



El complemento Adobe Flash Player no está actualizado

### **objetivos**

Es el momento de administrar un SGBD. Aquí veremos sólo los conceptos básicos que considere que os puede ir bien conocer.

Pienso que es un SGBD muy potente y flexible, y además es software libre.

Los objetivos concretos de este tema serán:

- Instalar el servidor PostgreSQL.
- Arrancar y parar la Base de Datos.
- crear usuarios
- Hacer copias de seguridad de la Base de Datos, y saber recuperarla en caso de necesidad.

### **conocimientos previos**

En este tema instalaremos y administraremos PostgreSQL sobre Linux. Harán falta, por tanto, unas nociones básicas sobre Linux.

Quien quiera también puede instalar el servidor en Windows o en otro Sistema Operativo. En este sentido también se verá la instalación en Windows 7. Caldrán evidentemente unas nociones básicas sobre el Sistema Operativo elegido para instalar el servidor.

Sin embargo, se recomienda utilizar una máquina virtual para evitar problemas.

Ahora que vamos a administrar PostgreSQL, podríamos plantearnos la conveniencia o no de utilizar PostgreSQL, sobre todo comparándolo con el otro SGBD más utilizado en Linux (y que antes era totalmente software libre), MySQL

Definiendo de forma muy rápida, PostgreSQL es un SGBD Relacional que incorpora conceptos de objetos. Vamos a refinar un poco esto.

Lo primero de todo es que es un SGBD Relacional, que ofrece apoyo prácticamente total al estándar SQL 92 / SQL 3, y que implementa perfectamente la integridad referencial, las restricciones, disparadores, transacciones, ...

Trate de ser muy flexible a la extensibilidad, y así permite al usuario definir tipos de datos nuevos, operadores nuevos, ...

Y sobre todo, incorpora también conceptos de BD orientadas a objetos. No quiere decir que sea un SGBD orientado a objetos, sino que incorpora cosas de estos, como son las clases y la herencia.

Algunos autores denominan a este tipo como SGBD objeto-relacionales (ORDBMS)

El origen de PostgreSQL fue un proyecto de la Universidad de Berkeley. Le dieron el nombre de Postgres y comenzó en 1986. En 1987 se lanzó la versión 1, que fue mejorando hasta la versión 4.2.

En 1994 surgió un descendiente de Postgres de dominio público y código abierto, llamado Postgre95, que además mejoraba cosas y era más eficiente. Además, el lenguaje de consultas pasó del original *Postqual* a *SQL* .

Posteriormente, como que el nombre no se aguantaba, en 1996 se pasó a **PostgreSQL** para reflejar la relación entre el SGBD original y las versiones más recientes con capacidades SQL. Las versiones partieron de la 6.0, para volver con la secuencia original.

En Linux, o mejor dicho, en el mundo del software libre, se utilizan principalmente dos SGBD: **MySQL** y **PostgreSQL** .

- En el primer intento por encima de todo que sea muy rápido y que gaste pocos recursos. Para conseguir esto incluso sacrifica cosas impensables en otros entornos (integridad referencial, subconsultas, transacciones, ...). Se ha impuesto en la creación de páginas web (formando la trilogía Apache - Php - MySQL), donde se busca sobre todo velocidad. Sin embargo en las últimas versiones empieza a incorporar algunos de los conceptos antes mencionados, sobre todo a partir de la versión 5.
- PostgreSQL podríamos definirlo como "más serio", y aunque pierde velocidad respecto a MySQL, puede ser perfectamente un SGBD para lugares donde la seguridad y la solvencia es algo más que anecdótica.

Hay muchas comparativas en Internet de los dos SGBD, y normalmente la conclusión no es que uno sea mejor que el otro, sino que depende del uso que se quiera hacer, contestando más bien la pregunta: **velocidad o potencia?**

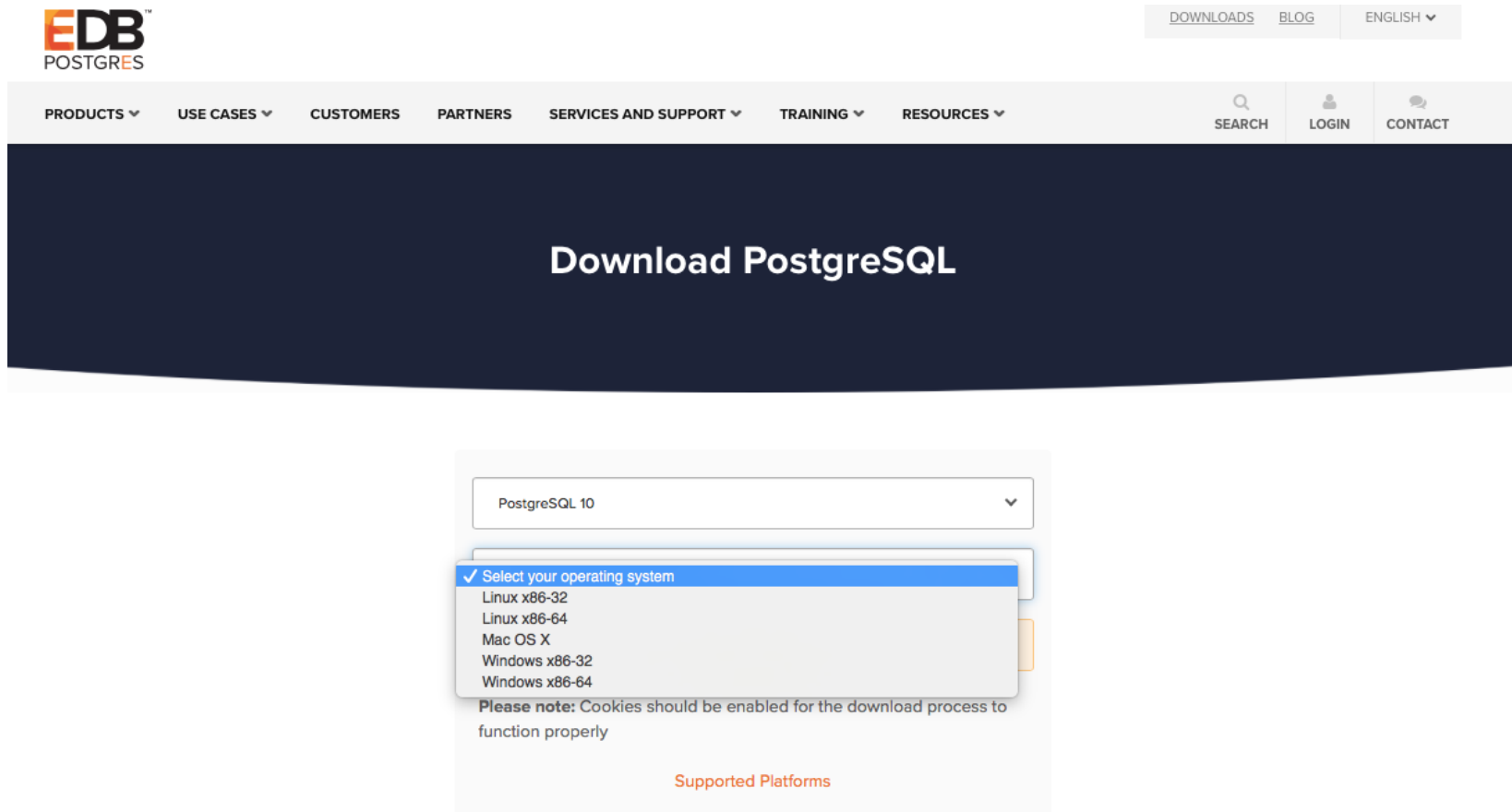
La instalación se puede realizar además de un nivel: instalación sencilla (en Linux y Windows), instalación por paquetes (también muy sencilla) y la instalación completa, compilando las fuentes (sólo en Linux, evidentemente).

Nosotros sólo vamos a ver la instalación sencilla en el Sistema Operativo elegido. En caso de duda, **os aconsejo que instale en una máquina virtual**.

## 2.1 Instalación automática en Linux

A partir de las últimas versiones, PostgreSQL ofrece la posibilidad de hacer una instalación automática, al estilo de Windows, ejecutando un programa de instalación.

De hecho, si accedemos a la página PostgreSQL, en la sección de Downloads ( [www.postgresql.org/download](http://www.postgresql.org/download) ) es el primero que nos ofrece, en las versiones de Sistemas Operativos más habituales.



El siguiente video muestra todos el proceso (para la versión 9.4.4 que era la última en el momento de realizar este vídeo.) Donde todo se puede hacer por defecto excepto la contraseña del usuario de la Base de Datos **postgres** , que será el Administrador de la Base de Datos (DBA). Os propongo la contraseña **postgres** .



El complemento Adobe Flash Player no está actualizado

Para la versión 9.6:

El lugar donde por defecto se guardan los programas y configuraciones es / **opt / PostgreSQL / 9.6 (9.6 o la versión que instalamos)**

La Base de Datos creada (Base de Datos grande, al estilo de Oracle, no al estilo de Access) por defecto se instala en / **opt / PostgreSQL / 9.6 / fecha**

Y durante el proceso de instalación se ha creado un usuario de Sistem Operativo llamado **postgres** , que será el propietario de todo el de PostgreSQL

En la Base de Datos sólo habrá un usuario, llamado también **postgres** , que como hemos comentado antes será el DBA

**Nosotros instalaremos la versión indicada en nuestro curso.**

## 2.2 Instalación automática en Windows

PostgreSQL se puede instalar perfectamente en cualquier versión de Windows XP, 7, 8 ....

En Windows Vista se debe desactivar un servicio primero, para que no hayan problemas. Si alguien lo está utilizando (cosa muy improbable) y tiene problemas, debería desactivar UAC (User Account Control), aunque sea temporalmente.

En estos apuntes mostraré la instalación en Windows 7, donde no debe haber ningún problema.






La instalación en Windows es prácticamente igual que la de Linux, y por tanto los comentarios serán similares. Tomaremos el archivo de instalación del mismo lugar ( <http://www.postgresql.org/download> ). La versión disponible es prácticamente la misma que en Linux

Durante la instalación, como en Linux, todo lo pondremos por defecto, excepto la contraseña del usuario **postgres** , que pondremos **postgres** .

### PostgreSQL Installers

These enterprise-class 64-bit PostgreSQL binaries are always free and Open Source. They are tested to run on Centos 6+, Ubuntu 12.04+, OSX 10.9+, Windows 7+ and Windows Server 2008+.

**Featured!** Command line Package Manager [usage instructions](#) for all versions of Postgres.

PostgreSQL 9.6.2 - Stable (09-Feb-17)	PostgreSQL 9.5.6 - Proven (09-Feb-17)
 <a href="#">postgresql-9.6.2-win64.exe</a>	 <a href="#">postgresql-9.5.6-win64.exe</a>
 <a href="#">postgresql-9.6.2-osx64.dmg</a>	 <a href="#">postgresql-9.5.6-osx64.dmg</a>
 <a href="#">postgresql-9.6.2-x64.rpm</a>	 <a href="#">postgresql-9.5.6-x64.rpm</a>
 <a href="#">postgresql-9.6.2-x64.deb</a>	 <a href="#">postgresql-9.5.6-x64.deb</a>

El siguiente video muestra todos el proceso (para la versión 9.4.5 que era la última en el momento de realizar este vídeo.)

**Nosotros instalaremos la versión indicada en nuestro curso.**





El complemento Adobe Flash Player no está actualizado

Para la versión 9.6:

El lugar donde por defecto se guardan los programas y configuraciones es **C: \ Archivos de programa (x86) \ PostgreSQL \ 9.6** (o su traducción: **C: \ Archivos de Programa (x86) \ PostgreSQL \ 9.6** )

La Base de Datos creada (Base de Datos grande, al estilo de Oracle, no al estilo de Access) por defecto se instala en **C: \ Archivos de programa (x86) \ PostgreSQL \ 9.6 \ fecha**

Y en la Base de Datos sólo hay un usuario de momento, el DBA, llamado también **postgres** (con contraseña **postgres** )

Los entornos de trabajo serán los programas que nos permitirán conectar e incluso administrar la Base de Datos.

- El **psql** va incorporado en PostgreSQL, y por lo tanto siempre está disponible. Es un editor de comandos SQL en modo línea.

Los otros dos entornos son gráficos.

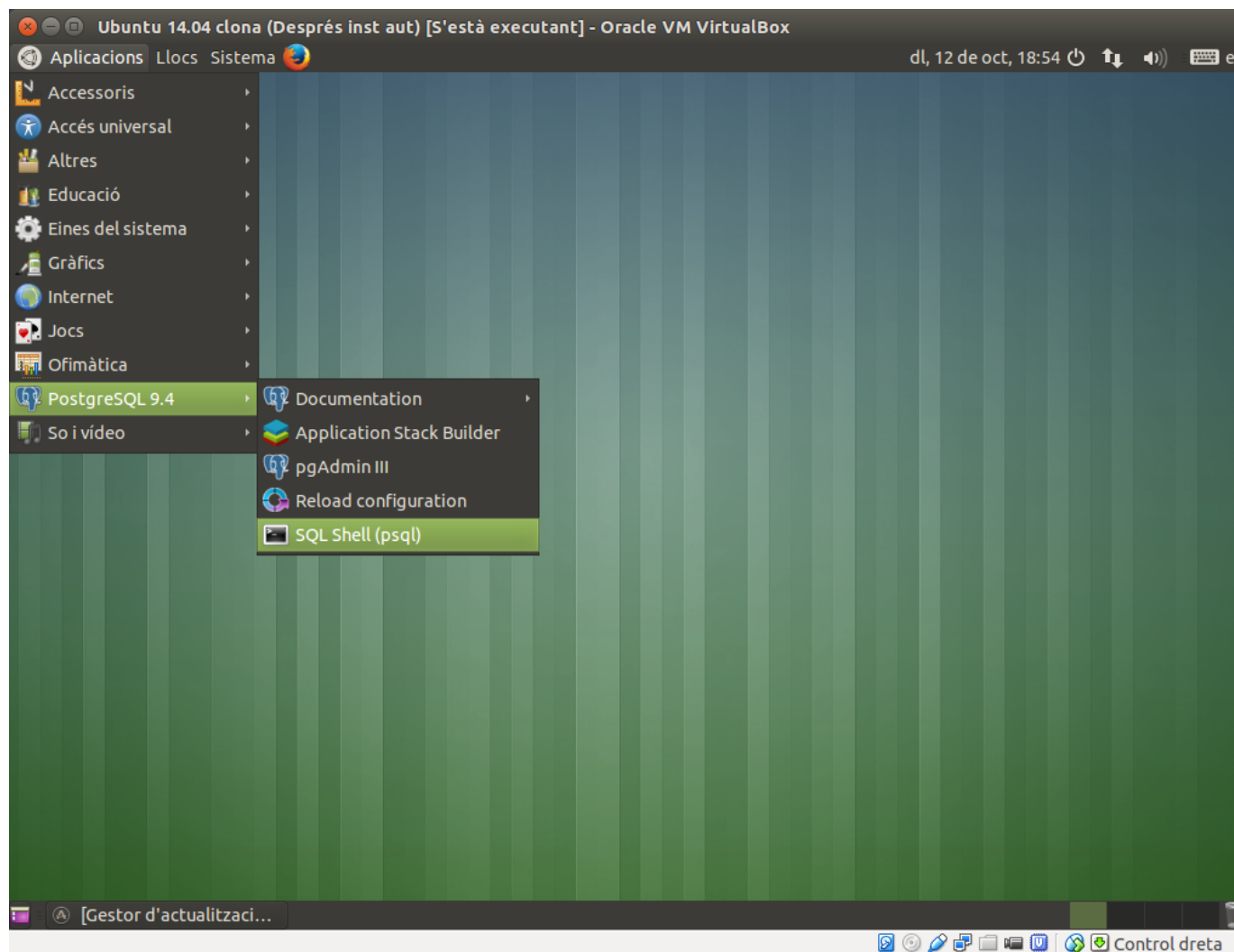
- El **PGADMIN** es el más habitual para administrar PostgreSQL.
- El **phpPgAdmin** es un conjunto de páginas PHP. Permite por tanto administrar desde cualquier lugar (si el servidor también tenemos un servidor web). Es prácticamente igual que el phpMyAdmin, que se utiliza muchísimo para administrar MySQL, y por lo tanto es muy conocido.

El programa más sencillo que podemos utilizar para conectarnos con el servidor y ejecutar las sentencias SQL es el **psql**, que sencillamente es un intérprete de sentencias SQL donde iremos poniendo los comandos y nos irá dando los resultados. Es, por tanto, bastante incómodo, pero como forma parte de PostgreSQL ya está instalado y siempre funciona.

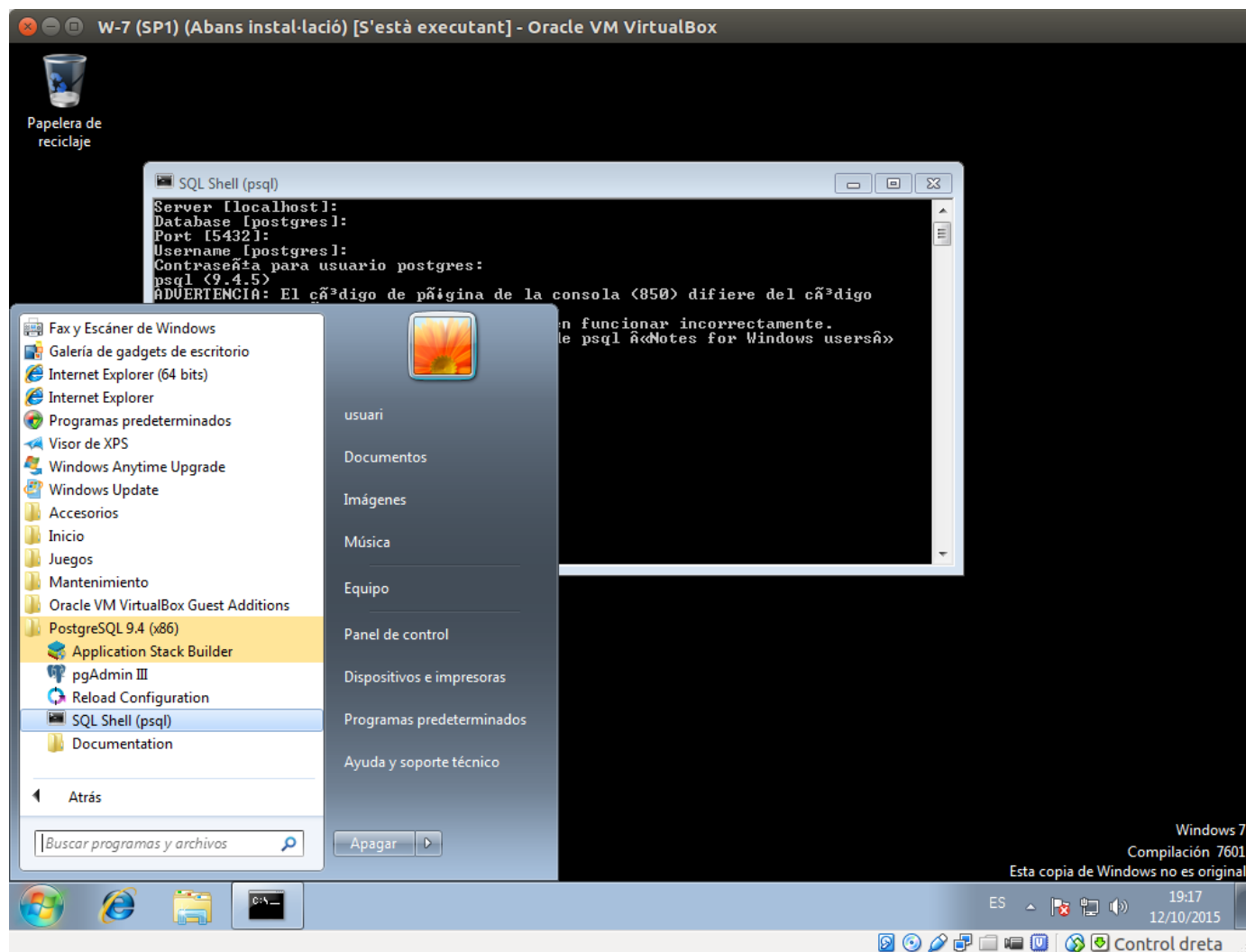
### **Nota**

Observe como ahora nos estamos conectando desde el mismo servidor

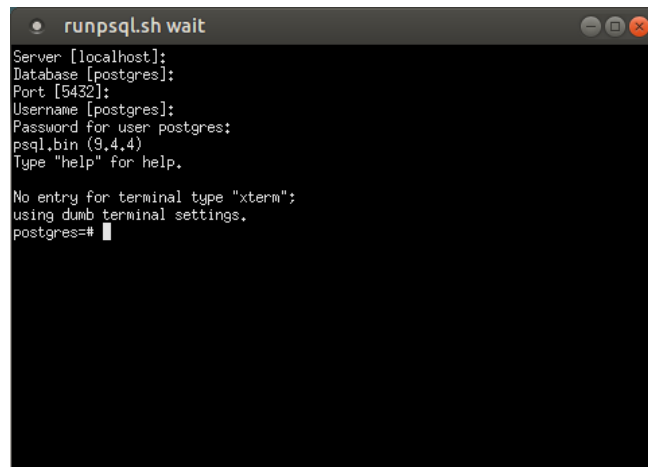
En la instalación que habíamos hecho tenemos en el menú un acceso directo.



Y desde Windows se haría de forma totalmente similar:



En realidad no es un acceso directo a psql, sino a un script que pide unos parámetros (usuario, Base de Datos, servidor, ...) y después llama a psql. Lo podemos dejar todo en blanco, excepto la contraseña (postgres)



```
runpsql.sh wait
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql.bin (9.4.4)
Type "help" for help.

No entry for terminal type "xterm";
using dumb terminal settings.
postgres=#
```

De todos modos puede ser muy conveniente conocer un poco tanto los parámetros de psql como los comandos, una vez hemos conectado. Recuerde que el lugar donde está situado es:

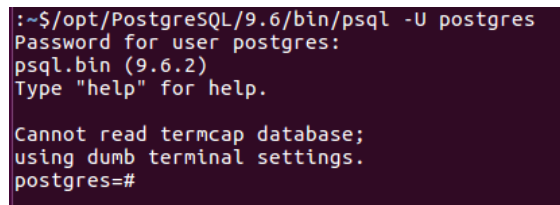
- En Linux: `/opt/PostgreSQL/9.6/bin/psql`
- En Windows: `c: \ Archivos de programa (x86) \ PostgreSQL \ 9.6 \ bin \ psql.exe`

#### Desde un terminal

Para conectar directamente como usuario postgres:

```
$ /opt/PostgreSQL/9.6/bin/psql -U postgres
```

tal y como nos muestra la siguiente imagen:



```
:-$/opt/PostgreSQL/9.6/bin/psql -U postgres
Password for user postgres:
psql.bin (9.6.2)
Type "help" for help.

Cannot read termcap database;
using dumb terminal settings.
postgres=#
```

Para conectarnos como distintos usuarios, o bases de datos distintas, tendremos que jugar con las opciones

El formato total es el siguiente:

```
psql [opciones] ... [base-de-datos [usuario]]
```

En las opciones podemos poner, entre otras:

- ? ayuda
- d *nom\_bd* nombre de la base de datos (no hay que poner -d)
- c para ejecutar tan sólo un comando

<b>-f <i>archivo</i></b>	para ejecutar las sentencias de su archivo
<b>-l</b>	para listar las Bases de Datos existentes
<b>-h <i>nombre</i></b>	nombre o dirección del servidor al que nos conectamos (por defecto local)
<b>-p <i>puerto</i></b>	puerto del servidor a través del cual nos conectamos (por defecto 5432)
<b>-U <i>nombre</i></b>	nombre del usuario
<b>-W</b>	para que nos pida obligatoriamente la contraseña.

A partir de ahora ya podremos poner las sentencias SQL, que pueden ocupar más de una línea y que deben finalizar para ;

Con las teclas de movimiento de cursor podemos ir a las sentencias anteriores (pero sólo una línea) para poder modificar algo y volver a ejecutar.

Aparte podremos poner algunos comandos (ya dentro de **psql**):

<b>\ C <i>nom_bd</i></b>	para conectar a otra Base de Datos
<b>\ e</b>	invoca un editor (por defecto <b>vino</b> en Linux y <b>notepad</b> en Windows) para editar la última sentencia
<b>\ l</b>	lista de las Bases de Datos existentes
<b>\ d</b>	lista de las tablas
<b>\ d <i>tabla</i></b>	descripción de la tabla
<b>\ g [ <i>archivo</i> ]</b>	ejecuta la actual sentencia y, en todo caso, envía el resultado al archivo
<b>\ y <i>archivo</i></b>	ejecuta la sentencia (o sentencias) del archivo
<b>\ w <i>archivo</i></b>	guarda en el archivo la sentencia del buffer
<b>\ q</b>	salir de psql

Aunque normalmente en una consulta SQL se debe poner la mesa o mesas que proporcionan los datos, PostgreSQL permite no poner ningún origen de datos. Esto nos permitirá hacer consultas para realizar cálculos utilizando los distintos operadores y funciones.

Por ejemplo, para que nos de la fecha de hoy:

```
SELECT NOW ();
```

### 3.1.1 Práctica guiada 1

La realización de esta práctica es obligatoria

En esta práctica guiada vamos a introducir unos datos de prueba, a partir de un archivo llamado **dades\_geo.sql** (el archivo lo tenéis en el curso). En realidad se creará un nuevo usuario **geo** y una nueva Base de Datos **geo** . En esta Base de Datos es donde se insertarán los datos. Primero deberá entrar en **psql** como usuario **postgres** . Después tendrá que ejecutar el comando (vea que la ruta puede cambiar en el servidor):

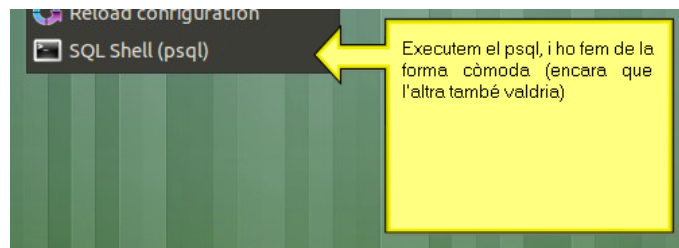
```
\ Y /home/administrador/Baixades/dades_geo.sql
```

1. Esta práctica consistirá en importar unos datos para poder practicar mínimamente. Lo más cómodo es trabajar en una máquina virtual

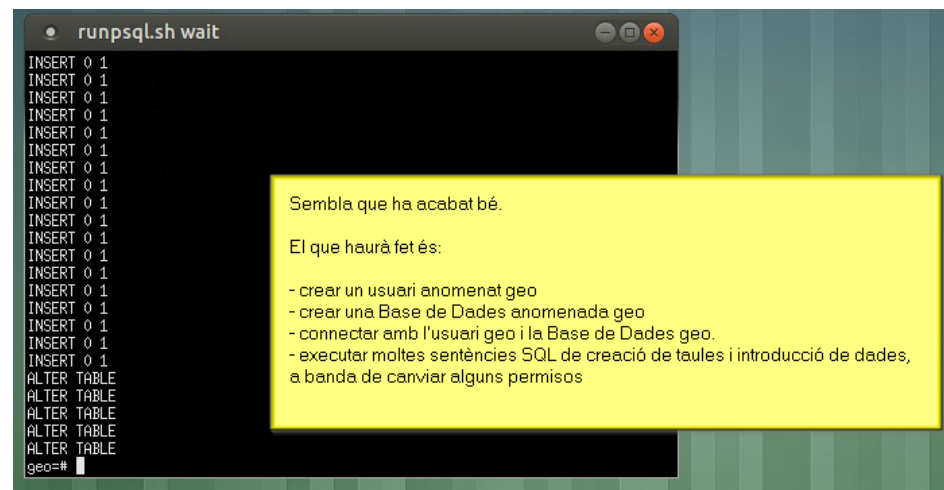
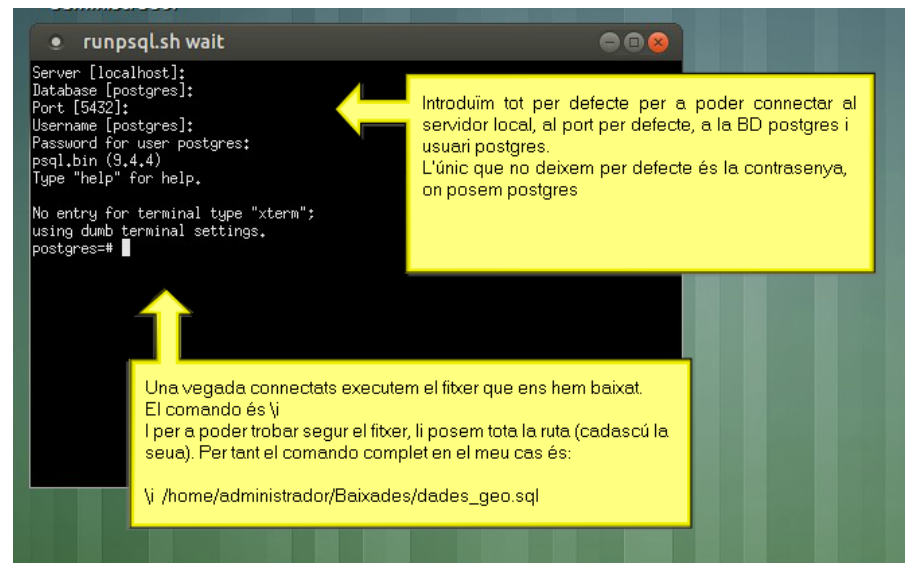
Los datos están en nuestro curso en el apartado - **recursos** - del tema. Guardamos el archivo para tenerlo disponible.



2. Ejecutamos el psql o desde un terminal







```
runsql.sh wait
(4 rows)
geo=# \l
      List of databases
  Name | Owner | Encoding | Collate | Ctype | Access privileges
-----+-----+-----+-----+-----+-----
 geo   | geo   | UTF8      | ca_ES.UTF-8 | ca_ES.UTF-8 |
 postgres | postgres | UTF8      | ca_ES.UTF-8 | ca_ES.UTF-8 |
 template0 | postgres | UTF8      | ca_ES.UTF-8 | ca_ES.UTF-8 | =c/postgres +
         |         |           |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8      | ca_ES.UTF-8 | ca_ES.UTF-8 | =c/postgres +
         |         |           |             |             | postgres=CTc/postgres
(4 rows)

geo=# \d
      List of relations
 Schema | Name      | Type | Owner
-----+-----+-----+-----
 public | comarques | table | geo
 public | instituts | table | geo
 public | poblacions | table | geo
(3 rows)

geo=#
```

Amb \l podem veure les Bases de Dades existents (Bases de Dades equivalents als esquemes d'Oracle o les Bases de Dades d'Access)

Amb \d podem veure les taules de l'actual Base de Dades (geo)

### ejercicio 1

Después de haber hecho la práctica guiada anterior, en la que hemos introducido los datos de prueba:

Cuántas filas hay en la mesa **poblaciones** de la Base de Datos **geo** ?.

Como el psql es tan árido, prácticamente se impone la utilización de alguna herramienta que haga al menos más agradables las cosas. Una de las más completas es **PGADMIN** . Nos permitirá disponer de editores SQL y de procedimientos mucho más agradables, además de la posibilidad de poder administrar la Base de Datos desde aquí.

Con la instalación automática de PostgreSQL ha instalado también pgAdmin 4

Podemos llamar el programa desde una terminal ( `/opt/PostgreSQL/9.6/pgAdmin 4 / bin / pgadmin4` ), pero siempre será más cómodo buscarlo con el ratón:

- En Ubuntu: **Aplicaciones** → **PostgreSQL 9.6** → **pgAdmin 4** .
- En Windows: **Inicio** → **Todos los programas** → **Postgresql 9 .6** → **pgAdmin 4** .

Tenemos varias maneras de poder arrancar o parar el servidor.

- **Automática** .

La inmensa mayoría de las veces el arranque y parada del sistema se hará de forma automática, cuando arrancamos o esperamos el SO. Esto lo conseguimos para tener el script en **/etc/init.d** , (que se pone automáticamente durante la instalación por paquetes, pero no en la instalación a mano) y por tener en los niveles de ejecución correspondientes los **start** y **kill**

- **Script en /etc/init.d** (sólo en Linux).

Lo que siempre podremos hacer, si estamos en Linux, es aprovechar el **script** situado en **/etc/init.d** para arrancar, parar, rearrancar y ver el estado (como cualquier otro servicio). Lo haremos así (fíjese que hacemos **sudo** para ejecutarlo como **root**):

```
$ Sudo /etc/init.d/postgresql-9.6 opción
```

donde la opción puede ser:

<b>start</b>	arranca el servidor
<b>stop</b>	lo para
<b>restart</b>	rearrancar (el para, y lo vuelve a poner en marcha)
<b>condrestart</b>	igual que el anterior, pero sólo en caso de que esté en marcha (si estaba parado, no hace nada)
<b>status</b>	ver el estado en que se encuentra actualmente (en marcha, parado, ...), y otras informaciones de interés

- Hay más formas, pero no las veremos aquí para exceder el fin que buscamos

### ejercicio 2

1.- Escribe exactamente lo que contesta la instrucción y adjunta una imagen

```
/etc/init.d/postgresql-9.6 status
```

## 5. Archivos de configuración

---

Básicamente son 3 los archivos de configuración:

- **postgresql.conf** donde están la mayor parte de parámetros
- **pg\_hba.conf** para controlar las conexiones
- **pg\_ident.conf** se combina con la anterior para las conexiones

Si hemos hecho una instalación automática, tanto en Linux como en Windows, los archivos se encontrarán en el directorio de datos, que en Linux es ***/opt/PostgreSQL/9.6/data*** , y en Windows es ***C: \ Archivos de programa ( x86) \ PostgreSQL \ 9.6 \ data*** .

### Nota

Para poder editar estos archivos, no habrá problemas desde Windows, pero desde Linux debe tener permiso. Deberá conectarse como superusuario ( **sudo su** ), y después, o bien utilizar **vino** , gedit, etc o bien ejecutar el explorador de archivos, con el que podrá navegar por todo el sistema de archivos (ya que el ha ejecutado como root), y cuando encuentra el archivo también el puede editar.

## 6. Gestión de usuarios y privilegios

---

Para poder proteger los datos entre las múltiples personas que se pueden conectar a una Base de Datos, que los datos no puedan ser vistas por otros (a no ser que lo queramos expresamente) y mucho menos que puedan ser manipuladas consciente o inconscientemente, los SGBD utilizan la autenticación para **usuarios** . En principio cada usuario sólo puede acceder a sus mesas, las que crea él (es el propietario). Pero también podremos dar permisos. Así un usuario podrá permitir la utilización de una mesa a otra, dándole diferentes grados de acceso: sólo consultar, o también añadir, o borrar, o actualizar, ..., o todas juntas, e incluso dar -le permiso para que a su vez dé permisos a otros.

Cuando muchos usuarios deben tener permisos similares, es conveniente la utilización de **grupos de usuarios** . Así, por ejemplo, si damos permiso de acceso a una determinada mesa a un grupo de usuarios, es como si hubiéramos dado el permiso uno a uno a todos los usuarios que pertenecen al grupo. Por tanto facilitarán mucho el trabajo del administrador.

Hasta la versión 8.0, **PostgreSQL** gestionaba los usuarios y grupos como tales ( **user** y **group** ), pero a partir de la versión 8.1 generaliza estos dos conceptos en uno: el **rol** ( **role** ). Antes eran dos entidades diferentes. Ahora es una sola, y un rol puede actuar como un usuario, un grupo o ambos. La única diferenciación será que algunos roles se pueden conectar (login) y se denominan **roles de entrada** , que sería el equivalente de usuario; y otros no (que sería el equivalente de grupo) y se llaman justamente **roles de grupo** .

Por otra parte, PostgreSQL es muy potente en cuanto a la limitación de la conexión de los usuarios, pudiendo dar más de un sistema de autenticación, y limitar mucho el acceso desde máquinas remotas, controlando tanto el usuario como la IP de la máquina desde donde se quiere conectar.

Por último, la seguridad de los ficheros queda garantizada por propietario, **postgres** .



## 6.1 Gestión de roles: utilización como usuarios

Un **rol** es una entidad capaz de recoger permisos y privilegios. Uno de estos permisos es el de conexión (login). Este **rol** o **usuario** será un nombre con una posible contraseña (que dependiendo del método de autenticación, servirá para controlar el acceso) que tendrá distintos permisos para crear tablas y otros objetos en una determinada base de datos, utilizarlos para hacer consultas o actualizaciones, ...

Aparte de esto, los permisos de un rol pueden asignarse a otro. Entonces es como si el segundo rol pertenecerá al primero, y el primer funciona como un grupo.

Los roles no están incluidos en ninguna base de datos particular. Por lo tanto son globales a toda la instalación de PostgreSQL (la gran Base de Datos).

Los roles, tanto si pueden hacer login o no, se guardarán en la tabla **pg\_authid**. Por comodidad (y compatibilidad con versiones anteriores) hay unas vistas que pueden facilitar la consulta.

- **pg\_roles** contiene todos los roles.
- **pg\_user** contiene los usuarios, es decir, los roles que pueden hacer un login.
- **pg\_shadow** contiene también las contraseñas.
- **pg\_group** contiene los grupos, es decir, los roles que no pueden hacer login.

Por lo tanto la primera manera de gestionar los roles sería manipular directamente las tablas o vistas, aunque parece demasiado fuerte.

Vamos a ver los modos normales de crear roles.

### CREATE ROLE

Tendremos que ejecutar esta sentencia SQL desde un usuario con permiso para crear roles, y conectado a cualquier BD

La sintaxis es la siguiente:

```
CREATE ROLE nombre [[WITH] opción [...]]
```

donde la opción puede ser (las subrayadas son las opciones por defecto):

<b>superuser</b>   <b>NOSUPERUSER</b>	permiso de superusuario (por defecto no)
<b>createuser</b>   <b>NOCREATEUSER</b>	similar al anterior (obsoleta)
<b>createdb</b>   <b>NOCREATEDB</b>	permiso para crear BD (por defecto no)
<b>CREATEROLE</b>   <b>NOCREATEROLE</b>	permiso para crear usuarios (por defecto no)
<b>inherente</b>   <b>NOINHERIT</b>	determina si el rol hereda las propiedades de los grupos (roles) a los que pertenece
<b>LOGIN</b>   <b>NOLOGIN</b>	permiso para conectarse (será un usuario)
<b>CONNECTION LIMIT</b> connlimit	Número máximo de conexiones concurrentes que puede haber (por defecto -1, que significa ilimitadas)
<b>[Encrypted</b>   <b>UNENCRYPTED]</b> <b>PASSWORD</b> 'password'	contraseña del usuario que puede ir encriptada o no
<b>VALID UNTIL</b> 'fecha'	fecha de caducidad del usuario

IN ROLE rolename [...]	roles (grupos) a los que pertenece
IN GROUP rolename [...]	similar al anterior (obsoleta)
ROLE rolename [...]	roles que pertenecerán a este rol (grupo)
USER rolename [...]	similar al anterior (obsoleta)
ADMIN rolename [...]	similar a la anterior pero además tendrán permiso para administrarlo (WITH ADMIN OPTION)

Para mantener la compatibilidad con versiones anteriores tenemos la sentencia equivalente:

```
CREATE USER nombre [[WITH] opción [...]]
```

donde por defecto sí tendrá el privilegio LOGIN.

Así, por ejemplo, desde una conexión por *psql* como *postgres* a *geo* podemos hacer:

```
CREATE ROLE geo1 LOGIN;
```

puede conectarse, no tiene password y no puede crear ni usuarios ni BD

```
CREATE ROLE GEO2 LOGIN PASSWORD 'GEO2';
```

puede conectarse, tiene password, no puede crear ni usuarios ni BD

Si queremos modificar algún aspecto del usuario, lo haremos con la sentencia

```
ALTER ROLE nombre [[WITH] opción [...]];
```

donde las opciones son las mismas que en CREATE ROLE, por ejemplo:

```
ALTER ROLE geo1 PASSWORD 'geo1';
```

Si queremos eliminar un rol, sencillamente

```
DROP ROLE nombre;
```

## createuser

Otra forma de crear roles, esta vez desde el sistema (sin conectar previamente a PostgreSQL, ni con psql ni con ninguna herramienta gráfica). Es un archivo ejecutable que proporciona PostgreSQL. En principio se intentará ejecutar como un usuario de BD con el mismo nombre que el usuario de SO Y como de momento sólo tenemos un superusuario (más concretamente un usuario con permiso para crear usuarios) más vale ejecutarlo como usuario de SO *postgres*, aunque lo podemos esquivar con la opción **-U**.

La sintaxis es:

```
createuser [ opciones ] nombre
```

Y en las opciones, entre otros, podremos poner:

- s** el nuevo usuario será superusuario (si no lo ponemos nos lo pedirá)
- S** el nuevo usuario no será superusuario
- d** el nuevo usuario podrá crear bases de datos (si no lo ponemos nos lo pedirá)

<b>-D</b>	el nuevo usuario no podrá crear bases de datos
<b>-r</b>	el nuevo usuario podrá crear usuarios (si no lo ponemos nos lo pedirá)
<b>-R</b>	el nuevo usuario no podrá crear usuarios
<b>-I</b>	el nuevo usuario podrá conectarse (por defecto)
<b>-L</b>	el nuevo usuario no podrá conectarse
<b>-P</b>	Pedirá la contraseña para el nuevo usuario

Aparte de estas opciones podemos poner otras, similares a las opciones de *psql*

<b>-h nombre</b>	nombre del servidor al que nos conectamos (por defecto local)
<b>-p puerto</b>	puerto del servidor a través del cual nos conectamos (por defecto 5432)
<b>-U nombre</b>	nombre del usuario que hace la operación (no lo que se crea)
<b>-W</b>	para que nos pida obligatoriamente la contraseña del usuario <i>que hace la operación</i>

Así, si en la consola no estamos como usuario postgres, podemos hacer:

```
createuser -U postgres
```

Aquí tenemos un ejemplo en el que le decimos expresamente que el usuario creado no sea superusuario, no pueda crear Bases de Datos y no pueda crear roles. Obsérvese que la contraseña que pide es la de quien ejecuta la orden. Por lo tanto se deberá poner **postgres** .

```
$ /opt/PostgreSQL/9.4/bin/createuser -U postgres -S -D -R geo3
Password:
```

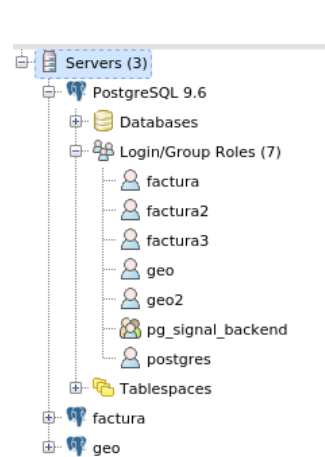
No existe el programa **alteruser** , para modificar un rol, pero sí **dropuser** para eliminarlo

```
dropuser [ opciones ] nombre
```

donde las opciones son algunas de createuser, con el mismo significado: **-h -P -U -W**

## Herramientas gráficas: pgAdmin

Para dar de alta un rol utilizado como un usuario con PgAdmin tendremos que ir a la opción **Roles de entrada** ( *Login Roles* ) habiéndonos autenticado como un usuario (un rol) que puede crear usuarios (roles), que en principio puede ser **postgres** .



Podemos observar que los roles no están dentro de las Bases de Datos.

Si intentamos crear un usuario nuevo, veremos que tenemos todas las posibilidades que teníamos en la creación para SQL, organizadas en pestañas.

The screenshot shows a dialog box titled 'Create - Login/Group Role'. It has a tabbed interface with the following tabs: 'General', 'Definition', 'Privileges', 'Membership', 'Parameters', 'Security', and 'SQL'. The 'General' tab is selected. It contains two input fields: 'Name' and 'Comments'. The 'Name' field is empty, and the 'Comments' field is a large text area. At the bottom of the dialog, there are four buttons: 'i' (information), '?' (help), 'Save' (with a floppy disk icon), and 'Cancel' (with a red 'X' icon). There is also a 'Reset' button (with a circular arrow icon) to the right of the 'Cancel' button.

Del mismo modo podremos modificar un usuario (yendo a sus propiedades) o eliminarlo.

Estos roles de entrada o usuarios son absolutamente independientes de los usuarios del SO (que ya se encarga él de que autentican). Pueden haber usuarios de ambos tipos con el mismo nombre, lo que puede facilitar la autenticación, pero no tiene porque ser así. Perfectamente puede haber usuarios de la BD que no existen en el SO y viceversa. Pero si no hacemos nada más, estos usuarios no podrán conectarse.

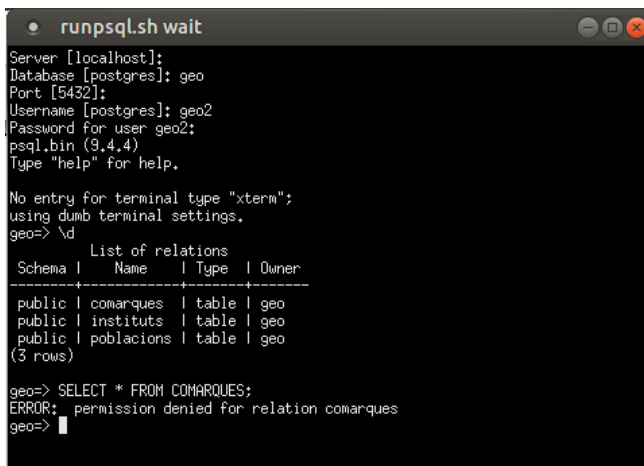
### ejercicio 3

Crea un usuario llamado **geo4** que se pueda conectar, con contraseña **geo4** y que pueda crear Bases de Datos. Hazlo preferiblemente desde **psql** . También lo puedes hacer desde **pgAdmin** , pero no te olvides de consular la sentencia SQL que se genera.

1. Escribe la sentencia SQL que se ha ejecutado para crearlo y adjunta una imagen.

De momento, la configuración es que todos los usuarios pueden conectar a todas las Bases de Datos. Pero en principio sólo quien crea una tabla (o cualquier objeto) puede acceder a ella (salvo el superusuario). Pero este propietario puede dar permisos a los otros roles para leerla, insertar, borrar, ... También lo podrá hacer el superusuario. Estos permisos no incluirán el de borrar la tabla o modificarla, que sólo lo podrá hacer su propietario.

Vamos a comprobar lo que acabamos de decir, que aunque todos los usuarios pueden conectarse a todas las Bases de Datos, no podrán acceder a los objetos de ella, a no ser que le damos con GRANT.



```
runpsql.sh wait
Server [localhost]:
Database [postgres]: geo
Port [5432]:
Username [postgres]: geo2
Password for user geo2:
psql.bin (9.4.4)
Type "help" for help.

No entry for terminal type "xterm";
using dumb terminal settings.
geo=> \d
          List of relations
Schema |   Name   | Type | Owner
-----+-----+-----+-----
public | comarques | table | geo
public | instituts | table | geo
public | poblacions | table | geo
(3 rows)

geo=> SELECT * FROM COMARQUES;
ERROR:  permission denied for relation comarques
geo=>
```

Nos hemos conectado a la Base de Datos **geo** como usuario **GEO2** . No hemos tenido problema en conectar. Incluso vemos las tablas de la Base de Datos **geo** (con \d podemos listarlas).

Pero no podemos acceder al contenido de ninguna mesa.

### GRANT

Para dar permisos utilizamos **GRANT** . Podremos dar 3 tipos de permisos:

- Permisos sobre tablas

Su sintaxis es

```
GRANT ({SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER} [, ...] | ALL [PRIVILEGES])
ON [TABLE] nombre_tabla [...]
TO {nombre_usuario | GROUP nombre_grupo | PUBLIC} [, ...]
[WITH GRANT OPTION]
```

Es decir, que sobre una mesa se puede dar permiso sólo para seleccionar, o insertar, o modificar, o borrar, o crear una clave externa sobre esta mesa, o crear un trigger sobre esta mesa. O muchos de ellos. O todos ( **ALL** ).

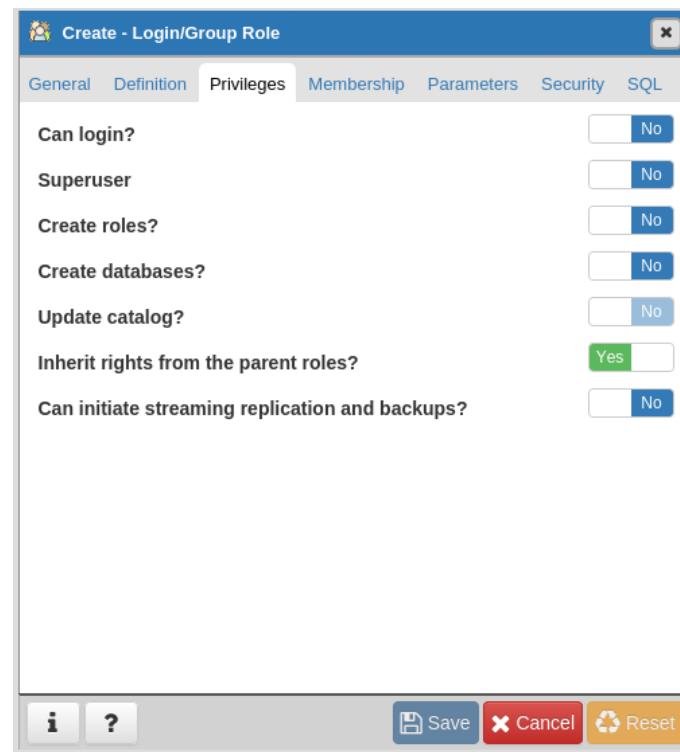
Se puede dar permiso a un usuario (o más) o a un grupo, o a **PUBLIC**, es decir en todo el mundo. En realidad la distinción entre usuario y grupo es para mantener compatibilidades, puesto que ya sabemos que ahora todo son roles.

Si además ponemos la opción **WITH GRANT OPTION**, los usuarios a los que hemos dado permiso pueden dar estos permisos a otros.

Cuando se da un permiso a un rol al que pertenecen una serie de miembros, estos heredarán los permisos sólo si tienen el privilegio **inherente**.

#### Nota

En **PgAdmin** se ve muy gráficamente en la segunda pestaña de las propiedades de un usuario



Por ejemplo, en el usuario y Base de datos **geo** ( entramos como usuario: geo y le damos permisos al usuario: GEO2 ) :

```
GRANT SELECT, UPDATE ON COMARCAS, POBLACIONES TO GEO2;
```

Ahora el usuario **GEO2** podrá acceder al contenido de las tablas comarcas y poblaciones. Incluso pueden actualizar las filas ya existentes y podemos comprobar como sí podemos acceder a su contenido.



```
runpsql.sh wait
Server [localhost]:
Database [postgres]: geo
Port [5432]:
Username [postgres]: geo2
Password for user geo2:
psql.bin (9.4.4)
Type "help" for help.

No entry for terminal type "xterm";
using dumb terminal settings.
geo> SELECT * FROM COMARQUES;
 nom_c | provincia
-----+-----
 Safor  | València
 Horta Sud | València
 Camp de Morvedre | València
 Foia de Bunyol | València
 Alacantí | Alacant
 Alt Maestrat | Castelló
 Plana Baixa | Castelló
 Horta Nord | València
 Ports | Castelló
 Racó | València
 Plana d'Utiel | València
```

## REVOKE

Para quitar los permisos utilizaremos **REVOKE**

- Para quitar permisos sobre tablas

```
REVOKE [GRANT OPTION FOR] {{SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER} [, ...] | ALL [PRIVILEGES]}
ON [TABLE] nombre_tabla [...]
FROM {nom_usu | GROUP nombre_grupo | PUBLIC} [, ...]
[CASCADE | RESTRICT]
```

Donde tendremos que tener en cuenta que sólo el usuario que ha dado un permiso o el superusuario puede quitarlo, excepto en el caso de la opción **CASCADE**, que quita también el permiso a todos los que se la han ido pasando por tener el permiso **WITH GRANT OPTION**, en su caso.

Una Base de Datos es un lugar donde puede haber uno o más de un esquema, y en cada esquema se pueden crear muchos objetos.

La jerarquía será la siguiente: **Servidor -> Base de Datos -> Esquema -> Tabla** .

Un usuario que se conecte a una Base de Datos podrá ver (si tiene permiso) los objetos de los diferentes esquemas de esta Base de Datos.

### CREACIÓN

- La creación del efectuaremos con la sentencia SQL:

```
CREATE DATABASE nombre
  [[WITH] [OWNER [=] nom_propietari ]
  [TEMPLATE [=] template ]
  [ENCODING [=] encoding ]
  [TABLESPACE [=] tablespace ]
  [CONNECTION LIMIT [=] num_conn ]]
```

En principio el propietario de la Base de Datos será quien la crea (deberá tener permiso para crear: **createdb** ). Si somos super usuarios podremos hacer que el propietario sea otro.

Cuando se crea una base de datos en realidad, para ir más rápido, lo que hace es copiarla de otro, una especie de plantilla. Podemos elegir esta plantilla o Base de Datos original: **template1** o **template0** (por defecto **template1** ).

La cláusula **encoding** servirá para especificar un conjunto de caracteres (aconsejable UTF-8).

Podemos especificar en qué **Tablespace** se guardará la Base de Datos y sus objetos.

También se puede decir cuántas conexiones concurrentes se pueden hacer. Por defecto (-1) no hay límite.

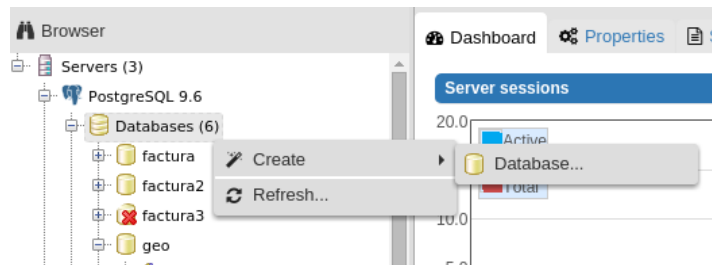
Todo esto también se podía hacer con una utilidad proporcionada por PostgreSQL, desde una sesión de usuario de SO **postgres** (u otro que tenga permiso para crear BD)

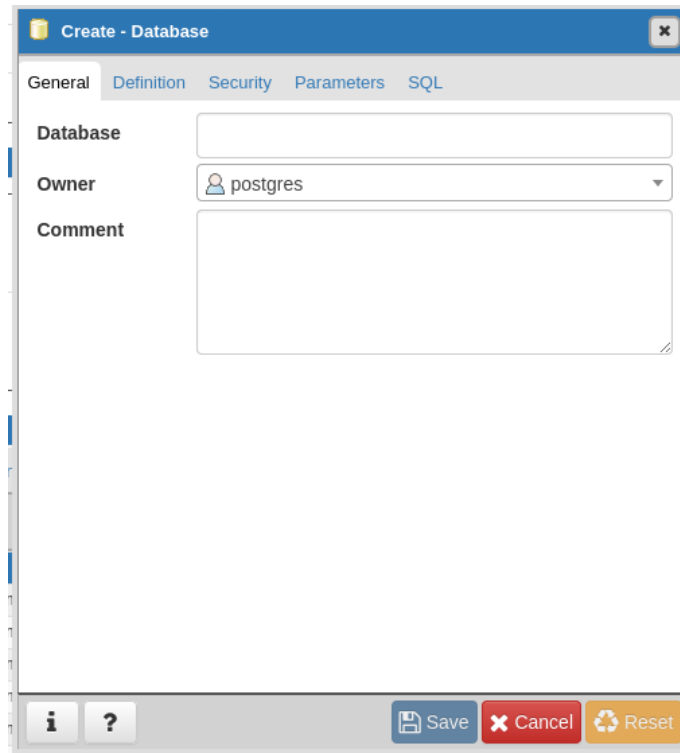
```
$ Createdb [ opciones ] nombre
```

donde algunas opciones, aparte de las que especificar el host, pedir contraseña, etc., son:

**-O** ( owner ) **-T** ( template ) **-E** ( encoding ) **-D** ( tablespace )

- También se puede hacer desde **PgAdmin** si apretamos el botón de creación de objetos, cuando estemos situados en el servidor o en una Base de Datos (no en un nivel inferior). Tendremos las mismas opciones que en el caso de la sentencia SQL, en diferentes pestañas





## MODIFICACIÓN

- La sentencia **ALTER DATABASE** permite variar los parámetros de la BD, o renombrar-la.
- En **pgAdmin** iremos a las propiedades de la Base de Datos.

## ELIMINACIÓN

- Se puede utilizar la sentencia SQL **DROP DATABASE** , o la utilidad **dropdb** .
- En **pgAdmin** , la papelera.

### ejercicio 4

Crear Base de Datos llamada **geo4** , el propietario de la cual debe ser **geo4** .

Pone tanto el usuario como estabas conectado, como la sentencia SQL que te ha permitido crearla (adjunta captura de pantalla)

## 8. Copia de seguridad y restauración

---

Esta tarea tan fundamental del Administrador de la Base de Datos es muy sencilla de hacer en PostgreSQL, y se puede planificar con un sencillo script que se ejecute periódicamente.

Hay básicamente dos maneras de hacer el backup:

- con ***pg\_dump***
- Copiando los archivos

Siempre tendremos, sin embargo, la posibilidad de hacerla cómodamente desde **PgAdmin** .

El método que utiliza **pg\_dump** consiste en generar un archivo de texto con los comandos SQL necesarios para rehacer la Base de Datos tal y como estaba en el momento de hacer el **dump**.

Utilizaremos el programa **pg\_dump**, situado donde todos los programas, y que tiene la siguiente sintaxis:

```
pg_dump [opciones] nom_bd
```

El resultado irá por la salida estándar, la pantalla. Por lo tanto seguramente siempre lo gastaremos así:

```
pg_dump nom_bd > fitxer_eixida
```

Si queremos que no se guarde en el directorio activo tendremos que poner la ruta también. Como cuestión de estilo, podríamos acostumbrarnos a poner siempre la extensión **.sql**

Ahora tendremos en este fichero las sentencias SQL para dejar la Base de Datos como estaba.

El puede ejecutar cualquier usuario, y sobre cualquier Base de Datos, pero evidentemente si el usuario no tiene permiso de acceso sobre la BD fallará.

Como **pg\_dump** es una aplicación cliente de PostgreSQL se podrá ejecutar desde cualquier lugar, en local o remoto. Y la autenticación será igual que lo que hemos visto hasta ahora.

Estas son algunas de las opciones:

<b>-a</b>	sólo guarda los datos
<b>-s</b>	sólo guarda la estructura
<b>--inserts</b>	incluye sentencias INSERT para los datos (sino pondrá sentencias COPY)
<b>--column-insertos</b>	lo mismo pero explicitando los nombres de las columnas
<b>-f archivo</b>	envía el resultado a un archivo; si no se especifica, la envía a la salida estándar.
<b>-F formato</b>	indica el formato con el que se guardará el archivo ( <b>p</b> archivo de texto; <b>t</b> de tipo tar; <b>c</b> de tipo custom, que es la más flexible; con el primer tendremos que restaurar utilizando <b>psql</b> ; con los dos últimos utilizaremos <b>pg_restore</b> )
<b>-n nombre</b>	incluye sólo el esquema especificado
<b>-t nombre</b>	incluye sólo la tabla especificada

Aparte estarán las habituales opciones de conexión: **-h** ( *host* ), **-p** ( *puerto* ), **-U** ( *usuario* ) y **-W**

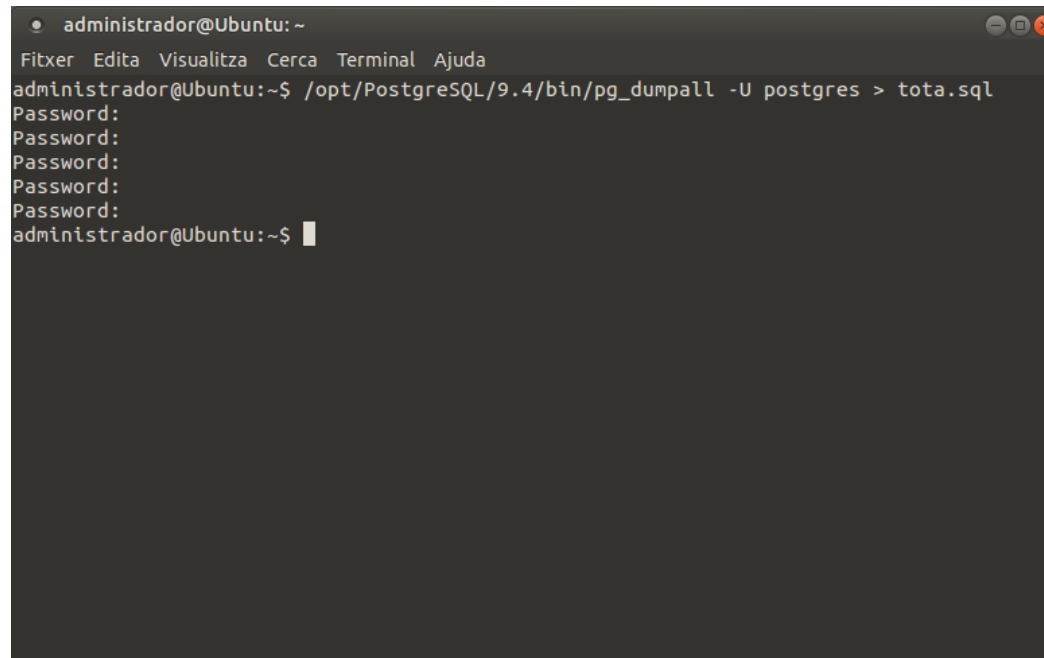
Esta sería la manera de hacer copia de seguridad de la Base de Datos **geo**, guardándola en un archivo del directorio activo llamado **geo.sql**.

```
$ /opt/PostgreSQL/9.6/bin/pg_dump -U geo> geo.sql
```

Para hacer un **dump** no hay que parar el servidor.

Una variante es el **pg\_dumpall** que hace el dump sobre todas las bases de datos (seguramente este es el candidato para copias de seguridad periódicas). Evidentemente tendremos que hacer como **postgres**, para poder tener acceso a todas las BD Tiene la ventaja adicional que también guarda los objetos que no pertenecen a una determinada base de datos, como son los usuarios. El inconveniente es

que tendremos que poner la contraseña de **postgres** tantas veces como Bases de Datos tengamos, porque se ha de conectar a cada una de ellas. Para poder evitar esto podríamos utilizar otro método de autenticación para el usuario **postgres** : **ident**. La siguiente imagen muestra el proceso de hacer una copia de seguridad de todo el cluster. Se ha introducido 5 veces la contraseña del usuario postgres, porque 5 son las bases de datos existentes.



```
administrador@Ubuntu: ~  
Fitxer Edita Visualitza Cerca Terminal Ajuda  
administrador@Ubuntu:~$ /opt/PostgreSQL/9.4/bin/pg_dumpall -U postgres > tota.sql  
Password:  
Password:  
Password:  
Password:  
Password:  
administrador@Ubuntu:~$
```

La copia de seguridad se tendrá guardado en el fichero **tota.sql** . Si vemos su contenido, tendrá guardado todas las sentencias para guardar las 5 Bases de Datos existentes, y también todos los roles (de usuario y de grupo) existentes.



Ubuntu 14.04 clona (Després usuaris) [S'està executant] - Oracle VM VirtualBox

Aplicacions Llocs Sistema

dj, 15 de oct, 08:52 es

tota.sql (~) - gedit

Fitxer Edita Visualitza Cerca Eines Documents Ajuda

Obre Desa Desfés

tota.sql x

```
-- PostgreSQL database cluster dump
--

SET default_transaction_read_only = off;

SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;

--
-- Roles
--

CREATE ROLE emp1;
ALTER ROLE emp1 WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB LOGIN NOREPLICATION PASSWORD
'md5cd6ef4e86ec3e4a45fc9b9936ab098b2';
CREATE ROLE emp2;
ALTER ROLE emp2 WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB LOGIN NOREPLICATION PASSWORD
'md51cce225a68f03917457c9a95937147f2';
CREATE ROLE empresa;
ALTER ROLE empresa WITH NOSUPERUSER INHERIT CREATEROLE CREATEDB LOGIN NOREPLICATION PASSWORD
'md595c943899e89329311489bcc90941ca2';
CREATE ROLE g_geo;
ALTER ROLE g_geo WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB NOLOGIN NOREPLICATION;
CREATE ROLE g_geo2;
ALTER ROLE g_geo2 WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB NOLOGIN NOREPLICATION;
CREATE ROLE geo;
ALTER ROLE geo WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB LOGIN NOREPLICATION PASSWORD
'md5ac0870ed1147c3a0ba959602774869d6';
CREATE ROLE geo1;
```

SQL Amplada de la tabulació: 8 Ln 1, Col. 1 INSER

administrador@Ubun... Carpeta de l'usuari tota.sql (~) - gedit

Control dreta

## 8.2 Restauración del Dump

Si el archivo de salida del **dump** (que en adelante de entrada) no tenía ningún formato (no hemos utilizado la opción -F) la ejecutaremos en el **psql** , ya que son comandos SQL. Lo podemos hacer de las dos formas siguientes:

```
psql nom_bd < fitxer_entrada
```

```
psql -f fitxer_entrada nom_bd
```

Sólo se ha especificado la Base de Datos a la que se conecta, pero como es el comando **psql** , podrían ir cualquiera de las opciones habituales (-U para especificar el usuario, -h para especificar el host, ... )

Como siempre, si el archivo no está en el directorio activo tendremos que poner la ruta. Incluso una vez hemos entrado en la base de datos, podremos hacer:

```
\ y fitxer_entrada
```

Habrà una restricción lógica, pero importante: tanto la Base de Datos como el usuario deben existir (no los crea esta restauración)

Para restaurar todas las Bases de Datos, lo tendremos que hacer como usuario **postgres** , y mejor acceso a **template1** .

Puestos a hacer virguerías, incluso podemos trasvasar la información de una BD a otra:

```
pg_dump BD1 | psql bd2
```

que incluso puede estar en otro lugar

```
pg_dump BD1 | psql -h host BD1
```

En cambio, si el archivo de salida tenía algún formato, porque hemos utilizado la opción **-Ft** o **-FC** , deberemos utilizar el programa **pg\_restore** , con el que se pueden especificar más cosas:

```
pg_restore [ opciones ] nombrearchivo
```

Y en las opciones, entre otros, podremos poner:

- |                   |   |
|-------------------|---|
| <b>-a</b>         | sólo restaura los datos   |
| <b>-s</b>         | sólo restaura la estructura   |
| <b>-c</b>         | borra los objetos antes de rellenarlos  |
| <b>-C</b>         | crea la BD antes de restaurar en ella   |
| <b>-d nom_bd</b>  | se conecta a la BD y restaura directamente en ella  |
| <b>-F formato</b> | indica el formato del archivo de entrada, aunque no hace falta que lo detecta automáticamente |
| <b>-t nombre</b>  | restaura sólo la tabla especificada   |

Aparte estarán las habituales opciones de conexión: **-h** ( *host* ), **-p** ( *puerto* ), **-U** ( *usuario* ) y **-W**



### ejercicio 5

1. Haz una copia de seguridad (o exportación) de la tabla **comarcas** de la Base de Datos **geo** (propietario **geo** ). Puedes utilizar el entorno y método que quiere. El fichero donde guardarla podría ser **comarques.sql** . Explica el método y entorno utilizado.
2. Restaurar (o importa) el archivo **comarques.sql** (donde está la mesa comarcas) en la Base de Datos **geo4** , por lo que el propietario de esta tabla sea **geo4** . Explica cómo lo has conseguido. Tendrás que controlar especialmente el propietario de la tabla, ya que seguramente en el fichero de exportación ha sido **geo** , y ahora queremos que sea **geo4** .
3. Haz una copia de seguridad (o exportación) de **toda** la Base de Datos **geo4** . Puedes utilizar el entorno y método que quiere, pero el formato debe ser PLA (SQL) y utilizando INSERT (no COPY). El fichero donde guardarla debe ser **geo4.sql** . Explica el método y entorno utilizado y **adjunta el archivo geo4.sql** .

Tienes que contestar todos los ejercicios en un archivo de texto, adjuntando imágenes en los casos que se especifica, y también siempre que lo consideras necesario. Pon siempre el nombre del ejercicio (Ejercicio 1, Exercicio 2, ...) y también el del subapartado.

Adjuntar archivo geo4.sql

Sube en un fichero comprimido ambos ficheros idONA el nombre: **Ex\_admin\_cognoms\_nom .odt (o pdf)**

