BASES DE DATOS ESPACIALES CON POSTGIS Y QGIS

PostgreSQL 17 & PostGIS 3.5

PSIG
Implementació, gestió i formació SIG

Carlos López Quintanilla
Consultor SIG

nov/des de 2025
Barcelona

Calendario y horario

Lloc:

Edifici MediaTIC
Cibernarium

Els dies (4):

- **Dimarts 25 de novembre**
- **Dijous 27 de novembre**
- **Dilluns 1 de desembre**
- **Dimecres 3 de desembre**

Horario:

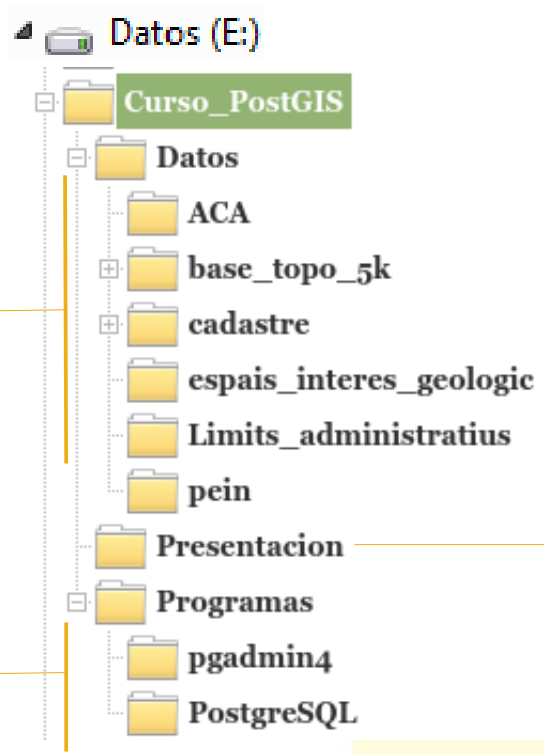
- **De 16:00 a 19:00 horas (3 h**

Total 12 hores de curs

Hay que hacer un descanso

Con descanso se rinde más!

Material del curs



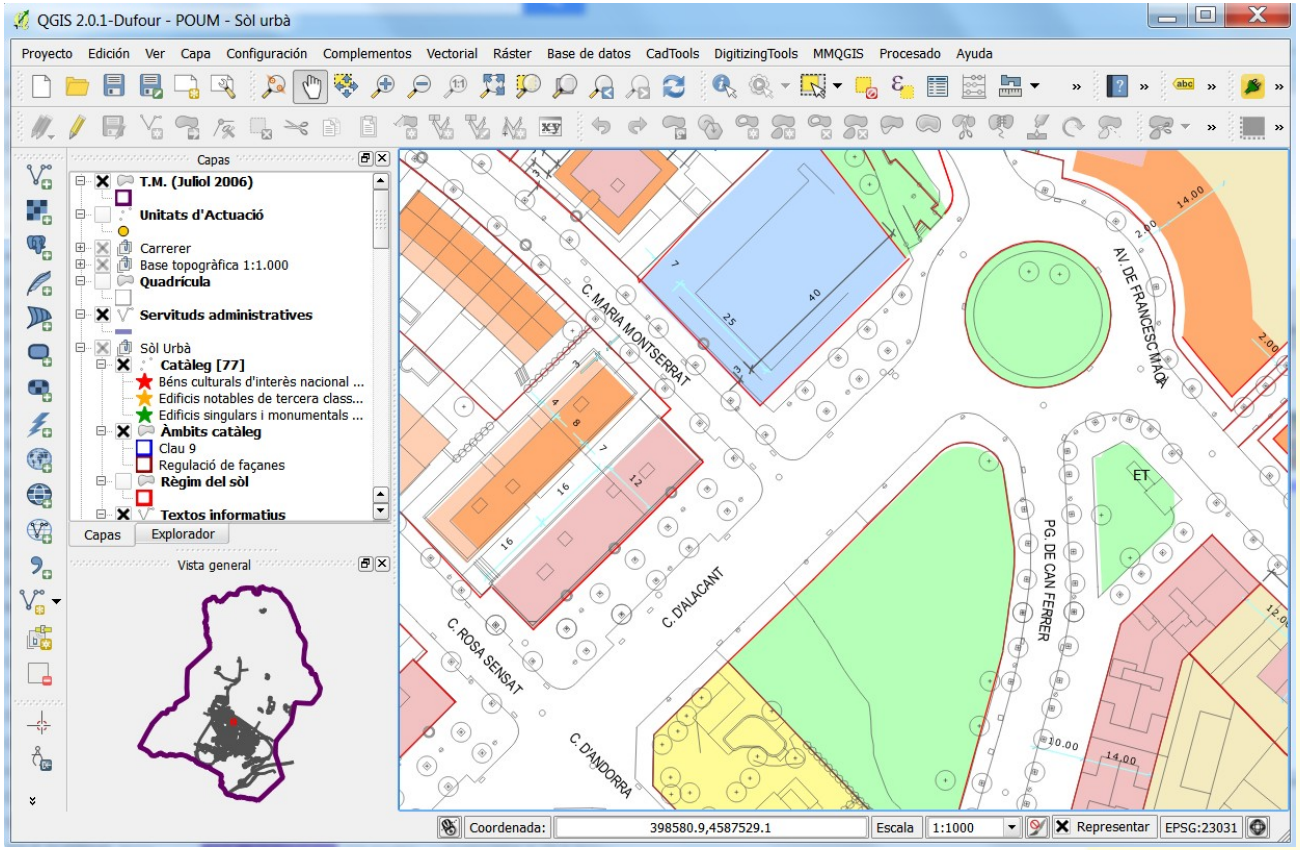
Dades per realitzar els exercicis (shp, tif i altres)

Aquesta presentació

Programes i aplicacions

Ens presentems !!!

- Nom
- Formació
- Professió
- Perqué PostGIS
- Experiències amb altres bases de dades



Índice

1. Introducción
2. Configuración y gestión básica
3. Query (SQL)
4. Programación
5. Gestión de la base de datos
6. Herramientas

Índice

1. Introducción

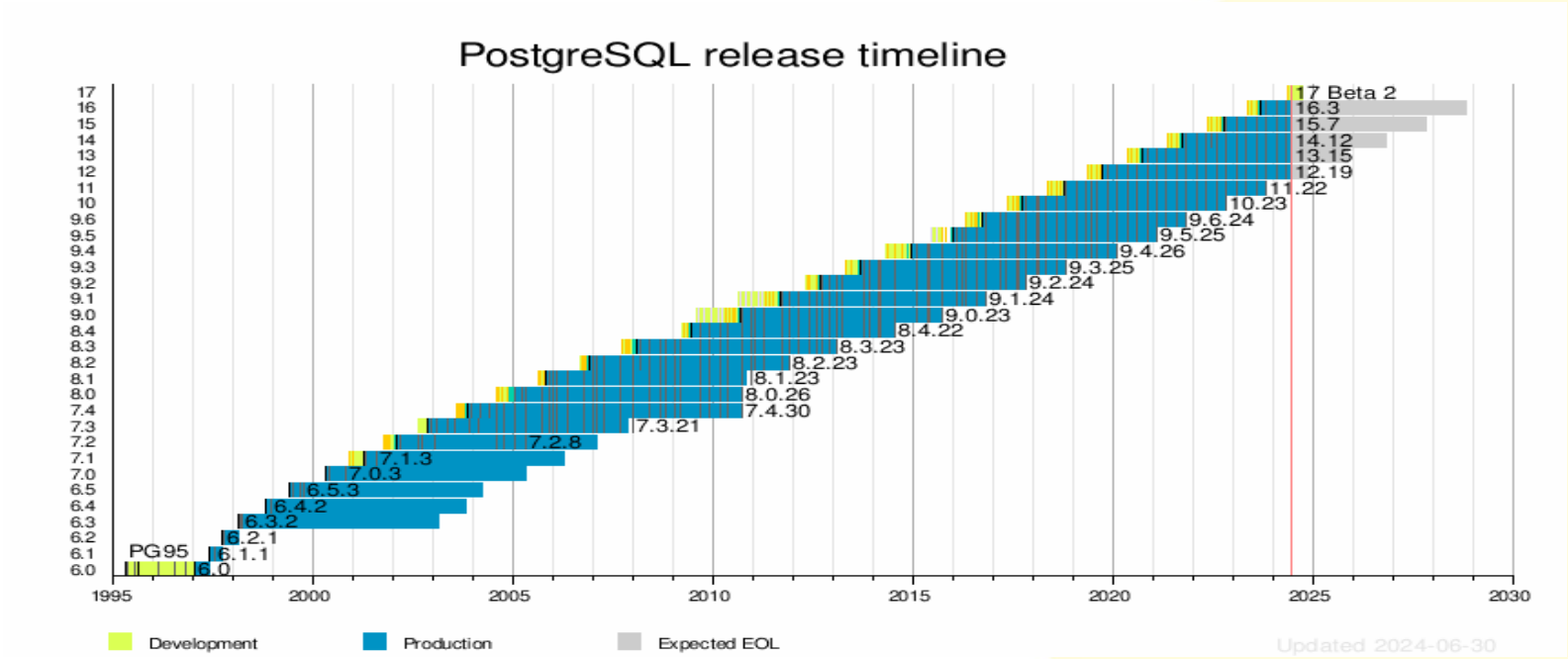
- 1.1 PostgreSQL
- 1.2 PostGIS
- 1.3 pgAdmin 3 y 4
- 1.4 Webs de referencia
- 1.5 QGIS
- 1.6 DB Manager (QGIS)

1.1. PostgreSQL

PostgreSQL

- PostgreSQL es un sistema de gestión de bases de datos relacional.

- Características principales:
  - Alta concurrencia
  - Amplia variedad de tipos de datos (texto ilimitado, geometrías, etc...).
  - Disparadores (triggers)
  - Funciones en múltiples lenguajes
  - Integritat transsacional



1.2. PostGIS

PostGIS

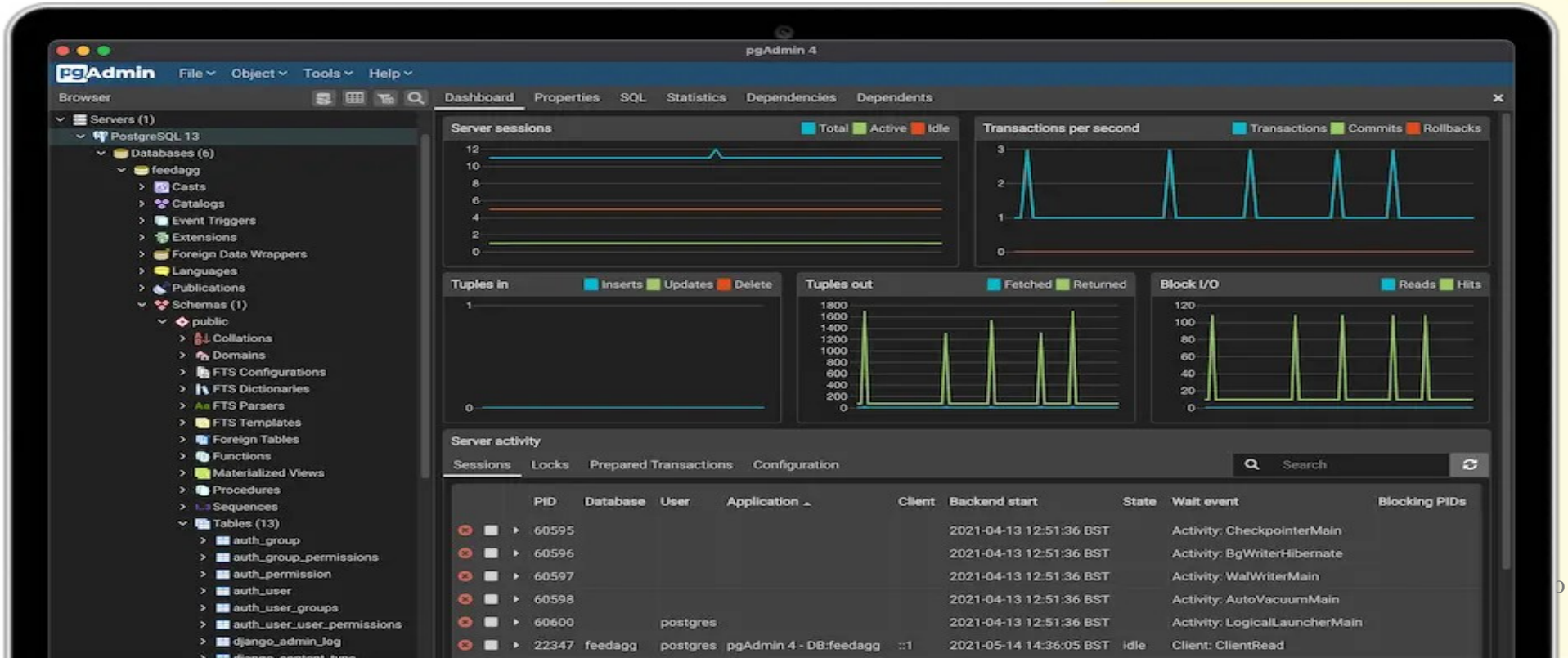
- PostGIS es un módulo que añade soporte de objetos geográficos o espaciales a la base de datos relacional PostgreSQL, y la convierte en una base de datos espacial.

- PostGIS añade 3 características:
  - Tipos de datos espaciales
  - Funciones que operan con los datos espaciales
  - Indexación espacial

1.3. pgAdmin 3 i 4

pgAdmin 4

- **pgAdmin** 4 es un programa que contiene herramientas para administrar y gestionar bases de datos de tipo PostgreSQL



1.4. Webs de referencia

1.3. Webs de referencia

Página web oficial de PostgreSQL:
https://www.postgresql.org/
Descarga del programa, documentación, manual, ejemplos de uso

Página web oficial de PostGIS:
http://postgis.net/
Descarga del módulo, documentación, manual, ejemplos de uso

Página web oficial de pgAdmin:
http://pgadmin.org/
Descarga del programa, documentación, código, etc...

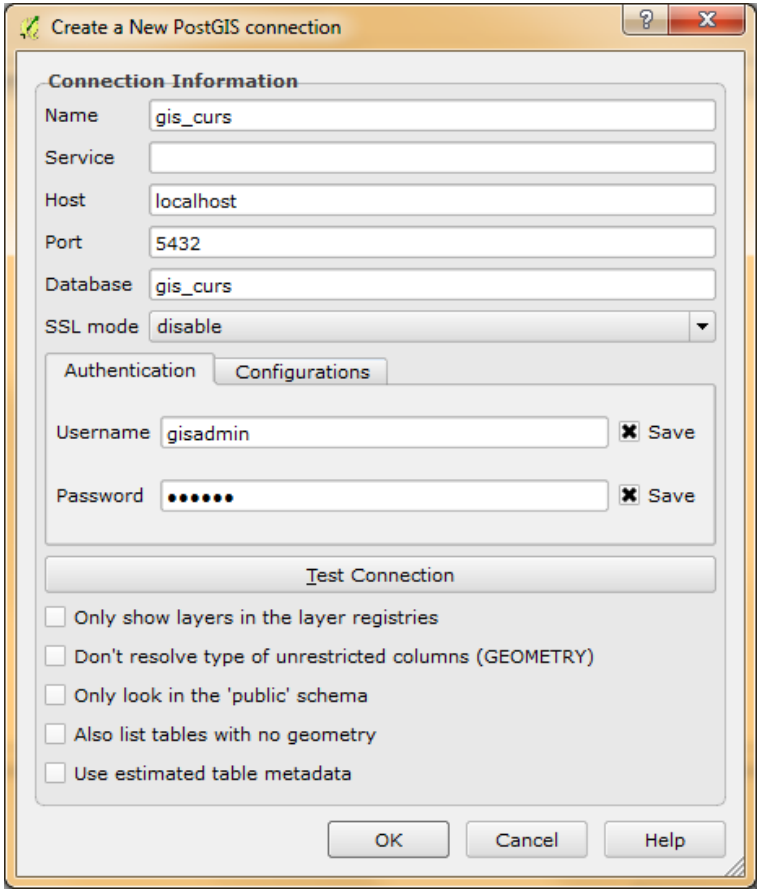
Tutorial SQL w3school (solo querys):
https://www.w3schools.com/sql/

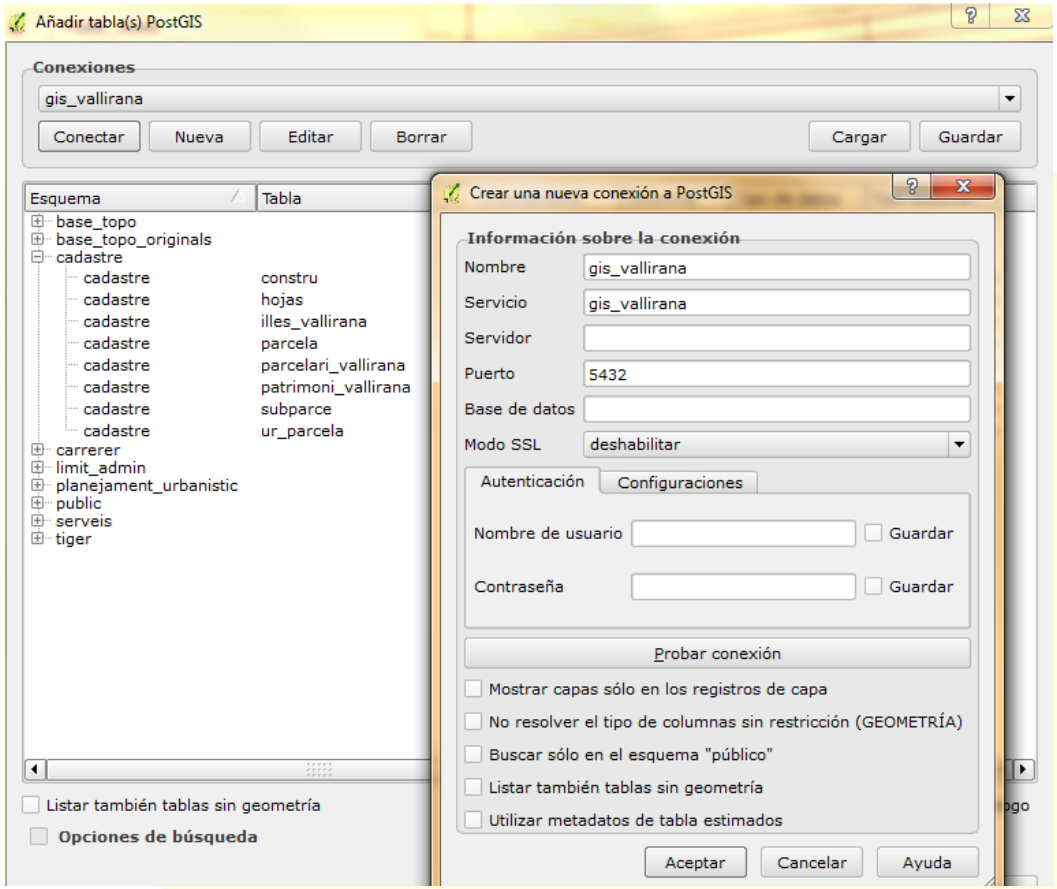
Foro de preguntas y respuestas:
- StackExchange: http://gis.stackexchange.com/

1.5 QGIS

QGIS

Para conectar con datos PostgreSQL primero hay que definir una conexión y después conectar

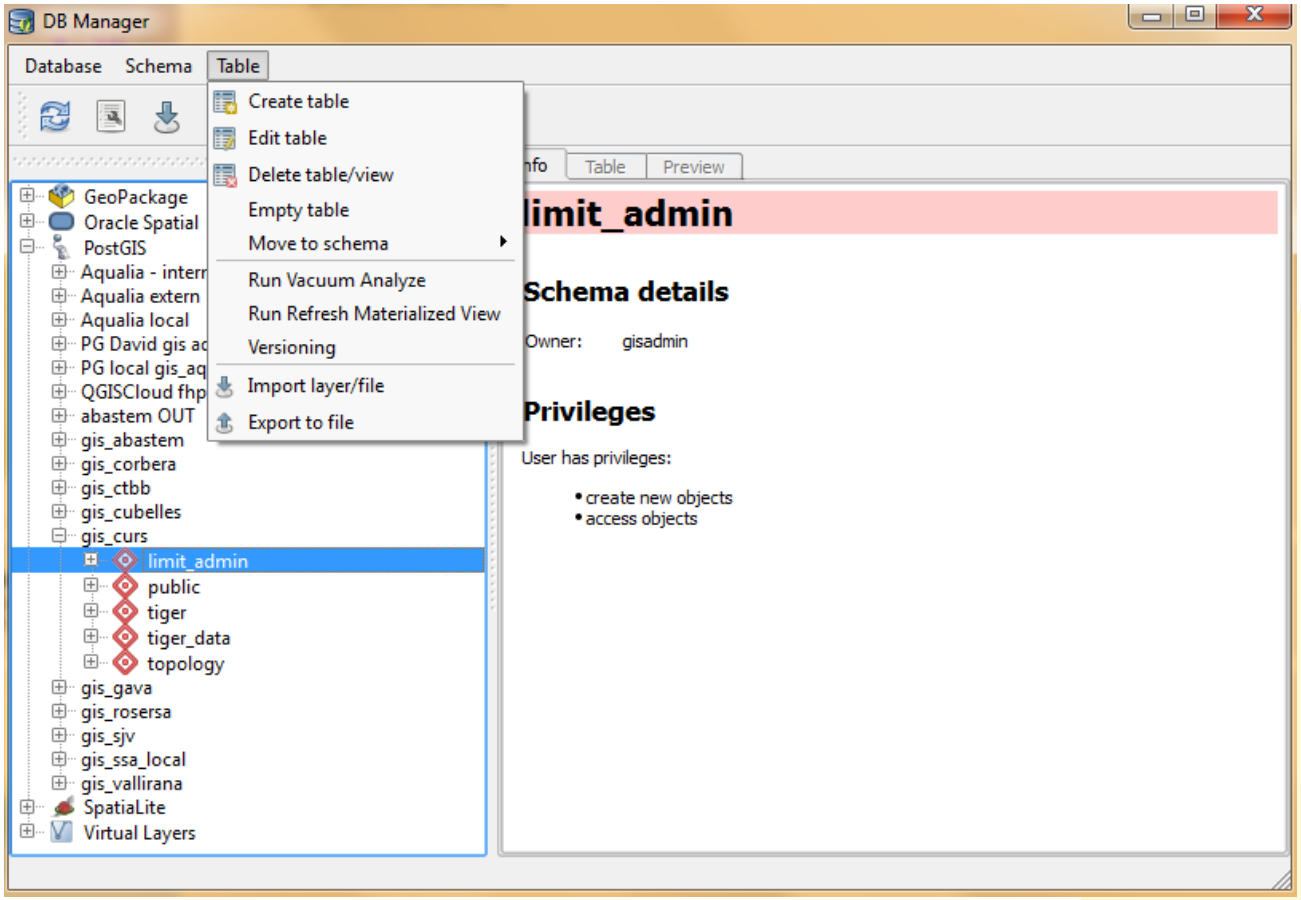


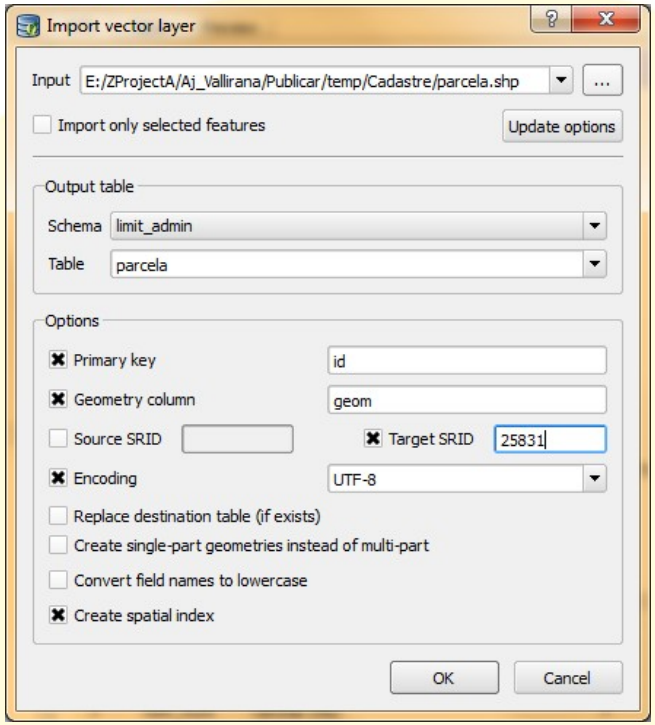


1.6 DB Manager (QGIS)

DB Manager (QGIS)

Plugin de QGIS para administrar PostGIS.





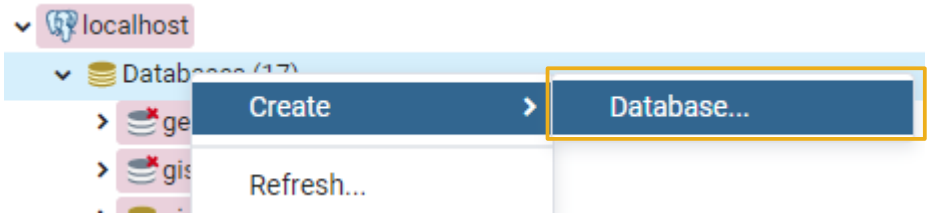
Índice

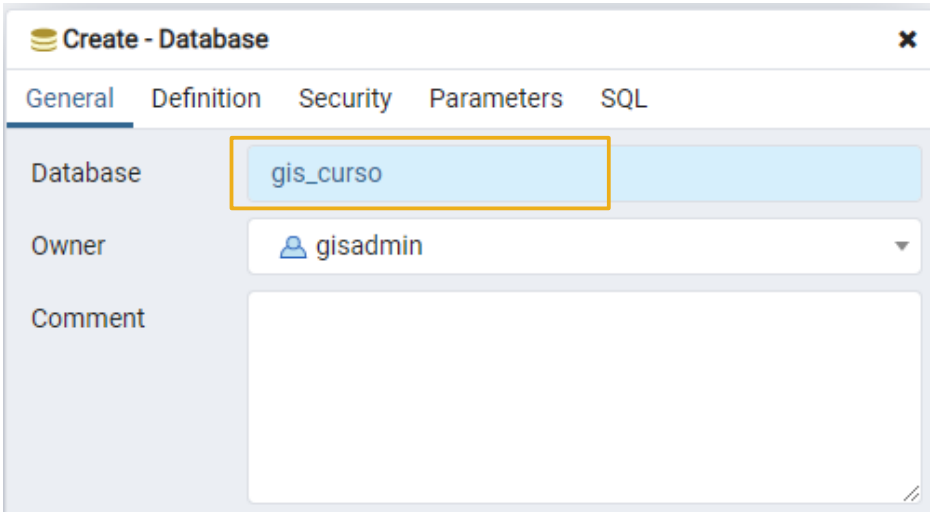
2. Configuración y gestión básica

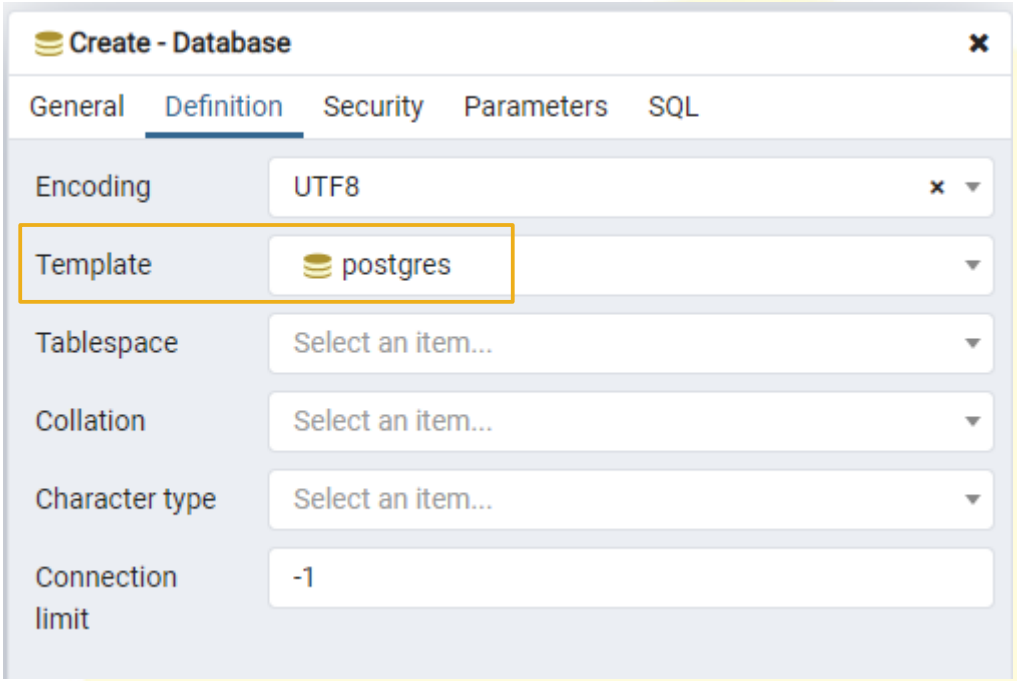
2.1 Generar la base de datos
2.2 Configuración para conectar (QGIS & pgAdmin)
2.3 Fichero de configuración pg_service.conf
2.4 Migrar capas (plugin DB Manager)
2.5 Jugar con las capas y pgAdmin

2.1 Generar la base de datos

Generar la base de dades



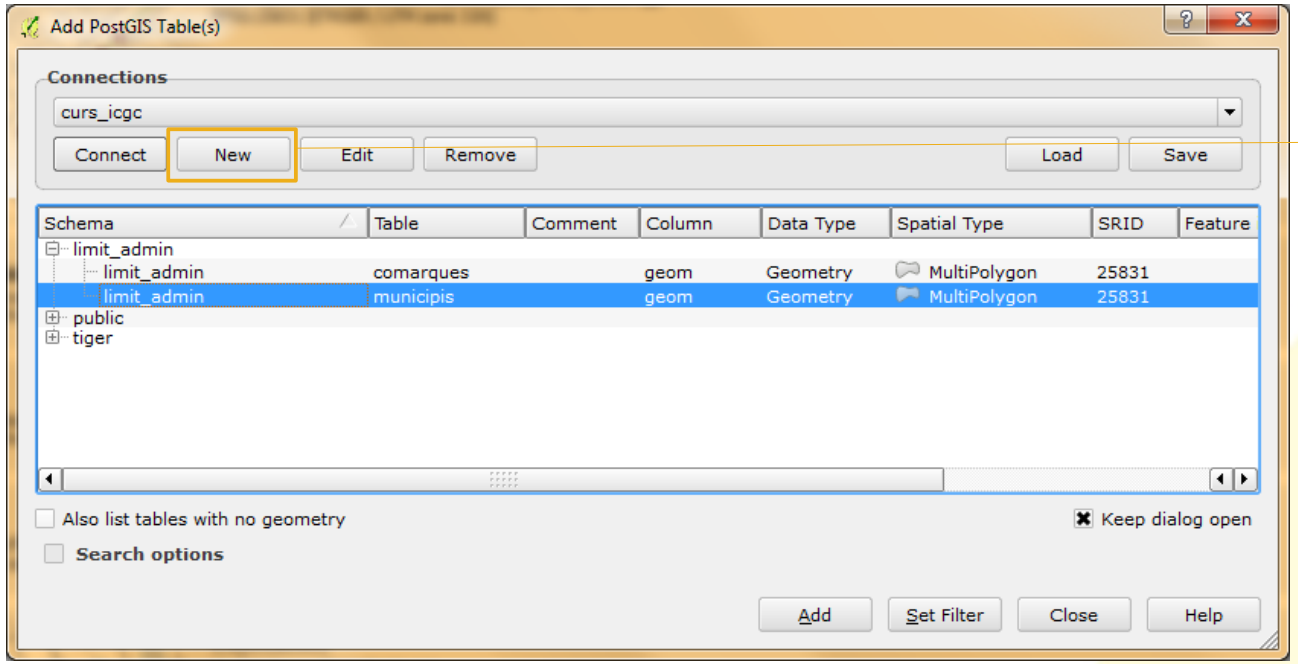


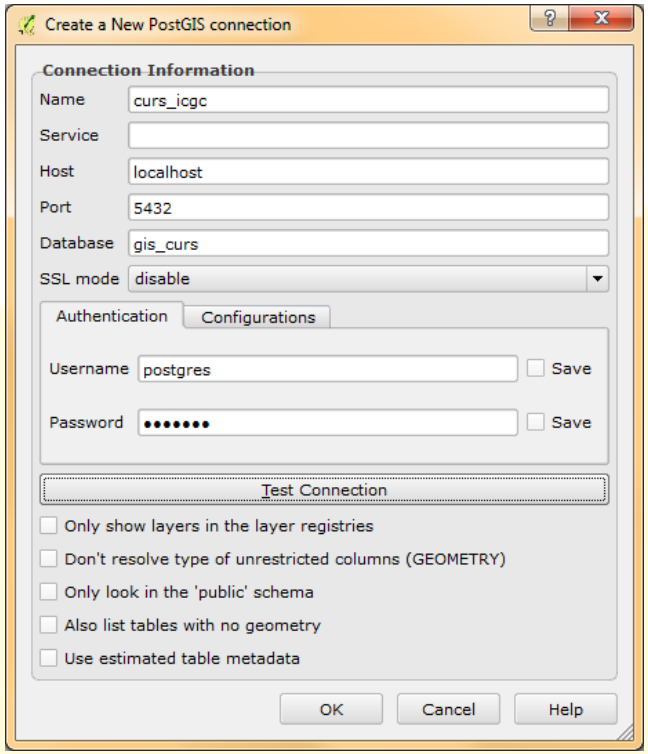


2.2 Configuración para conectar (QGIS & pgAdmin)

QGIS conexión a PostGIS

Para conectar con una base de datos de tipo PostGIS utilizaremos el icono del elefante

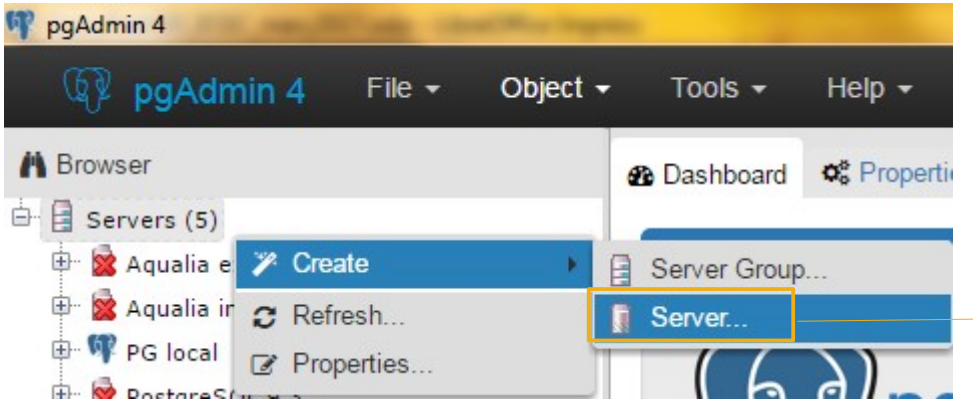


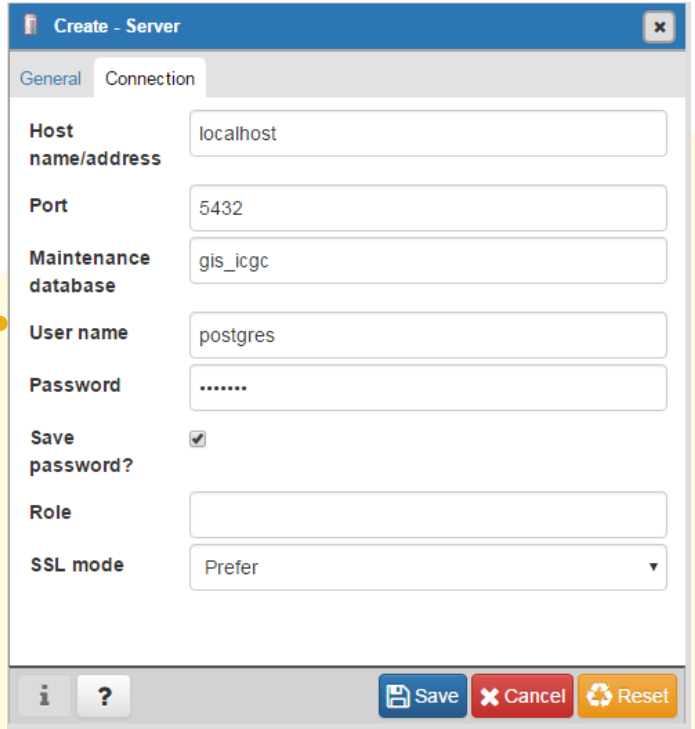


Name: el que queráis
Host: localhost
Port: 5432
Username: Postgres
Password: ?????

pgAdmin 4 connexión a PostGIS

Para connectar con una base de datos de tipo PostGIS utilizaremos la herramienta Servers >> Create >> Server...





Name: el que queráis
Host: localhost
Port: 5432
Username: postgres
Password: ?????

2.3 Fichero de configuración pg_service.conf

pg_service.conf

Para conectar con una base de datos podemos definir los parámetros de conexión con un fichero local de tipo INI.

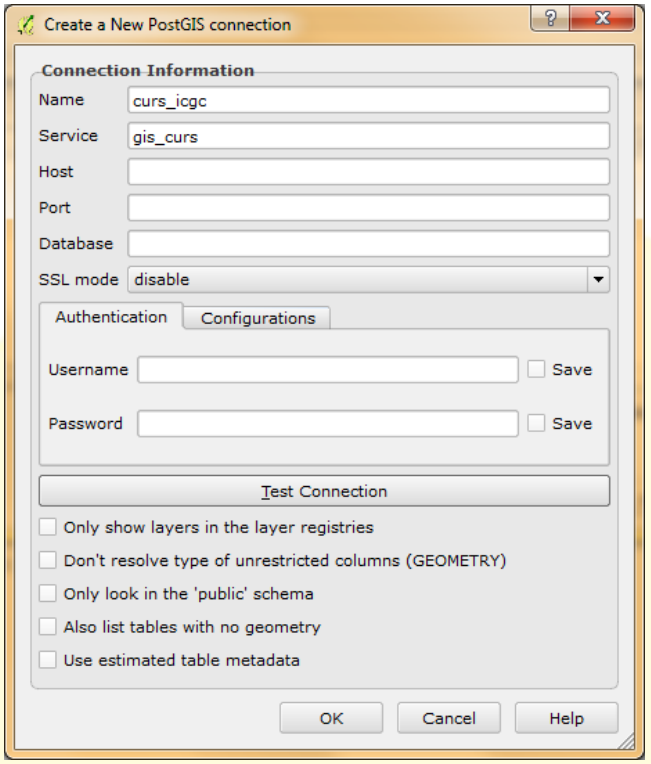
Primero hay que declarar una variable de entorno (usuario o sistema):

- Nombre de la variable = PGSYSCONFDIR
- Valor de la variable = ubicación del fichero pg_service.conf

Contenido del fichero _service.conf

```
[gis_curso]
host=localhost
port=5432
dbname=gis_curso
user=postgres
password=postgres
```

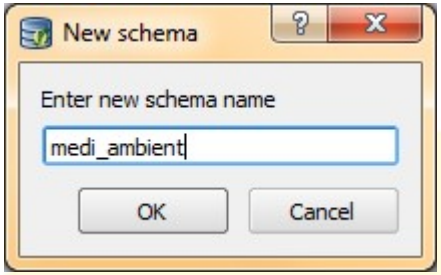
Atención:
En Windows, hay que guardar pg_service.conf en formato Unix.
Para hacerlo con el Notepad++ ir a:
Editar --> Conversión fin de línia --> Convertir en formato UNIX.
Guardar

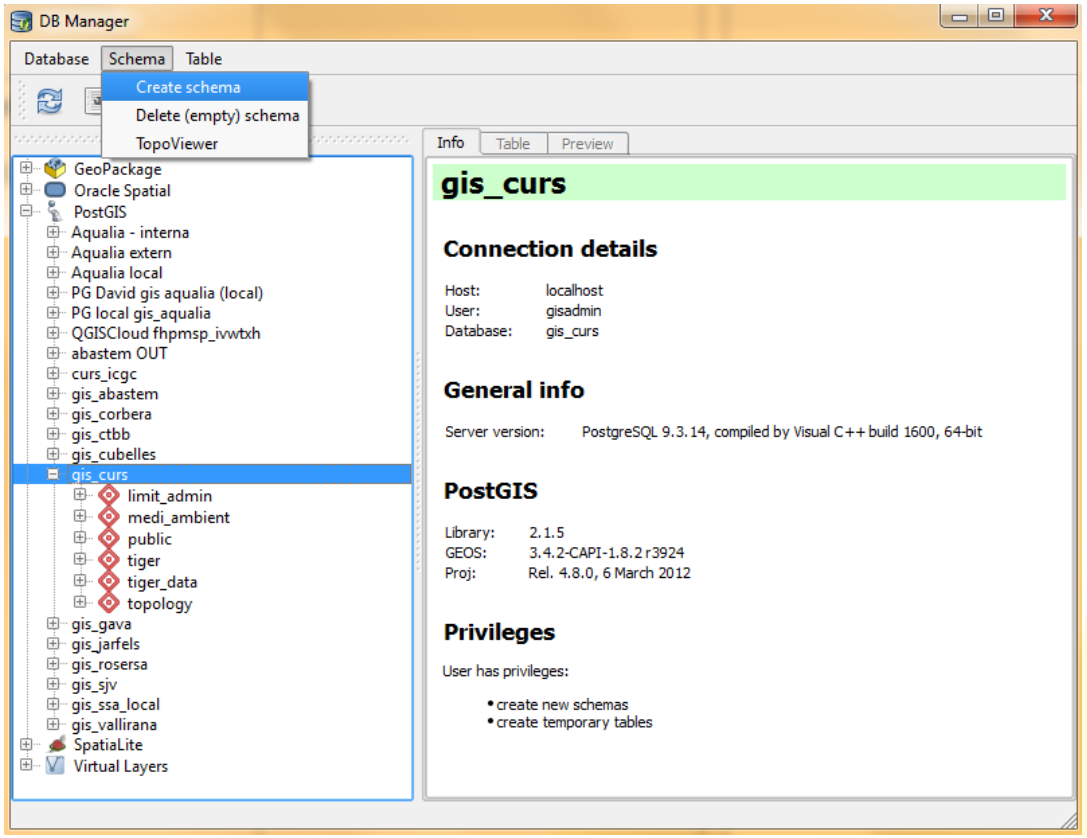


2.4 Migrar capas (plugin DB Manager)

DB Manager (QGIS)

Abrimos el DB Manager y localizamos la base de datos gis_curso
El el menú Schema hacemos dos nuevos Schema
nombre=**medio_ambiente y limit_admin**

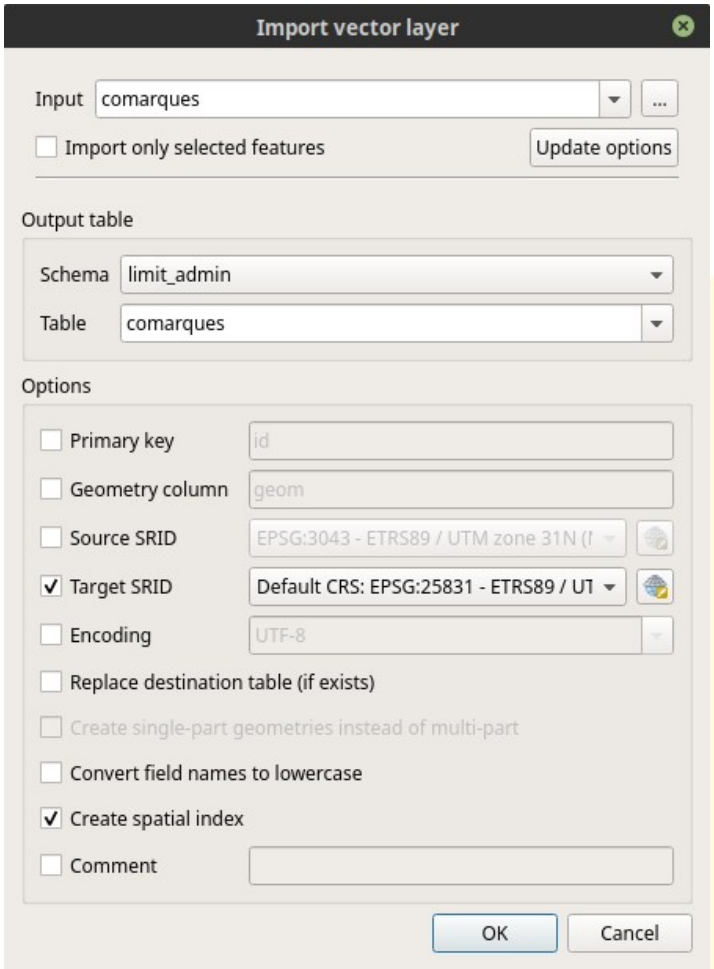




DB Manager (QGIS)

Utilizaremos la herramienta «**Import Layer**» para importar la capa de

- espais_pein
  (schema medio_ambiente)
- munis_poly
- comarcas
- provincias
  (schema limit_admin)



2.5 Jugar con las capas y pgAdmin

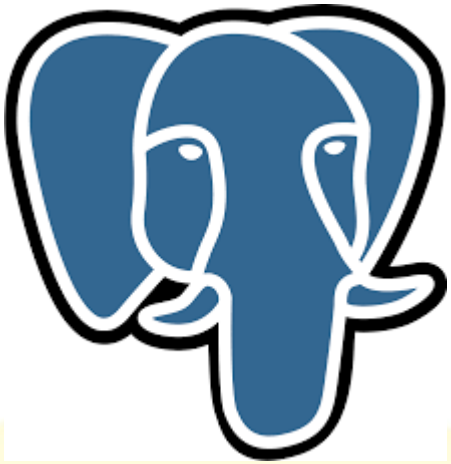
Índice

3. Query (SQL)

- 3.1 Introducción a SQL
- 3.2 Select
- 3.3 Alias
- 3.4 Tipo de datos
- 3.5 From
- 3.6 Sensibilidad a mayúsculas
- 3.7 Where
- 3.8 Funciones PostGIS
- 3.9 Order by
- 3.10 Limit
- 3.11 Agregación
- 3.12 Group by
- 3.13 Having
- 3.14 Join
- 3.15 Subquery
- 3.16 Union
- 3.17 Create table
- 3.18 Alter table
- 3.19 Insert
- 3.20 Update
- 3.21 Delete
- 3.22 Drop & drop cascade
- 3.23 Views

3.1 Introducción a SQL

Introducción a SQL



Structured Query Language (SQL) es un lenguaje de programación diseñado para trabajar con bases de datos.

La primera versión se hizo el año 1986 y la última (SQL3) salió el año 1999. La última ampliación se hizo el año 2006.

Muchas bases de datos añaden sentencias propias al lenguaje base, el cual funciona para todas. Nosotros utilizaremos algunas cosas propias de la versión de postgres.

-- Dos guiones y un espacio hacen que la línea sea un comentario

/
Una barra y un asterisco hacen que sea comentario hasta encontrar un asterisco y una barra.
En este caso se pueden comentar múltiples líneas simultáneamente
**/*

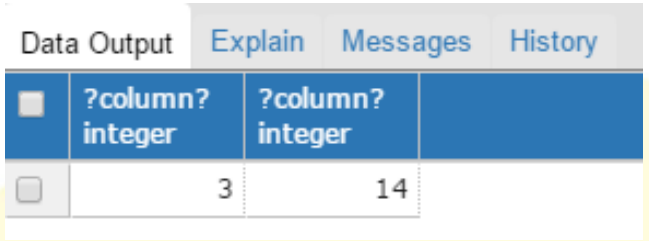
3.2 Select

Select

La instrucción ***Select*** selecciona datos que son enviados al usuario en forma de tabla.
Es la instrucción más básica y la más compleja al mismo tiempo, ya que tiene muchas variantes y posibles modificaciones.

Al final de cada sentencia hay que añadir un **punto y coma** (;), esto sirve para darle la instrucción al postgresql para que se ejecute

```
select 3, 12+2;
```

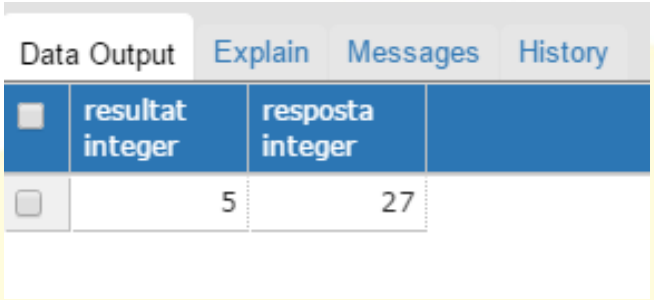


3.3 As

As

En el ejemplo anterior podemos ver que las columnas no tienen nombre. Para cambiar el nombre de las columnas, utilizaremos ***AS***, el cual cambia el nombre de estas.

```
select 3+2 as resultado, 6+21 as respuesta;
```



3.4 Tipo de datos

Tipo de datos

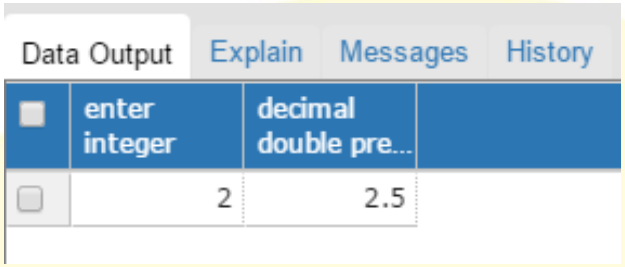
SQL distingue entre diferentes tipo de datos. Cada tipo tiene sus operaciones que pueden dar diferentes resultado, aunque representen la misma cosa.

Ejemplo: el número 5

Por suerte, SQL permite cambiar el tipo de dato, cambiando su comportamiento. Cambiar el tipo de un dato se llama **casting** Vigilar por que, si conversión puede fallar si esta no es posible.

Para convertir un tipo de datos, hay 2 opciones:
:: → dato::tipo – versión Postgres
Cast → cast(dato as tipo) – versión SQL general

select 5/2 as enter, 5::float/2 as decimal;



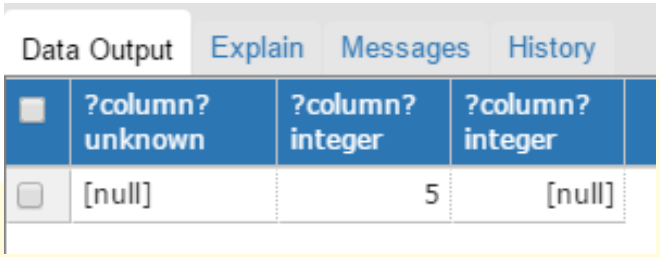
Tipo de datos: null

A veces se da el caso que no hay datos, ja sea en la base de datos donde buscamos o bien que no nos interesa devolver un resultado.

Esta ausencia de dato o valor se representa como ***null***.

Casi todas las operaciones (incluidas todas las matemáticas) devuelven null si null es uno de sus operantes.

Veamos un ejemplo:
select null, 5+0, 5+null;



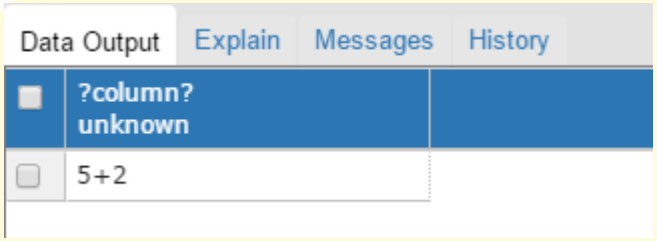
Por lo tanto, hay que estar muy alerta con no confundir con el valor 0.

Tipos de datos: literales

Si utilizamos la sentencia:
```
select 5+2;
```

El resultado que obtenemos es:

Pero, ¿Cómo podemos hacer para que realmente tome 5+2 como un solo valor?
Si lo ponemos todo entre comillas simples, PostgreSQL lo interpreta como una sola cosa:
```
select '5+2';
```

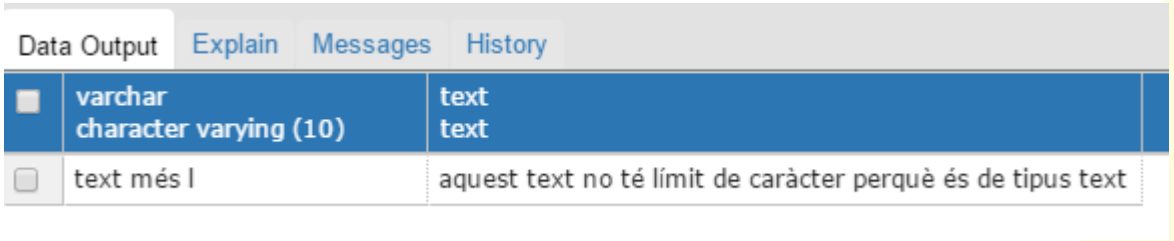


Tipo de datos: text

Hay dos tipos básicos de texto:
• Cadena de caracteres con límite
• Cadena de caracteres sin límite

La primera opción sería varying char(n) o bien **varchar**(n);
Y la segunda sería **text**.

```
select 'texto más largo que puedes escribir'::varchar(10),
'este texto no tiene límite de caracteres porque es de tipo text'::text;
```



3.5 From

From

Una base de datos sirve para guardar y leer datos.
Vamos a hacer ahora esto último.

FROM nos permite acceder a los datos de una tabla de la base datos. El contenido de las columnas resultantes será ejecutado para todas las filas de la tabla. Podemos especificar la columna de cada interación por su nom.

```
select nomcomar, geom from limit_admin.comarques;
```

A veces nos interesa coger todos los campos y valores de una tabla.
Esto se puede hacer utilitzando un asterisco.

```
select * from limit_admin.comarques;
```

También se pueden hacer operaciones con los datos de cada columna.

```
select id, id+5 from limit_admin.comarques;
```

3.6 Sensibilidad a mayúsculas

Sensibilidad a mayúsculas

El lenguaje SQL solo distingue entre mayúsculas y minúsculas en dos casos:
1- Valores literales (como por ejemplo text)
2- Nombres identificados entre comillas dobles (“)

También ignora las nuevas líneas, espacios extras y tabulaciones.

Hasta ahora hemos mostrado las sentencias con el formato mayúsculas para las instrucciones y minúsculas para los nombres. Esto es debido a motivos históricos.

Lo más importante es utilizar siempre el mismo sistema.

Recomendación!!

Definir todo con **minúsculas**, nombre de base de datos, esquemas, tablas, vistas, campos, roles, etc...

3.7 Where

Where

La instrucción where introduce condicionales al select.
Solo se devuelven las filas que cumplan la condición escrita en el where. Esta puede ser cualquier expresión que sea de tipo booleano.

```
select columna
   from basic.taula
   where columna > 3
;
```

Operaciones booleanas

Op	Definición
a = b	Igual que
a <> b a != b	Diferente que
a > b	Más grande que
a >= b	Más grande o igual que
a < b	Más pequeño que
a <= b	Más pequeño o igual que
not a	$\bar{a}$ (álgebra de Boole)
a or b	a·b (álgebra de Boole)
a and b	a+b (álgebra de Boole)

Pequeño ejercicio:

¿Qué id tiene la comarca del Bages?
¿Cuántos municipios hay en la comarca del Baix Llobregat?
¿Cuántos municipios tienen más de 1000 metros de altitud y más de 1000 habitantes?
¿Cuántos municipios tiene más de 200 km2?

3.8 Funciones PostGIS

Funciones PostGIS

boolean **ST_Intersects**(geometry geomA , geometry geomB);
Devuelve un booleano si 2 geometrías se intersectan.

geometry **ST_Intersection**(geometry geomA , geometry geomB);
Devuelve la geometría de la intersección entre 2 geometrías.

geometry **ST_Union**(geometry g1, geometry g2);
Devuelve la geometría de la unión de entre 2 geometrías.

geometry **ST_Transform**(geometry g1, integer srid);
Devuelve una geometría transformada a otro SRC.

geometry **ST_SetSRID**(geometry geom, integer srid);
Devuleve la geometría definida en un SRC.

float **ST_Area**(geometry g1);
Devuelve el área de una geometría.

Funciones de PostGIS
(última versión):
postgis.net/docs/reference.html

Funciones PostGIS

PostGIS dispone de un gran número de funciones que hacen muchos cálculos. No nos interesa saber como funcionan internamente, lo importante es el resultado a partir de los datos que le proporcionamos. Las funciones se escriben con unos paréntesis después de su nombre.

```
select nomcomar, ST_Centroid(geom)
   from limit_admin.comarcas
   where ST_Area(geom) > 1000000000
;
```

Funciones de PostGIS (última versión): postgis.net/docs/reference.html

Funciones de PostgreSQL

PostgreSQL también tiene sus propias funciones. Aquí algunas de las más importantes:

• coalesce(a, b, ...): Devuelve el primer valor no null del listado o null si todo lo son.
• nullif(a, b): Si *a* igual a *b*, devuelve null. Si no, devuelve *a*.
• abs(a): Valor absoluto d'*a*.
• round(a): Redondea *a*.
• lower(a): Convierte *a* a minúsculas.
• date_trunc(medida, a): Trunca la fecha/hora *a* a la medida *medida*.

Funciones de PostgreSQL (última versión):
www.postgresql.org/docs/current/static/functions.html

3.9 Order by

Order by

Order by se pone después del *where* y ordena las filas según los campos y criterios elegidos.

Ejemplos:

```
select id, ST_Area(geom) as area
   from limit_admin.comarques
   order by area, id;

select id, ST_Area(geom) as area
   from limit_admin.comarques
   order by area desc, id;
```

En el primer caso, se ordena de forma ascendente (de menor a mayor) según el área de la comarca (en m^2). En caso de empate se desempatada por id ascendente.

En el segundo se ordena descendente y se desempata por id ascendente.

En ambos casos se pueden observar que los alias de una columna puede utilizarse como nombre de columna.
También ss pueden utilizar operaciones dentro del *order by*.

Notas:
Por defecto order by es ascendente. Aunque se puede utilizar ASC para ser explícito.

Carácter y textos serán ordenados alfabéticamente.

3.10 Limit

Limit

Limit hace que el resultado final devuelva las primeras filas del resultado que tendría si no le ponemos límit.
Esto permite hacer una carga parcial de una tabla muy grande, y hacer testeo y pruebas con una muestra parcial.

```
select id, ST_Area(geom) as area
   from limit_admin.comarques
   order by area, id
   limit 10
;
```

Pequeño ejercicio:
¿Cuáles son las 5 comarcas de Catalunya con más área (en km²)?
¿Cuáles son los 10 municipios de Catalunya con más habitantes?

3.11 Agregación

Agregación

La agregación consiste en utilizar más de una fila para calcular un resultado final.

Los casos más habituales son.:
- Sumatorios
- Contar files
- Cálculos estadísticos (como por ejemplo la media, mín. , máx.)

Cuando se utiliza la agregación hay que estar alerta con los campos que se agregan (si se intenta utilizar una columna sin agregar, no sbrá que valor coger).

Cuando se hace una agregación, las instrucciones WHERE, ORDER BY, LIMIT, etc. afectan a los datos antes de agregarse.

```
select sum(value)
   from exemple
;
```

3.12 Group by

Group by

Agrupa las columnas en grupos, definidos por las columnas deseadas.

Todas las columnas de la tabla final han de estar definidas en los grupos, o ser una función agregativa o bien no ser utilizadas.

Esto hace que group by permita hacer agregaciones parciales según unos campos concretos.

```
select codiprov, ST_Union(geom) as geom
   from limit_admin.munis_poly
   group by codiprov
;
```

Pequeño ejercicio:
¿Cuántos municipios hay en cada comarca?
¿Cuántos habitantes hay en cada provincia de Catalunya?

3.13 Having

Having

Como ya hemos visto, *where* afecta a los valores antes de la agregación. Para hacerlo después se puede utilizar *having*.

```
select codiprov, ST_Union(geom) as geom
   from limit_admin.munis_poly
   group by codiprov
   having codiprov = '08'
;
```

Ejemplo de select simple (todo lo que hemos visto)

```
select codiprov , sum(ST_Area(geom)) as area
   from limit_admin.munis_poly
   where ST_Area(geom) > 10000000
   group by codiprov
   having codiprov in ('08','17', '25')
   order by area desc
   limit 2
;
```

3.14 Join

Join (1)

Permite agrupar dos (o más) tables en una sola en la misma query. La condición que agrupa las diferentes filas de las tables es la condición ***on***.

```
select m.nommuni, p.nomprov
   from limit_admin.munis_poly as m
      left join limit_admin.provincies as p
         on m.codiprov = p.codiprov
   group by m.nommuni, p.nomprov
;
```

Como se puede ver en el ejemplo se pueden utilizar alias por las tablas. Esto es muy útil para no tener que reescribir el nombre de la tabla entera cada vez.

```
    limit_admin.munis_poly as m
```

Join (2)

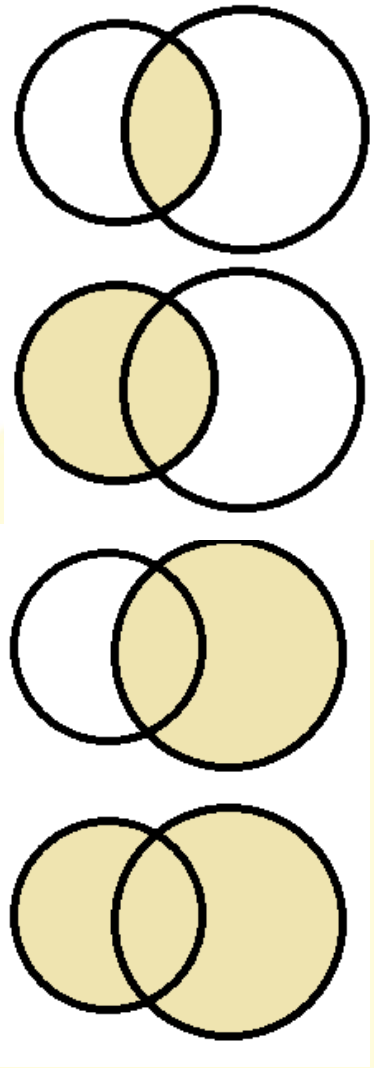
Hay diferentes maneras de cruzar las tablas. Vamos a ver caso a caso:

Inner join
Las filas se juntan solo si la condición se cumple.

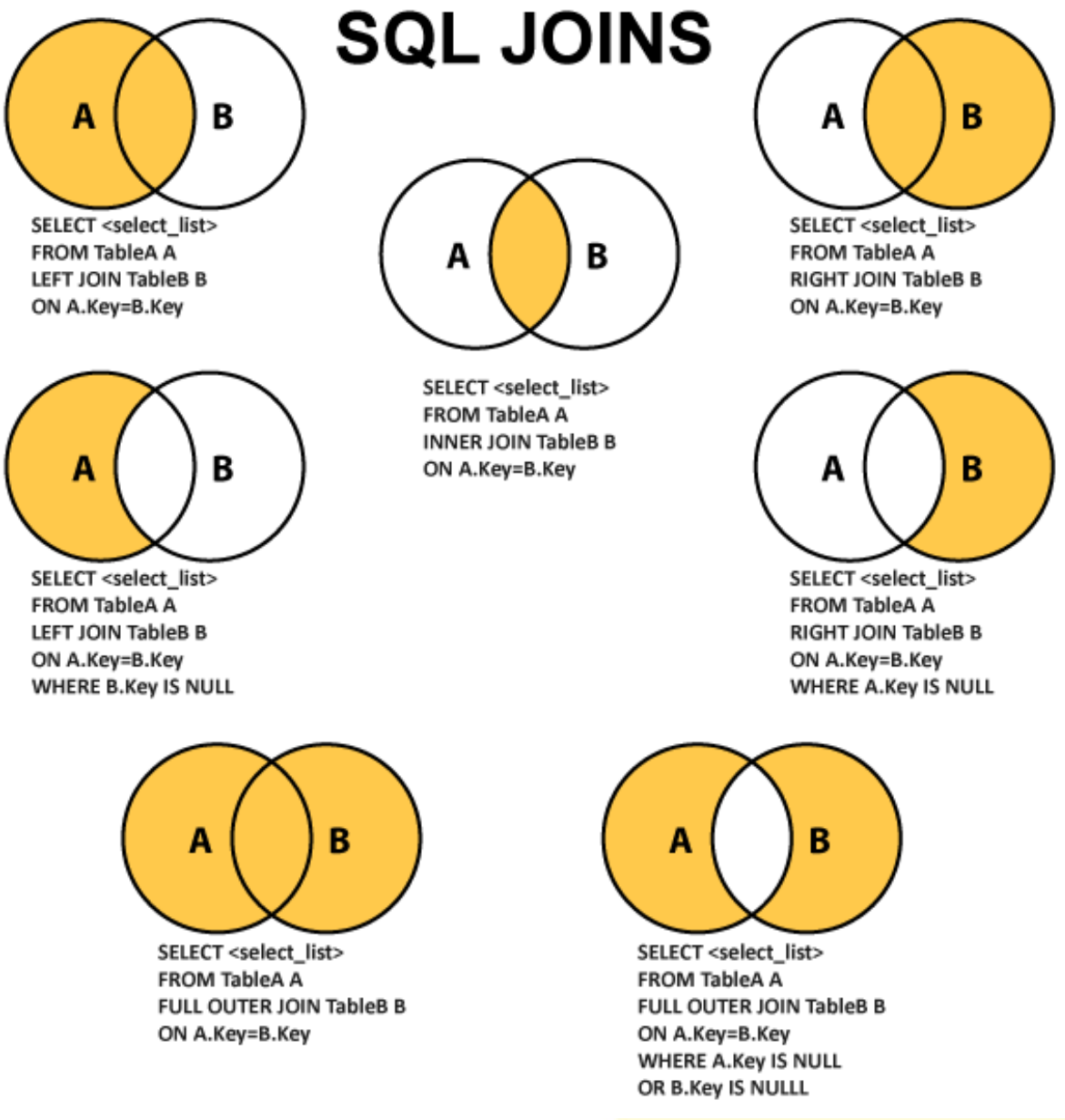
Left join
Las filas se juntan si la condición se cumple. Además se mantienen las filas de la izquierda (primera tabla) con los valores de la tabla de la derecha como null.

Right join
Equivalente al anterior pero girando el orden de las tablas.

Full join
Se muestran las filas que se pueden cruzar con la condición. Además és todas las otras (tanto las de la derecha como las de la izquierda) que no es pueden emparejar con ninguna fila, dejando los valores de la otra tabla como null (como si fuese *right join* y *left join* a la vez).



Join (3)


SQL JOINS
A
B
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key=B.Key
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key=B.Key
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key=B.Key
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key=B.Key
WHERE B.Key IS NULL
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key=B.Key
WHERE A.Key IS NULL
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key=B.Key
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key=B.Key
WHERE A.Key IS NULL
OR B.Key IS NULLL

Join (4)

Se pueden encadenar múltiples JOIN ... ON ... JOIN ... ON ... para juntar más de dos tables. La manera de como se juntaran dependerá de la relación especificada en ON.

Los joins son muy comunes en los modelos relacionales ya que los datos finales suelen estar en otra tabla (para evitar repeticiones).

Ejercicio:
¿Cuantos municipios están en el interior del Parc Natural del Montserrat?

Ejemplo:

```
select *
   from taula1 as p
      left join taula2 as s
         on p.t2 = s.id
      left join taula3 as t
         on s.t3 = t.id
   where p.id > 100
   limit 100
;
```

3.14 Subquery

Subquery (1)

Las subquerys permiten utilizar el resultado de una query en otra query (de aquí su nombre).

Estas subquerys pueden servir de tablas hijas (si es limita a una fila por cada subquery) o valores (si tienen una hija y una columna).

```
-- Provincia que tiene el municipi más pequeño
select p.id, p.nomprov
   from limit_admin.provincies as p
   where codiprov = (
      select codiprov
         from limit_admin.munis_poly
         order by ST_Area(geom) asc
         limit 1
   )
;
```

Nota:
La subquery no acaban en punto y coma (;) ya que el comando no se ejecuta hasta que la query principal se acaba.

En lugar de utilizar la subquery para hacer una tabla para un *from*, esta tendrá que tener obligatoriamente un alias (ejemplo en el siguiente apartado).

3.16 Union

Union

La instrucción UNION une las filas de los resultados de múltiples queries en la primera. Para hacerlo es necesario que las columnas tengan el mismo tipo de datos. Mantendrán el nombre de las columnas de la primera query.

Para aplicar instrucciones al resultado se tiene que usar una subquery.

```
   select 1 as resultat
union
   select 2
;
```

result at
1
2

```
select resultat
   from (
         select 1 as resultat
      union
         select 2
   ) as _
   order by resultat desc
;
```

result at
2
1

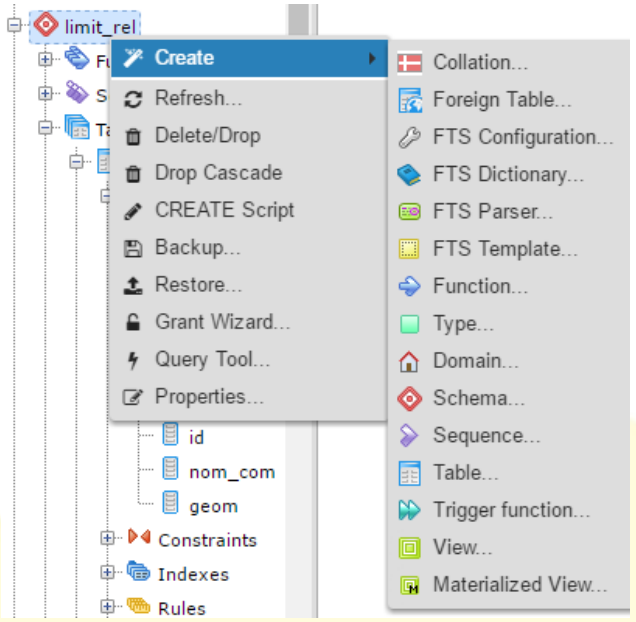
Ejercicio:
Listar los nombres de municipios y de espais del PEIN en una sola tabla

3.17 Create

Create

Crea un objecto. Cada objecto tiene su estructura particular de la instrucción.

Recomendable hacerlos desde el programa pgAdmin excepto si se quiere guardar la instrucción en un fichero o bien si haciéndolo con el pgAdmin es más lento.



Ejemple:

```
 -- create table
create schema test_schema;
create table test_schema.test (
                id serial primary key,
                geom geometry('MultiPolygon', 25831)
);
```

3.18 Alter table

Alter table

Utilizaremos esta instrucción para modificar la estructura de una tabla existente.
Modificar una columna actual o añadir una nueva.

Ejemplo:

```
 -- alter table
alter table test_schema.test add column num float;
```

	id [PK] serial	n integer	num double precis
*			

Ejercicio
Añadir una nueva columna a la capa d'espais del PEIN para guardar la superficie en hectarees

3.19 Insert

Insert

Sirve para añedir filas a una tabla. No hace falta especificar todas las columnas, solo las que interesan (excepto que la tabla tenga alguna que no pueda ser null o sin valor por defecto).

VALUES sirve para llenar las tablas literalmente. También se puede llamar con un SELECT aunque es poco habitual.

INSERT permite usar un SELECT para conseguir los datos. Esto permite hacer copias y guardar cálculos en otra tabla.

Ejemplo:

```
 -- insert into
insert into test_schema.test(n) values
   (5),
   (15)
;
```

	id [PK] serial	n integer	num double precision
1	1	5	
2	2	15	
*			

3.20 Update

Update

UPDATE actualiza valores de la tabla.

Ejemplo:

```
 -- update
update test_schema.test
   set n = 3
   where id=1
;
```

	id [PK] serial	n integer	num double precision
1	1	3	
2	2	15	
*			

Ejercicio:
Calcular la superficie de los espias del PEIN en hectarees

3.21 Delete

Delete

Elimina filas o registros de la tabla según el criterio elegido. Si no hay criterio, elimina todos los valores de la tabla (pero no la tabla en si).

Ejemplo:

```
 -- delete algunos registros
delete from test_schema.test
   where id < 2
;

-- delete todo el contenido de una tabla
delete from test_schema.test;
```

	id [PK] serial	n integer	num double precision
1	2	15	
*			

3.22 Drop & drop cascade

Drop & drop cascade

Drop elimina un objecto (tabla, función, vista, esquema, base de datos, etc.).

A veces nos podemos encontrar el caso que otro objeto necesita el que queremos eliminar. Podemos hacer dos coses:
1- Lo modificamos por que ya no necesitamos el objeto a eliminar, o
2- Lo eliminamos todo, el objeto y la dependencia.

Este segundo caso es muy habitual (por ejemplo eliminar un esquema con todo lo que tenemos dentro). Para hacerlo hay que llamar DROP CASCADE, que elimina todos los otros objetos que dependen del eliminado.
Como no requiere confirmación en la operación, hay que ir alerta cuando utilicemos DROP CASCADE ya que podría ser que elimines alguna cosa que no recuerdes que tuviese dependencia.

Ejemplo:

```
 -- drop
drop table test_schema.test;

 -- drop cascade
drop table test_schema.test
cascade;
```

3.23 Views

Views

Son querys en forma de tabla. Externamente funcionan idénticamente a las tables. Esto permite hacer tables dinámicas que obtienen los valores de otras tables.

Muy útil para hacer una capa con información y cargarla en QGIS. La información puede estar guardada en diferentes lugares pero QGIS lo ve como una sola tabla con información.

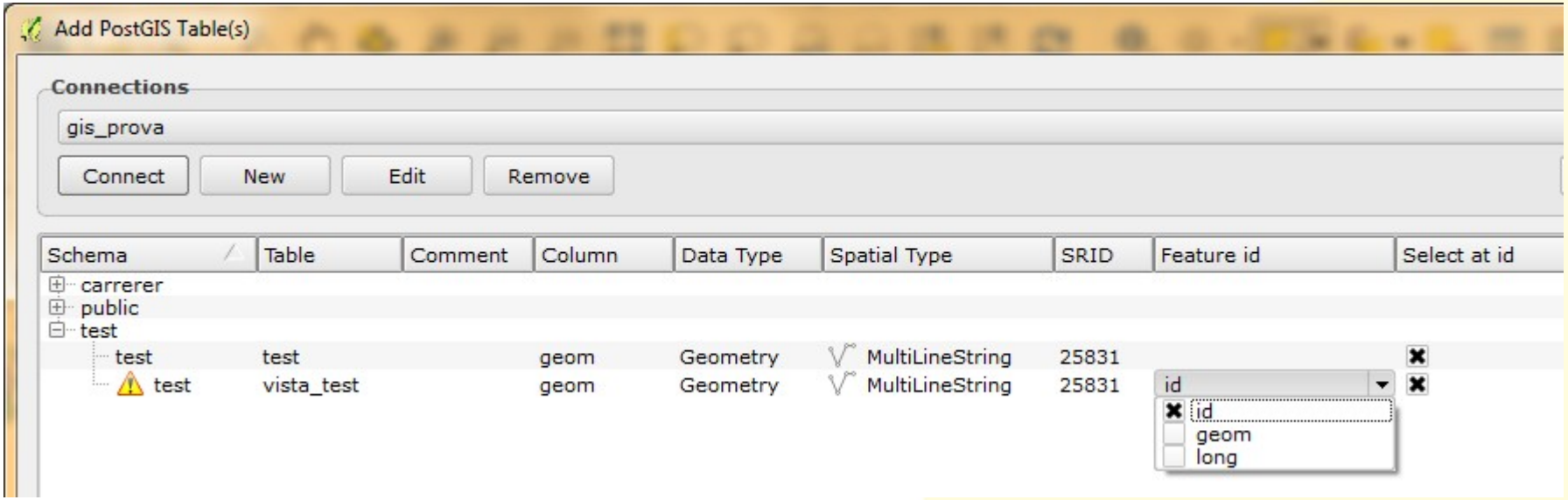
Ejemplo:

```
create view test.vista_test as
   select * from test.test
;
```

Views

Recordar que QGIS necesita que las tables y las vistas de PostgreSQL tengan una clave primaria (PK)

Las tablas siempre tienen PK, pero las vistas no, antes de añadir la vista al QGIS hay que definir que campo o campos serán la PK



Ejercicio:
Convertir en vista los municipios que intersectan con Montserrat y cargar la vista en QGIS

Índex

4. Programación

- 4.1 Funciones
- 4.2 PL/pgSQL
- 4.3 Otros lenguajes procedurales
- 4.4 Do
- 4.5 Triggers

4.1 Funciones

Funciones (1)

Durante el curso hemos trabajado con funciones de PostgreSQL y PostGIS.
Cada función hace un proceso.
Normalmente no sabemos como funciona internamente y no hace falta saberlo; solo nos interesa saber que hace.

Podemos definir nuestras propias funciones.
El formato básico es el siguiente:

```
create or replace nombre(param1 tipus1) returns tipus_r as $$
<codi>
$$ language llenguatge;
```

Por ejemplo:

```
create or replace suma(a int, b int) returns int as $$
   select a+b;
$$ language sql;
```

Los parámetros, en este caso *a* y *b*, se pueden llamar por su nombre o bien con *$n* donde *n* es la posición (seria *$1* y *$2*).

Funciones (2)

Si queremos devolver una tabla hay que definirla. Por ejemplo:

```
create or replace function prov(id bigint) returns table (
   id bigint,
   nom_prov text
) as $$
   select id, nom_prov
      from limit_rel.provincies
      where id = $1
   ;
$$ language sql;
```

Aquí utilizamos *$1* en lugar de *id* para evitar confusiones.

Las funciones con lenguaje SQL solo poden fer una sola query (con sus subquerys).
Para hacer múltiples querys se necesita un lenguaje procedural.

4.2 Programación procedural (PL/pgSQL)

Programación procedural

Utilizando un lenguaje procedural, es decir, un lenguaje que pueda ejecutar múltiples comandos uno detrás de otro, se abren posibilidades que anteriormente no teníamos o eran muy complicado de hacer.

Nosotros nos centraremos en PL/pgSQL, ya que es el lenguaje procedural propio del PostgresSQL y se parece mucho al SQL.

La estructura básica del *PL/pgSQL* es:

```
declare
   -- variables que se utilizarán
begin
   -- código
end;
```

En el caso de no utilizar ninguna variable, el apartado *declare* se omite.

Si nos fijamos en el ejemplo de la suma pero esta vez utilizando PL/pgSQL. En lugar de utilizar un *select* utilizaremos la instrucción *returns*, que devuelve el cálculo. Esto es debido a que no sabe que cálculo tiene que devolver, ya qu pueden haber diversos select.

```
create or replace suma(a int, b
int) returns int as $$
begin
   return a+b;
end;
$$ language plpgsql;
```

PL/pgSQL

PL/pgSQL añade muchas cosas al lenguaje SQL. Algunas de estas son:
- Variables
- Condicionales
- Bucles
- Tablas temporales
- Posibilidad de ejecutar strings como un comando

```
declare
  a int := 3;
  x int;
  r int := 0;
begin
  if a = 3 then
    perform a();
  elsif a = 4 then
    perform b();
  else
    perform c();
  end if;

for x in get_values() loop
  r := r+x;
end loop;

execute 'select a from ' ||
quote_indent('table'::text) || '
where x > 3;';
end;
```

4.3 Otros lenguajes

Otros lenguajes

PostgreSQL permite el uso de varios lenguajes de programación: Python, TLC, Pearl y un lenguaje propio (PL/pgSQL). También permite ejecutar funciones de librerías dinámicas (.dll / .so). Además hay un repositorio no oficial para poder utilizar otros lenguajes como el Shell (bash), Java, PHP, etc.

Estos lenguajes se tienen que instalar, pero se utilizan de manera muy similar a PL/pgSQL:

```
create or replace suma(a int, b int) returns int as $$
# codi en una (petita) variant de Python 3
# alerta amb les tabulacions
return a+b;
$$ language plpython3u;
```

4.4 Do

Do

SQL y PL/pgSQL permiten ejecutar código de otros lenguajes con la instrucción do. Esta es muy similar a la definición de las funciones:

```
do $codi1$
begin
-- código en plpgsql

do $codi2$
#código en python
$func2$ language plpython3u;

end;
$codi2$ language plpgsql;
```

Esta instrucción es muy útil para probar código de otros lenguajes como una query directamente.

4.4 Triggers

Triggers (1) = Disparadores

Cada trigger ejecuta una función que es un poco especial cada vez que una condición se cumpla.
Muy útil para automatizar cambios, puede hacer desde calcular alguna columna hasta modificar otras tablas para hacer que los valores sean consistentes.

El trigger se puede activar de tres maneras (no todos los tipos los tienen):
- Antes de la instrucción, esto permite modificar los valores que esta aplica. *(before)*
- Después de la instrucción, esto permite utilizar funciones que lean los cambios hechos. *(after)*
- En lugar de la instrucción, esta no hace ningún cambio pero se ejecuta nuestra función. *(instead of)*

El trigger se pueden ejecutar
- Por cada fila. *(for each row)*
- Una vez por sentencia. *(for each statement)*

Triggers (2)

Un ejemplo muy utilizado: calcular automáticamente el área de una geometría

```
-- create function
create or replace function test_schema.area_trigger() returns trigger as $$
begin
    new.area := ST_Area(new.geom);
    return new;
end;
$$ language plpgsql;

-- create trigger
create trigger area_trigger
before insert or update of geom
on test_schema.test
for each row
execute procedure test_schema.area_trigger();
```

Triggers (3)

Como hemos visto, la función devuelve el tipo *trigger* y, dentro la función, devuelve *new*.

new es una fila (tipo *record*) que se corresponde con la fila que está cambiando y como quedará si la modificamos con un trigger **antes** de que suceda, estamos cambiando lo que cambia.

Las funciones de trigger también tienen el tipo *old* que corresponde con la fila con los valores antes de cambiarlos.

new y *old* pueden no estar disponibles dependiendo de lo que haga el trigger. (si haces un *delete* no habrá ningún *new)*.

En el caso de utilizar un *for each statement*, no hay ni *new* ni *old*.

Triggers (4)

Una manera especial de utilizar los triggers es utilizarlos para editar el contenido de una vista. Se comporta, pues, como si la vista fuese editable.

Ejemplos:

```
-- Dada la vista
create view exemple.vista as
        select true as taula, id, geom from taula1
    union
        select false as taula, id, geom from taula2
;
```

Continua....

...

```
-- Preparar la vista para un insert
-- función
create function vista_insert() returns trigger as $$
begin
    if new.taula then
        insert into taula1(id, geom) select new.id, new.geom;
    else
        insert into taula2(id, geom) select new.id, new.geom;
    end if;
return new;
end;
$$ language plpgsql;


-- trigger
create trigger vista_insert
    instead of insert on exemple.vista
    for each row execute procedure vista_insert()
;
```

Continua....

```
-- Preparar la vista per una actualització de geometria
-- Funció
create function vista_update_geom() returns trigger as $$
begin
    if new.taula then
        update taula1
        set geom=new.geom
        where id=new.id
        ;
    else
        update taula2
         set geom=new.geom
        where id=new.id
        ;
    end if;
return new;
end;
$$ language plpgsql;

-- Fem un trigger
create trigger vista_update_geom
    instead of update of geom on exemple.vista
    for each row execute procedure vista_update_geom()
;
```

Triggers (5)

Como punto final, los triggers activan otros triggers.
Puede ser muy útil pero alerta con los bucles infinitos (el PostgreSQL no lo revisa).

(Ejercicios)

- Crear un Trigger para actualizar el área de una capa de polígonos
- Crear un Trigger para añadir las coordenadas X e Y en 2 campos para una capa de puntos

Índice

5. Gestión de la base de datos

- 5.1 Nomenclatura
- 5.2 Permisos
- 5.3 Roles
- 5.4 Integridad de los datos
- 5.5 Eficiencia (indexación)
- 5.6 Seguridad

5.1 Nomenclatura

Nomenclatura

Una de las cosas que normalmente está infravalorada y que afecta mucho la manera de trabajar en un proyecto con SQL o cualquier otro lenguaje de programación, es la nomenclatura.

La nomenclatura es en programación todas las normas de estilos de escritura del código. Habitualmente se pueden elegir entre diferentes estilos de código. El que se elija es una decisión de los desarrolladores. Eso si: hay que mantener la consistencia en todo el proyecto, ya que cambiar a menudo de estilo hace que sea más difícil leerlo incluso, puede introducir errores.

En SQL antiguamente se escribían las instrucciones en mayúscula y los nombres en minúscula. Actualmente, sobretodo si se utiliza un editor que tenga colores, y si además se trabaja en un proyecto un poco grande, muchas veces se escribe todo en **minúsculas,** ya que es más rápido. Los nombres se escriben en minúsculas para evitar problemas con la interpretación de los nombres con mayúsculas del PostgreSQL.

5.2 Permisos

Permisos

PostgreSQL tiene un sistema de permisos propio (y diferente a otras bases de datos como Access o Oracle).

Los permisos pueden ser otorgados (o quitados), entre otros, a los siguientes objetos:
- Base de datos
- Esquema
- Tabla o vista
- Función

Cada objete tiene ciertos permisos y funcionan de maneras diferentes.

Para otorgar permisos a un usuario o grupo se utiliza **GRANT**.
Para quitar permisos se utiliza **REVOKE**.
Los dos tienen la misma estructura.

Es recomendable utilizar pgAdmin para hacer las gestiones que sean necesarias.

Nota:

Para ver todas las opciones buscar GRANT en la documentación de PostgreSQL.

5.3 Roles

Roles

Un rol es a lo que otorgan permiso. Puede ser un usuario, un grupo o los dos a la vez.

Normalmente se diseñan los permisos pensando en grupos y usuarios. Los usuarios se pueden conectar, y pueden tener permisos propios. En muchas ocasiones es mucho más útil hacer que haya ciertos grupos que tengan ciertos permisos y hacer que el usuario pertenezca al grupo, especialmente si el acceso es personal personal y hay muchos usuarios. Además este sistema permite hacer grupos de permisos que son más fáciles de manejar.

Internamente, para PostgreSQL, la única diferencia entre un usuario y un rol es que el usuario tiene la posibilidad de conectarse. Técnicamente puedes hacer que un usuario se le otorgue los permisos de otro usuario pero suele ser mala idea.

Igual que con los permisos es recomendable utilizar el pgAdmin.

5.5 Integridad de los datos

Integridad de los datos

Dentro de una base de datos nos interesa asegurarnos de que todos los datos que haya sean posibles y tengan sentido. Es mejor que se queje antes de guardarlo que alguna cosa funcione mal posteriormente, o que no hayan errores más adelante. Por esto, se pueden generara restricciones que no permitan añadir o actualizar los datos si estos no cumplen las restricciones.

Hay diferentes tipos de restricciones en una o más columnas:
- Not null: El valor de la columna no puede ser null.
- Unique: El valor no puede ser repetido en una otra fila.
- Clave primaria (primary key): Identificador de la fila, implica la restricción not null y unique.
- Clave foránea (foreign key): La(as) columna(as) hace referencia a una clave primaria de otr tabla. Esta ha de existir.
- Check: una condición (booleana) a cumplir. (ej: valor > 100).

Estas restricciones no son excluyentes y se pueden añadir juntas.

Además, se puede definir un valor por defecto que, si se quiere, puede generarse con una función. Si no se especifica este valor es null.

Ejemplo de restricciones:

```
create table exemple (
  id serial primary key,
  m int not null references
model(id),
  valor float check (valor > 0)
);
```

Serial (y las variantes bigserial y smallserial) son especiales. En realidad son de tipo int (bigint, smallint) pero generan una serie y poseen un valor por defecto que genera un autoincremento del valor de la tabla, evitando valores repetidos.

Este valor por defecto se puede ver en el QGIS si la geometría tiene etiqueta que muestra la id y todavía no se ha guardado.

5.6 Eficiencia

Eficiencia (1)

El tema de la eficiencia es muy extenso y complicado, ya que se necesita de conocimientos de como funciona una base de datos. No obstante, vamos a ver como ver o buscar para hacer que tu base de datos sea más eficiente.

La primera herramienta que hay que conocer es la instrucción EXPLAIN, especialmente su variante EXPLAIN ANALYZE. Esta instrucción permite analizar querys, analiza el rendimiento de las diferentes partes y cuanto tarda en ejecutarse. Esto permite entender las querys para modificarlas y hacerlas más eficientes.

La segunda herramienta son los INDEX. Esta herramienta guarda los datos de alguna columna de una tabla para hacer comparaciones y búsquedas más rápido.
También existe el INDEX SPATIAL, muy recomendable aplicarlo a las capas

Eficiencia (2)

En el cas de las funciones, se pueden añadir *categories* que ayudan al planificador a hacer las sentencias más eficientes. Hay tres que son muy útiles:

Strict
La función devuelve null si uno de sus parámetros es null.

Immutable/stable/volatile
Estos argumentos sirven para decidir como tratará la optimización de la función.

Immutable siempre devuelve el mismo si tiene los mismos argumentos. Normalmente funciones matemáticas.

Stable devuelve lo mismo siempre que no se modifiquen los valores de la base de datos. Normalmente selects y operaciones.

Volatile (por defecto) cada vez puede devolver un valor diferente. Por ejemplo si se utiliza la hora actual en la función.

```
create or replace suma(a int, b
int) returns int as $$
begin
   return a+b;
end;
$$
  language plpgsql
  strict
  immutable
;
```

Eficiencia (3)

Parallel safe/parallel restricted/parallel unsafe
Permite especificar si se puede y como ejecutar la función en paralelo. Hacerlos mal puede generar errores extraños o resultados incorrectos.

Si solo es una operación matemática o un select de una tabla, y solo es una tabla normal, se puede utilizar parallel safe sin preocuparse de que esté mal. Cualquier llamada a una función que no sea parallel safe puede comportar problemas.

```
create or replace suma(a int, b
int) returns int as $$
begin
   return a+b;
end;
$$
  language plpgsql
  strict
  immutable
  parallel safe
;
```

5.7 Seguridad

Seguridad

En una base de datos es muy importante la seguridad. No solo por ataques exteriores, sinó también para evitar errores humanos. Por defecto PostgreSQL es muy seguro, pero hay unas cuantas acciones que se pueden hacer para mejorar la seguridad:

- Una cuenta por persona (no compartir el usuario).
- Tener contraseñas complicadas y guardarlas en **KeePass2** o LastPass.
- Solo dar permisos a aquello que sea necesario hacer. Ni más ni menos.
- Revisar los permisos periódicamente.
- Quitar los permisos cuando alguien deje de trabajar en alguna cosa.
- Utilizar encriptación.
- Utilizar **pg_service.conf**, es mejor que tenerlo en el proyecto y permite crear uno para cada usuario.
- Para lecturas para aplicaciones utilizar un usuario de la aplicación donde se ejecuten funciones con las sentencias dentro.

Índice

6. Herramientas

6.1 pg_dump (copias de seguridad)
6.2 ogr2ogr (GDAL)
6.3 Audit
6.4 pgRouting

6.1 pg_dump (copias de seguridad)

pg_dump (copias de seguridad)

Herramienta propia de PostgreSQL que permite guardar datos de la base de datos en diferentes formatos, normalmente preparados para cargarlos en otra base de datos.

Es una herramienta muy útil para hacer copias de seguridad.

https://www.postgresql.org/docs/13/app-pgdump.html

6.2 ogr2ogr (GDAL)

ogr2ogr (llibreria GDAL)

http://www.gdal.org/ogr2ogr.html

En la librería GDAL tenemos una herramienta muy potente para hacer importaciones o cargas de datos de forma masiva, con muchos parámetros y opciones.

Las opciones más interesantes son las siguientes:

-f format_name:
output file format name (default is ESRI Shapefile), otros valores:
    -f "ESRI Shapefile"
    -f "TIGER"
    -f "MapInfo File"
    -f "GML"
    -f "PostgreSQL"
-append: Append to existing layer instead of creating new
-overwrite: Delete the output layer and recreate it empty
-update: Open existing output datasource in update mode rather than trying to create a new one
-nln name: Assign an alternate name to the new layer

Ejemplo:
ogr2ogr -append -f PostgreSQL PG:dbname=warmerda abc.tab

```
REM Declarem variable PG de sortida
SET PG="host=localhost port=5432 dbna
SET srid="25831"
REM Capes altimetria
ogr2ogr -skipfailures -a_srs EPSG:%sr
ogr2ogr -skipfailures -a_srs EPSG:%sr
ogr2ogr -skipfailures -a_srs EPSG:%sr
ogr2ogr -skipfailures -a_srs EPSG:%sr
REM
pause
```

6.3 Audit

Audit

Es una herramienta que genera triggers automáticos para guardar todo lo que se hace en una tabla, funciona como un log.

https://wiki.postgresql.org/wiki/Audit_trigger

6.4 pgRouting

pgRouting

Es una librería para hacer todo tipo de cálculo de rutas.
http://pgrouting.org/

pgRouting library contains following core features:

- Dijkstra Algorithm
- Johnson's Algorithm
- Floyd-Warshall Algorithm
- A* Algorithm
- Bi-directional Algorithms * Bi-directional Dijkstra * Bi-directional A*
- Traveling Sales Person
- Driving Distance
- Turn Restricted Shortest Path (TRSP)
- many more!!!

Gracias por vuestra asistencia

Bases de Datos Espaciales con PostGIS y QGIS

PSIG

Implementació, gestió i formació SIG

Carlos López Quintanilla
Consultor SIG

+34 699 680 261
carlos.lopez@psig.es
www.psig.es