Puesta en marcha de un servidor de bases de datos utilizando PostgreSQL 7.

José A. Rocamonde Quintela.
jose@universodigital.net

22 de mayo 2001

Distribución:

1. Breve historia, características y funcionamiento.
2. Instalación y configuración.
3. Mantenimiento y seguridad.
4. Interfaces con entornos y lenguajes

Parte 1: Breve historia, características y funcionamiento

- Historia y evolución
- Características fundamentales
- Funcionamiento interno

Breve historia de Postgres

- **Ingres** *(1977 - 85)*
  Universidad de california en Berkeley
- **Postgres** *(1986 - 94)*
  POSTerior a In**GRES**. Michael Stonebraker
- **Postgres95** *(1995)*
  Andrew Yu & Jolly Chen publican el código en Internet
- **PostgreSQL 6** *(1996 - 99)*
  PostgreSQL Global Development Group
- **PostgreSQL 7** *(2000 - ?)*
  Foreign Keys, mayor velocidad y rendimiento

Características fundamentales

- Soporte SQL92
- Transacciones
- Integridad referencial
- Bloqueos de tabla y fila
- Constraints, rules y triggers
- Objetos: clases y herencia
- Multiples tipos de datos predefinidos
- Soporte de tipos y funciones del usuario
- Conectividad TCP/IP, JDBC y ODBC
- Interfaz con diversos lenguajes

Transacciones

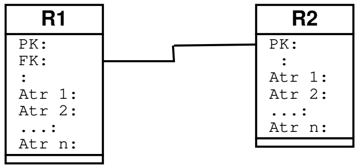
Permiten el paso entre dos estados consistentes manteniendo la integridad de los datos.

Los conjuntos de acciones de una transaccion se encierran entre:

```
BEGIN WORK;                         BEGIN WORK;
  ...                                 ...
  SENTENCIAS SQL                      SENTENCIAS SQL
  ...                                 ...
COMMIT WORK;                        ROLLBACK WORK;
```


Integridad referencial

- Permite mantener la consistencia entre las tuplas de dos relaciones o tablas.
- La especificación de claves foráneas se realiza en la creación de las tablas mediante el constraint `foreign key`.
- Definimos una clave foránea `FK` (*Foreign key*) que para cada tupla en la tabla R_1, el atributo `FK` toma uno de los valores del atributo `PK` (*Primary key*) de la tabla R_2, o bien toma el valor nulo.



Ejemplo de código SQL

Para establecer la integridad referencial entre `libro` y `genero`.

```
create table libro (
  cod_libro integer not null,
  cod_genero integer,
  titulo text,
  autor  text,
 primary key (cod_libro),
 foreign key (cod_genero)  -- clave foranea en libro
      references genero    -- clave primaria en genero
      on delete set null   -- se anula si se borra en genero
      on update cascada    -- se actualiza en cascada );

create table genero (
  cod_genero integer not null,
  descripcion text,
 primary key (cod_genero) );
```

Bloqueos de tablas y de filas

- Permite diferentes niveles de bloqueo `sh()` (compartido) y `ex()` (exclusivo) tanto a nivel de tabla como de fila
- Determinadas acciones como `UPDATE, DELETE, INSERT, ALTER TABLE, DROP TABLE`,o `VACUUM` llevan asociados algunos de los niveles de bloqueo.
- Es posible indicar explicitamente un tipo de bloqueo mediante la sentencia `LOCK`:

```
LOCK [ TABLE ] name
LOCK [ TABLE ] name IN [ ROW | ACCESS ] { SHARE | EXCLUSIVE } MODE
LOCK [ TABLE ] name IN SHARE ROW EXCLUSIVE MODE
```


Constraints, rules y triggers

Tienen la función y el objetivo de mantener la BD integra y consistente previniendo contra la entrada de datos inválidos.

Constraints : “obligadores” o “forzadores”

`PRIMARY KEY` Fuerza la existencia de clave primaria.
`FOREIGN KEY` Fuerza la existencia de clave foránea.
`UNIQUE` Obliga a que los datos de esa columna no estén repetidos.
`NOT NULL` Impide la existencia de nulos.
`CHECK` Restringe la entrada de datos chequeando determinada condición.

```
CREATE TABLE socio (
  nombre      text     CHECK (leght(trim(nombre))>=2),
  apellidos   text     CHECK (leght(trim(nombre))>=2),
  edad        integer  CHECK (edad >= 0),
  genero      char(1)  CHECK (genero IN ('M','F')) )
```


Rules : Reglas de entrada, permiten modificar los datos cuando se accede a una tabla. Modifican los efectos de `SELECT, INSERT, UPDATE,` y `DELETE`.

Reglas `DO`: Ejecutan comandos SQL despues de realizar la consulta.

Reglas `DO INSTEAD`: Reemplazan la consulta del usuario por las acciones descritas en la regla.

Sintaxis:

```
CREATE RULE nombre_regla
AS ON { SELECT | INSERT | UPDATE | DELETE }
TO objeto
[clausula where]
DO [INSTEAD]
[NOTHING | accion ]
```


Triggers : “disparadores”

- Permiten chequear o modificar una tupla antes de que esta sea añadida a la base de datos.
- Llaman a funciones speciales que se ejecutan en el servidor (*server-side*) cada vez que una tupla es añadida o modificada.
- Las funciones (*server-side*) pueden ser escritas en cualquier lenguaje que soporte Postgres excepto SQL: plpgsql, C ...

  Sintaxis:

```
CREATE TRIGGER nombre_trigger
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE }
ON nombre_tabla
FOR EACH { ROW | STATEMENT }
EXECUTE PROCEDURE nombre_funcion(argumentos)
```

Objetos: clases y herencia

- Debido a que la sintaxis de SQL hace referencia a tablas, se emplean los términos tabla y clase indistintamente.
- Las tuplas de una tabla son instancias de una clase, y cada instancia (tupla) tiene un *identificador de objeto* (`OID`) que es único.
- La herencia de clases se define utilizando la claúsula `INHERITS` cuando creamos una nueva clase:

```
CREATE TABLE socio (
   id      integer,
   nombre  text    );
```

```
CREATE TABLE socio_biblioteca (
     fecha_alta     date,
     num_max_libros integer
) INHERITS (socio);
```


Multiples tipos de datos predefinidos

- **Numéricos**: integer, float, int2, int4, int8, float4, float8, y serial.
- **Monetarios:** money.
- **Texto y carácter:** char, char(n), text, varchar(n) y name.
- **Fecha y hora:** abstime, date, datetime, interval, reltime, time, timespan y timestamp.
- **Booleanos:** bool o boolean.
- **Geométricos:** point, line, lseg, box, path, polygon y circle.
- **IPV4:** inet, cidr y macaddr.

- **Conjuntos enumerados:** se pueden emular con el *constraint* `CHECK`

```
CREATE TABLE ejemplo (
    mes  char(3) CHECK (mes IN
         ('Ene','Feb','Mar','May','Jun','Jul','Ago','Sep','Oct','Nov','Dic')),
    dia int2 CHECK (dia>0 and dia<32)   -- rango: [1..31]  )
```

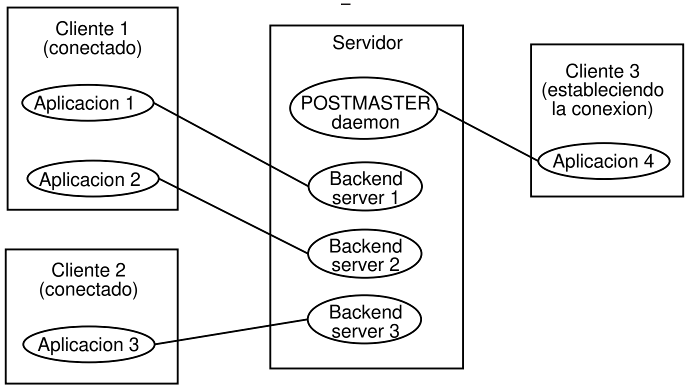
- **Arrays:** Vectores o matrices multidimensionales.
  - Se definen utilizando `atributo tipo[n]` donde n es el tamaño del array. Se puede omitir para dejar el tamaño indeterminado.
  - El acceso al contenido se realiza utilizando un índice `atributo[i]`.
- **Blobs:** (*Binary Large Objects*) permite almacenar en la BD archivos binarios (imagenes, documentos PS, ...).

  Debemos cargarlos y descargarlos directamente utilizando archivos del disco mediante las funciones `lo_import(file)` y `lo_export(file)`. La funcion `lo_unlink(objeto)` elimina el BLOB de la BD.

Funcionamiento interno - Arquitectura cliente/servidor

- Modelo ACS: *un proceso por usuario.*
- Procesos que intervienen de manera concurrente:
  1. Un proceso *daemon* supervisor: **postmaster**
  2. La aplicación del usuario: (por ejemplo el programa *psql*)
  3. Al menos un proceso *postgres backend server*: (el servidor **postgres** en si)

Ejemplo de sesiones simultáneas



Funcionamiento interno - Realización de una consulta

Las consultas se llevan a cabo en cuatro pasos:

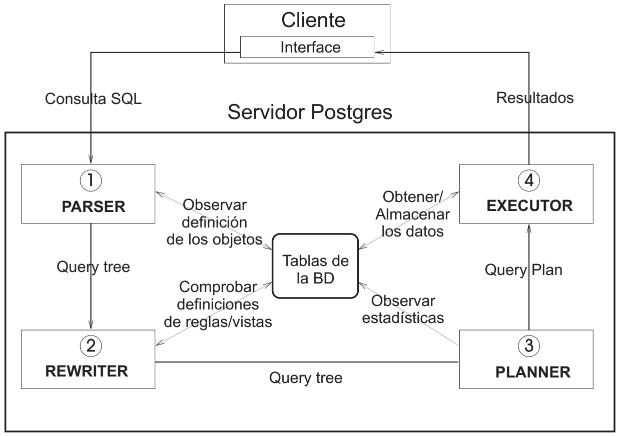
1. **parser**: chequea la sintaxis de la consulta y genera el *árbol de consulta* (*query tree*)

2. **rewriter**: procesa el *query tree* y comprueba la existencia de reglas reescribiendo de nuevo el *query tree* si es neceario.

3. **planner**: genera el *plan de consulta* a partir del *query tree* analizando todos los posibles caminos y escogiendo el de menor coste

4. **executor**: recorre recursivamente el árbol del *plan de consulta* y accede al disco para obtener los datos necesarios. Realiza los joins, cálculos y ordenaciones necesarias para devolver las tuplas resultado de la consulta.

Realización de una consulta



Funcionamiento interno - Las tablas del sistema

- Almacenan datos relacionados con el SGBD: usuarios y grupos, tipos de datos, funciones, operadores, etc.

- Los nombres de todas las tablas del sistema empiezan por *pg_* : pg_database, pg_tables, pg_user ...

- Se puede obtener un listado completo de estas tablas on el comando de *psql* `\dS` y una descripción de su contenido con `\d pg_nombretabla`

- Admiten la mayoría de las sentencias SQL para manipular su contenido al igual que las tablas del usuario: `SELECT, UPDATE, INSERT, ....` No admiten `DROP TABLE` ni `ALTER TABLE`

Las tablas más destacadas son:

pg_database : Lista de las bases de datos
pg_tables : Lista de tablas de la BD actual y del sistema (pg_*)
pg_user : Lista de usuarios con * en el campo passwd.
pg_shadow : Lista de usuarios (sólo accesible por el root)
pg_group : Lista de grupos de usuarios

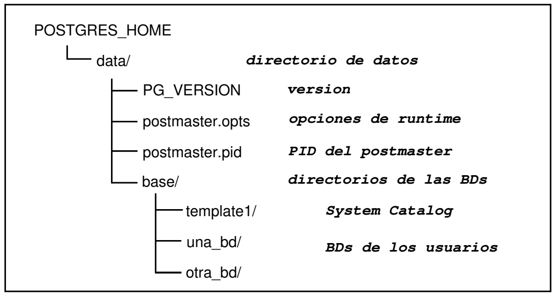
El *System Catalog* cuenta con un total de 33 tablas:

pg_aggregate	pg_am	pg_amop	pg_amproc	pg_attrdef	pg_attribute
pg_class	pg_database	pg_description	pg_group	pg_index	pg_indexes
pg_inheritproc	pg_inherits	pg_ipl	pg_language	pg_listener	pg_log
pg_opclass	pg_operator	pg_proc	pg_relcheck	pg_rewrite	pg_rules
pg_shadow	pg_statistic	pg_tables	pg_trigger	pg_type	pg_user
pg_variable	pg_views	pg_xactlock			

Funcionamiento interno - Directorio de datos

Se guardan todos los datos almacenados en la BD, incluyendo las tablas del sistema.

Está definido por la variable `POSTGRES_DATA` dentro del arhivo de configuración *postmaster.init*. Generalmente suele ser `$POSTGRES_HOME/data/`.



Parte 2: Instalación y configuración

- Instalación. Archivos de configuración.
- Creando y eliminando usuarios.
- Creando y eliminando BDs.

Instalación

Formas de instalación utilizando debian:

1. apt-get install postgresql
2. apt-get install task-database-pg
3. apt-get install lista_de_paquetes

- Paquetes estandar:

postgresql	Paquete principal.
postgresql-client	Programas y aplicaciones *front-end* (psql, createdb ...)
postgresql-contrib	Facilidades y mejoras adicionales (más tipos de datos ...)
postgresql-pl	Lenguaje procedimental PL/pgSQL

- Paquetes adicionales:

postgresql-doc	Documentacion (Guías del programador, del usuario ...)
postgresql-test	Test para comprobar un correcto funcionamiento
odbc-postgresql	Soporte para acceder por ODBC al *backend server*

- Interfaces con lenguajes de programación:

postgresql-dev	C y C++
python-pygresql	Python.
libpgjava	Java (JDBC)
libpgperl	Perl.
libdbd-pg-perl	Perl 5 usando DBI.
php3-pgsql	PHP3
php4-pgsql	PHP4
libpgtcl	TCL

Archivos de configuración

En Debian se encuentran en el directorio `/etc/postgresql`

postmaster.init	Opciones para el *postmaster daemon*
pg_options	Opciones para el *backend server*
pg_hba.conf	Control de las máquinas que pueden acceder y tipo de autenticación.
pg_ident.conf	Control de los usuarios (autenticación *ident*)
postgresql.env	Establece el entorno del usuario postgres (equivalente al .profile)
pg_geqo	Optimización de las consultas utilizando GEQO *(Genetic Query Optimization)*

Archivos de configuración - postmaster.init

Opciones más habituales para el *postmaster daemon*:

POSTGRES_HOME, POSTGRES_DATA y POSTGRES_LOG : Directorio home, directorio de datos y fichero de log.

PGBACKENDCOUNT y PGBUFFERS : Número máximo de procesos *backend server* simultáneos y número de buffers de memoria reservados.

PGDEBUG y PGECHO : Nivel de debug (0..4) y volcado o no de las consultas en el fichero de log.

PGFSYNC : Activar o desactivar la llamada a `fsync()` despues de cada transacción.

PGALLOWTCPIP y PGPORT: Permitir o no conexiones tcp/ip y número de puerto donde escucha el postmaster.

Archivos de configuración - pg_options

Opciones para el *backend server*. Sobre todo tipos y niveles de *trace* y *log*:

all : Rastrear todas las acciones (1), no rastrear nada (-1), o especificificar que acciones (0).

deadlock_timeout, nofsync, syslog, hostlookup, showportnumber: Timeout para interbloqueos, sincronizar o no a disco las transacciones, enviar los logs al syslog, activar la resolución de nombres y mostrar el puerto tcp/ip.

query, plan, parse, rewriten : Permiten rastrear el parser, el rewriter, el planner y/o el executor. También opciones para pretty_trace.

Otras opciones para bloqueos, memoria y estadísticas

Archivos de configuración - pg_hba.conf

- Control de las máquinas que pueden acceder y tipo de autenticación. (trust, password, crypt, ident, krb4, krb5 y reject)
- Tres tipos de registros: host, hostssl y local:
  - Registro **host**:
    Identifica un conjunto de máquinas pertenecientes a una red.
    `host DBNAME IP_ADDRESS ADDRESS_MASK AUTHTYPE [AUTH_ARGUMENT]`
  - Registro **hostssl**:
    Igual que el anterior pero auteticando por TCP/IP sobre SSL.
  - Registro **local**:
    Identifica el tipo de autenticación a usar conectándose en local.
    `local DBNAME AUTHTYPE [AUTH_ARGUMENT]`

Ejemplo de configuración: pg_hba.conf

```
# tipo_reg    BD         IP_red      máscara        tipo_auth   args_auth

# Permitir acceso a todos los usuarios locales:
 host      all      127.0.0.1     255.255.255.255    trust

# Permitir el acceso a la BD template1 a los ususarios de la red
# 192.168.1.0 que tengan cuenta en la máquina y en Postgres:
 host    template1  192.168.1.0   255.255.255.0      ident     sameuser

# Permitir el acceso a la BD ventas a los usuarios de la red
# 192.168.2.0 utilizando SSL y siempre que tenga password en pg_shadow:
 hostssl  ventas    192.168.2.0   255.255.255.0      crypt

# Denegar el acceso a la BD sueldos a los usuarios locales:
 local    sueldos      reject
```


Archivos de configuración - pg_ident.conf

Control de los usuarios utilizando autenticación *ident*.

- Contiene una tabla que identifica cada usuario con su nombre de usuario en Postgres.
- Cada registro contiene tres campos:

  `nombre_del_mapa usuario usuario_postgres`

- Ejemplo:

```
contabilidad   luis      usuario1
contabilidad   maria     usuario2
contabilidad   paco      miguel
comercial      carlos    com1
comercial      esteban   com2
```


Creando y eliminando usuarios

- Utilizando los front-ends `createuser` y `dropuser` desde la linea de comandos del shell
- Utilizando sentencias SQL:
  - Crear usuario:
    ```
    INSERT INTO pg_shadow
     (usename,usecreatedb,usesuper,passwd)
    VALUES ('jose',Y,Y,'mipasswd');
    ```
  - Eliminar usuario:
    ```
    DROP USER jose;
    ```
  - Listar los usuarios:
    ```
    SELECT * FROM pg_shadow;
    ```

Creando y eliminando BDs

- Utilizando los front-ends `createdb` y `dropdb` desde la linea de comandos del shell
- Utilizando sentencias SQL:
  - Crear una BD:
    ```
    INSERT INTO pg_database
     (datname,datdba,encoding,datpath)
    VALUES ('mibd',31,7,'mibd');
    ```
  - Eliminar una BD:
    ```
    DROP DATABASE mibd;
    ```
  - Listar los BDs:
    ```
    SELECT * FROM pg_database;
    ```


Parte 3: Mantenimiento y seguridad

- Seguridad y control de acceso.
- Backups y recuperación de datos.
- Optimizaciónes para aumento del rendimiento.

Seguridad y control de acceso

- Nivel 0: sobre **hosts y usuarios**:
  (qué máquinas y/o usuarios se pueden conectar)
  Utilizando las opciones de `pg_hba.conf`
- Nivel 1: sobre **usuarios y BDs**:
  (a que BDs se pueden conectar)
  Utilizando `pg_ident.conf` y mapas de usuarios.
- Nivel 2: sobre **tablas**:
  (a que tablas se puede acceder)
  Sentencias SQL:
  - Privilegios: `GRANT` y `REVOKE`
  - Vistas: `CREATE VIEW` y `DESTROY VIEW`

Backups y recuperación de datos

- Snapshot del directorio de datos (*POSTGRES_DATA*)
  Pros :
  - Rapidez y sencillez

  Contras :
  - Recomendable parar el postmaster y usuarios
  - Incompatibilidad entre versiones
- Volcado (dump) y recuperación
  Pros :
  - Backup "en caliente" y en remoto
  - Backups parciales (sólo algunas BDs)
  - Compatibilidad entre versiones

  Contras :
  - Gran lentitud con BDs grandes
  - Consumo de recursos del sistema

Backups - Snapshot del directorio de datos

Para asegurarnos de guardar las BD en un estado consitente, deberemos parar el servidor Postgres:

1. Detener el servidor:
   `/etc/init.d/postgresql stop`

2. Realizar la copia con tar y gzip:
   `tar -cvzf pgdata-backup.tgz /home/databases/postgres/data/`

3. Levantar el servidor:
   `/etc/init.d/postgresql start`

4. Copiar el tgz a algún soporte (disco, cinta, CD ...)

Para recuperar la copia: descomprimir y desempaquetar el fichero `pgdata-backup.tgz` en el directorio `POSTGRES_DATA`.

Backups - Volcado (dump) y recuperación

- Existen tres comandos que nos facilitan esta labor:
  - **pg_dump -d *database***: Vuelca el contenido de la BD *database* por STDOUT.
  - **pg_dumpall -d**: Vuelca el contenido de todas las BDs por STDOUT.
  - **psql -e *database*** Recupera el contenido de la BD *database* desde STDIN.
- Opciones más usuales de `pg_dump` y `pg_dumpall`:

```
pg_dump -h host -p puerto -d -u nombre_base_datos
```

```
pg_dumpall -h host -p puerto -d -u
```


Backups - Volcado (dump) y recuperación

1. Volcado y recuperación de una sóla BD
   - **Sin comprimir:**
     - Volcado: `pg_dump -d database > db.dump`
     - Recuperación: `psql -e database < db.dump`
   - **Comprimiendo:**
     - Volcado: `pg_dump -d database | gzip -c > db.dump.gz`
     - Recuperación: `psql -e database < gzip -c -d db.dump.gz`

2. Volcado y recuperación de todas las BDs
   - **Sin comprimir:**
     - Volcado: `pg_dumpall -d > todas-dbs.dump`
     - Recuperación: `psql -e template1 < todas-dbs.dump`
   - **Comprimiendo:**
     - Volcado: `pg_dumpall -d | gzip -c > todas-dbs.dump.gz`
     - Recuperación: `psql -e template1 < gzip -c -d todas-dbs.dump.gz`

Optimizaciones para aumento del rendimiento

1. Optimizando el servidor Postgres
   - Creación de índices
   - Limpieza y análisis de la BD: sentencia *vacuum*
   - Buffers de *shared-memory*, fsync() y conexiones simultáneas

2. Consideraciones sobre el hardware y el sistema operativo
   - Distribución de los datos en arrays de discos.
   - Sistemas redundantes y reparto de carga

Creación de índices

- Por defecto acceso secuencial al fichero dónde se almacena una tabla.
- Índices: ficheros adicionales que indexan la tabla por un deteminado campo creando punteros que permiten un acceso rápido.
- Sintaxis:
  ```
  CREATE INDEX indice ON tabla (columna)
  CREATE INDEX indice ON tabla USING [ BTREE | RTREE | HASH ] (columna)
  DROP INDEX indice
  ```
- Sentencia `CLUSTER`:
  - Reordena la tabla colocando juntos los valores duplicados.
  - Acelera el acceso en consultas que acceden a muchos valores duplicados.
  - Sintaxis: `CLUSTER indice ON tabla`

Limpieza y análisis de la BD: sentencia *VACUUM*

- **Actualizaciones**: se crea una copia de la tupla actualizada manteniendo ambas en la tabla y marcando la vieja.
- **Eliminaciones**: se marcan las tuplas como borradas pero no se eliminan de la tabla (eliminación perezosa).
- Acelera los procesos de `UPDATE` y `DELETE` pero se genera *basura* en las tablas aumentando su tamaño.
- **Sentencia** `VACUUM`: Borra fisicamente del fichero las tuplas *caducadas*.
  `VACUUM` : Analiza todas las tablas de la BD
  `VACUUM tabla` : Analiza sólo la tabla `tabla`

Buffers de *shared-memory*, fsync() y conexiones simultáneas

- **Buffers de *shared-memory***:
  - Establecen la cantidad de memoria reservada para el *server backend*. Cuanta más memoria más rápido se podran realizar las consultas.
  - Se especifican el la variable `PGBUFFERS` del archivo `postmaster.init`. Cada buffer es de 8kb.
  - El número máximo de buffers depende de la cantidad de memoria de la máquina y de los parámetros del Kernel. Se puede cambiar y consultar el el archivo `/proc/sys/kernel/shmmax`.

- **fsync()** :
  - Fuerza el volcado a disco de las transacciones terminadas.
  - Se activa o desactiva en la variable `PGFSYNC`.
  - Desactivada aumenta el rendimiento pero tiene el incoveniente de que un fallo en el sistema puede provocar pérdida de datos.
- **Conexiones simúltáneas:**
  - El número máximo de conexiones o clientes concurrentes se especifica en la variable `PGBACKENDCOUNT` y en principio está limitado a 1024.
  - Para permitir más de 1024 clientes concurrentes deberemos cambiar el valor de `MAXBACKENDS` en el fichero `include/config.h` y recompilar.
  - Si aumentamos el valor de `PGBACKENDCOUNT` deberíamos aumentar también `PGBUFFERS` de forma que tuviese al menos el doble.

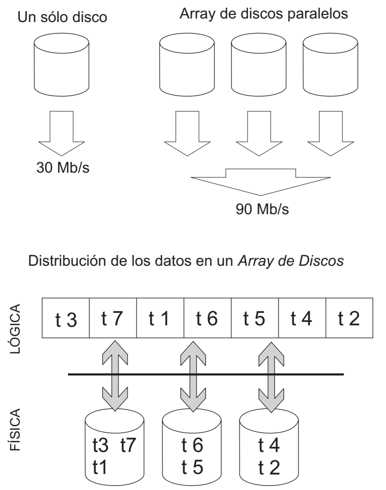
Distribución de los datos en arrays de discos

- Cada tabla y cada índice se almacenan en un fichero diferente.
  - Fácil reparto de datos (BDs, tablas e índices)

- Reparto de ficheros en varios discos

  - Acceso en paralelo (lecturas y escrituras)
  - Aumento del rendimiento

- Discos redundantes (RAID 1)
  - Protección ante fallos hardware
  - Lecturas más rápidas



Sistemas redundantes y reparto de carga

- Actualmente Postgres no soporta servidores redundates que compartan y actualicen los mismos datos.

- Para entornos orientados a *sólo-consultas*, se puede montar un sistema redundante:

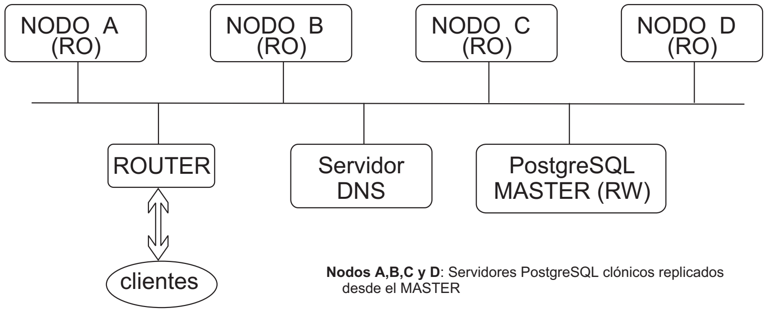
  - Formado por :
    * 1 servidor maestro (lectura-escritura)
    * n-servidores esclavos (sólo-lectura)

  - Mayor tolerancia a fallos

  - Reparto de carga

  - Tiempo de respuesta menor

Sistema de servidores Postgres redundantes (sólo consultas)



Nodos A,B,C y D: Servidores PostgreSQL clónicos replicados desde el MASTER

Servidor DNS: Entradas redundantes identificando las IPs de A,B,C y D con un sólo nombre.

MASTER: Servidor PostgreSQL para actualizaciones

Parte 4: Interfaces con entornos y lenguajes

- Clientes modo terminal y clientes gráficos.
- Clientes Windows.
- Interfaz con algunos lenguajes.

Clientes modo terminal y clientes gráficos

- **psql** : Cliente interactivo en modo texto que actúa de front-end contra Postgres. Tambien procesa archivos de texto y permite formatear la salida.
  Incluido con PostgreSQL.
- **pgaccess** : Cliente gráfico desarrollado por Constantin Teodorescu en Visual Tcl.
  Disponible como paquete deb y rpm
  `http://flex.flex.ro/pgaccess/`
- **gnome-db** : Ofrece una arquitectura de acceso a BDs bajo gnome.
  Disponible como paquete deb y rpm
  `http://www.gnome-db.org`

Clientes Windows

- **ODBC** : Acceso ODBC para clientes windows. Instala un nuevo *DSN* bajo el *ODBC data source* permitiendo que cualquier aplicación lo utilice.
  `ftp://ftp.postgresql.org/pub/odbc/`
- **pgadmin** : Aplicación cliente vía ODBC bajo Windows desarrollada en Visual Basic.
  `http://www.greatbridge.org/project/pgadmin/`
- **ZDO** : Zeos Database Objects son un conjunto de componentes Delphi para acceder directamente a servidores Postgres sin utilizar DBE. Disponibles tambien para C++ Builder
  `http://www.zeos.dn.ua`

Interfaz con algunos lenguajes

- Cuenta con interfaces y librerías para acceder desde la mayoría de los lenguajes de programación y scripting actuales:

  C y C++, Java, Delphi, Python, Perl, PHP, TCL, Ruby, Bash ...

- Diversas utilidades y programas que permiten:
  - Migrar desde otros sistemas de almacenamiento como MS Access o ficheros DBF.
  - Migrar desde otros SGBD como mySQL y Oracle.
- Se ejecuta bajo diversos sistemas operativos además de Linux:

  Solaris, FreeBSD, NetBSD, QNX, Tru64, HPUX, IRIX, SCO, AIX, SunOS e incluso Win NT.

Ejemplo de código bajo C (`libpgeasy`)

```
#include <stdio.h>
#include <libpq-fe.h>
#include <libpgeasy.h>                          /* libpgeasy header file */
int main() {
    char       state_code[3];                   /* holds state code entered by user */
    char       query_string[256];               /* holds constructed SQL query */
    char       state_name[31];                  /* holds returned state name */

    connectdb("dbname=test");                   /* connect to the database */
    printf("Enter a state code:  ");            /* prompt user for a state code */
    scanf("%2s", state_code);
    sprintf(query_string,                       /* create an SQL query string */
            "SELECT name FROM statename \
             WHERE code = '%s'", state_code);
    doquery(query_string);                      /* send the query */
    while (fetch(state_name) != END_OF_TUPLES)  /* loop through all rows returned */
        printf("%s\n", state_name);             /* print the value returned */
    disconnectdb();                             /* disconnect from the database */
    return 0;
}
```


Ejemplo de código bajo Python (`pygresql`)

```
#!/usr/bin/python

import sys
from pg import DB                                  # load database routines
conn = DB('test')                                  # connect to the database

sys.stdout.write('Enter a state code:  ')          # prompt user for a state code
state_code = sys.stdin.readline()
state_code = state_code[:-1]

for name in conn.query(                            # send the query
        "SELECT name \
         FROM statename \
         WHERE code = '"+state_code+"'").getresult():
        sys.stdout.write('%s\n' % name)            # print the value returned
```


Ejemplo de código bajo Perl (`libpgperl`)

```
#!/usr/bin/perl

use Pg;                                             # load database routines
$conn = Pg::connectdb("dbname=test");               # connect to the database
                                                    # did the database connection fail?
die $conn->errorMessage unless PGRES_CONNECTION_OK eq $conn->status;

print "Enter a state code: ";                       # prompt user for a state code
$state_code = <STDIN>;
chomp $state_code;
$result = $conn->exec(                              # send the query
        "SELECT name \
         FROM statename \
         WHERE code = '$state_code'");
                                                    # did the query fail?
die $conn->errorMessage unless PGRES_TUPLES_OK eq $result->resultStatus;
while (@row = $result->fetchrow) {                  # loop through all rows returned
        print @row, "\n";                           # print the value returned
}
```


Referencias

Enhancement of the ANSI SQL Implementation of PostgreSQL. *Georg Gottlob and Katrin Seyr*: 1998. Viena University of Technology

Efficient Data Organization and Load Balancing on Parallel Disks. *Szu-Wen Kuo*: 1999. Ph.D. Thesis, University of Illinois.

Parallel systems in the data warehouse. *Stephen Morse and David Isaac*: 1998. Prentice Hall.

PostgreSQL: Introduction and Concepts. *Bruce Momjian*:2001. Addison-Wesley

The PostgreSQL development team tutorial and guides edited by *Thomas Lockart*:

- The PostgreSQL Tutorial
- The PostgreSQL User's Guide
- The PostgreSQL Administrator's Guide
- The PostgreSQL Programmer's Guide

Slides creados en LaTeX usando la clase FoilTeX desarrollada por IBM.