UNIVERSIDAD NACIONAL AUTONOMA DE NICARAGUA
LEÓN

FACULTAD DE CIENCIAS

D E P A R T A M E N T O D E C O M P U T A C I O N

TITULO:

“Guía para la configuración y gestión de sistemas de base de datos PostgreSQL bajo entorno Linux”

Autores:

Br. Julio César Maradiaga Acosta.
Br. Omara Nadieshka Moraga Ramírez.

Tutor:

Msc. Ernesto Espinoza

DEDICATORIA

A Dios y María Santísima por habernos permitido llegar hasta el final de esta etapa del camino de la enseñanza y el saber.

A nuestras madres quienes han sido mujeres admirables, que nos han brindado cuidados, amor y comprensión, quienes con sus sabios consejos orientaron nuestros pasos por el camino recto de la vida, convirtiéndose por sus virtudes en nuestras mejores amigas.

AGRADECIMIENTO

A Dios nuestro señor y María Santísima que nos dieron la fortaleza, sabiduría, los dones y bendiciones para realizar con éxito y orgullo nuestro trabajo monográfico. Pero sobre todo por guiar nuestros pasos por el camino del bien y del saber.

A nuestras madres por el apoyo incondicional y consejos que nos dieron durante el transcurso de nuestras vidas.

Yo Julio de una manera muy especial quisiera agradecer a mi hermano Juan Rafael por haberme brindado aquellos consejos de aliento en los momentos de confusión y angustia, además por el apoyo económico.

Yo Omara agradezco a mi compañero Julio César por brindarme su confianza y darme aliento y motivación en situaciones difíciles.

No podíamos olvidar manifestar la gratitud que sentimos por el apoyo brindado por el Lic. Ernesto Espinoza y el Ing. Ricardo Espinoza en el desarrollo del presente trabajo monográfico, al proporcionarnos ciertas facilidades para la investigación y la realización de algunas pruebas.

A todos mil gracias...

INTRODUCCIÓN

En los últimos años, el software de base de datos ha experimentado un auge extraordinario, a raíz de la progresiva automatización de los procesos de casi la totalidad de las empresas de hoy en día.

No es extraño que existan multitud de gestores de base de datos tales como: Oracle, Microsoft SQL Server, Borland Interbase, entre otros programas que permiten manejar la información de modo sencillo.

Las soluciones software que hemos citado son comerciales. Pero, en el mundo del software libre, siempre que se necesita algo se implementa. Así tenemos MySQL, gestor muy usado en los servidores web y PostgreSQL, que será el gestor sobre el cual trataremos.

Con el desarrollo de esta guía pretendemos proporcionarle al usuario la información necesaria sobre las herramientas y comandos utilizados en la creación y administración de base de datos realizados en este potente gestor.

Postgres fue desarrollado originalmente en el departamento de ciencias de la computación de la Universidad de California en Berkeley, fue pionero en muchos de los conceptos de base de datos relacionales orientados a objetos que ahora están disponibles en algunas bases de datos comerciales. En 1977 comenzó como un proyecto denominado *Ingres* el que más tarde fue desarrollado comercialmente por la *Relational Technologies/Ingres Corporation*.

En 1986 un equipo dirigido por *Michael Stonebraker* de Berkeley continuó el desarrollo del código de *Ingres* para crear un sistema de bases de datos objeto-relacionales llamado *Postgres*.

En 1994, Andrew Yu y Jolly Chen añadieron un intérprete del lenguaje SQL a Postgres, dando lugar a Postgres95 el cual fue publicado posteriormente en la Web para encontrar su propio espacio en el mundo como un descendiente de dominio público del código original de *Postgres* de Berkeley.

En 1996 se hizo evidente que el nombre de Postgres95 no resistiría el paso del tiempo, por lo cual se eligió como nuevo nombre PostgreSQL, para reflejar la relación entre el *Postgres* original y las versiones más recientes con capacidades SQL.

El proyecto *PostgreSQL* sigue actualmente un activo proceso de desarrollo a nivel mundial gracias a un equipo de desarrolladores y contribuidores de código abierto.

Esperamos que las instrucciones aquí plasmadas proporcionen una mejor comprensión de este potente gestor de base de datos.

Cabe mencionar que los lectores deben estar familiarizados con el entorno Linux, sistema operativo anfitrión del gestor de base de datos a tratar.

JUSTIFICACIÓN

Hemos decidido desarrollar una ***"Guía para la configuración y Gestión de sistemas de base de datos PostgreSQL bajo entorno Linux"***, por diversas razones:

1. PostgreSQL, es el sistema de gestión de base de datos de código abierto más avanzado hoy en día.

2. Puede ser integrado en plataformas Windows como Linux.

3. Permite que aplicaciones como Access, Visual Basic, Fox Pro, Visual C++, entre otros, utilicen PostgreSQL como servidor.

4. Incluye las características de los gestores comerciales de base de datos.

5. Evita bloqueos innecesarios utilizando el control de concurrencia multi-versión.

6. Tamaño de una base de datos limitado sólo por el hardware.

Por lo expuesto anteriormente es que deseamos incorporar esta guía, al Departamento de Computación de la Facultad de Ciencias para que sirva como punto de partida en el desarrollo de futuros trabajos monográficos que se realicen sobre este potente gestor.

OBJETIVO GENERAL

1. Dar a conocer las herramientas necesarias para la administración y configuración de base de datos PostgreSQL bajo entorno Linux, así como la configuración de estaciones Windows para consultar dicha base de datos.

OBJETIVOS ESPECIFICOS

1. Enumerar los pasos a seguir para una correcta instalación y configuración del motor de Base de Datos de PostgreSQL sobre plataforma *Linux.*

2. Desarrollar una guía con las sentencias y comandos más utilizados en *PostgreSQL* para la gestión y administración de base de datos.

3. Elaborar una lista de los problemas más comunes que se pueden presentar si un proceso en el servidor termina de forma anormal.

4. Enseñar los pasos necesarios para realizar una conexión remota de clientes Windows a un servidor de base de datos en Linux.

5. Enseñar algunos aspectos de seguridad implementada en una base de datos PostgreSQL.

MARCO TEÓRICO

¿Qué es PostgreSQL?

Es un Sistema de Gestión de Base de Datos Objeto-Relacional (ORDBMS) de código abierto más avanzado hoy en día, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL, contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java).

Características de PostgreSQL

PostgreSQL ha generado algunas características que son propias en el mundo de las bases de datos orientadas a objetos. Esto ha llevado a que, algunas Bases de Datos comerciales hayan incorporado recientemente estas ventajas en las que **PostgreSQL** fue pionera. La siguiente es una breve lista de algunas de esas características:

- **DBMS Objeto-Relacional**

PostgreSQL aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas. Ejemplos de su avanzada funcionalidad son consultas SQL declarativas, control de concurrencia multi-versión, soporte multi-usuario, transacciones, optimización de consultas, herencia, y arrays.

- **Altamente Extensible**

PostgreSQL soporta operadores, funciones, métodos de acceso y tipos de datos definidos por el usuario.

- **Soporte SQL Comprensivo**

PostgreSQL soporta la especificación SQL99 e incluye características avanzadas tales como las uniones (joins) SQL92.

- **Integridad Referencial**

PostgreSQL soporta integridad referencial, la cual es utilizada para garantizar la validez de los datos en una relación asociada a otra por medio del o los campos que contienen ese dato.

- **Lenguajes Procedurales**

PostgreSQL tiene soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL. Este lenguaje es comparable al lenguaje procedural de Oracle, PL/SQL. Otra ventaja de PostgreSQL es su habilidad para usar Perl, Python, o TCL como lenguaje procedural embebido.

- **MVCC (Control de Concurrencia Multi-Versión)**

Es la tecnología que PostgreSQL usa para evitar bloqueos innecesarios manteniendo la consistencia de los datos en un modelo multiversión. Esto significa que mientras se consulta una base de datos, cada transacción ve una imagen de los datos (una versión de la base de datos).

La principal diferencia entre multiversión y el modelo de bloqueo es que en los bloqueos multiversión derivados de una consulta (lectura) de datos no entran en conflicto con los bloqueos derivados de la escritura de datos y de este modo la lectura nunca bloquea la escritura y la escritura nunca bloquea la lectura.

- **Cliente/Servidor**

PostgreSQL usa una arquitectura de proceso por usuario cliente/servidor. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL.

- **Write Ahead Logging (WAL)**

La característica de PostgreSQL, conocida como *Write Ahead Logging* incrementa la dependencia de la base de datos al registro de cambios, antes de que estos sean escritos en la base de datos. Esto garantiza que en el hipotético caso de que la base de datos se caiga, existirá un registro de las transacciones a partir del cual podremos restaurar la base de datos. Esto

puede ser enormemente beneficioso, ya que cualquiera de los cambios que no fueron escritos en la base de datos pueden ser recuperados usando el dato que fue previamente registrado. Una vez que el sistema ha quedado restaurado, un usuario puede continuar trabajando desde el punto en que lo dejó cuando cayó la base de datos.

Otras características que aportan potencia y flexibilidad adicional son:

- Restricciones (contraints).
- Disparadores (triggers).
- Reglas (rules).
- Integridad transaccional.

Tomando en cuenta esas características y sumando la estabilidad, performance, disponibilidad y la eficiencia de un sistema operativo como **Linux**, muchas empresas e instituciones lo están adoptando como servidor de Bases de Datos.

Limites de una base de datos en PostgreSQL

- ✓ ***Tamaño máximo de una base de datos:*** ilimitado, solo limitado por el hardware.
- ✓ ***Tamaño máximo para una tabla:*** hasta 64 Tb. (Terabytes).
- ✓ ***Tamaño máximo para un campo:*** 1 Gb.
- ✓ ***Cantidad máxima de registros:*** ilimitado.
- ✓ ***Cantidad máxima de atributos en una tabla:*** hasta 1600.
- ✓ ***Cantidad máxima de índices por tabla:*** ilimitado.

Principales mejoras en PostgreSQL

Durante el desarrollo de versiones anteriores a PostgreSQL se hacia hincapié en identificar y entender los problemas del motor de datos. Con PostgreSQL el énfasis ha pasado a aumentar características y capacidades, aunque el trabajo continúa en todas las áreas.

Las principales mejoras en PostgreSQL incluyen:

1. Los bloqueos de tablas han sido sustituidos por el control de concurrencia multi-versión, el cual permite a los accesos de sólo lectura continuar leyendo datos durante la actualización de registros y permite

copias de seguridad en caliente desde pg_dump mientras la base de datos permanece disponible para consultas.

2. Se han implementado importantes características del motor de datos, incluyendo subconsultas, valores por defecto, restricciones a valores en los campos (constraints) y disparadores (triggers).

3. Los tipos internos han sido mejorados, incluyendo nuevos tipos de fecha/hora, de rango amplio y soporte para tipos geométricos adicionales.

4. La velocidad del código del motor de datos han sido incrementado aproximadamente en 20%-40% y su tiempo de arranque ha bajado en un 80% desde que la versión 6.0 fue lanzada.

Arquitectura de un sistema PostgreSQL

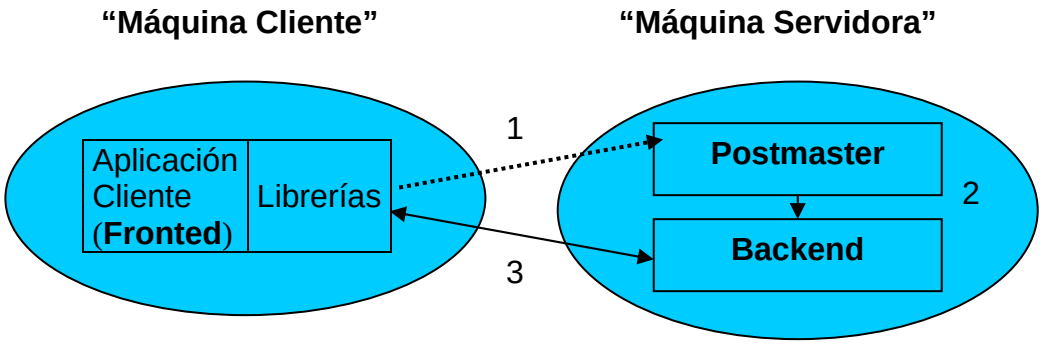
Una vez estudiado algunos conceptos, características y otros aspectos de PostgreSQL pasaremos a explicar un poco sobre cómo interactúan las partes de un sistema PostgreSQL.

En la jerga de las bases de datos, PostgreSQL utiliza un modelo denominado cliente/servidor de "proceso por usuario"; una sesión consiste en:

1. Un proceso denominado supervisor (postmaster).
2. La aplicación de interfaz de usuario (frontend en inglés).
3. Uno o más procesos servidores de acceso a la base de datos (backend en inglés).

Un único postmaster maneja una colección dada de base de datos en un único host. A tal colección se le denomina una *instalación* o un *sitio*.

Las aplicaciones frontend que quieren acceder a una base de datos dada en una instalación, realizan llamadas a la librería. La librería envía el requerimiento del usuario a través de la red al postmaster, quien en su momento arranca un nuevo proceso servidor backend y se conecta el proceso cliente al nuevo servidor. A partir de aquí el proceso cliente y el servidor se comunican entre ellos sin intervención del postmaster.



En consecuencia, el proceso postmaster está siempre corriendo, esperando llamadas, mientras que los proceso cliente y servidor vienen y van.

Instalación de PostgreSQL en Red Hat Linux 8.0

Ahora, vamos a describir el proceso de instalación de PostgreSQL para Linux. En nuestro caso hemos utilizado la distribución Red Hat Linux 8.0, por lo tanto, de aquí en adelante siempre que hablemos de Linux estaremos haciendo referencia a dicha versión.

Linux utiliza un sistema de gestión de paquetes denominado RPM (Red Hat Package Manager) que permite instalar fácilmente el gestor de base de datos. Además PostgreSQL viene incluido en la distribución estándar de Linux por lo que simplemente hemos de tener presente a la hora de realizar la instalación del sistema operativo seleccionar en los grupos de paquetes a instalar la opción *"Servidor de base de datos SQL".*

Hay que tener en cuenta que si se realiza la instalación de PostgreSQL de esta manera solo se instalaran los paquetes básicos como son:

PostgreSQL-7.2.2-1.i386.rpm
PostgreSQL-server-7.2.2-1.i386.rpm
Postgresql-libs-7.2.7-1.i386.rpm

Los cuales permiten utilizar PostgreSQL pero sin muchas de sus capacidades, como por ejemplo el paquete PostgreSQL-odbc-7.2.2-1.i386.rpm que permite conectar PostgreSQL con programas que tienen un

drivers ODBC compatible (tal como Acces) y Postgresql-tcl-7.2.2-1.i386.rpm que permite instalar aplicaciones PostgreSQL que requieren soporte tcl tales como pgacces (cliente gráfico de base de datos), entre otros paquetes que aumentan considerablemente la capacidad de PostgreSQL.

En caso de que haya realizado una instalación como la antes descrita y quisiera agregar paquetes que le den a PostgreSQL la potencialidad que usted necesita en el CD 3 de instalación de Red Hat Linux encontrará dichos paquetes y para su instalación puede seguir el numeral 3 de los pasos descritos a continuación.

Si no tiene instalado PostgreSQL para proceder a realizar su instalación siga los pasos siguientes:

> **Nota:** Si ya tiene Linux instalado es recomendable antes de proceder a realizar la instalación, verificar primero si ya tiene o no instalado PostgreSQL. Esto se puede hacer mediante el siguiente comando:
>
> **rpm –qa|grep -i postgres.**
>
> Con esta orden se listaran los paquetes PostgreSQL instalados si existen.

1. En la distribución de Red Hat Linux 8.0, en el CD número 2, están los paquetes básicos. Copiar los archivos a un directorio temporal de compilación, por lo general se utiliza usr/src/ (el cual será nuestro caso) para almacenar los códigos basados en RPM, pero puede elegir otro directorio si usted desea.

   Una vez copiado los archivos mediante la siguiente orden:

   $cp postgresql-nombre_paquete-7.2.2-1.i386.rpm /usr/src/

2. Nos movemos al directorio mencionado, pero antes de proceder a realizar la instalación de los paquetes individuales nos tenemos que convertir en el usuario root, de lo contrario no podremos realizar la instalación. Otra cosa que hay que tener presente es el orden en que se van extrayendo los paquetes:

   Primero se procede con las librerías mediante la orden:
   [jyo@localhost src]$ rpm –Uvhi postgresql-libs-7.2.2-1.i386.rpm

Después se procede con los siguientes paquetes, en el orden respectivo:
[jyo@localhost src]$ rpm –Uvhi postgresql-7.2.2-1.i386.rpm
[jyo@localhost src]$ rpm –Uvhi postgresql-server-7.2.2-1.i386.rpm

3. Hasta aquí, ya tiene instalado PostgreSQL (ya operativo) en su sistema pero sin muchos de los paquetes que proporcionan a PostgreSQL soporte adicional, y esos paquetes son:

   - *PostgreSQL-odbc-7.2.2-1.i386.rpm*: que permite conectar PostgreSQL con programas que tenga un drivers ODBC.
   - *Postgresql-jdbc-7.2.2-1.i386.rpm*: Activa el soporte Java.
   - *Postgresql-perl-7.2.2-1.i386.rpm*: esta opción es necesaria si usted pretende utilizar lenguajes procedurales.
   - *Postgresql-python-7.2.2-1.i386.rpm:* Esta opción sólo se requiere si usted planea usar el lenguaje procedural pl/Python.
   - *Postgresql-tcl-7.2.2-1.i386.rpm:* Esta opción instalará aplicaciones PostgreSQL y aplicaciones que requieren Tcl, tales como *pgaccess* (un popular cliente gráfico de bases de datos) y el lenguaje procedural pl/Tcl.

Para proceder a la instalación de cualquiera de los paquetes anteriores utilice la orden:

rpm –Uvhi postgresql-nombre_paquete-7.2.2-1.i386.rpm

Nota: si no tiene los CD de instalación puede visitar el sitio web de PostgreSQL (www.postgresql.org) para descargar los paquetes que usted necesite.

Comandos de PostgreSQL

Esta es la información de referencia para los comandos y utilidades de soporte de PostgreSQL.

Initdb

Descripción*:*

Crea un nuevo sistema de base de datos de PostgreSQL. Un sistema de base de datos, es una colección de bases de datos que son todas administradas por el mismo usuario y manejadas por un sólo postmaster. Crear un sistema de base de datos consiste en *definir los directorios* en los cuales los datos de la base de datos serán almacenados. No se debe ejecutar el initdb como root.

Esto se debe a que no se puede ejecutar el servidor de la base de datos ni siquiera como root, pero el servidor necesita tener acceso a los archivos que initdb crea. Además, durante la fase de la inicialización cuando no hay usuarios y ningún control de acceso instalado, postgres solamente se conectará con el nombre de usuario actual de Linux, así que uno debe iniciar una sesión bajo la cuenta que poseerá el proceso del servidor.

Aunque initdb procurará crear el directorio de datos respectivo, lo cierto es que no tendrá el permiso para hacerlo. Por lo tanto, es una buena idea crear el directorio de datos antes de ejecutar initdb *y* entregar la propiedad de él al súper usuario de la base de datos.

Sintaxis*:*

initdb [-D *dbdir*]
[-i *sysid*]
[-W]
[--noclean | -n] [--debug | -d]

Opciones*:*

-D, *dbdir*

Esta opción especifica en qué parte del sistema de archivos será almacenada la base de datos. Ésta es la única información requerida por el initdb.

-i, *sysid*

Selecciona el id del sistema para el súper usuario (root) de la base de datos. Por omisión apunta al id de aquel usuario que este ejecutando initdb.

-W pwprompt

Ocasiona que el initdb pregunte por el password del súper usuario (root) de la base de datos. Si uno no planea usar la autentificación a través de password, entonces realmente no es

importante. De cualquier manera uno no podrá utilizar la autentificación a través de password hasta que haya establecido un password.

-n noclean

Por defecto, cuando initdb determina que un error ha ocurrido evita que se cree totalmente el sistema de base de datos, remueve cualquier archivo que pudo haber creado, antes de determinar que no puede acabar el trabajo. Esta opción inhibe "tidying-up" y es por lo tanto, útil para depurar.

-d debug

Imprime la salida de depuración de la "carga inicial backend" y algunos otros mensajes de poco interés para el público en general. La "carga inicial backend" es la aplicación que el initdb usa para crear las tablas del catálogo. Esta opción genera una enorme cantidad de salida.

Salidas*:*

initdb creará los ficheros en el área de datos especificada que son las tablas del sistema y el marco de trabajo para una instalación completa.

Postmaster

Descripción*:*

Postmaster administra la comunicación entre los procesos del cliente y del servidor, así como la asignación de buffers compartidos y semáforos. *Sólo un postmaster debe estar ejecutándose a la vez para una instalación Postgres dada.* Aquí una instalación significa un directorio de base de datos y un número de puerto de postmaster. Se puede ejecutar más de un postmaster en una misma máquina si cada uno de ellos tiene un directorio y un número de puerto diferente.

Sintaxis*:*

postmaster [-B *nBuffers*] [-D *DataDir*] [-N *nBackends*]
[-S] [-d [*DebugLevel*] [-i] [-l] [-o *BackendOptions*]
[-p *port*] postmaster [-n | -s]...

Opciones:

-B *nBuffers*

Indica el número de buffers de memoria compartida que postmaster asignará y administrará para los procesos servidores que inicie. El valor predeterminado para esta opción es 64 buffers, siendo cada buffer de 8 kilobytes (o lo que sea que esté indicado en BLCKSZ en config.h).

-D *DataDir*

Especifica el directorio a usar como raíz del árbol de directorios de las bases de datos. Si no se especifica -D, el nombre de directorio predeterminado es el valor de la variable de entorno PGDATA. Si PGDATA no está especificada, entonces se utiliza el directorio $POSTGRESHOME/data. Si no se especifica ni la variable de entorno ni esta opción de línea de comando, el directorio predeterminado es el utilizado al momento de la compilación.

-N *nBackends*

El máximo número de procesos servidores (backend) que postmaster tiene permitido iniciar. En la configuración predeterminada este valor está usualmente definido en 32, y puede ser fijado hasta un valor máximo de 1024 si su sistema puede soportar esa cantidad de procesos. Tanto el valor predeterminado como el máximo puede modificarse cuando se compila Postgres (vea el archivo src/include/config.h)

-S

Indica que el proceso postmaster debe iniciarse en modo silencioso. Esto anulará la vinculación con la terminal del usuario (que tiene el control) e iniciará su propio grupo de proceso. Esta opción no debería utilizarse en conjunto con las opciones de depuración ya que cualquier mensaje enviado a la salida estándar y a la salida de error estándar será descartado.

-d [*DebugLevel*]

Este argumento *DebugLevel* determina la cantidad de información de depuración que producirá el servidor. Si *DebugLevel* es uno, postmaster rastreará todo el tráfico de conexión y nada más. Para niveles iguales o mayores a 2 se activa la depuración y el proceso del servidor y postmaster muestran más información incluyendo el entorno del servidor y tráfico de proceso. Note que si no se especifica ningún archivo para que los servidores del backend envíen su información, esta

información será exhibida en la terminal de su proceso postmaster padre.

-i

Esta opción habilita las comunicaciones mediante TCP/IP o mediante el socket de dominio Internet. Sin esta opción solamente es posible la comunicación a través del socket de dominio Unix local.

-l

Este parámetro habilita la comunicación mediante el socket SSL. También es necesario especificar la opción -i. Además, debió habilitarse SSL en el momento de la compilación.

-o *BackendOptions*

Las opciones de postgres que se especifican en *BackendOptions* son pasadas a todos los procesos iniciados en el servidor por este postmaster. Si la cadena de opciones contiene espacios, entonces debe encerrársela entre comillas.

-p *port*

Especifica el puerto TCP/IP por el cual postmaster deberá esperar por conexiones solicitadas desde las aplicaciones del lado del cliente. El valor predeterminado es el especificado en la variable de entorno PGPORT o, si PGPORT no fue especificada, se toma como predeterminado el valor establecido cuando Postgres fue compilado (normalmente 5432). Si se especifica un puerto distinto del predeterminado, a todas las aplicaciones cliente (incluyendo psql) deberá especificárseles el mismo puerto ya sea mediante las opciones de línea de comando o utilizando la variable de entorno PGPORT.

Existen algunas opciones de línea de comandos disponibles para realizar depuraciones en caso de que un proceso en el servidor termine de forma anormal. Estas opciones controlan el comportamiento de postmaster en estas situaciones, y *ninguna de ellas está pensada para ser utilizada en situaciones normales*.

La estrategia usual para esta situación es notificar a todos los demás procesos en el servidor que deben terminar y reinicializar la memoria y semáforos compartidos. Esto es así debido a que un proceso de servidor que funcione de manera errática podría corromper alguno de estos recursos compartidos antes de terminar.

Estas opciones especiales son:

-n

postmaster no reinicializará las estructuras compartidas. Un programador podría luego examinar la memoria compartida y los estados de los semáforos.

-s

postmaster detendrá todos los demás procesos del servidor enviándoles la señal SIGSTOP, pero no hará que terminen. Esto permite a los programadores del sistema realizar vuelcos de núcleo a mano para todos los procesos del servidor.

Salidas:

semget: No space left on device

Si aparece este mensaje, debería ejecutar el comando ipcclean. Una vez hecho esto, pruebe iniciar postmaster nuevamente. Si aún no funciona, probablemente necesite configurar el núcleo (kernel) de su sistema para que pueda utilizar memoria compartida y semáforos. Si ejecuta múltiples instancias de postmaster en un sólo host, o tiene un kernel con muy poca memoria compartida o un límite de semáforos muy pequeño, tal vez deba reconfigurar su kernel para incrementar sus parámetros de memoria compartida y semáforos.

Tal vez pueda posponer la reconfiguración del kernel disminuyendo lo especificado con -B para reducir la utilización de memoria compartida por parte de Postgres, o disminuyendo lo especificado con -N para reducir la cantidad de semáforos que utiliza Postgres.

StreamServerPort: cannot bind to port

Si se encuentra con este mensaje, debe asegurarse de que no existen otros procesos de postmaster ejecutándose en el momento. La manera más fácil de determinar esto es mediante el comando:

```
$ ps aux | grep postmaster
```

Si está seguro de que no existen otros procesos de postmaster en ejecución, y aun así sigue recibiendo este error, intente especificar un puerto diferente utilizando la opción -p. También puede obtener este mensaje de error si finaliza postmaster y lo vuelve a iniciar inmediatamente utilizando el

mismo número de puerto; simplemente espere unos segundos hasta que el sistema operativo cierre el puerto antes de intentar nuevamente. Finalmente, puede que obtenga este mensaje de error si especifica un número de puerto que su sistema operativo considere reservado.

IpcMemoryAttach: shmat() failed: Permission denied

Una explicación plausible es que otro usuario intentó iniciar un proceso postmaster en el mismo puerto el cual ha adquirido recursos compartidos y luego ha finalizado. Dado que las claves de memoria compartidas de Postgres se basan en el número de puerto asignado al proceso postmaster, estos conflictos tiene más probabilidad de ocurrir si existe más de una instalación en un mismo servidor. Si no hay otros procesos postmaster en ejecución, ejecute ipcclean e intente nuevamente. Si existen otros postmaster ejecutándose, deberá contactar a los propietarios de estos procesos para coordinar la asignación de puertos y/o la remoción de los segmentos de memoria compartida no utilizados.

Nota*:*

Siempre que sea posible *evite* utilizar SIGKILL para forzar la finalización de postmaster. En su lugar debería utilizarse SIGHUP, SIGINT o SIGTERM. La utilización es:

$ kill –s SIGTERM No_proceso_postmaster

Createdb

Descripción:

createdb crea una nueva base de datos PostgreSQL. El usuario que ejecuta este comando se convierte en el propietario de la base de datos. createdb es un script shell que envuelve un comando SQL *CREATE DATABASE* a través del terminal interactivo de PostgreSQL *psql*. Así pues, no hay nada especial sobre la creación de bases de datos por este u otros métodos.

Sintaxis:

createdb [opciones] nombre_database [descripción]

Opciones:

-h, *--host:*
Especifica el nombre de host (hostname) de la máquina sobre la que está ejecutándose el postmaster.

-p, *--puerto:*
Especifica el puerto TCP/IP o el socket local Unix en el cual el postmaster esta escuchando para recibir conexiones.

-U, --nombre de usuario:
Nombre del usuario que usa createdb.

-W, *--password*
Fuerza a que se teclee un password.

-q, *--quiet*
No muestra respuesta alguna.

-D, *--localización de la base de datos:*
Especifica la localización alternativa de la base de datos para esta instalación. Esta es la localización de las tablas del sistema.

dbname

Especifica el nombre de la base de datos que será creada. El nombre debe ser único entre todas las bases de datos PostgreSQL en esta instalación. Los nombres de las bases de datos han de comenzar con un carácter alfabético y su longitud está limitada a 31 caracteres. El valor por omisión es crear una base de datos con el mismo nombre que el usuario en curso del sistema.

Descripción

Opcionalmente esto especifica un comentario que será asociado con la base de datos nuevamente creada.

Salidas:

CREATE DATABASE

Nota: Si se recibe como salida el mensaje anterior es porque la base de datos fue creada exitosamente.

Ejemplo:

1. Para crear la base de datos demo utilizando el servidor por defecto de base de datos:

$ **createdb** demo
CREATE DATABASE

Nota: La respuesta sería la misma si se hubiese ejecutado el comando de SQL **CREATE DATABASE**.

2. Para crear la base de datos anterior utilizando el postmaster en la máquina (host) localhost, puerto 5000:
   $ **createdb -p** 5000 **-h** localhost demo
   CREATE DATABASE

Dropdb

Descripción*:*

dropdb destruye una base de datos PostgreSQL existente. El usuario que ejecute este comando debe ser un superusuario de la base de datos o su propietario. dropdb es un envoltorio del comando SQL Drop Database por medio del terminal interactivo *psql* de PostgreSQL. Por esto no hay nada de especial en borrar bases de datos por medio de este u otros métodos.

Sintaxis*:*

dropdb [*opciones*] *nombre_bd*

Opciones*:*

-h*,* --host
Especifica el nombre de host de la máquina sobre la que el postmaster esté corriendo.

-p, *--puerto*
Especifica el puerto TCP/IP o el socket Unix local en el que postmaster atiende conexiones.

-U, *--nombre_usuario*
Nombre del usuario que se va a conectar.

-W, --password
Fuerza la introducción de un password.

-e, --echo
Muestra en pantalla las consultas que dropdb genera y envía al backend.

-i, --interactivo
Antes de hacer algo destructivo, pide confirmación a través del prompt.

nombre_bd:
Especifica el nombre de la bases de datos que va a ser borrada. Debe ser una de las existentes en esta instalación de PostgreSQL.

Salida*:*
DROP DATABASE

Nota: Si se recibe como salida el mensaje anterior es porque la base de datos fue borrada exitosamente.

Ejemplo*:*

1. Para destruir la base de datos demo en el servidor de bases de datos por defecto:
   $ **dropdb demo**
   DROP DATABASE

2. Para destruir la base de datos demo usando el postmaster del host localhost, puerto 5000, con verificación y echando un vistazo a la consulta en marcha:

   $ **dropdb -p 5000 -h localhost -i -e demo**
   Database "demo" will be permanently deleted.
   Are you sure? (y/n) **y**
   DROP DATABASE "demo"
   DROP DATABASE

Create user

Descripción:
Añadirá un nuevo usuario PostgreSQL. Debe ser un súper usuario de bases de datos para usar este comando.

Use *alter user* para cambiar la palabra clave y los privilegios de un usuario y *drop user* para eliminar a un usuario. PostgreSQL viene con un script *createuser* que tiene la misma funcionalidad que este

comando (de hecho llama a este comando) pero puede ser ejecutado desde la línea de comandos.

Sintaxis*:*

Create user nombre_usuario [WITH]
[SYSID uid]
[PASSWORD 'palabra clave']]
[CREATEDB | NOCREATEDB] [CREATEUSER | NOCREATEUSER]
[IN GROUP nombre_grupo [,…]]
[VALID UNTIL 'abstime']

Entradas*:*

***nombre*_usuario:**

El nombre del usuario.

uid:

La orden SYSID puede ser usada para escoger el identificador de usuario PostgreSQL del usuario que se esta creando. No es nada necesario que corresponda a los identificadores de usuarios de UNIX, pero algunas personas eligen mantener los números iguales. Si no se especifica, se usará por defecto el número más alto asignado más uno.

palabra clave:

Pide la palabra clave del usuario. Si no va a usar autenticación por palabra clave puede omitir esta opción, de otra manera el usuario no será capaz de conectar con el servidor de autenticación de palabras claves.

CREATEDB | NOCREATEDB:

Estas órdenes definen la capacidad de un usuario para crear bases de datos. Si se especifica CREATEDB, el usuario definido tendrá permiso para crear sus propias bases de datos. Usando NOCREATEDB se degenerará a un usuario la capacidad de crear bases de datos. Si se omite esta orden, NOCREATEDB se usa por defecto.

CREATEUSER | NOCREATEUSER:

Estas órdenes determinan si a un usuario se le permitirá crear nuevos usuarios. Esta opción harán del usuario un súper usuario

que podrá pasar por encima de todas las restricciones de acceso. Si se omite esta orden se cogerá la orden de NOCREATEUSER como valor por defecto del usuario.

nombre_grupo:

El nombre de un grupo dentro del cual se coloca al usuario como un nuevo miembro.

abstime:

Esta orden pone un valor absoluto a la fecha en la que la palabra clave del usuario pierde su validez. Si se omite esta orden el login valdrá para siempre.

Salida*:*

CREATE USER

Nota: Si se recibe como salida el mensaje anterior es porque el usuario ha sido creado correctamente.

Ejemplo*:*

1. Para crear un usuario pepe en la base de datos por defecto:
   create user pepe;
   CREATE USER

2. Para crear al mismo usuario usando un password
   create user pepe with password 'pepito';
   CREATE USER

Alter User

Descripción:

Modifica la información de la cuenta de usuario. Se usa para cambiar los atributos de la cuenta de un usuario de PostgreSQL. Sólo un súper usuario de una base de datos puede cambiar privilegios y fechas de caducidad de palabras clave con esta orden. Ordinariamente los usuarios sólo pueden cambiar su propia palabra clave.

Sintaxis:

ALTER USER nombre_de_usuario
[WITH PASSWORD ´palabra clave´]
[CREATEDB|NOCREATEDB][CREARTEUSER|
NOCREATEUSER]
[VALID UNTIL ´abstime´]

Ejemplo:

1. Cambiar la palabra clave de un usuario:
   ALTER USER david WITH PASSWORD ´hu8jmn3´;

2. Cambiar la validez de un usuario hasta la fecha:
   ALTER USER José VALID UNTIL ´31 En 2030´;

3. Cambiar la validez de un usuario hasta la fecha, especificando que su autorización expirara al mediodía del 4 de Mayo de 1998.
   ALTER USER chris VALID UNTIL ´4 May 12:00:00 1998´;

4. Dar a un usuario la capacidad de crear otros usuarios y nuevas bases de datos.
   ALTER USER Maria CREATEUSER CREATEDB;

Entrada:

Nombre_de_usuario:

El nombre del usuario cuyos detalles van a ser modificados.

palabra clave:

La nueva palabra clave que va a ser usada en esta cuenta.

CREATEDB | NOCREATEDB:

Estas cláusulas definen la capacidad de un usuario para crear bases de datos. Si se especifica CREATEDB, el usuario podrá definir sus propias bases de datos. Usando NOCREATEDB se deniega a un usuario la capacidad de crear bases de datos.

CREATEUSER | NOCREATEUSER:

Estas cláusulas determinan si un usuario está autorizado a crear nuevos usuarios él mismo. Está opción hace ser además al

usuario un súper usuario que puede pasar por encima de todas las restricciones de acceso.

abstime:

La fecha (y opcionalmente la hora) en la que la palabra clave de este usuario expirará.

Salida:

ALTER USER

Nota: Mensaje recibido si la modificación es correcta.

ERROR: ALTER USER usuario "nombre de usuario" no existe

Nota: Mensaje de error recibido si el usuario especificado no existe en la base de datos.

Drop user

Descripción*:*

Retira de la base de datos el usuario dado. No retira tablas, vistas u otros objetos que pertenezcan al usuario. Si el usuario es dueño de una base de datos, se producirá un error.

Use *create user* para adicionar nuevos usuarios y *alter user* para modificar las propiedades de un usuario. PostgreSQL viene con un guión *drop user* que tiene la misma función de este comando (de hecho invoca este comando) pero que puede ser ejecutado desde la shell.

Sintaxis:

drop user *nombre_usuario*

Entrada:

nombre_usuario:

El nombre de un usuario existente.

Salida*:*

DROP USER

Nota: El mensaje que se obtiene si el usuario ha sido retirado con éxito.

Ejemplo*:*

1. Para borrar al usuario pepe del servidor de bases de datos por defecto:

$ **drop user pepe**
DROP USER

Create table

Descripción:

Crea una nueva clase o tabla en la base de datos actual. Cada tipo de dato puede ser un tipo simple, un tipo complejo (set) o un tipo array. Cada atributo puede ser específico para ser no nulo, y puede tener un valor por defecto, especificado por la cláusula DEFAULT. Una tabla no puede tener más de 1600 campos.

Sintaxis:

```
CREATE [TEMPORARY | TEMP] TABLE nombre_tabla (
    campo tipo
    [NULL | NOT NULL]  [UNIQUE]  [DEFAULT value]
    [column_constraint_clause  |  PRIMARY KEY  }  [ … ]  ]
    [ ,  …  ]
    [ , PRIMARY  KEY  ( column [ , … ]  )  ]
    [, … CHECK  ( condition ) ]
    [, table_constraint_clause ]
    )  [INHERITS (inherited_table   [, …]   )  ]
```

Entrada:

TEMPORARY:

Se crea la tabla sólo para esta sesión, y es eliminada automáticamente con el fin de la sesión. Las tablas permanentes existentes con el mismo nombre no son visibles mientras la tabla temporal existe.

nombre_tabla:

El nombre de una nueva clase o tabla a crear.

campo:
El nombre de un campo.

tipo:
El tipo del campo. Puede incluir especificaciones de array.

DEFAULT value:
Un valor por defecto para el campo.

column_constraint_clause:
La cláusula opcional de restricciones (constraint) especifica una lista de restricciones de integridad o comprueba que las nuevas inserciones o actualizaciones deben satisfacer, para que la inserción o la actualización tenga éxito. Cada restricción debe evaluarse a una expresión booleana. Aunque SQL92 requiere la column_constraint_clause para referirse a ese campo solamente PostgreSQL permite que múltiples campos sean referenciados dentro de un único campo constraint.

table_constraint_clause:
La cláusula opcional CONSTRAINT especifica una lista de restricciones de integridad que las nuevas inserciones o las actualizaciones deberán satisfacer para que una sentencia insert o update tenga éxito. Cada restricción debe ser evaluada a una expresión booleana. Se pueden referenciar múltiples campos con una única restricción. Solo se puede definir una única cláusula PRIMARY KEY por tabla; PRIMARY KEY column (una restricción de tabla) y PRIMARY KEY (una restricción de campo) son mutuamente excluyente.

INHERITS inherited_table:
La cláusula opcional INHERITS especifica una colección de nombres de tabla de las cuales esta tabla hereda todos los campos. Si algún campo heredado aparece más de una vez PostgreSQL informa de un error. PostgreSQL permite automáticamente a la tabla creada heredar funciones de las tablas superiores a ella en la jerarquía de herencia.

> **Nota:** La herencia de funciones se realiza siguiendo las convenciones del Common Lisp Object System (CLOS).

Salida:

CREATE TABLE

Nota: Mensaje devuelto si la tabla se ha creado con éxito.

ERROR:

Nota: Mensaje devuelto si la creación de la tabla falla. Este mensaje viene normalmente acompañado por algún texto explicativo, como:

ERROR: Relation 'table' already exists

Nota: Ocurre en tiempo de ejecución, si la tabla especificada ya existe en la base de datos.

ERROR: DEFAULT: type mismatched:

Nota: Si el tipo de datos o el valor por defecto no corresponden al tipo de datos de la definición del campo.

Alter table

Descripción:

Cambia la definición de una tabla existente. La orden ADD COLUMNS, añade una nueva columna a la tabla usando la misma sintaxis que CREATE TABLE. La orden ALTER COLUMN le permite poner o eliminar los valores por defecto de la columna. Nótese que los valores por defecto solo se aplicaran a las líneas nuevas que inserte. La cláusula RENAME causa que el nombre de una tabla o de una columna cambie sin que se modifique ninguno de los datos contenidos en la tabla afectada. De este modo, la tabla o la columna permanecerán del mismo tipo y tamaño después de que este comando sea ejecutado. Usted debe ser el creador de esta tabla para poder cambiar su esquema.

" * " siguiendo a un nombre de una tabla indica que la orden debe ejecutarse sobre esa tabla y todas las tablas que estén bajo ella en la jerarquía subsecuente; por defecto, el atributo no será añadido a o

renombrado en ninguna de las subclases. Esto siempre se debe hacer cuando se añade o modifica un atributo en una superclase.

Ejemplo:

1. Para añadir a una columna de tipo VARCHAR a una tabla:
   ALTER TABLE distribuidores ADD COLUMN direcciones VARCHAR (30);

2. Para renombrar una columna existente:
   ALTER TABLE distribuidores RENAME COLUMN direcciones TO ciudad;

3. Para renombrar una tabla existente:
   ALTER TABLE distribuidores RENAME TO proveedores;

Sintaxis:

```
ALTER TABLE nombre_tabla  [*]
ADD  [COLUMN]  nombre_columna tipo_de_columna

ALTER TABLE nombre_tabla  [*]
ALTER [COLUMN] nombre_columna  {SET DEFAULT valor |
DROP DEFAULT}

ALTER TABLE nombre_tabla  [  *  ]
RENAME  [COLUMN]  nombre_columna  TO  nuevo_nombre

ALTER TABLE nombre_tabla
RENAME TO  nuevo_nombre_tabla
```

Entradas:

nombre_tabla:
El nombre de una tabla existente para modificarla.

nombre_columna:
Nombre de una columna nueva o ya existente.

tipo_de_columna:
Tipo de la nueva columna.

nuevo_nombre:
Nuevo nombre para una columna ya existente.

nuevo_nombre_tabla:
Nuevo nombre para la tabla.

Salida:

ALTER:
Mensaje recibido de la columna o la tabla que se ha renombrado.

ERROR:
Mensaje recibido si la tabla o la columna no son validas.

Drop table

Descripción:

Elimina tablas y vistas de una base de datos. Solo su propietario puede destruir una tabla o vista. Una tabla puede ser vaciada de sus filas, pero no destruida, usando Delete o Truncate.

Si una tabla a de ser destruida tiene un índice secundario, este debe ser removido primero. La remoción de solo un índice secundario no afecta el contenido de las tablas subyacente.

Sintaxis:

DROP TABLE nombre [, . . .]

Ejemplo:

1. Para destruir dos tablas, cintas y distribuidores:
   DROP TABLE cintas, distribuidores;

Entradas:

nombre:
El nombre de una tabla vista existente para eliminarla.

Salidas:

DROP:
El mensaje devuelto si el comando concluyo exitosamente.

ERROR Relation "nombre" Does Not Exist!

Si la tabla o vista especificada no existe en la base de datos.

Insert into

Descripción:

Permite la inserción de nuevas filas en una clase o una tabla. Se puede insertar una fila a la vez o varias como el resultado de una consulta. Las columnas en el resultado pueden ser dictadas en cualquier orden.

Cada columna que no este presente en la lista de origen será insertada usando el valor por defecto, que puede ser tanto un valor por defecto declarado default o bien null. PostgreSQL rechazará la nueva columna si se inserta null en una columna declarada como not null.

Debe tener privilegios de inserción en la tabla para añadir en ella, así como también privilegios de selección en cualquier tabla especificada en una cláusula where.

Sintaxis:

INSERT INTO tabla [(columna1, columna2,...)]
{VALUES (valor1, valor2,...) | SELECT consulta}

Entrada:

tabla:

El nombre de una tabla existente

columna1,...2:

El nombre de una columna en tabla.

Valor:

Un valor valido a asignar en columna.

Consulta:

Una consulta valida.

Salida:

INSERT oid 1:

Nota: mensaje devuelto si solo se ha insertado una fila. Oid es el numero OID de la fila insertada.

INSERT 0 #:

Nota: mensaje devuelto si solo se ha insertado mas de una fila. # es el numero de la fila insertada.

Ejemplo:

1. Inserta una fila en la tabla prueba:
   INSERT INTO prueba VALUES ('001','Wanda', 15);

Nota: el ejemplo anterior inserta en la tabla prueba con atributo código, nombre y edad los valores anteriores.

Update

Descripción:

Cambia el valor de las columnas especificadas por todas las filas que satisfacen la condición dada. Solamente necesita indicar las columnas que serán modificadas.

Debe tener permiso de escritura sobre la tabla para poder modificarla, así como permiso de lectura de cualquier tabla cuyos valores sean mencionados en la condición WHERE.

Sintaxis:

UPDATE nombre_tabla SET columna=expresión[,...]

[FROM lista]

[WHERE condición]

Entrada:

nombre_tabla:

El nombre de una tabla existente.

columna:

El nombre de la columna en la tabla.

expresión:

Una expresión valida o valor a ser asignado a la columna

lista:

Es una extensión no estándar de postgres que permite la aparición de columnas de otras tablas en la condición WHERE.

Salida:

UPDATE #

Nota: mensaje obtenido si se actualizo correctamente. El símbolo de # representa el numero de filas que han sido actualizadas. Si # es igual a 0 ninguna fila fue actualizada.

Delete

Descripción:

Borra las filas que satisfacen la cláusula WHERE de la tabla especificada. Si la condición (cláusula WHERE) esta ausente, el efecto es borrar todas las filas de la tabla. El resultado es una tabla valida, pero vacía.

Para modificar la tabla usted debe poseer acceso de escritura a la misma, así como acceso de lectura a cualquier tabla cuyos valores son leídos en la *condición.*

Sintaxis:

DELETE FROM nombre_table [WHERE condition]

Ejemplo:

1. Borra todos los registros de la tabla especificada excepto los que tengan como contenido de nombre_campo valor:
DELETE FROM nombre_tabla WHERE nombre_campo < > ´valor´;
SELECT * FROM films;

2. Borrar todos los registros de la tabla especificada:
DELETE FROM nombre_tabla;
SELECT * FROM nombre_tabla;

Entrada:

Nombre_table:

El nombre de una tabla existente.

condición:

Esta es una consulta SQL de selección la cual devuelve las filas a ser borradas.

Salida:

DELETE count:

Nota: Mensaje devuelto si los ítems son borrados exitosamente. El valor count es la cantidad de filas borradas. Si count es 0, ninguna fila fue borrada.

Truncate

Descripción:

Remueve rápidamente todas las filas de una tabla. Tiene el mismo efecto que el DELETE pero al no recorrer la tabla resulta más rápido. Es más efectivo en tablas grandes.

Sintaxis:

TRUNCATE [TABLE] nombre_tabla

Entrada:

Nombre_tabla:

el nombre de la tabla a truncar.

Salidas:

TRUNCATE

Nota: mensaje retornado si la tabla ha sido vaciada (truncada) exitosamente.

Copy

Descripción:

Mueve datos entre tablas de PostgreSQL y ficheros de sistemas de archivos estándar. Copy indica al servidor de postgres que lea o escriba de o a un fichero. El fichero a de ser directamente visible para el servidor, y el nombre completo a de especificarse desde el punto de vista del servidor. Si se especifica stdin o stdout, los datos van de la aplicación cliente al servidor(o viceversa).

Nota: la palabra clave BINARY obliga a que todos los datos se almacenen como objetos binarios en lugar de como textos. Esto es algo más rápido que el comportamiento normal de postgres, pero el resultado no es generalmente portable, y los fichero generados son algo mas grande aunque este es un factor que depende de los datos en sí. Por defecto, cuando se copia un texto se usa un tabulador ("\t") como delimitador. El delimitador puede cambiarse por cualquier otro carácter empleando la palabra clave USING DELIMITERS. Los caracteres dentro de los campos de datos que resulten coincidir con el delimitador serán encerrados entre comillas.

Sintaxis:

```
COPY [BINARY] tabla [WITH OIDS]
FROM {'nombre_archivo' | stdin}
[[USING] DELIMITERS 'delimitador']
[WITH NULL AS 'null string']

COPY [BINARY] tabla [WITH OIDS]
TO {'nombre_archivo' | stdout}
[[USING] DELIMITERS 'delimitador']
[WITH NULL AS 'null string']
```

Entrada:

BINARY:

Cambia el comportamiento de formato de campos, forzando a todos los datos a almacenarse o leerse como objetos binarios, en lugar que como texto.

Tabla:

El nombre de una tabla existente.

WITH OIDS:

Copia el identificador de objeto interno único (OID) para cada fila.

Nombre_archivo:

La ruta absoluta del fichero entrada o salida.

Stdin:

Especifica que la entrada viene de un conducto o terminal.

Stdout:

Especifica que la salida va a un conducto o terminal.

Delimitador:

Un carácter que delimita los campos de entrada o salida.

Null string:

Una cadena para representar valores null. El valor por defecto es "\N", aunque se puede elegir usar una cadena vacía.

Salida:

COPY

Nota: este mensaje aparece cuando la copia se ha completado satisfactoriamente.

ERROR: reason

Nota: este mensaje aparece cuando la copia falló por la razón indicada en el mensaje de error.

Begin

Descripción:

Por defecto, PostgreSQL ejecuta las transacciones en modo no encadenado (también conocido como "autocommit" en otros sistemas de base de datos). En otras palabras, cada estado de usuario es

ejecutado en su propia transacción y un commit se ejecuta implícitamente al final (si la ejecución fue exitosa, de otro modo se ejecuta un rollback). BEGIN inicia una transacción de usuario en modo encadenado, o sea, todos los estados de usuarios después de un comando BEGIN se ejecutaran en una transacción única hasta un explicito COMMIT, ROLLBACK, o aborte la ejecución. Los estados en modo encadenado se ejecutan mucho más rápido, porque la transacción START/COMMIT requiere una actividad significativa de CPU y de disco. La ejecución de múltiples estados dentro de una transacción también es requerida para la consistencia cuando se cambian muchas tablas relacionales.

Nota: utilice COMMIT o ROLLBACK para terminar una transacción.

Sintaxis:

BEGIN [WORK | TRANSACTION]

Entrada:

WORK | TRANSACTION:

Palabras claves opcionales. No tienen efecto.

Salida:

BEGIN

Nota: esto significa que una transacción ha sido comenzada.

NOTICE: BEGIN: already a transaction in progress.

Nota: esto indica que una transacción ya esta en progreso. La transacción en curso no se ve afectada.

Commit

Descripción:

Realiza la transacción actual. Todos los cambios realizados por la transacción son visibles a las otras transacciones, y se garantiza que se conservan si se produce alguna caída del sistema.

Nota: las palabras claves WORK y TRANSACTION pueden ser omitidas. Use ROLLBACK para abortar una transacción.

Sintaxis:

COMMIT [WORK | TRANSACTION]

Entrada:

WORK | TRANSACTION:

Palabras claves opcionales. No tienen efecto.

Salida:

COMMIT

Nota: esto significa que una transacción se realizó con éxito.

NOTICE: COMMIT: no transaction in progress

Nota: si no hay transacciones en progreso.

Rollback

Descripción:

Deshace la transacción actual y provoca que todas las modificaciones originadas por las mismas sean descartadas.

Nota: utilice COMMIT para terminar una transacción de forma exitosa. ABORT es un sinónimo de ROLLBACK.

Sintaxis:

ROLLBACK

Entrada:

ninguna.

Salida:

ABORT

Nota: mensaje devuelto si la operación es exitosa.

NOTICE: ROLLBACK: no transaction in progress.

Nota: Si no hay transacciones en progreso actualmente.

Create group

Descripción:

Permite crear un nuevo grupo en la base de datos. Esta orden solamente podrá ser ejecutada por un usuario administrativo.

Sintaxis:

CREATE GROUP name
[WITH [SYSID gid]
[USER username [. . .]]]

Entrada:

name:

El nombre del grupo.

Gid:

La cláusula SYSID puede ser usada para elegir el número id del grupo PostgreSQL del grupo nuevo. El uso de esta cláusula es opcional.
En caso de no especificar el número id del grupo, se asignará el número mayor ya asignado mas uno, empezando por 1.

Username:

Una lista de los usuarios a incluir en el grupo. Los usuarios tienen que existir antes de incluirlos en el grupo.

Salidas:

CREATE GROUP:

Nota: Mensaje que será devuelto siempre que la orden termine con éxito.

Drop group

Descripción:

Elimina el grupo especificado de la base de datos. Los usuarios del grupo no se eliminan.

Sintaxis:

DROP GROUP name

Entrada:

name:

El nombre de un grupo existente.

Salida:

DROP GROUP

Nota: Mensaje devuelto si el grupo es eliminado satisfactoriamente.

Create table as

Descripción:

Permite a una tabla ser creada a partir del contenido de una tabla existente. Equivale en funcionamiento a: SELECT INTO, pero quizás con una sintaxis más directa.

Sintaxis:

CREATE TABLE nombre_tabla [(column [. . .])]
AS select_clause

Entrada:

nombre_tabla:

El nombre de una nueva tabla a ser creada.

column:

El nombre de una columna. Se pueden especificar múltiples nombre de columna usando una lista de nombres de columna delimitada por comas.

select_clause:

Una sentencias de consulta valida.

Pg_dump

Descripción:

Es un utilitario para volcar una base de datos Postgres en un fichero de script conteniendo comandos de consulta. Los ficheros de script son en formato de texto y pueden ser usados para reconstruir la base de datos, incluso en otras máquinas y con otras arquitecturas. Pg_dump producirá las consultas necesarias para regenerar todos los tipos definidos por el usuario, funciones, tablas, índices, agregados, y operadores. Adicionalmente, todos los datos son copiados en formato de texto el cual puede ser nuevamente copiado, también puede ser importado a herramientas para su edición.

Pg_dump es útil para verter el contenido de una base de datos que se vaya a mudar de una instalación de Postgres a otra. Después de ejecutar pg_dump, se debe examinar el script de salida a ver si contiene alguna advertencia.

Nota: pg_dump tiene pocas limitaciones. Las limitaciones surgen principalmente de la dificultad para extraer cierta meta-información de los catálogos del sistema.

Pg_dump no maneja objetos grandes. Los objetos grandes son ignorados y se debe lidiar con ellos de forma manual.

Sintaxis:

Pg_dump [base_de_datos]
pg_dump [-h huésped] [-puerto]
[-t tabla]
[-a] [-c] [-d] [-D] [-n] [-N]
[-o] [-s] [-u] [-v] [-x]
[base_de_datos]

Entrada:

base_de_datos:

Especifica el nombre de la base de datos que se va a extraer. base_de_datos tiene como estándar el valor de la variable de entorno USER.

-a:

Vuelca sólo los datos, no el esquema (las definiciones).

-c:

Limpia el esquema antes de crearlo.

-d:

Vuelca los datos como propios insertos de cadenas.

-D:

Vuelca los datos como insertos con nombres de atributos.

-n:

Suprime las dobles comillas de los identificadores, a menos que sean absolutamente necesarios. Esto puede causar problemas al cargar la misma si los datos volcados contienen palabras reservadas usadas por los identificadores.

-N:

Incluye comillas dobles en los identificadores. Este es el estándar.

-o:

Vuelca los identificadores de objetos (OIDs) para cada tabla.

-s:

Vuelca sólo el esquema (las definiciones), no los datos.

-t tabla:

Vuelca los datos para la tabla únicamente.

-u:
Usa autenticación por medio de clave de acceso. Pide un nombre de usuario y clave de acceso.

-v:
Especifica el modo verbose (parlanchín).

-x:
Evita el volcado de ACLs (comandos grant/revoke) y la información de propiedad de la tabla.

Pg_dump también acepta los siguientes argumentos de línea de comando para parámetros de conexión:

-h huésped:
Especifica el nombre de la máquina en la cual se esta ejecutando el postmaster. El estándar es usar un socket de dominio local Unix en vez de una conexión IP.

-p puerto:
Especifica el puerto de Internet TCP/IP o extensión de archivo socket de dominio local Unix en el cual postmaster está esperando que se efectúen conexiones. El número estándar de puerto es 5432, o el valor de la variable de ambiente PGPORT (si está establecida).

-u:
Usa autenticación con clave de acceso. Pide nombre_de_usuario y clave_de_acceso.

Salida:

Pg_dump creará un fichero o escribirá a stdout.

La conexión con la base de datos ´template´ falló. connectDB() falló: ¿Está el postmaster ejecutándose y aceptando conexiones en el ´Socket de Unix´en el puerto ´puerto´?

Nota: este error se obtiene cuando pg_dump no puede unirse al proceso postmaster en el huésped y puerto especificados. Si ve usted este mensaje, verifique que postmaster se este ejecutando en el huésped indicado, y que usted especificó el puerto correcto. Si su site usa algún sistema de autenticación,

verifique que usted tiene las credenciales de autenticación requeridas.

La conexión con la base de datos ´base_de_datos´ falló. Fatal 1: SetUserId: el usuario ´nombre_de_usuario´ no está en ´pg_shadow´

Nota: el usuario no posee una entrada válida en la relación pg_shadow y no le será permitido tener acceso a Postgres.

DumpSequence (tabla): SELECT falló

Nota: el usuario carece del permiso para leer la base de datos.

Ipcclean

Descripción:

Limpia la memoria compartida y el espacio de semáforos de "backends" abortados, borrando todas las instancias que son propiedad del usuario postgres. Solamente el DBA (administrador de base de datos) debe ejecutar este programa ya que puede causar algún tipo de comportamiento extraño, es decir caídas, si se ejecuta durante una ejecución multiusuario. Este programa se debe ejecutar si aparecen mensajes como por ejemplo *semget: no queda espacio libre en el dispositivo,* al ejecutar el proceso postmaster o el servidor "backend".

Si se ejecuta esta orden mientras el proceso postmaster está corriendo, eliminará la memoria compartida y los semáforos almacenados por el postmaster. Esto puede provocar el fallo general de los servidores "backend" iniciados por ese postmaster.

Sintaxis:

ipcclean.

Entrada:

Ninguna.

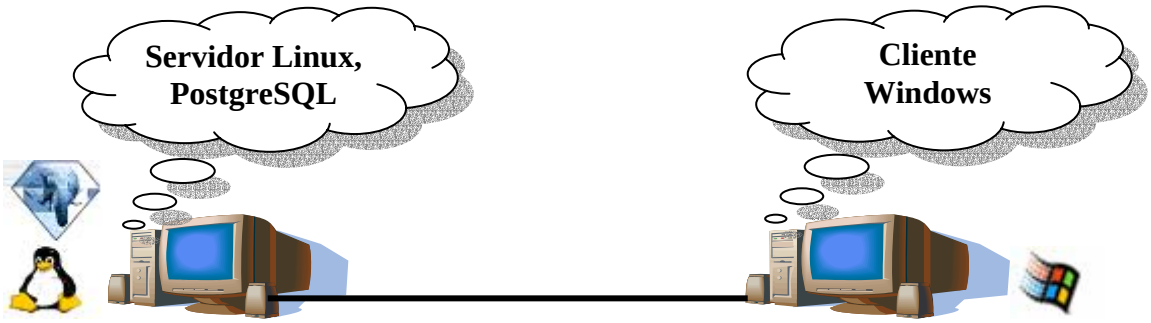
Salida:

Ninguna.

Accediendo a un Servidor PostgreSQL desde clientes Windows

Los sistemas que son usados actualmente por compañías, instituciones, agrupaciones pequeñas o grandes, tratan de utilizar sistemas homogéneos, es decir, las mismas plataformas, las mismas aplicaciones para no tener problemas de conectividad, eso seria lo ideal, pero bueno estamos en un mundo globalizado, en donde cada vez mas las empresas se asocian y comparten información, cuando sucede esto lo primero que tienen que hacer es interconectar sus sistemas, para poder compartir información o los recursos deseados, pero también se enfrentan muchas veces al primer problema, *"la plataforma que usa la otra empresa es distinta a la que usamos nosotros!!"*, la base de datos, no son las mismas, ahí es donde entran los sistemas abiertos que hacen que no importen las diferencias.

Bien nos enfrentaremos a escenarios con sistemas heterogéneos. Existen muchos estándares para lograr conectividad, uno de ellos es el estándar ODBC (Open Data Base Conectivity), el cual permite hacer conexión con un origen de Datos, o sea, mediante un controlador acceder a una base de datos desde plataformas distintas de la plataforma en que está el origen de datos (nuestro Gestor de Base de Datos).



Para esto necesitaremos tener el drivers ODBC para PostgreSQL, el cual lo podemos obtener del siguiente servidor:

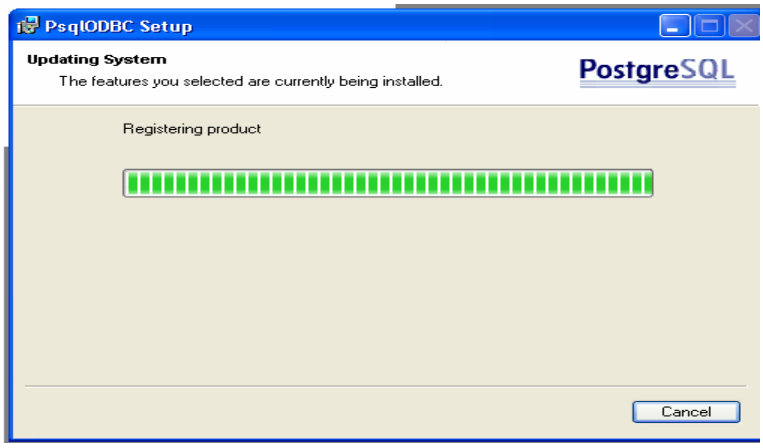
ftp: http://odbc.postgresql.org/ftpsite/odbc/versions/full/

A continuación se describe la instalación y configuración de PsqlODBC en una plataforma Windows XP.

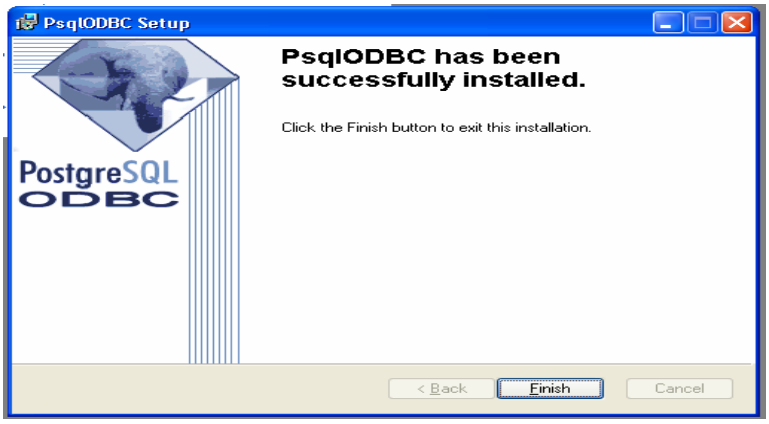
Instalación del driver ODBC

1. En primer lugar, lanzamos la ejecución del fichero PsqlODBC.exe con lo cual aparece el asistente de instalación.

2. Damos click en siguiente (next).

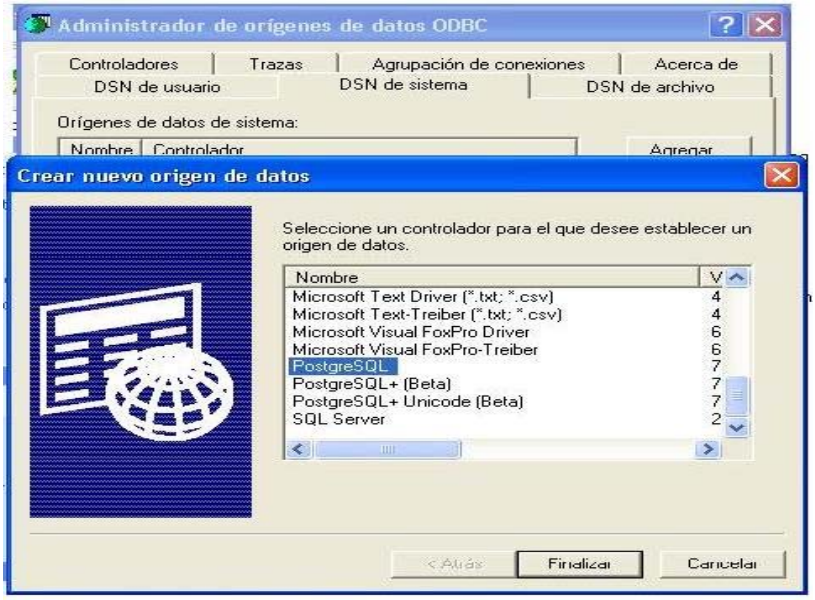


3. Y una vez instalados todos los elementos, damos click en finalizar.

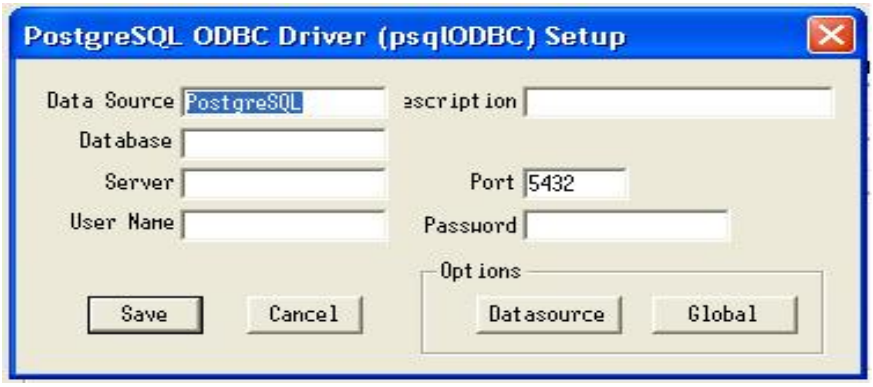


Configuración del Driver ODBC

1. Seguidamente, desde el panel de control, seleccionamos la opción Rendimiento y Mantenimiento, luego Herramientas del Sistema y ahora de click en origen de datos ODBC y configuraremos el DSN (data source names) del sistema, añadiendo el driver PsqlODBC que acabamos de instalar. Para ello damos clic en la opción "Agregar" nos aparece el siguiente cuadro. Seleccionamos *PostgreSQL*



2. Después dé click en finalizar con lo que aparecerá la figura siguiente:



Y ahora procedemos a configurarlo. Para ello debemos insertar los datos correspondientes a nuestra base de datos (*Database* = nombre de la base de datos a la que se quiere acceder), nuestro servidor (*Server* = nombre o número IP del servidor donde está alojada nuestra base de datos), el nombre de usuario (*User Name* = nombre del *usuario postgres* autorizado para acceder a la base de datos, ver configuración del archivo pg_hba.conf) y el *Password* de acceso (Contraseña del *usuario postgres*).

3. Si usted desea puede seleccionar las opciones avanzadas Data Source y Global que nos permitirán de manera general:

    - Seleccionar opciones de acceso (read-only, exclusive...)

- Especificar Conversiones de tipo.
- Elegir Opciones de cache y de métodos de indexación.
- Decidir si se exportan o no las tablas internas.

Una vez configurado el sistema, cerramos el panel de control y aunque no es necesario reiniciar la máquina, no estaría de más.

Para que la configuración tenga éxito, el servidor de PostgreSQL debe estar correctamente configurado y funcionando. Pruebe haciendo un Telnet desde Windows a Linux al puerto 5432 (donde se instala por defecto el servidor) no debe dar lugar al siguiente mensaje *"conection refused by server".*

Finalmente, ahora que ya tenemos el servidor y el cliente instalados, podemos empezar a trabajar con el sistema.

Seguridad

La seguridad de la base de datos en PostgreSQL está implementada en los siguientes niveles:

1. Protección de los ficheros de la base de datos. Todos los ficheros almacenados en la base de datos están protegido contra escritura por cualquier cuenta que no sea la del súper usuario de postgres.

2. Las conexión de los clientes al servidor de la base de datos están permitidas por defecto, únicamente por sockets Unix locales y no mediante socket TCP/IP. Ha de arrancarse el demonio con la opción **– i** para permitir la conexión de clientes no locales.

3. Las conexiones de los clientes remotos se pueden restringir por dirección IP y/o por nombre de usuario mediante el fichero pg_hba.conf situado en PG_DATA.

4. A cada cliente de postgres se le asigna un nombre de usuario y una contraseña, esta es opcional. Por defecto, los usuarios no tienen permiso de escritura a base de datos que no hayan creado.

5. Los usuarios pueden ser incluidos en grupos, y el acceso a las tablas pueden restringirse en base a esos grupos.

Autentificación de usuario

Autentificación: Es el proceso mediante el cual el servidor de la base de datos y el postmaster se aseguran de que el usuario que esta solicitando acceso a la base de datos es en realidad quien dice ser.

La autenticación de clientes es una de las características centrales de postgreSQL, sin ella, se podría sacrificar la conectividad remota o por el contrario se podría permitir que cualquiera se pudiera conectar a nuestra base de datos y recibir o incluso modificar nuestros datos.

Todos los usuarios que quieran utilizar postgres tienen que estar autorizados a hacerlo (ver configuración del archivo pg_hba.conf).

Configuración del archivo pg_hba.conf

A partir de postgresql 7.1.x, los accesos de clientes basados en máquinas (host) se encuentran especificados en el archivo pg_hba.conf. Los permisos y restricciones descritos en este archivo no deberían ser confundidos con los permisos de un usuario de postgresql a los objetos dentro de la base de datos.

El archivo pg_hba.conf esta localizado en el directorio de datos de postgresql y es instalado automáticamente con la ejecución del comando initdb cuando postgresql es instalado.

Mediante la configuración de este archivo se puede restringir los accesos a la base de datos a máquinas especificas, así como permitir acceso a un rango de direcciones IP usando mascara de red. Cada máquina configurada tiene su propio registro de máquinas, el cual es una línea simple en el archivo pg_hba.conf.

Dicho simplemente, el archivo pg_hba.conf le permite determinar quién está autorizado para conectarse, a qué base de datos, desde qué máquina, y para graduar cómo deben probar su autenticidad para obtener el acceso.

Estructura del archivo pg_hba.conf

Este archivo define entradas secuenciales que definen las características que postgreSQL debería usar durante el proceso de autenticación del cliente. Este archivo esta diseñado para ser fácilmente actualizable y ajustarlo así a sus necesidades.

Varias reglas sintácticas se aplican a este archivo:

1. Sólo se puede usar un registro de máquina por línea en el archivo, consecuentemente los registros de máquinas no pueden extenderse a varias líneas.

2. Cada registro de máquina debe contener múltiples campos, los cuales deben estar separados por tabulaciones o por espacios en blancos. El número de campos en un registro de máquina esta directamente relacionado con el tipo de entrada que está siendo definida.

Sintaxis de un registro de máquina en el archivo pg_hba.conf

Tipo_registro base_de_datos IP Mascara_de_red Metodo_autenticacion

Descripción:

Tipo_registro:

Host: una entrada host es usada para especificar máquinas remotas que están autorizadas para conectar al servidor postgresql. El postmaster debe ser ejecutado con la opción –i (TCP/IP) para que una entrada de máquina funcione correctamente.

Local: una entrada local es semánticamente lo mismo que una entrada host. Sin embargo, no necesita especificar una máquina a la que le esté permitido conectarse. La entrada local es usada para

conexiones de clientes que son iniciadas desde la misma máquina en la que esta operando el servidor postgresql.

Hostssl: una entrada hostssl es usada para especificar máquinas (remotas o locales) que están autorizadas para conectar al servidor postgresql usando SSL. El uso de SSL le garantiza que todas las comunicaciones entre el cliente y el servidor estén encriptadas. Para que esto funcione, tanto el cliente como el servidor deben soportar SSL. El proceso postmaster debe ser iniciado con las opciones –l (SSL) y –i (TCP/IP).

base_de_datos:

Es el nombre de la base de datos a la que la máquina especificada esta autorizada a conectar. La palabra clave base_de_datos tiene tres posibles valores:

All: especifica que el cliente puede conectar a cualquier base de datos alojada en el servidor postgresql.
Sameuser: especifica que el cliente sólo puede conectar a una base de datos en la que coincidan los nombres de usuarios autenticados de los clientes.
Name: un nombre determinado puede ser especificado, de forma que el cliente solo puede conectarse a la base de datos especificada por ese nombre.

Ip y Mascara_de_red:

Especifican una dirección IP o rango de direcciones, que están autorizadas a conectar al servidor postgresql. Dicho rango puede ser especificado describiendo una red IP con su mascara de red asociada. De lo contrario, para una única dirección IP, el campo mascara de red debería ser establecido a 255.255.255.255.

Metodo_autenticacion:

Especifica el tipo de autenticación que el servidor debería usar para un usuario que intenta conectarse a postgresql. Este método consta de una lista de opciones:

Trust: permite a cualquier usuario del host (máquina) definido conectarse a una base de datos postgresql sin el uso de una contraseña, como cualquier usuario postgresql. Usted está

autorizando la autenticación basada en máquina con el uso de este método, para cualquier usuario de la máquina especificada. Ésta es una condición insegura si la máquina especificada no es una máquina segura o proporciona acceso a usuarios desconocidos para usted.

Reject: automáticamente deniega el acceso a postgresql para esa máquina o usuario. Esta puede ser una configuración prudente para sitios en la que nadie esta autorizado a conectar a su servidor de base de datos.

Password: especifica que debe existir una contraseña para una conexión de usuario. El uso de este método requerirá que el usuario que conecta proporcione una contraseña. Si usted usa este método, la contraseña será enviada en texto plano.

Crypt: es similar al método password. Cuando se usa crypt, la contraseña no enviada en texto plano, sino a través de un formato simple de encriptación. El uso de este método no es muy seguro, pero es mejor que usar el método password.

Ident: especifica que un mapa de identidad debería ser usado cuando una máquina esta solicitando conexiones desde una IP valida listada en el archivo pg_hba.conf. Este método requiere una opción. La opción requerida puede ser tanto el termino especial sameuser, o un mapa nombrado que es definido dentro del archivo pg_ident.conf.

Control de acceso

PostgreSQL proporciona mecanismos para permitir a los usuarios limitar el acceso que otros usuarios tendrán a sus datos.

1. *Súper usuario de la base de datos:*
   El súper usuario de la base de datos, es aquel usuario con el que se ignora todos los controles de acceso descrito anteriormente.

2. *Privilegios de acceso:*
   El uso de los privilegios de acceso para limitar la lectura, escritura.

3. *Borrado de clases y modificación de estructuras:*
   Los comandos que borran y modifican la escritura de una clase, como alter, drop table y drop index, sólo funcionan con el propietario de la clase. Estas operaciones no están permitidas nunca en los catálogos del sistema.

ANEXOS

Prácticas de PostgreSQL

A continuación presentamos una serie de prácticas que tienen por objetivos:

1. Familiarizar al alumno con el entorno de trabajo y los conceptos fundamentales de implementación de base de datos PostgreSQL, así como su diseño.
2. Ser capaz de formular consultas simples en SQL mediante las sentencias más comunes.

Práctica 1:

El objetivo es lograr que el estudiantes sepa cómo crear un sistema de ficheros mediante la opción initdb y sus variantes para poder iniciar un servidor de base de datos en el mismo usando el comando postmaster con sus opciones.

1. ¿Qué es un sistema de ficheros?

2. Crear un sistema de base de datos llamado prueba.
   Initdb prueba ó
   Initdb –D prueba

3. Crear un sistema de base de datos llamado prueba1 que impida que se creen todos los ficheros en caso que ocurra un error y remueva todos aquellos ficheros que se hayan creado.
   Initdb prueba1 –n

4. Crear un sistema de base de datos llamado prueba2 que imprima la salida de depuración del backend.
   Initdb prueba2 –d

5. ¿Qué es el postmaster?
   Es el proceso que administra la comunicación entre el proceso cliente y el servidor, aunque el postmaster no interactúa directamente con el usuario crea un proceso en segundo plano llamado frontend que es el que interactúa directamente con el usuario.

6. ¿Cómo se detiene el servidor de base de datos (postmaster)?
   Ps aux | grep postmaster
   Kill –s SIGTERM No_proceso_de_postmaster

7. Iniciar el servidor de base de datos para el sistema de ficheros llamado prueba.
   Postmaster –D prueba

8. Iniciar el servidor de base de datos para el sistema de ficheros llamado prueba1 que permita conexiones remota usando el protocolo TCP/IP y atienda peticiones por el puerto 5430.
   Postmaster –D prueba1 –i –p 5430

9. Iniciar el servidor de base de datos para el sistema de ficheros llamado prueba1 que permita el intercambio entre cliente / servidor de manera encriptada.
   Postmaster –D prueba1 –i –l

10. ¿Qué sucede si se intenta iniciar un proceso postmaster con las opciones por defecto cuando ya existe otro proceso postmaster ejecutándose con las mismas opciones? Argumente.

Práctica 2:

Con esta practica se pretende lograr que el estudiante sea capaz de crear y borrar bases de datos en el sistema de fichero creado en la práctica 1, así como también crear, modificar y borrar tablas.

1. ¿Qué comandos se pueden utilizar en PostgreSQL para crear base de datos y diga cuándo utilizar uno u otro?
   - 1.1 *createdb:* permite crear una base de datos desde el prompt de linux el cual puede ser usada para ser pasada como parámetro cuando se desea utilizar el psql.
   - *1.2* *Create database:* también permite crear base de datos pero desde el psql.

2. Crear una base de datos llamada registros para un servidor que escuche peticiones por el puerto por defecto.
   Createdb registros

3. Crear una base de datos llamada registros1 para un servidor que escucha peticiones por el puesto 5430.
   Createdb –p 5430 registros1

4. ¿Qué comandos se pueden utilizar en PostgreSQL para borrar base de datos y diga cuándo utilizar uno u otro?
   4.1 dropdb: permite borrar base de datos desde el prompt de linux.
   4.2 drop database: permite borrar base de datos desde el psql.

5. Borrar la base de datos que se llama registros1 pidiendo la confirmación antes de borrar la base de datos.
   Dropdb –p 5430 –i registros1

Práctica 3:

SQL es un lenguaje de programación de base de datos cuyo origen esta estrechamente relacionado con la invención de las base de datos relacionales por E. F. Codd a principios de los 70.

En esta práctica introduciremos algunas sentencias SQL, que modificaran y consultaran la base de datos a la que estamos conectados y que nos permitirán comprobar el funcionamiento de algunos de los conceptos vistos en nuestra guía.

1. Crear una tabla llamada clases que tenga como atributos cod_asig, nombre_asig y carrera de tipo texto de una longitud definida por usted.
   Create table clases (cod_asig varchar(10),nombre_asig varchar(30),carrera varchar(25));

2. Cambiar el nombre del atributo llamada nombre_asig de la tabla creada anteriormente por el nombre asignaturas.
   Alter table asignaturas Rename nombre_asig To asignaturas

3. Listar la estructura de la tabla.
   \d clases

4. Agregar el atributo anio_acad a la tabla clases el cual será de tipo numérico y contendrá el año académico en que dicha asignatura es impartida y que contenga valor por defecto 1.
   Alter table clases Add anio_acad int default 1;

5. Cambie el nombre de la tabla clases por el nombre asignaturas y luego visualice los resultados.
   Alter table clases rename to asignaturas.

6. Inserte cinco registros a la tabla asignaturas (una por cada año académico).
   Insert into asignaturas values('001','Base de Datos I','Computacion',3);
   Insert into asignaturas values('002','Introduccion a la Informatica','Computacion',1);
   Insert into asignaturas values('003','Sistema Operativo','Computacion',4);
   Insert into asignaturas values('004','Redes I','Computacion',5);
   Insert into asignaturas values('005','Programacion I','Computacion',2);

7. Visualice los resultados
   Select * from asignaturas;

8. Cambiar el código de asignatura para Redes I. Poner como nuevo código 010 y visualice los resultados.
   Update asignaturas set cod_asig='010' where asignaturas.asignaturas nombre='Redes I';

9. Eliminar todos los registros de la tabla asignaturas.
   Truncate asignaturas; ó
   Delete from asignaturas;

10. Visualice los registros de la tabla asignaturas.
    Select * from asignaturas;

11. Borre la tabla asignaturas.
    Drop table asignaturas;

Práctica 4:

En esta práctica trataremos de que el estudiante aprenda a consultar las tablas obteniendo los registros con los atributos deseados. Utilizaremos la cláusula SELECT, FROM y WHERE.

La cláusula select lista los atributos que se desean mostrar en los resultados. Utilice * si desea incluir todos los atributos de la tabla mencionada en la cláusula from.

La cláusula from indica la tabla o las tablas sobre las que se van a realizar las consultas.

Y la cláusula where indica las condiciones que deben cumplir los registros que serán mostrados en el resultado de la consulta.

Antes de iniciar con esta práctica el estudiante deberá crear la tabla estudiantes con atributos:

- nombre texto
-apellidos texto
-asignatura texto
-becado true/false
-ciudad texto

Luego deberá insertar 10 registros, de los cuales 6 deberán ser becados, 3 de los registros debe tener como ciudad león, 4 de Managua y 3 de Masaya. Como asignatura 3 registros contendrán Redes, 5 de Programación y 2 de Informática.

1. Visualice todos los registros de la tabla estudiantes.
   Select * from Estudiantes;

2. Seleccionar todos aquellos estudiantes que sean becados.
   Select * from Estudiantes where becado=true;

3. Seleccionar el nombre y el apellido de los estudiantes que sean de la ciudad de León.
   Select nombre, apellidos from estudiantes where ciudad='leon';

4. Visualizar todos los estudiantes que lleven la asignatura de Redes.
   Select * from estudiantes where asignatura='Redes';

5. Seleccionar todos los estudiantes de sean de managua y que no sean becados.
   Select * from estudiantes where ciudad='managua' and becado=false;

6. Seleccione a los estudiantes que sean de la ciudad de Managua o Masaya
   Select * from estudiantes where ciudad='Managua' or ciudad='Masaya';

7. Seleccionar el nombre, apellidos, ciudad y becado de los estudiantes que llevan la asignatura de Programación.
   Select nombre,apellidos,ciudad,becado from estudiantes where asignatura='Programacion';

Práctica 5:

Con esta práctica se pretende que el estudiante aprenda a utilizar funciones agregadas del lenguaje SQL y que pueden ser implementada en PostgreSQL, entre las cuales tenemos: Count, Max, Min y operadores como el Between y el In.

Count: devuelve el total de registros seleccionado.
Max: utilizado para devolver el valor más alto de un campo especificado.
Min: utilizado para devolver el valor más bajo de un campo especificado.
Between: utilizado para especificar un intervalo de valores.
In: utilizado para especificar registros de una base de datos.

1. Recordemos un poco. En la práctica 4 ya existe una tabla llama Estudiantes. Agregarle a dicha tabla dos nuevos atributos llamado no_carnet de tipo texto de longitud 10, y edad de tipo entero. Teniendo en cuenta que el número de carnet tiene que empezar con 00-00001-0 y terminar con 00-00010-0 y la edad tiene que ir entre 18 y 30,
   Alter table Estudiante add no_carnet varchar(10);
   Alter table Estudiante add edad int;

2. Seleccionar de la tabla Estudiantes las personas cuyo número de carnet este entre el 00-00002-0 y el 00-00007-0.
   Select * from Estudiante where no_carnet between '00-00002-0' and '00-00007-0';

3. Seleccionar a los estudiantes cuyo número de carnet este comprendido entre 00-00001-0 y 00-00004-0 y del 00-00008-0 al 00-00010-0.
   Select * from Estudiantes where no_carnet not between '00-00005-0' and '00-00008-0';

4. Seleccionar el nombre, apellido y número de carnet de los estudiantes vivan ya sea en León, Managua y Masaya.
   Select nombre,apellido,no_carnet from Estudiantes where ciudad in('leon','managua','masaya');

5. Calcular el total de los estudiantes becados.
   Select count(becado) as total from Estudiantes where becado=true;

6. Calcular el total de los estudiantes que llevan la asignatura de programación.
   Select count(asignatura) as total from Estudiantes where asignatura='programacion';

7. Calcular el total de estudiantes cuya edad este comprendida entre 20 y 27.
   Select count(edad) as total from Estudiantes where edad between 20 and 27;

8. Escriba la consulta que dé como resultado la edad mínima de los estudiantes.
   Select min(edad) as menor from Estudiantes;

9. Escriba la consulta que dé como resultado la edad máxima de los estudiantes.
   Select max(edad) as mayor from Estudiantes;

10. Obtener el número de carnet, nombre, apellido de los estudiantes que tengan edad entre 20 y 25.
    Select no_carnet,nombre,apellido from Estudiantes where edad between 20 and 25;

C.A.R.P.I.S

C.A.R.P.I.S: significa **C**ontrol **A**utomatizado de los **R**egistros **P**ara **I**ngeniería en **S**istemas y es una pequeña aplicación que nosotros hicimos con el propósito de hacer una breve demostración en nuestro trabajo monográfico de las ventajas que significa trabajar con ADO.

La versión de Visual Basic que utilizamos es la 6.0 la cual trae incluido esta tecnología (el MSADODC.OCX). Por eso consideramos importante poner un poco del código que se utilizó en el desarrollo de nuestra aplicación.

ADO, siglas que en ingles significan Microsoft ActiveX Data Objects y utiliza la interfaz OLE DB para tener acceso a un amplio conjunto de orígenes de datos.

Esperamos que los estudiantes se sientan rápidamente cómodos con la programación con ADO y para efectos de prácticas se presentan algunas interfaces con su código original y otras en las cuales el estudiante tendrá que completar el código.

Formulario para registrar asignaturas

Este es el formulario en el que se lleva a cabo el registro de las asignaturas que se imparten en la carrera de Ingeniería en sistemas, indicando el docente que impartirá dicha asignatura así como también el semestre y el año académico.


```
Private Sub Form_Activate()
'con el siguiente código validamos los controles
'en caso que existan o no registros en la base de datos.

'si no hay registros
  If Adodc1.Recordset.RecordCount = 0 Then
      invalidareditores
      invalidarbotones
      cmdnuevo.Enabled = True
'si hay registros
   Else
      invalidareditores
      validarbotones
      cmdgrabar.Enabled = False
      cmdcancelar.Enabled = False
   End If
'fin del procedimiento
End Sub
```

```
'******** CODIGO PARA LOS BOTONES DE DESPLAZAMIENTO ******

                  ******************  BOTON  INICIO  *****************

Private Sub cmdinicio_Click()
'ir al primer registro
    Adodc1.Recordset.MoveFirst
'realizar las validaciones correspondientes
    cmdinicio.Enabled = False
    cmdanterior.Enabled = False
    cmdsiguiente.Enabled = True
    cmdultimo.Enabled = True
End Sub

                  '*********** BOTON ANTERIOR ***********

Private Sub cmdanterior_Click()
'moverse al registro anterior
   Adodc1.Recordset.MovePrevious
   'en caso que se esté en el primer registro
    If Adodc1.Recordset.BOF = True Then
       'indicar que ya se está en el primer registro y después moverse al
       'primer registro
       MsgBox "Primer Registro.", vbInformation, "Atención"
       Adodc1.Recordset.MoveFirst
       cmdinicio.Enabled = False
       cmdanterior.Enabled = False
       cmdsiguiente.Enabled = True
       cmdultimo.Enabled = True
    Else
'en caso de no estar en el primer registro habilitar todos los botones de
'desplazamiento
       cmdsiguiente.Enabled = True
       cmdultimo.Enabled = True
       cmdinicio.Enabled = True
       cmdanterior.Enabled = True
    End If
'fin del procedimiento
End Sub
```

```
'********** CODIGO DEL BOTON SIGUIENTE **********

Private Sub cmdsiguiente_Click()
'ir al siguiente registro
   Adodc1.Recordset.MoveNext
'en caso que se esté en el ultimo registro mandar un mensaje indicándolo
   If Adodc1.Recordset.EOF = True Then
      MsgBox "Último Registro.", vbInformation, "Atención"
       'ir al ultimo registro
      Adodc1.Recordset.MoveLast
      cmdinicio.Enabled = True
      cmdanterior.Enabled = True
      cmdsiguiente.Enabled = False
      cmdultimo.Enabled = flase
   Else
        'validar botones de desplazamiento
      cmdsiguiente.Enabled = True
      cmdultimo.Enabled = True
      cmdinicio.Enabled = True
      cmdanterior.Enabled = True
   End If
'fin del procedimiento
End Sub

********** BOTON ULTIMO **********

Private Sub cmdultimo_Click()
'ir al ultimo registro
   Adodc1.Recordset.MoveLast
'realizar las validaciones correspondientes
   cmdinicio.Enabled = True
   cmdanterior.Enabled = True
   cmdsiguiente.Enabled = False
   cmdultimo.Enabled = False
End Sub

********** BOTON NUEVO **********

Private Sub cmdnuevo_Click()
'el método AddNew me permite agregar un registro
```

```
    Adodc1.Recordset.AddNew
'realizar las validaciones correspondientes
    validareditores
    invalidarbotones
    cmdgrabar.Enabled = True
    cmdcancelar.Enabled = True

    'actualizar la tabla historial
    With formmenu_secretaria.Adodc2
        .Recordset.AddNew

          'agregar la fecha en que se agrego el registro
        .Recordset.Fields("fecha") = Date
          'agregar la hora en que se agrego el registro
        .Recordset.Fields("hora") = Time
        .Recordset.Fields("usuario") = nombre_usuario
        .Recordset.Fields("tipo_usuario") = tipo_usuario
        .Recordset.Fields("operacion") = "agrego registro"
        .Recordset.Fields("tabla_acc") = "asignaturas"
        .Recordset.Update
    End With
    'fin del proceso
End Sub

                ********************  BOTON  EDITAR  ******************

Private Sub cmdeditar_Click()
    validareditores
    invalidarbotones
    cmdgrabar.Enabled = True
    cmdcancelar.Enabled = True
    'actualizar la tabla historial
    With formmenu_secretaria.Adodc2
        .Recordset.AddNew
        .Recordset.Fields("fecha") = Date
        .Recordset.Fields("hora") = Time
        .Recordset.Fields("usuario") = nombre_usuario
        .Recordset.Fields("tipo_usuario") = tipo_usuario
        .Recordset.Fields("operacion") = "editar registro"
        .Recordset.Fields("tabla_acc") = "asignaturas"
```

```
        .Recordset.Fields("elmen_acc") = nombre.Text
        .Recordset.Update
    End With
'fin del proceso
End Sub

********************  BOTON  GRABAR  *****************

Private Sub cmdgrabar_Click()
'evitar que dejen el texbox nombre vacío
    If Len(nombre.Text) = 0 Then
        MsgBox "Debe especificar un nombre de asignatura.", vbInformation,
        "Atención"
'establecer el enfoque en el texbox nombre
        nombre.SetFocus
        Exit Sub
    End If

   'obligar  a que se especifique un docente
    If Len(combo_docente.Text) = 0 Then
        MsgBox "Debe especificar un docente.", vbInformation, "Atención"
        combo_docente.SetFocus
        Exit Sub
    End If
    'grabar el Nuevo registro
    Adodc1.Recordset.Update
    invalidareditores
    validarbotones
    cmdgrabar.Enabled = False
    cmdcancelar.Enabled = False

    'actualizar la tabla historial
    With formmenu_secretaria.Adodc2
        .Recordset.AddNew
        .Recordset.Fields("fecha") = Date
        .Recordset.Fields("hora") = Time
        .Recordset.Fields("usuario") = nombre_usuario
        .Recordset.Fields("tipo_usuario") = tipo_usuario
        .Recordset.Fields("operacion") = "guardar registro"
        .Recordset.Fields("tabla_acc") = "asignaturas"
```

```
        .Recordset.Fields("elmen_acc") = nombre.Text
        .Recordset.Update
    End With
End Sub

            ********************  BOTON CANCELAR  *****************

Private Sub cmdcancelar_Click()
'evitar que se guarden los cambios
    Adodc1.Recordset.CancelUpdate
'si no hay registros
    If Adodc1.Recordset.RecordCount = 0 Then
        invalidareditores
        invalidarbotones
        cmdnuevo.Enabled = True
    Else
        invalidareditores
        validarbotones
        cmdgrabar.Enabled = False
        cmdcancelar.Enabled = False
    End If


'actualizar nuestro recodset
    Adodc1.Refresh

    'actualizar la tabla historial
    With formmenu_secretaria.Adodc2
        .Recordset.AddNew
        .Recordset.Fields("fecha") = Date
        .Recordset.Fields("hora") = Time
        .Recordset.Fields("usuario") = nombre_usuario
        .Recordset.Fields("tipo_usuario") = tipo_usuario
        .Recordset.Fields("operacion") = "cacelar operación"
        .Recordset.Fields("tabla_acc") = "asignaturas"
        .Recordset.Update

    End With
'fin del proceso
End Sub
```

********** BOTON BORRAR **********

```
Private Sub cmdborrar_Click()
  Dim res As String
'consultar si se desea borrar el registro
   res = MsgBox("Desea Borrar el registro actual?", vbYesNo + vbCritical,
    "Confirmar")
   If res = vbNo Then
      Exit Sub
   Else
      'actualizar la tabla historial
      'antes de borrar el registro
      With formmenu_secretaria.Adodc2
         .Recordset.AddNew
         .Recordset.Fields("fecha") = Date
         .Recordset.Fields("hora") = Time
         .Recordset.Fields("usuario") = nombre_usuario
         .Recordset.Fields("tipo_usuario") = tipo_usuario
         .Recordset.Fields("operacion") = "borró el registro " & nombre.Text
         .Recordset.Fields("tabla_acc") = "asignaturas"
         .Recordset.Update
      End With
      'borrar el registro
      Adodc1.Recordset.Delete
      'ir al registro anterior
       Adodc1.Recordset.MovePrevious
   'en caso que ya se este en le primer registro nos devolvemos al 'primer
   'registro
      If Adodc1.Recordset.BOF = True Then
         Adodc1.Recordset.MoveFirst
       'si no hay registros realizar las validaciones correspondientes
         If Adodc1.Recordset.RecordCount = 0 Then
            invalidareditores
            invalidarbotones
            cmdnuevo.Enabled = True
         End If
      End If
   End If
End Sub
```

********** BOTON BUSCAR **********

```
Private Sub cmdbuscar_Click()
  Dim ap As String, aux As String
'pedir el parámetro de búsqueda
   ap = InputBox("Escriba el nombre de la asignatura a buscar:", "Atención")
'si no se escribió nada cancelar el proceso de búsqueda
   If Len(ap) = 0 Then
      Exit Sub
   End If
   Adodc1.Recordset.MoveFirst
   aux = "nombre_asig like '" & ap & "*'"
'buscar el registro
   Adodc1.Recordset.Find aux
'en caso de que no se encuentre el registro
   If Adodc1.Recordset.EOF Then
      MsgBox "Ese registro no existe o verifique que" + _
      vbCrLf + "sus parámetros esten escritos correctamente.", vbInformation,
      "Atención"
      'ir al primer registro
      Adodc1.Recordset.MoveFirst
   End If

   'actualizar la tabla historial
   With formmenu_secretaria.Adodc2
      .Recordset.AddNew
      .Recordset.Fields("fecha") = Date
      .Recordset.Fields("hora") = Time
      .Recordset.Fields("usuario") = nombre_usuario
      .Recordset.Fields("tipo_usuario") = tipo_usuario
      .Recordset.Fields("operacion") = "buscó registro"
      .Recordset.Fields("tabla_acc") = "asignaturas"
      .Recordset.Update
   End With
End Sub
```

Formulario menú para la secretaria:

La figura siguiente muestra algunas de las opciones que puede hacer el usuario secretaria. Entre ellas tenemos la generación de reportes tales como el listado de los estudiantes de cada año de la carrera y el listado de asignaturas.

A continuación se mostrara el código necesario para cargar el reporte que nos permitirá visualizar el listado de los de II año de la carrera.


```
Private Sub mnII_sistemas_Click()
   'primero llamamos al comando secundario de nuestro dataenvironment
   'que contiene la consulta que devuelva el resultado deseado.
   DataEnvironment1.listado_II
   'invocamos a nuestro reporte
   lista_II.Show 1
End Sub
```

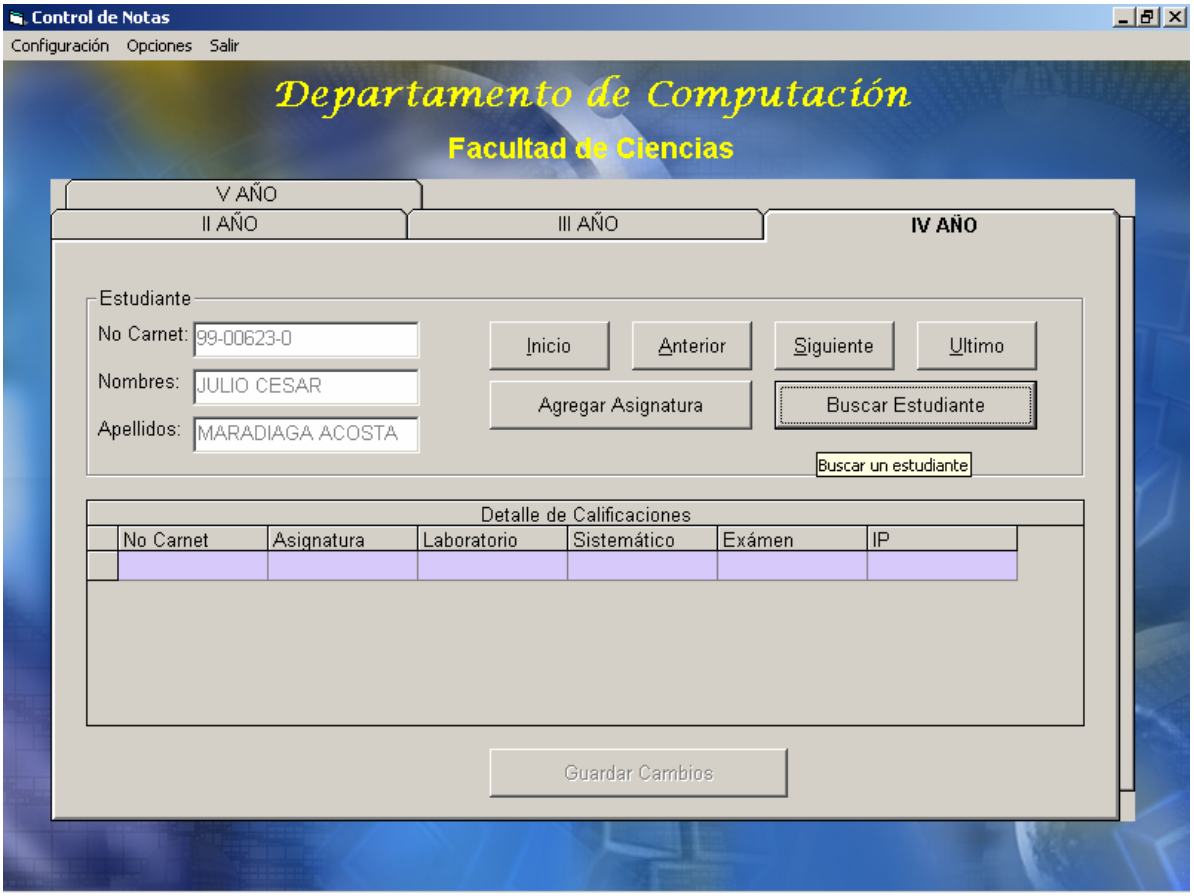
La consulta SQL para el botón secundario es el siguiente:

```
select estudiantes.no_carnet, estudiantes.apellido,estudiantes.nombre from
estudiantes where annodeestudio=2 order by estudiantes.apellido
```

El código correspondiente para la generación del listado de III,IV y V es similar al mostrado anteriormente.

Formulario para el control de notas

En la figura siguiente se muestra el formulario para el control de notas de los estudiantes de la carrera. Como podrá observar cada pestaña corresponde con un año académico y en cada una están disponibles las operaciones que se pueden realizar, como son los botones de desplazamientos, que no profundizaremos sobre ello porque consideramos que el estudiante a esta altura ya puede realizar estas operaciones y los botones de búsqueda, agregar asignaturas al detalle de calificaciones y el de guardar cambios.



A continuación presentamos el código correspondiente a la pestaña de cuarto año plan 98.

```
'evento para la carga inicial del formulario
Private Sub Form_Load()
'con la siguiente consulta se actualiza el recortset del detalle de
'calificaciones
Adodc6.RecordSource = "SELECT notas.no_carnet, notas.nombre,
notas.NLP, notas.NSP, notas.NEP, notas.IP, notas.NLS, notas.NSS,
notas.NES, notas.IIP, notas.NDE, notas.NE, notas.NF, notas.sem" _
& " From notas" _
& " WHERE notas.sem like '" & semestre & "'" & " and notas.no_carnet like '"
& CStr(tnocarnet4.Text) & "'"
Adodc6.Refresh
'inhabilitar el botón para guardar cambio en el detalle
cmdguardarcambios4.Enabled = False
'aquí el estudiantes deberá desarrollar el código correspondientes para las
'otras pestañas


End Sub

'código correspondiente al botón agregar asignaturas
Private Sub cmdagregarasignaturas4_Click()
'c es para capturar la respuesta del usario
Dim c As Integer
   'si el botón esta a true es porque se han hecho
   'modificaciones sin haber guardado los cambios aún
   If Me.cmdguardarcambios4.Enabled = True Then
      c = MsgBox("Desea guardar los cambios?", vbQuestion + vbYesNo,
"Confirmar")
      If c = vbYes Then
         cmdguardarcambios4_Click
      Else
'cancelar la operación
         Adodc6.Recordset.CancelUpdate
         cmdguardarcambios4.Enabled = False
      End If
   End If
```

```
    asignatura = ""
    anno_acad = 4
'filtrar las lista de asignaturas
    formlista.Adodc1.RecordSource = "select * from asignaturas where anno="
& 4 & "and asignaturas.semestre like '" & semestre & "'"
    formlista.Adodc1.Refresh
'en caso de no haber asignaturas definidas para este año académico
    If formlista.Adodc1.Recordset.RecordCount = 0 Then
        MsgBox "No hay asignaturas definidas para este semestre en este
año.", vbInformation, "Atención"
        Exit Sub
    End If
'cargar el formulario con las asignaturas
    formlista.Show 1
End Sub

'código correspondiente al botón buscar estudiantes
Private Sub cmdbuscarestudiantes4_Click()
    Dim ap As String, criterio As String
    'c es para capturar la respuesta del usario
Dim c As Integer
    'si el botón esta a true es porque se han hecho modificaciones
    'sin haber guardado los cambios aún
    If Me.cmdguardarcambios4.Enabled = True Then
        c = MsgBox("Desea guardar los cambios?", vbQuestion + vbYesNo,
"Confirmar")
        If c = vbYes Then
            cmdguardarcambios4_Click
        Else
            cmdguardarcambios4.Enabled = False
        End If
    End If

    ap = InputBox("Escriba los apellidos del estudiante.", "Confirmar")

'en caso de no escribir nada abortar la búsqueda
    If Len(ap) = 0 Then
        Exit Sub
    Else
```

```
        criterio = "apellido like '" & ap & "'"
        Adodc5.Recordset.Find criterio
        'si eof es true, es porque el registro no fue encontrado
        If Adodc5.Recordset.EOF = True Then
            MsgBox "Ese registro no existe", vbInformation, "Atención"
            Adodc5.Recordset.MoveFirst
        End If
    'actualizar el grid
    Adodc6.RecordSource = "SELECT notas.no_carnet, notas.nombre,
notas.NLP, notas.NSP, notas.NEP, notas.IP, notas.NLS, notas.NSS,
notas.NES, notas.IIP, notas.NDE, notas.NE, notas.NF, notas.sem" _
    & " From notas" _
    & " WHERE notas.sem like '" & semestre & "'" & " and notas.no_carnet like
'" & CStr(tnocarnet4.Text) & "'"
    Adodc6.Refresh
    End If
End Sub

'código correspondiente al botón guardar cambio
Private Sub cmdguardarcambios4_Click()
Dim r As Integer

'lo que se consigue con este código es actualizar el campo
'nota final de IP y IIP con la suma de sus sistemáticos
'prueba y examen respectivamente
    If parcial = "I" Then
        For r = 2 To 4
            If DataGrid1(2).Columns(r).Text = "" Then
                DataGrid1(2).Columns(r).Text = 0
            End If
        Next r
        DataGrid1(2).Columns(5).Text = CInt(DataGrid1(2).Columns(4)) +
CInt(DataGrid1(2).Columns(3)) + CInt(DataGrid1(2).Columns(2))
    ElseIf parcial = "II" Then
        For r = 6 To 8
            If DataGrid1(2).Columns(r).Text = "" Then
                DataGrid1(2).Columns(r).Text = 0
            End If
        Next r
```

```
        DataGrid1(2).Columns(9).Text = CInt(DataGrid1(2).Columns(6)) +
CInt(DataGrid1(2).Columns(7)) + CInt(DataGrid1(2).Columns(8))

         'obtener el IIP
        DataGrid1(2).Columns(9).Text = CInt(DataGrid1(2).Columns(6)) +
CInt(DataGrid1(2).Columns(7)) + CInt(DataGrid1(2).Columns(8))

        'esto es para encontrar la nota de entrada
            If DataGrid1 (2).Columns (6).Text <> "" And
DataGrid1(2).Columns(7).Text <> "" And DataGrid1(2).Columns(8).Text <> ""
Then
                DataGrid1 (2).Columns (10).Text =
((CDbl(DataGrid1(2).Columns(5).Text) +
CDbl(DataGrid1(2).Columns(9).Text)) / 2) * 0.6
            End If

            'esto es para encontrar la nota final
            If DataGrid1 (2).Columns (11).Text <> "" Then
                DataGrid1 (2).Columns (12).Text =
CDbl(DataGrid1(2).Columns(10).Text) +
CDbl(DataGrid1(2).Columns(11).Text)
            End If
    End If

    Adodc6.Recordset.Update
    Adodc6.Recordset.Requery
    'actualizar el grid
    Adodc6.RecordSource = "SELECT notas.no_carnet, notas.nombre,
notas.NLP, notas.NSP, notas.NEP, notas.IP, notas.NLS, notas.NSS,
notas.NES, notas.IIP, notas.NDE, notas.NE, notas.NF, notas.sem" _
    & " From notas" _
    & " WHERE notas.sem like '" & semestre & "'" & " and notas.no_carnet like
'" & CStr(tnocarnet4.Text) & "'"
    Adodc6.Refresh
'inhabilitar el boton guardar
    cmdguardarcambios4.Enabled = False
End Sub
```

```
'el siguiente evento se dispara antes de borrar un registro
Private Sub DataGrid1_BeforeDelete(Index As Integer, Cancel As Integer)
    Dim res As Integer
'preguntar si el botón guardar esta habilitado
   If cmdguardarcambios4.Enabled = True Then
        MsgBox "No se puede eliminar el registro, debe guardar algunos
cambios primero, y después intente de nuevo.", vbInformation, "Atención"
        Cancel = True
        Exit Sub
    Else
'preguntar si realmente se desea borrar el registro
        res = MsgBox("Desea borrar el registro actual?", vbQuestion +
vbYesNo, "Confirmar")
        If res = vbNo Then
            Cancel = True
            Exit Sub
        End If
    End If
End Sub
```

Ejercicio propuesto 1:

El estudiante al finalizar con el desarrollo de esta sencilla pero importante práctica será capaz de realizar **operaciones básicas** con los registros de una base de datos.

Se requiere que el estudiante desarrolle el código correspondiente a la interfaz mostrada en la figura siguiente:

El código a desarrollar es para:

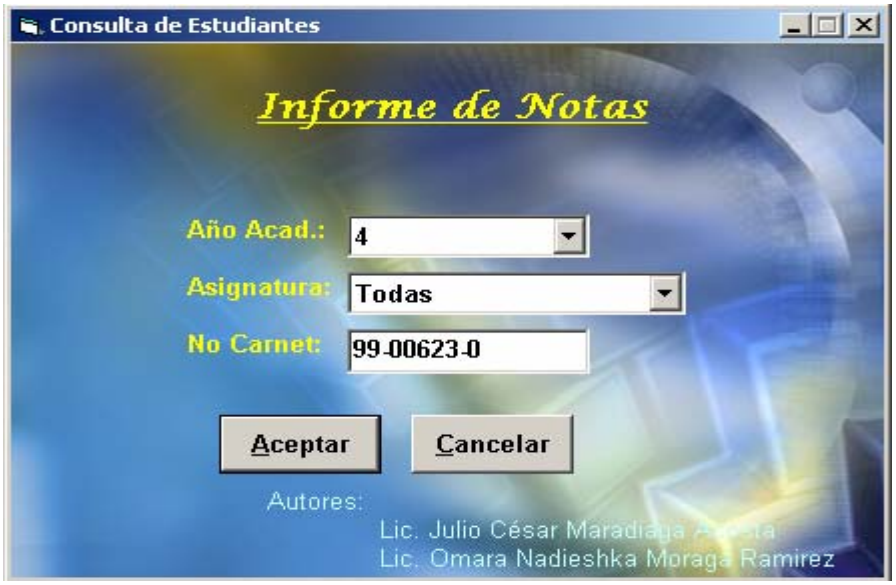
1. Botones de edición como son:
   Nuevo, Editar, Grabar, Cancelar, Borrar y Buscar.
2. Botones de desplazamientos como son:
   Inicio, Anterior, Siguiente, Último
3. Validaciones correspondientes a la carga inicial del formulario así como también validaciones que van a estar en dependencia de las operaciones realizadas por el usuario.

4. Para el textbox del nombre, no se debe permitir dejarlo vacío al dar de alta a un registro.
5. Los tipo de usuarios serán:
    a. Administrador.
    b. Estudiantes.
    c. Docente.
    d. Secretaria.
6. Para el textbox del password, no se debe visualizar los caracteres escritos, en su lugar deber visualizarse un *.

Ejercicio propuesto 2:

Al finalizar con el desarrollo del presente ejercicio el estudiante será capaz de trabajar con la **generación de reportes**.

Se requiere que el estudiante desarrolle el código para el formulario mostrado en la siguiente figura:



El código a desarrollar es para:

1. Botón Aceptar:
    Una vez proporcionado la información necesaria (año académico, asignatura y el número de carnet del estudiante que desea ver su

informe de notas) al dar clic en este botón se generará el informe de notas el cual contendrá:

- ✓ Si es el primer parcial:
  - ➢ No de Carnet del estudiante.
  - ➢ Nombre de la asignatura.
  - ➢ Nota de la evaluación en el laboratorio.
  - ➢ Nota del sistemático.
  - ➢ Nota del examen.
  - ➢ Nota final del Primer Parcial.

- ✓ Si es el segundo parcial:
  - ➢ No de Carnet del estudiante.
  - ➢ Nombre de la asignatura.
  - ➢ Nota de la evaluación en el laboratorio.
  - ➢ Nota del sistemático.
  - ➢ Nota del examen.
  - ➢ Nota final del Segundo Parcial.
  - ➢ Nota de entrada al examen final.
  - ➢ Nota del examen final
  - ➢ Nota final de la asignatura.

2. Boton Canclar:
   Finalizar la aplicación

GLOSARIO

Sitio o site: es la máquina en la que está instalado Postgres. También se puede interpretar site de forma más precisa como cualquier conjunto concreto de programas binarios y bases de datos Postgres instalados.

Superusuario de Postgres: es el usuario llamado Postgres que es dueño de los ficheros de la base de datos Postgres. Como Superusuario de la base de datos, no le es aplicado ningún mecanismo de seguridad y puede acceder a cualquiera de los datos de forma arbitraria.

Administrador de la base de datos o DBA: es la persona responsable de instalar Postgres con mecanismos para hacer cumplir una política de seguridad para un site.

El postmaster: es el proceso que actúa como una puerta de control para las peticiones al sistema Postgres. Las aplicaciones frontend se conectan al Postmaster manteniendo los registros de los errores del mismo.

Frontend: es la aplicación sobre la que trabaja el usuario (ej. Programa psql.)

Backend: es el servidor de acceso a la base de datos.

Bibliografía

1	www.nextec.com.ar/postgres/caracteristicas.htm
2	www.programacion.com/bbdd/tutorial/intropostgresql/3/
3	www.postgresql.cl
4	gborg.postgresql.org/Project/psqlodbc/genpage.php?howto-visualbasic
5	gborg.postgresql.org/project/psqlodbc/projdisplay.php
6	www.forosdelweb.com/s195b67ev9798d17e21c7ea740088578d/msg150185.html