Bases de datos en PostgreSQL

Marc Gibert Ginestà
Oscar Pérez Mora

UOC
Formación de Posgrado

Objetivos

El objetivo principal de esta unidad es conocer el gestor de bases de datos relacionales con soporte para objetos PostgreSQL, y comentar tanto sus características comunes a otros gestores de bases de datos, como las que lo distinguen de sus competidores de código abierto.

Además, se ofrece la oportunidad de aplicar los conocimientos aprendidos en el módulo referido a SQL sobre un sistema gestor de base de datos real y examinar las diferencias entre el estándar y la implementación que hace de él el SGBD.

Por último, se presentan las tareas de administración del SGBD más habituales que un usuario debe llevar a cabo como administrador de Post-greSQL.

1. Características de PostgreSQL

En este apartado comentaremos las características más relevantes de este SGBD con soporte para objetos, tanto sus prestaciones más interesantes o destacadas, como las limitaciones en su diseño o en implementación de los estándares SQL.

También es interesante conocer un poco su historia, ya que tanto por las personas que han participado en su desarrollo como por su relación con otros gestores de bases de datos, nos ayudará a tener una mejor visión de la evolución del mismo.

1.1. Breve historia

La historia de PostgreSQL se inicia en 1986 con un proyecto del profesor Michael Stonebraker y un equipo de desarrolladores de la Universidad Berkeley (California), cuyo nombre original era POSTGRES. En su diseño se incluyeron algunos conceptos avanzados en bases de datos y soporte parcial a la orientación a objetos.

POSTGRES fue comercializado por Illustra, una empresa que posteriormente formó parte de Informix (que comercializaba el conocido SGBD del mismo nombre, recientemente absorbida por IBM y su DB/2). Llegó un momento en que mantener el proyecto absorbía demasiado tiempo a los investigadores y académicos, por lo que en 1993 se liberó la versión 4.5 y oficialmente se dio por terminado el proyecto.

En 1994, Andrew Yu y Jolly Chen incluyeron SQL en Postgres para posteriormente liberar su código en la web con el nombre de Postgres95. El proyecto incluía múltiples cambios al código original que mejoraban su rendimiento y legibilidad.

En 1996 el nombre cambió a PostgreSQL retomando la secuencia original de versiones, por lo que se liberó la versión 6.0. En el año 2004 la última versión estable oficial es la 7.4.6, mientras que la versión 8.0 está ya en fase final de estabilización.

1.2. Prestaciones

PostgreSQL destaca por su amplísima lista de prestaciones que lo hacen capaz de competir con cualquier SGBD comercial:

Los desarrolladores de proyectos basados en software libre tienen muy en cuenta PostgreSQL cuando los requerimientos de un proyecto exigen prestaciones de alto nivel.

- Está desarrollado en C, con herramientas como Yacc y Lex.

- La API de acceso al SGBD se encuentra disponible en C, C++, Java, Perl, PHP, Python y TCL, entre otros.

- Cuenta con un rico conjunto de tipos de datos, permitiendo además su extensión mediante tipos y operadores definidos y programados por el usuario.

- Su administración se basa en usuarios y privilegios.

- Sus opciones de conectividad abarcan TCP/IP, *sockets* Unix y *sockets* NT, además de soportar completamente ODBC.

- Los mensajes de error pueden estar en español y hacer ordenaciones correctas con palabras acentuadas o con la letra 'ñ'.

- Es altamente confiable en cuanto a estabilidad se refiere.

- Puede extenderse con librerías externas para soportar encriptación, búsquedas por similitud fonética (soundex), etc.

- Control de concurrencia multi-versión, lo que mejora sensiblemente las operaciones de bloqueo y transacciones en sistemas multi-usuario.

- Soporte para vistas, claves foráneas, integridad referencial, disparadores, procedimientos almacenados, subconsultas y casi todos los tipos y operadores soportados en SQL92 y SQL99.

- Implementación de algunas extensiones de orientación a objetos. En PostgreSQL es posible definir un nuevo tipo de tabla a partir de otra previamente definida.

1.3. Limitaciones

Las limitaciones de este tipo de gestores de bases de datos suelen identificarse muy fácilmente analizando las prestaciones que tienen previstas para las próximas versiones. Encontramos lo siguiente:

- Puntos de recuperación dentro de transacciones. Actualmente, las transacciones abortan completamente si se encuentra un fallo durante su ejecución. La definición de puntos de recuperación permitirá recuperar mejor transacciones complejas.

- No soporta *tablespaces* para definir dónde almacenar la base de datos, el esquema, los índices, etc.

- El soporte a orientación a objetos es una simple extensión que ofrece prestaciones como la herencia, no un soporte completo.

4. Creación y manipulación de tablas

4.1. Creación de tablas

Una vez conectados a una base de datos, la sentencia SQL **create table** permite crear las tablas que necesitemos:

```
demo=#     create table persona (
demo(#     nombre varchar(30),
demo(#     direccion varchar(30)
demo(# );
CREATE
```

El comando **drop table** permite eliminar tablas:

```
demo=# drop table persona;
```

La tabla recién creada aparece ahora en la lista de tablas de la base de datos en uso:

```
demo=# \dt
List of relations
Name     |Type   |Owner
---------+-------+--------
 persona | table | quiron
(1 row)
```

Podemos consultar su descripción mediante el comando *\d tabla*:

```
demo=# \d persona
Table "persona"
Column     |Type                    | Modifiers
-----------+------------------------+-----------
    nombre | character varying(30)  |
 direccion | character varying(30)  |
```

La tabla está lista para insertar en ella algunos registros.

```
demo=# insert into persona values ( 'Alejandro Magno' , 'Babilonia' );
INSERT 24756 1
demo=# insert into persona values ( 'Federico García Lorca ' , 'Granada 65' );
INSERT 24757 1
```

El número con el que responde el comando *insert* se refiere al OID del registro insertado.


Este aspecto se explicará en detalle más adelante.

Las consultas se realizan con la sentencia SQL **select.** En este caso solicitamos que nos muestre todas las columnas de los registros en la tabla *persona*:

```
demo=# select * from persona;
nombre                 |direccion
-----------------------+------------
 Alejandro Magno       | Babilonia
 Federico García Lorca | Granada 65
(2 rows)
demo=#
```

Las tablas creadas en PostgreSQL incluyen, por defecto, varias columnas ocultas que almacenan información acerca del identificador de transacción en que pueden estar implicadas, la localización física del registro dentro de la tabla (para localizarla muy rápidamente) y, los más importantes, el OID y el TABLEOID. Estas últimas columnas están definidas con un tipo de datos especial llamado identificador de objeto (OID) que se implementa como un entero positivo de 32 bits. Cuando se inserta un nuevo registro en una tabla se le asigna un número consecutivo como OID, y el TABLEOID de la tabla que le corresponde.

En la programación orientada a objetos, el concepto de OID es de vital importancia, ya que se refiere a la identidad propia del objeto, lo que lo diferencia de los demás objetos.

Para observar las columnas ocultas, debemos hacer referencia a ellas específicamente en el comando *select*:

```
demo=# select oid, tableoid, * from persona;
oid    |tableoid  |nombre                 |direccion
-------+----------+-----------------------+------------
 17242 |    17240 | Alejandro Magno       | Babilonia
 17243 |    17240 | Federico García Lorca | Granada 65
(2 rows)
demo=#
```

Estas columnas se implementan para servir de identificadores en la realización de enlaces desde otras tablas.

Ejemplo de la utilización de OID para enlazar dos tablas

Retomamos la tabla *persona* y construimos una nueva tabla para almacenar los teléfonos.

```
demo=# create table telefono (
demo(# tipo char(10),
demo(# numero varchar(16),
demo(# propietario oid
demo(# );
CREATE
```

La tabla *teléfono* incluye la columna *propietario* de tipo OID, que almacenará la referencia a los registros de la tabla *persona*. Agreguemos dos teléfonos a 'Alejandro Magno', para ello utilizamos su OID que es 17242:

```
demo=# insert into telefono values( 'móvil' , '12345678', 17242 );
demo=# insert into telefono values( 'casa' , '987654', 17242 );
```

Las dos tablas están vinculadas por el OID de persona.

```
demo=# select * from telefono;
tipo        |numero       | propietario
------------+-------------+-------------
      móvil |    12345678 |   17242
       casa |      987654 |   17242
(2 rows)
```

La operación que nos permite unir las dos tablas es *join*, que en este caso une *teléfono* y *persona*, utilizando para ello la igualdad de las columnas *telefono.propietario* y *persona.oid*:

```
demo=# select * from telefono join persona on (telefono.propietario = perso-na.oid);
tipo        |numero       | propietario |nombre           | direccion
------------+-------------+-------------+-----------------+-----------
      móvil |    12345678 |       17242 | Alejandro Magno | Babilonia
       casa |      987654 |       17242 | Alejandro Magno | Babilonia
(2rows)
```

Los OID de PostgreSQL presentan algunas deficiencias:

- Todos los OID de una base de datos se generan a partir de una única secuencia centralizada, lo que provoca que en bases de datos con mucha actividad de inserción y eliminación de registros, el contador de 4 bytes se desborde y pueda entregar OID ya entregados. Esto sucede, por supuesto, con bases de datos muy grandes.

- Las tablas enlazadas mediante OID no tienen ninguna ventaja al utilizar operadores de composición en términos de eficiencia respecto a una clave primaria convencional.

- Los OID no mejoran el rendimiento. Son, en realidad, una columna con un número entero como valor.

5. Manipulación de datos

5.1. Consultas

Las consultas a la base de datos se realizan con el comando **select**, que se implementa en PostgreSQL cumpliendo en gran parte con el estándar SQL:

Notación

Omitiremos las referencias comunes a SQL y sólo se mostrarán algunas de las posibilidades de consulta con PostgreSQL. Por lo que respecta a las funciones auxiliares, se han visto algunas en el apartado de tipos de datos y, en todo caso, se recomienda la consulta de la documentación del producto para las operaciones más avanzadas.

```
demo=# select parte, tipo
demo-# from productos
demo-# where psugerido > 30
demo-# order by parte
demo-# limit 5
demo-# offset 3;
parte       | tipo
------------+----------
Monitor     | 1024x876
Monitor     | 1024x876
Procesador  | 2.4 GHz
Procesador  | 1.7 GHz
Procesador  | 3 GHz
(5 rows)
```

Al igual que MySQL, PostgreSQL admite la sentencia **explain** delante de **select** para examinar qué está ocurriendo durante una consulta:

Al igual que en el módulo de MySQL, vemos que no aprovecha los índices (básicamente porque no tenemos ninguno definido).

```
demo=# explain select productos.clave, parte||' '||tipo||' '||especificación as producto,
proveedores.empresa , precio  from productos natural join precios natural join proveedores;
                                    QUERY PLAN
------------------------------------------------------------------------------
Hash Join  (cost=45.00..120.11 rows=1000 width=104)
   Hash Cond: (("outer".empresa)::text = ("inner".empresa)::text)
   -> Hash Join  (cost=22.50..72.61 rows=1000 width=104)
      Hash Cond: ("outer".clave = "inner".clave)
      -> Seq Scan on precios  (cost=0.00..20.00 rows=1000 width=32)
      -> Hash (cost=20.00..20.00 rows=1000 width=76)
         -> Seq Scan on productos  (cost=0.00..20.00 rows=1000 width=76)
   -> Hash (cost=20.00..20.00 rows=1000 width=24)
      -> Seq Scan on proveedores  (cost=0.00..20.00 rows=1000 width=24)
(9 rows)
demo=#
```

Veamos como mejorar el rendimiento de este **select**:

```
demo=# create index empresa_idx on precios (empresa);
CREATE INDEX
demo=# create index clave_idx on precios (clave);
CREATE INDEX
demo=# explain select productos.clave, parte||' '||tipo||' '||especificación as producto,
proveedores.empresa , precio  from productos natural join precios natural join proveedores;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=29.00..56.90 rows=20 width=104)
   Hash Cond: ("outer".clave = "inner".clave)
   ->  Seq Scan on productos  (cost=0.00..20.00 rows=1000 width=76)
   ->  Hash  (cost=28.95..28.95 rows=20 width=32)
      ->  Hash Join  (cost=1.25..28.95 rows=20 width=32)
         Hash Cond: (("outer".empresa)::text = ("inner".empresa)::text)
         ->  Seq Scan on proveedores  (cost=0.00..20.00 rows=1000 width=24)
         ->  Hash  (cost=1.20..1.20 rows=20 width=32)
            ->  Seq Scan on precios  (cost=0.00..1.20 rows=20 width=32)
(9 rows)
demo=#
```

5.2. Actualizaciones e inserciones

PostgreSQL cumple con el estándar SQL en todos los sentidos para las sentencias de actualización, inserción y borrado. No define modificadores ni otros mecanismos para estas sentencias.

En determinadas cláusulas, y para tablas heredadas, es posible limitar el borrado o la actualización a la tabla *padre* (sin que se propague a los registros de las tablas hijas) con la cláusula **only**:

```
demo=# update only persona set nombre='Sr. '||nombre;
UPDATE 2
demo=# select * from persona;
nombre                     | direccion
---------------------------+-------------
 Sr. Alejandro Magno       | Babilonia
 Sr. Federico García Lorca | Granada 65
 Juan                      | Treboles 21
(3 rows)
demo=#
```

7. Administración de PostgreSQL

En las tareas administrativas como la instalación, la gestión de usuarios, las copias de seguridad, restauraciones y el uso de prestaciones internas avanzadas, es donde realmente se aprecian las diferencias entre gestores de bases de datos. PostgreSQL tiene fama de ser más complejo de administrar que sus competidores de código abierto, lo que se debe, sobre todo, a que ofrece más prestaciones (o más complejas).

El contenido de los siguientes apartados contempla las opciones de uso común para la administración de un servidor PostgreSQL. Existen tantas alternativas que no es posible incluirlas todas en este módulo, por lo que sólo se presentarán algunos temas de importancia para el administrador, desde una perspectiva general, que permita obtener una visión global de las posibilidades prácticas de las herramientas administrativas.

Bibliografía

El manual de PostgreSQL es la referencia principal que se debe tener siempre a mano para encontrar posibilidades y resolver dudas. En especial se recomienda leer los siguientes capítulos:

Capítulo III. Server Administration.

Capítulo V. Server Programming.

Capítulo VII. Internals.

De la misma manera, también son muy útiles las listas de correo que se describen en el sitio oficial www.postgresql.org.

7.1. Instalación

PostgreSQL está disponible para la mayoría de distribuciones de GNU/Linux. Su instalación es tan sencilla como ejecutar el instalador de paquetes correspondiente.

En Debian, el siguiente procedimiento instala el servidor y el cliente respectivamente:

```
# apt-get install postgresql
# apt-get install postgresql-client
```

En distribuciones basadas en RPM, los nombres de los paquetes son un poco diferentes:

```
# rpm -Uvh postgresql-server
# rpm -Uvh postgresql
```

Una vez instalado, se escribirá un *script* de inicio que permite lanzar y apagar el servicio PostgreSQL; de este modo, para iniciar el servicio, deberemos ejecutar el siguiente comando:

```
# /etc/init.d/postgresql start
```

Notación

Además del **start** también podremos utilizar los parámetros **restart, stop, reload** que permiten reiniciar, detener y recargar el servidor (releyendo su configuración), respectivamente.

Si se desea realizar una instalación a partir del código fuente, puede obtenerse del sitio oficial www.postgresql.org. A continuación, se describe el proceso de instalación de forma muy simplificada. En la práctica podrán encontrarse algunas diferencias; lo más recomendable es leer cuidadosamente la documentación incluida en los archivos INSTALL y README. Cualquier duda no resuelta por la documentación, puede consultarse en la lista de distribución.

```
# tar xzvf postgresql-7.4.6.tar.gz
# cd postgresql-7.4.6
# ./configure
# make
# make install
```

Con este proceso se instala la versión 7.4.6. El archivo se descomprime utilizando *tar*. Dentro del directorio recién creado se ejecuta *configure*, que realiza una comprobación de las dependencias de la aplicación. Antes de ejecutar *configure,* debemos instalar todos los paquetes que vamos a necesitar.

La compilación se realiza con *make* y, finalmente, los binarios producidos se copian en el sistema en los lugares convenientes con *make install*.

Después de instalados los binarios, se debe crear el usuario *postgres* (responsable de ejecutar el proceso *postmaster*) y el directorio donde se almacenarán los archivos de las bases de datos.

```
# adduser postgres
# cd /usr/local/pgsql
# mkdir data
# chown postgres data
```

Una vez creado el usuario *postgres*, éste debe inicializar la base de datos:

```
# su - postgres
# /usr/local/pgsql/initbd -D /usr/local/pgsql/data
```

initdb

El ejecutable *initdb* realiza el procedimiento necesario para inicializar la base de datos de postgres, en este caso, en el directorio /usr/local/pgsql/data.

El *postmaster* ya está listo para ejecutarse manualmente:

```
# /usr/local/pgsql/postmaster -D /usr/local/pgsql/data
```

Bibliografía

El proceso de compilación tiene múltiples opciones que se explican en la documentación incluida con las fuentes.

7.1.1. Internacionalización

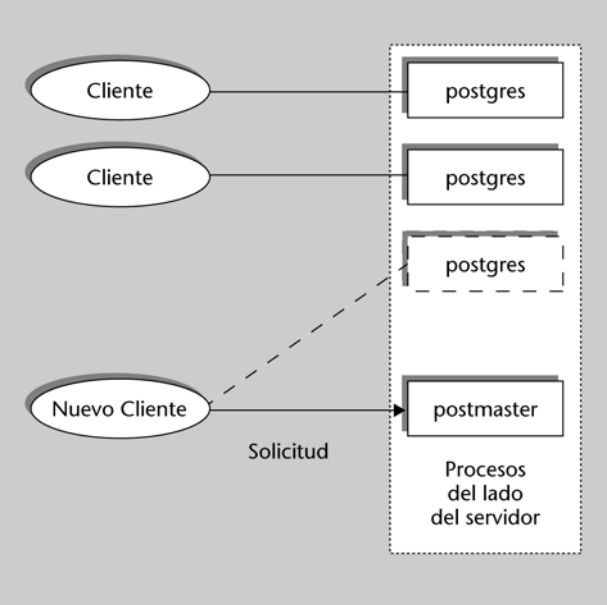
Por omisión, PostgreSQL no está compilado para soportar mensajes en español, por lo que es necesario compilarlo a partir de las fuentes incluyendo las

siguientes opciones de configuración, para que tanto el servidor como el cliente **psql** adopten la configuración establecida por el programa **setlocales** y las variables de entorno respectivas:

```
# configure --enable-nls -enable-locale
```

7.2. Arquitectura de PostgreSQL

El siguiente gráfico muestra de forma esquemática las entidades involucradas en el funcionamiento normal del gestor de bases de datos:



PostgreSQL está basado en una arquitectura cliente-servidor. El programa servidor se llama **postgres** y entre los muchos programas cliente tenemos, por ejemplo, **pgaccess** (un cliente gráfico) y **psql** (un cliente en modo texto).

Un proceso servidor *postgres* puede atender exclusivamente a un solo cliente; es decir, hacen falta tantos procesos servidor *postgres* como clientes haya. El proceso **postmaster** es el encargado de ejecutar un nuevo servidor para cada cliente que solicite una conexión.

Se llama **sitio** al equipo anfitrión (*host*) que almacena un conjunto de bases de datos PostgreSQL. En un *sitio* se ejecuta solamente un proceso *postmaster* y múltiples procesos *postgres*. Los clientes pueden ejecutarse en el mismo sitio o en equipos remotos conectados por TCP/IP.

Es posible restringir el acceso a usuarios o a direcciones IP modificando las opciones del archivo pg_hba.conf, que se encuentra en /etc/postgresql/pg_hba.conf.

Este archivo, junto con /etc/postgresql/postgresql.conf son particularmente importantes, porque algunos de sus parámetros de configuración por defecto

provocan multitud de problemas al conectar inicialmente y porque en ellos se especifican los mecanismos de autenticación que usará PostgreSQL para verificar las credenciales de los usuarios.

Para habilitar la conexión a PostgreSQL desde clientes remotos, debemos verificar el parámetro tcpip_socket = true en el fichero */etc/postgresql/postgresql.conf*.

A continuación, para examinar los métodos de autenticación y las posibilidades de conexión de clientes externos, debemos mirar el fichero */etc/postgresql/pg_hba.conf*, donde se explicita la acción que hay que emprender para cada conexión proveniente de cada *host* externo, o grupo de *hosts*.

7.3. El administrador de postgres

Al terminar la instalación, en el sistema operativo se habrá creado el usuario **postgres**, y en PostgreSQL se habrá creado un usuario con el mismo nombre.

Él es el único usuario existente en la base de datos y será el único que podrá crear nuevas bases de datos y nuevos usuarios.

Normalmente, al usuario *postgres* del sistema operativo no se le permitirá el acceso desde un *shell* ni tendrá contraseña asignada, por lo que deberemos convertirnos en el usuario *root*, para después convertirnos en el usuario *postgres* y realizar tareas en su nombre:

```
yo@localhost:~$ su
Password:
# su - postgres
postgres@localhost:~$
```

El usuario **postgres** puede crear nuevas bases de datos utilizando el comando **createdb**. En este caso, le indicamos que el usuario propietario de la misma será el usuario postgres:

```
postgres@localhost:~$ createdb demo --owner=postgres
create database
```

El usuario **postgres** puede crear nuevos usuarios utilizando el comando **createuser**:

Se ha creado el usuario yo con permisos para crear bases de datos y sin permisos para crear usuarios.

```
postgres@localhost:~$ createuser yo
Shall the new user be allowed to create databases? (y/n) y
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
```

Los siguientes comandos eliminan bases de datos y usuarios, respectivamente:

```
postgres@localhost:~$ dropdb demo
postgres@localhost:~$ dropuser yo
```

Es recomendable que se agreguen los usuarios necesarios para operar la instalación de PostgreSQL, y recurrir, así, lo menos posible al ingreso con *postgres*.

También disponemos de sentencias SQL para la creación de usuarios, grupos y privilegios:

```
demo=# create user marc password 'marc21';
CREATE USER
demo=# alter user marc password 'marc22';
ALTER USER
demo=# drop user marc;
DROP USER
```

Los grupos permiten asignar privilegios a varios usuarios y su gestión es sencilla:

```
create group migrupo;
```

Para añadir o quitar usuarios de un grupo, debemos usar:

```
alter group migrupo add user marc, ... ;
alter group migrupo drop user marc, ... ;
```

7.3.1. Privilegios

Cuando se crea un objeto en PostgreSQL, se le asigna un dueño. Por defecto, será el mismo usuario que lo ha creado. Para cambiar el dueño de una tabla, índice, secuencia, etc., debemos usar el comando *alter table*. El dueño del objeto es el único que puede hacer cambios sobre él, si queremos cambiar este comportamiento, deberemos asignar privilegios a otros usuarios.

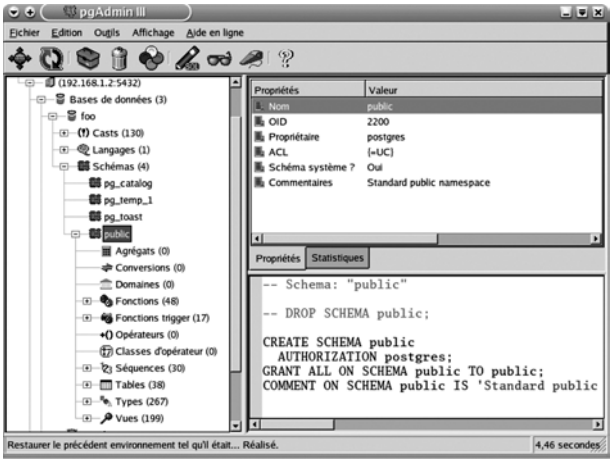
Los privilegios se asignan y eliminan mediante las sentencias `grant` y `revoke`. PostgreSQL define los siguientes tipos de operaciones sobre las que podemos dar privilegios:

select, insert, update, delete, rule, references, trigger, create, temporary, execute, usage, y all privileges.

8. Cliente gráfico: pgAdmin3

El máximo exponente de cliente gráfico de PostgreSQL es el software pgAdmin3 que tiene licencia "Artist License", aprobada por la FSF.

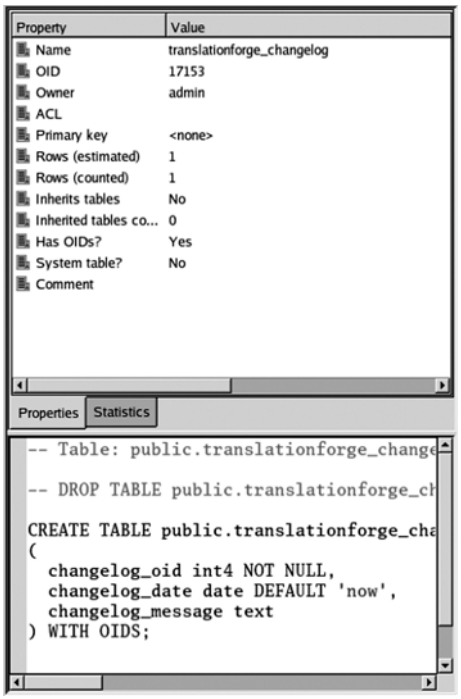
pgAdmin3 está disponible en http://www.pgadmin.org.



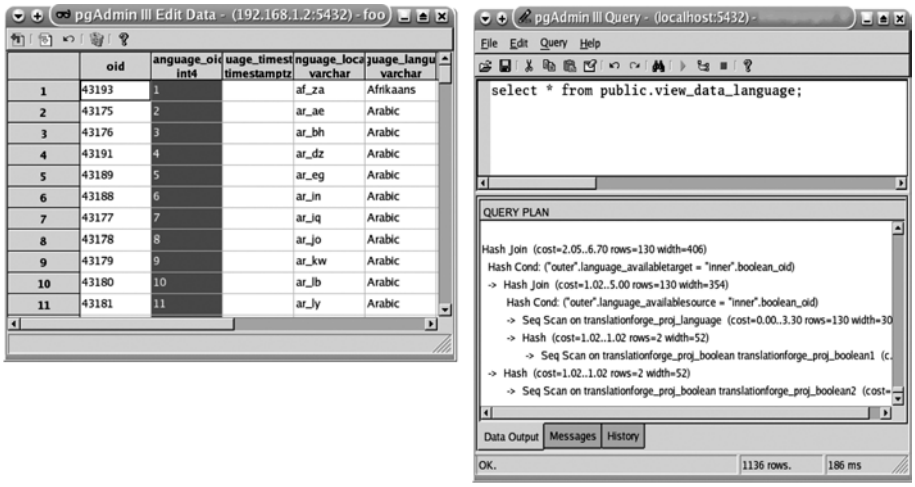
En pgAdmin3 podemos ver y trabajar con casi todos los objetos de la base de datos, examinar sus propiedades y realizar tareas administrativas.

- Agregados
- Casts
- Columnas
- Constraints
- Conversiones
- Bases de datos
- Dominios
- Funciones
- Grupos
- Índices
- Lenguajes (PLpgsql, PLpython, PLperl, etc.)
- Clases de operadores
- Operadores
- Servidores PostgreSQL
- Reglas
- Esquemas
- Secuencias
- Tablas
- Triggers
- Tipos de datos
- Usuarios
- Vistas

Una característica interesante de pgAdmin3 es que, cada vez que realizamos alguna modificación en un objeto, escribe la/s sentencia/s SQL correspondiente/s, lo que hace que, además de una herramienta muy útil, sea a la vez didáctica.



pgAdmin3 también incorpora funcionalidades para realizar consultas, examinar su ejecución (como el comando **explain**) y trabajar con los datos.



Todas estas características hacen de pgAdmin3 la única herramienta gráfica que necesitaremos para trabajar con PostgreSQL, tanto desde el punto de vista del usuario como del administrador. Evidentemente, las acciones que podemos realizar en cada momento vienen condicionadas por los permisos del usuario con el que nos conectemos a la base de datos.

Resumen

PostgreSQL implementa las características necesarias para competir con cualquier otra base de datos comercial, con la ventaja de tener una licencia de libre distribución BSD.

La migración de bases de datos alojadas en productos comerciales a PostgreSQL se facilita gracias a que soporta ampliamente el estándar SQL. PostgreSQL cuenta con una serie de características atractivas como son la herencia de tablas (clases), un rico conjunto de tipos de datos que incluyen arreglos, BLOB, tipos geométricos y de direcciónes de red. PostgreSQL incluye también el procesamiento de transacciones, integridad referencial y procedimientos almacenados. En concreto, hay procedimientos documentados para migrar los procedimientos almacenados desarrollados en lenguajes propietarios de bases de datos comerciales (PL/SQL) a PL/PGSQL.

La API se distribuye para varios lenguajes de programación como C/C++, Perl, PHP, Python, TCL/Tk y ODBC.

Por si esto fuera poco PostgreSQL es extensible. Es posible agregar nuevos tipos de datos y funciones al servidor que se comporten como los ya incorporados. También es posible insertar nuevos lenguajes de programación del lado del servidor para la creación de procedimientos almacenados. Todas estas ventajas hacen que muchos programadores lo elijan para el desarrollo de aplicaciones en todos los niveles.

Entre sus deficiencias principales podemos mencionar los OID. PostgreSQL está aún en evolución, se espera que en futuras versiones se incluyan nuevas características y mejoras al diseño interno del SGBD.

Bibliografía

Documentación de PostgreSQL de la distribución: http://www.postgresql.org/docs/

Silberschatz, A.; Korth, H.; Sudarshan,S. (2002). *Fundamentos de bases de datos* (4.ª ed.). Madrid: McGraw Hill.

Worsley, John C.; Drake, Joshua D. (2002). *Practical PostgreSQL*. O'Reilly.