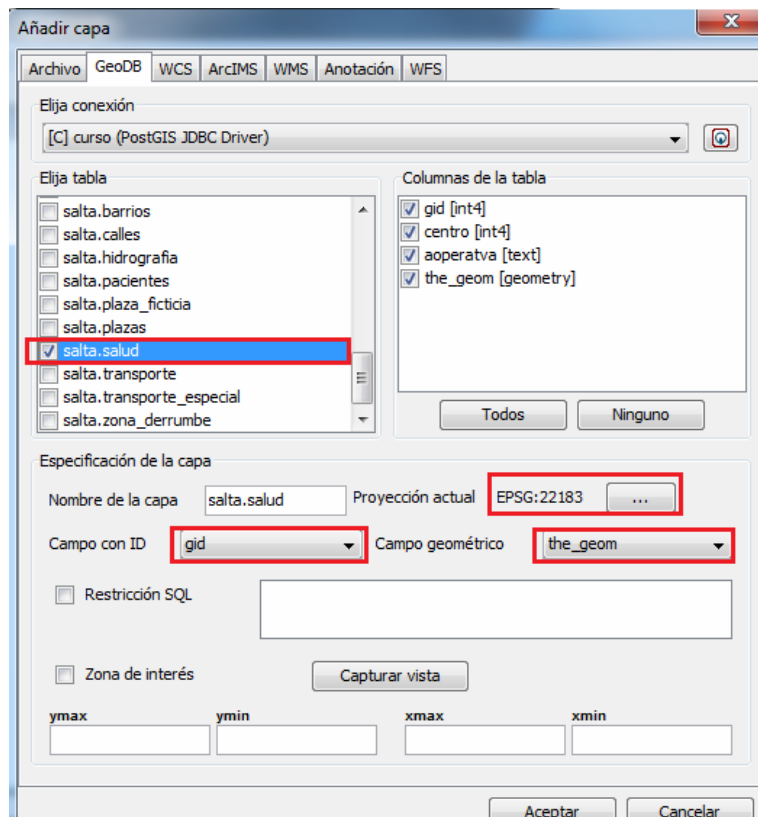
Accedemos, en primer lugar, a la base de datos desde la herramienta de añadir capas, usando la solapa "GeoDB". Antes de agregar una capa, debemos configurar el sistema de referencia de la vista, desde el menú "Vista, Propiedades". En este caso usaremos datos de Salta, por lo cual colocaremos el EPSG:22183.



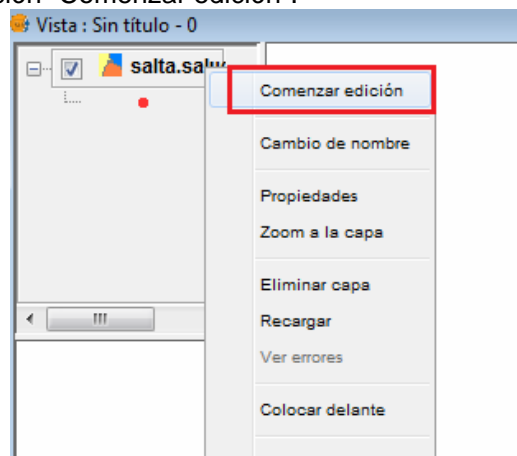
Recordemos que siempre que nos conectamos a una base de datos es necesario colocar los parámetros de conexión, o completarlos en caso de que ya estén almacenados.



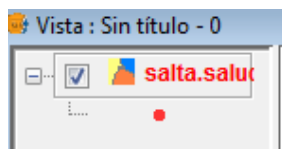
Visualizamos el listado de capas y seleccionamos la capa a añadir en la vista. En este caso modificaremos los puntos de los Centros de Salud. Comprobamos que el campo de la clave primaria (gid) y el campo geométrico (the_geom) sean reconocidos por gvSIG, y luego añadimos la capa.



Una vez que tenemos la capa agregada en la vista, la seleccionamos en la tabla de contenidos de la izquierda, y hacemos clic sobre ella con el botón derecho. Seleccionamos la opción "Comenzar edición".



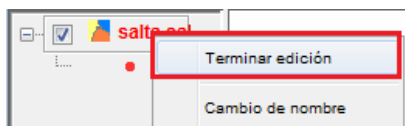
Vemos que la capa se vuelve de color rojo en la tabla de contenidos.



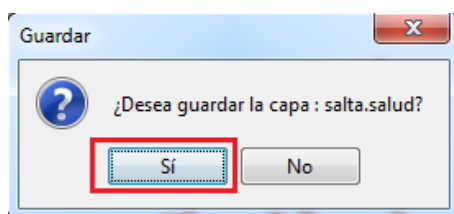
Utilizamos las herramientas de edición que hemos visto en el curso introductorio para modificar la capa, agregar elementos, modificar la estructura de la tabla de atributos, etc.



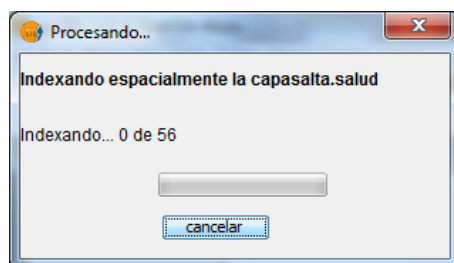
Cuando terminamos de editar vamos de nuevo al menú de la capa cliqueando con el botón derecho sobre ella y seleccionamos la opción "Terminar edición".



Emergerá una ventana que nos pedirá confirmación para guardar los cambios realizados.



Es probable que aparezca una ventanita con una barra de estado que nos muestra la evolución de la transacción de la información con la base. Finalmente se cerrará el proceso y la capa volverá a su estado normal.



Es importante tener en cuenta que gvSIG “bloquea” la tabla que se está visualizando. Esto es un inconveniente no resuelto en las versiones actuales de gvSIG que nos impide continuar editando una tabla consultada desde gvSIG con las herramientas de Postgres. Estas tablas se podrán consultar, utilizar para realizar joins y otras operaciones, pero no podrán ser actualizadas ni eliminadas hasta que gvSIG no se cierre.

Como hemos visto, casi no hay diferencias entre la edición de una capa creada a partir de un archivo shapefile y una capa que en realidad es una tabla o vista de Postgres. Pero en el caso de la tabla de Postgres, todos los usuarios y sistemas clientes se benefician instantáneamente del trabajo realizado por cada usuario.



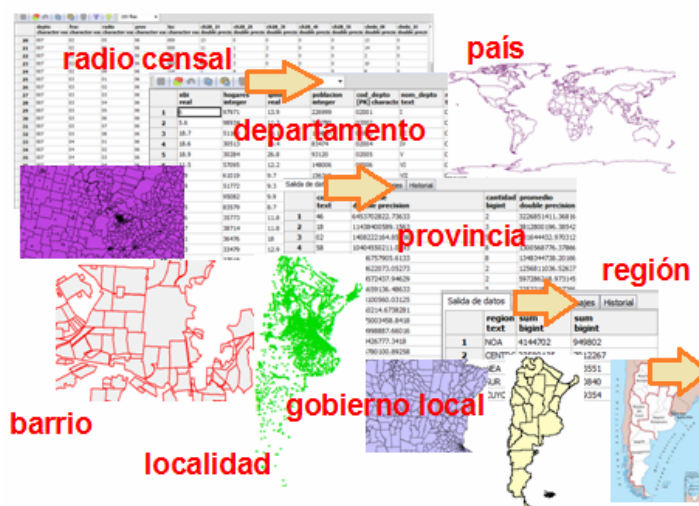
ACTIVIDAD 3

- Modificar la tabla “salta.plazas” desde gvSIG.
- Crear dos grandes parques en la zona Norte de la ciudad.
- Crear una impresión de pantalla que muestre los nuevos parques y agregarla en el documento.

4 SQL: consultas de agregación.

Las consultas de agregación son aquellas que realizan operaciones de distinto tipo entre los valores existentes en distintos registros de la misma columna. El objetivo de estas consultas es conseguir tablas que resuman los datos, llevándolos a otro nivel de agregación menos detallado.

La agregación implica generalmente una operación matemática o lógica sobre un campo para lograr una tabla de resumen con tantos registros como datos diferentes se encuentren en el campo que se define para efectuar la agregación.



Por ejemplo, si tenemos una tabla en donde los datos se encuentran desagregados a nivel de departamento (datos_depto), y queremos llevarlos a nivel de región, entonces efectuaremos la agregación sobre un campo que contenga valores

distintos que se correspondan a las entidades del nivel de agregación que queremos lograr. Con los campos restantes, podemos realizar alguna operación matemática que nos de como resultado un solo dato para cada entidad agregada. En la tabla “datos_deptos” podemos sumar el campo “poblacion” para obtener el dato de población para las regiones.

Archivo Editar Vista Herramientas Ayuda							
	nbi real	hogares integer	ipmh real	poblacion integer	cod_depto [PK] caracte	nom_depto text	region text
448	29.8	1542	64.5	6768	70119	VALLE FERTIL	CUYO
449	27.8	3250	70	6768	70119	25 DE MAYO	CUYO
450	22.1	855	63.2	3958	70133	ZONDA	CUYO
451	22.3	4525	57.1	16856	74007	AYACUCHO	CUYO
452	37.6	1192	76.4	3870	74014	BELGRANO	CUYO
453	19.1	3683	51.6	12469	74021	CORONEL PRIN	CUYO
454	14.4	5205	44.2	18333	74028	CHACABUCO	CUYO
455	10.9	30439	35.8	11	74028	CHACABUCO	CUYO
456	23.8	3090	50.2	10952	74042	GOBERNADOR L	CUYO
457	11.7	5742	38.1	19656	74049	JUNIN	CUYO
458	10.5	46284	34.9	167682	74056	LA CAPITAL	CUYO
459	47.5	1484	81.2	5146	74063	LIBERTADOR GF	CUYO
460	5.4	2248	14.4	7634	78007	CORPEN AIKE	SUR
461	12	19890	25	72174	78014	DESEADO	SUR
462	9.2	25096	18.2	91419	78021	GUER AIKE	SUR
463	11.2	1982	19.4	6580	78028	LAGO ARGENTIN	SUR
464	10.6	1792	29.1	6059	78035	LAGO BUENOS A	SUR
465	6.9	1943	20.3	6139	78042	MAGALLANES	SUR
466	12.6	883	23.9	2846	78049	RIO CHICO	SUR
467	9	12486	30.4	41258	82007	BELGRANO	CENTRO
468	8.4	24771	25.5	78441	82014	CASEROS	CENTRO
469	9.1	47677	29.8	161200	82021	CASTELLANOS	CENTRO

La consulta resultante es:

```
SELECT region, SUM ( poblacion ) FROM datos_depto
GROUP BY region;
```

En esta consulta pedimos dos campos: uno con el nombre de las regiones, y otro con el de la población. Pero como el dato de población está desagregado a nivel de departamento, le aplicamos la operación “SUM” (sumatoria). La consulta termina con la orden “GROUP BY” que indica el campo cuyos valores distintos se convertirán en los registros de la nueva tabla. El resultado es una tabla resumida.

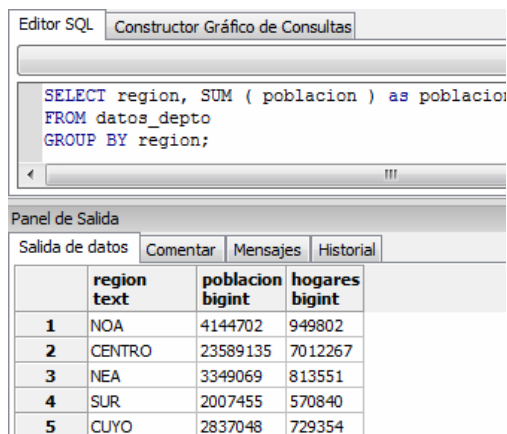
Editor SQL Constructor Gráfico de Consultas	
<pre>SELECT region, SUM (poblacion) FROM datos_depto GROUP BY region;</pre>	
Panel de Salida	
Salida de datos Comentar Mensajes Historial	
region text	sum bigint
1 NOA	4144702
2 CENTRO	23589135
3 NEA	3349069
4 SUR	2007455
5 CUYO	2837048

Se pueden agregar varios campos a la vez en las consultas de agregación. Solo es necesario tener en cuenta que la operación debe dar como resultado un solo valor para el nuevo nivel de agregación.

```
SELECT region, SUM ( poblacion ), SUM ( hogares)
FROM datos_depto
GROUP BY region;
```

Notamos que ambos campos agregados se denominan “sum” en la tabla resultante. Colocamos alias a las columnas para poder identificar de qué se trata cada una.

```
SELECT region, SUM ( poblacion ) as poblacion, SUM ( hogares) as
hogares
FROM datos_depto
GROUP BY region;
```



The screenshot shows a SQL query editor with two tabs: 'Editor SQL' and 'Constructor Gráfico de Consultas'. The 'Editor SQL' tab is active, displaying the following query:

```
SELECT region, SUM ( poblacion ) as poblacion
FROM datos_depto
GROUP BY region;
```

Below the query editor is a 'Panel de Salida' (Output Panel) with tabs for 'Salida de datos', 'Comentar', 'Mensajes', and 'Historial'. The 'Salida de datos' tab is active, showing a table with the following data:

	region text	poblacion bigint	hogares bigint
1	NOA	4144702	949802
2	CENTRO	23589135	7012267
3	NEA	3349069	813551
4	SUR	2007455	570840
5	CUYO	2837048	729354

Las consultas de agregación son muy útiles a la hora de presentar los mismos datos pero utilizando diferentes niveles de agregación territorial. Si tenemos datos almacenados con el gran nivel de desagregación, como el radio censal, podemos llevarlos al nivel departamental o provincial utilizando una consulta de agregación, y así, evitamos almacenar dos veces los mismos datos.

Como ejemplo, tenemos la tabla “datos_radio” en donde cada registro corresponde a un radio censal. Si queremos hacer un mapa por radio censal del Conurbano Bonaerense, simplemente creamos una vista con la consulta de JOIN entre la tabla “radios_censales” y la tabla “datos_radio” utilizando el campo “cod_rad” para establecer la correspondencia.

```
SELECT a.gid, a.cod_rad, a.the_geom, b.th_t
FROM radios_censales a join datos_radio b
ON a.cod_rad = b.cod_rad;
```

El dato obtenido es la cantidad total de hogares por radio censal.

Editor SQL

Constructor Gráfico de Consultas

```
SELECT a.gid, a.cod_rad, a.the_geom, b.th_t
FROM radios_censales a join datos_radio b
ON a.cod_rad = b.cod_rad;
```

Panel de Salida

Salida de datos

Comentar

Mensajes

Historial

	gid integer	cod_rad text	the_geom geometry	th_t double precision
1	1	06805090	0106000020	212
2	2	06805090	0106000020	491
3	3	06805090	0106000020	343
4	4	06805090	0106000020	53
5	5	06805100	0106000020	291
6	100	06749050	0106000020	323
7	6	06805160	0106000020	391
8	7	06805100	0106000020	314
9	8	06805130	0106000020	507
10	9	06805160	0106000020	321
11	10	06805160	0106000020	531
12	11	06805130	0106000020	337
13	12	06805100	0106000020	354
14	13	06805160	0106000020	375
15	14	06805160	0106000020	372

Pero si luego queremos llevar estos mismos datos a nivel de partido del Conurbano, entonces tendremos que agregar los datos usando un campo que contenga valores distintos que se correspondan con el nuevo nivel de agregación. En la tabla “datos_radio” el campo del código de departamento no existe como tal, pero lo podemos crear concatenando los campos “prov” y “depto” usando la sintaxis “prov||depto” en donde los campos se unen mediante dos caracteres especiales llamados “pipes”.

La consulta que nos trae los datos de hogares agregados a nivel de departamento es:

```
SELECT prov||depto AS cod_depto, SUM (th_t) as hogares
FROM datos_radio
GROUP BY cod_depto;
```

Hemos asignados alias a los campos concatenados (cod_depto) y al campo sumariado (hogares). De esta manera será más fácil establecer la correspondencia entre campos en el paso que sigue.

Editor SQL		Constructor Gráfico de Consultas
<pre>SELECT prov depto AS cod_depto, SUM (th_t) as hogares FROM datos_radio GROUP BY cod_depto;</pre>		
Panel de Salida		
Salida de datos		
Comentar Mensajes Historial		
	cod_depto text	hogares double precision
1	06294	6380
2	06392	11211
3	06126	28323
4	06588	18699
5	06638	73026
6	06147	8731
7	06707	11773
8	06826	15059
9	06175	8370
10	06763	44458
11	06210	17890
12	06525	13313

En este caso, no juntaremos dos tablas, sino que juntaremos una tabla con una tabla resultante de una subconsulta.

```
SELECT a.gid, a. nom_depto, a.the_geom, b.hogares
FROM partidos a join DATOS_AGREGADOS b
ON a.cod_depto = b.cod_depto;
```

Ahora reemplazaremos “DATOS_AGREGADOS” por la consulta que agrega los datos a nivel de departamento:

```
SELECT a.gid, a.nom_depto, a.the_geom, b.hogares
FROM partidos a join (SELECT prov||depto AS cod_depto, SUM
(th_t) as hogares FROM datos_radio GROUP BY cod_depto) b
ON a.cod_depto = b.cod_depto;
```

La consulta nos trae todos los datos necesarios para poder generar una vista visible desde gvSIG: el campo con la clave primaria (gid), el campo con la geometría para dibujar los elementos (the_geom) y el campo que es la variable para poder armar un mapa temático (hogares). También trajimos el campo “nom_depto”, para poder etiquetar los departamentos con su nombre.

Podemos hacer una vista y ver los resultados desde gvSIG:

```
CREATE VIEW v_partidos_hogares AS
SELECT a.gid, a.nom_depto, a.the_geom, b.hogares
FROM partidos a join (SELECT prov||depto AS cod_depto, SUM
(th_t) as hogares FROM datos_radio GROUP BY cod_depto) b
ON a.cod_depto = b.cod_depto;
```

Recordemos que el campo geométrico de la vista debe ser registrado en la tabla “geometry_columns”:

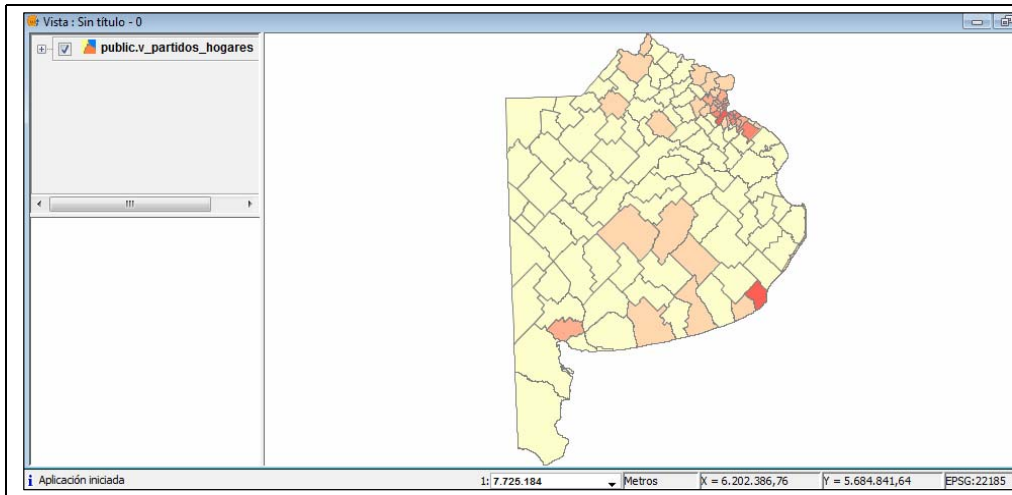
```
INSERT INTO geometry_columns VALUES ('', 'public',
```



```
'v_partidos_hogares', 'the_geom', 2, 22185, 'MULTIPOLYGON');
```

Y al agregar la vista como capa de gvSIG, podemos generar el mapa temático. Tengamos en cuenta que el sistema de referencia de esta capa es EPSG:22185. Ante cualquier duda, siempre podemos consultar el SRID del campo geométrico con:

```
SELECT SRID (the_geom) FROM tabla ;
```



Veamos otro ejemplo, pero esta vez agregando datos extraídos de los campos geométricos de niveles de agregación más pequeños. Vamos a suponer que nos pidieron un reporte en donde debemos informar el porcentaje de superficie protegida por provincia, es decir, una comparación entre la superficie total de cada provincia, y la superficie de los parques nacionales, provinciales, áreas protegidas, etc.

En la tabla “provincias” tenemos un campo geométrico del cual podemos extraer la superficie.

```
SELECT nom_prov, AREA (the_geom) FROM provincias;
```

Editor SQL

Constructor Gráfico de Consultas

```
SELECT nom_prov, AREA (the_geom) FROM provincias;
```

<

!!!

Panel de Salida

Salida de datos

Comentar

Mensajes

Historial

	nom_prov text	area double precision
1	SANTA FE	133493744335.563
2	SAN LUIS	75888565816.1816
3	SALTA	155607256258.805
4	LA PAMPA	143741613440.688
5	CORDOBA	164251097757.342
6	JUJUY	53307789246.6777
7	CORRIENTES	89237449168.625

Lo mismo podemos hacer con la tabla “areas_protegidas”:

```
SELECT AREA (the_geom) FROM areas_protegidas;
```

Pero en este caso, el resultado viene desagregado a nivel de área protegida, y para poder realizar la operación de cálculo del porcentaje, debemos llevar este resultado a nivel de provincia, de manera tal que puedan juntarse las tablas mediante un campo en común. Utilizaremos para realizar la agregación el campo “cod_prov”.

```
SELECT cod_prov, SUM (AREA (the_geom)) AS superficie
FROM areas_protegidas
GROUP BY cod_prov;
```

Hemos sumado el campo “area”, renombrado como “superficie” para cada valor distinto del campo “cod_prov”.

Como información extra para el reporte, podemos agregar la cantidad de áreas protegidas y la superficie promedio de las reservas por provincia.

```
SELECT cod_prov, SUM (AREA (the_geom)) AS superficie,
COUNT (*) as cantidad, AVG (AREA (the_geom)) as promedio
FROM areas_protegidas
GROUP BY cod_prov;
```

El comando “COUNT” cuenta los valores distintos de los campos o combinaciones de campos que se colocan a continuación entre paréntesis. En este caso no calculamos COUNT para un campo en particular, sino para la combinación de todos, y por eso colocamos “(*)”. Además le hemos asignado el alias “cantidad” para hacer más clara la respuesta. Luego, la función “AVG” calcula el promedio entre un conjunto de valores numérico. En este caso, suma todas las superficies protegidas de cada provincia y luego divide por el total de casos por provincia. Le colocamos también, el alias “promedio”.

Editor SQL

Constructor Gráfico de Consultas

```

SELECT cod_prov, SUM (AREA (the_geom)) AS superficie,
COUNT (*) as cantidad, AVG (AREA (the_geom)) as promedio
FROM areas_protegidas
GROUP BY cod_prov;

```

Panel de Salida

Salida de datos

Comentar

Mensajes

Historial

	cod_prov	superficie	cantidad	promedio
	text	double precision	bigint	double precision
1	46	6453702822.73633	2	3226851411.36816
2	18	11438400589.1563	3	3812800196.38542
3	02	1408222164.85156	5	281644432.970312
4	58	10404550211.0293	8	1300568776.37866
5	78	10786757905.6133	8	1348344738.20166
6	86	2513622073.05273	2	1256811036.52637
7	94	1194572437.94629	2	597286218.973145
8	90	1176659136.48633	5	235331827.297266
9	50	7228100560.03125	7	1032585794.29018
10	06	40060214.6738281	1	40060214.6738281
11	38	19275003458.8418	8	2409375432.35522
12	10	5519998887.66016	1	5519998887.66016

Ahora que obtuvimos los datos al nivel de agregación deseado, estamos listos para generar el reporte que nos han solicitado. Primero debemos juntar las tablas:

```
SELECT a.nom_prov, AREA (a.the_geom) as sup_prov,
b.superficie, b.cantidad, b.promedio
FROM provincias a join TABLA_AGREGADA b
ON a.cod_prov = b.cod_prov;
```

Los campos utilizados para realizar la correspondencia son el campo “cod_prov” de cada tabla. Reemplacemos “TABLA_AGREGADA”:

```
SELECT a.nom_prov, AREA (a.the_geom) as sup_prov,
b.superficie, b.cantidad, b.promedio
FROM provincias a join
(SELECT cod_prov, SUM (AREA (the_geom)) AS superficie,
COUNT (*) as cantidad, AVG (AREA (the_geom)) as promedio
FROM areas_protegidas
GROUP BY cod_prov ) b
ON a.cod_prov = b.cod_prov;
```

Editor SQL Constructor Gráfico de Consultas

```
SELECT a.nom_prov, AREA (a.the_geom) as sup_prov,
b.superficie, b.cantidad, b.promedio
FROM provincias a join
(SELECT cod_prov, SUM (AREA (the_geom)) AS superficie,
COUNT (*) as cantidad, AVG (AREA (the_geom)) as promedio
FROM areas_protegidas
GROUP BY cod_prov ) b
ON a.cod_prov = b.cod_prov;
```

Panel de Salida

Salida de datos Comentar Mensajes Historial

	nom_prov text	sup_prov double precision	superficie double precision	cantidad bigint	promedio double precision
1	SAN LUIS	75888565816.1816	813450793.431641	1	813450793.431641
2	SALTA	155607256258.805	21338794441.8066	7	3048399205.97238
3	LA PAMPA	143741613440.688	99444182.1542969	3	33148060.718099
4	CORDOBA	164251097757.342	3993682176.40625	3	1331227392.13542
5	JUJUY	53307789246.6777	19275003458.8418	8	2409375432.35522
6	CORRIENTE	89237449168.625	11438400589.1563	3	3812800196.38542
7	CHACO	99889377890.0879	254036059.904297	2	127018029.952148
8	FORMOSA	75795109006.8809	1343426777.3418	4	335856694.335449
9	MISIONES	30443856771.2852	4146831821.83594	7	592404545.976563

Ahora que tenemos los dos datos de superficie, nos queda solo calcular el porcentaje:

```
SELECT a.nom_prov, AREA (a.the_geom) as sup_prov, b.superficie,
(b.superficie * 100) / AREA (a.the_geom) as porcentaje,
b.cantidad, b.promedio
FROM provincias a join
(SELECT cod_prov, SUM (AREA (the_geom)) AS superficie,
COUNT (*) as cantidad, AVG (AREA (the_geom)) as promedio
FROM areas_protegidas
GROUP BY cod_prov ) b
ON a.cod_prov = b.cod_prov;
```

Finalmente hemos obtenido un reporte muy completo que podrá ser incorporado a un informe.

Editor SQL Constructor Gráfico de Consultas

```

SELECT a.nom_prov, AREA (a.the_geom) as sup_prov, b.superficie,
(b.superficie * 100) / AREA (a.the_geom) as porcentaje,
b.cantidad, b.promedio
FROM provincias a join
(SELECT cod_prov, SUM (AREA (the_geom)) AS superficie,
COUNT (*) as cantidad, AVG (AREA (the_geom)) as promedio
FROM areas_protegidas
GROUP BY cod_prov ) b
ON a.cod_prov = b.cod_prov;

```

Panel de Salida

Salida de datos Comentar Mensajes Historial

	nom_prov text	sup_prov double precision	superficie double precision	porcentaje double precision	cantidad bigint	promedio double precision
1	SAN LUIS	75888565816.1816	813450793.431641	1.07190165564862	1	813450793.431641
2	SALTA	155607256258.805	21338794441.8066	13.7132386720553	7	3048399205.97238
3	LA PAMPA	143741613440.688	99444182.1542969	0.06918259770009	3	33148060.718099
4	CORDOBA	164251097757.342	3993682176.40625	2.4314493059318	3	1331227392.13542
5	JUJUY	53307789246.6777	19275003458.8418	36.1579493939398	8	2409375432.35522
6	CORRIENTE	89237449168.625	11438400589.1563	12.8179376435806	3	3812800196.38542
7	CHACO	99889377890.0879	254036059.904297	0.25431739116828	2	127018029.952148
8	FORMOSA	75795109006.8809	1343426777.3418	1.77244520780337	4	335856694.335449
9	MISIONES	30443856771.2852	4146831821.83594	13.6212433693594	7	592404545.976563
10	ENTRE RIOS	78539994937.5293	105895803.556641	0.13483041810846	2	52947901.7783203



ACTIVIDAD 4

Crear una vista con la consulta anterior y hacer un mapa temático que muestre el porcentaje de superficie NO protegida por provincia. Hacer una impresión de pantalla y añadirla al documento.

Bibliografía

OLAYA, Víctor. (2011). *Sistemas de Información Geográfica*. Versión 1.0, Rev. 24 de marzo de 2011. Proyecto “Libro Libre SIG”.

http://forge.osor.eu/docman/view.php/13/577/Libro_SIG.zip

THE POSTGIS TEAM. Manual de PostGIS 1.5.3.

<http://www.postgis.org/download/postgis-1.5.3.pdf>

OBE, Regina y HSU Leo. (2011). *PostGIS in Action*. Editorial Manning. Stamford.

Índice

1 SQL ESPACIAL. ANÁLISIS TOPOLÓGICOS	1
1.1 TOUCH. ESTÁ EN CONTACTO.....	2
1.2 CONTIENE	4
1.3 CONTIENE PROPIAMENTE	5
1.4 WITHIN. ESTÁ CONTENIDO POR.	6
1.5 OVERLAPS. SOLAPA.	7
1.6 CROSSES. SE CRUZA.	7
1.7 INTERSECTS	8
1.8 DISJOINT	9
2 ÍNDICES ESPACIALES.	11
3 EDICIÓN DE INFORMACIÓN GEOGRÁFICA DESDE UN SIG	17
4 SQL: CONSULTAS DE AGREGACIÓN.....	20

Usted es libre de compartir - copiar, distribuir, ejecutar y comunicar públicamente y de hacer obras derivadas de este documento

Este documento es una obra compartida bajo la licencia Creative Commons.

Atribución-NoComercial-CompartirIgual 2.5 Argentina (CC BY-NC-SA 2.5)



Atribución — Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciante (pero no de una manera que sugiera que tiene su apoyo o que apoyan el uso que hace de su obra).



No Comercial — No puede utilizar esta obra para fines comerciales.



Compartir bajo la Misma Licencia — Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Aviso: Al reutilizar o distribuir la obra, tiene que dejar muy en claro los términos de la licencia de esta obra. La mejor forma de hacerlo es enlazar a la siguiente página:

<http://creativecommons.org/licenses/by-nc-sa/2.5/ar/>

Programa Nacional Mapa Educativo
Ministerio de Educación
República Argentina

Teléfono/Fax: 54 11 4129-1408
Correo electrónico: mapaedu_nac@me.gov.ar
www.mapaeducativo.edu.ar

