Objetos y jerarquía en PostgreSQL

info@todopostgresql.com

Todo PostgreSQL
by Abatic

abatic

1. ORDBMS
2. Jerarquía de los Objetos
3. Bases de Datos
4. Esquemas
    - search_path
5. Roles
    - Usuarios
    - Grupos
6. Control de Acceso

Todo PostgreSQL
by Abatic

ORDBMS

Los Sistemas Gestores de Bases de Datos Objeto-Relacional (SGBDRO) es una extensión de la base de datos relacional tradicional, a la cual se le proporcionan características de la programación orientada a objetos (POO).

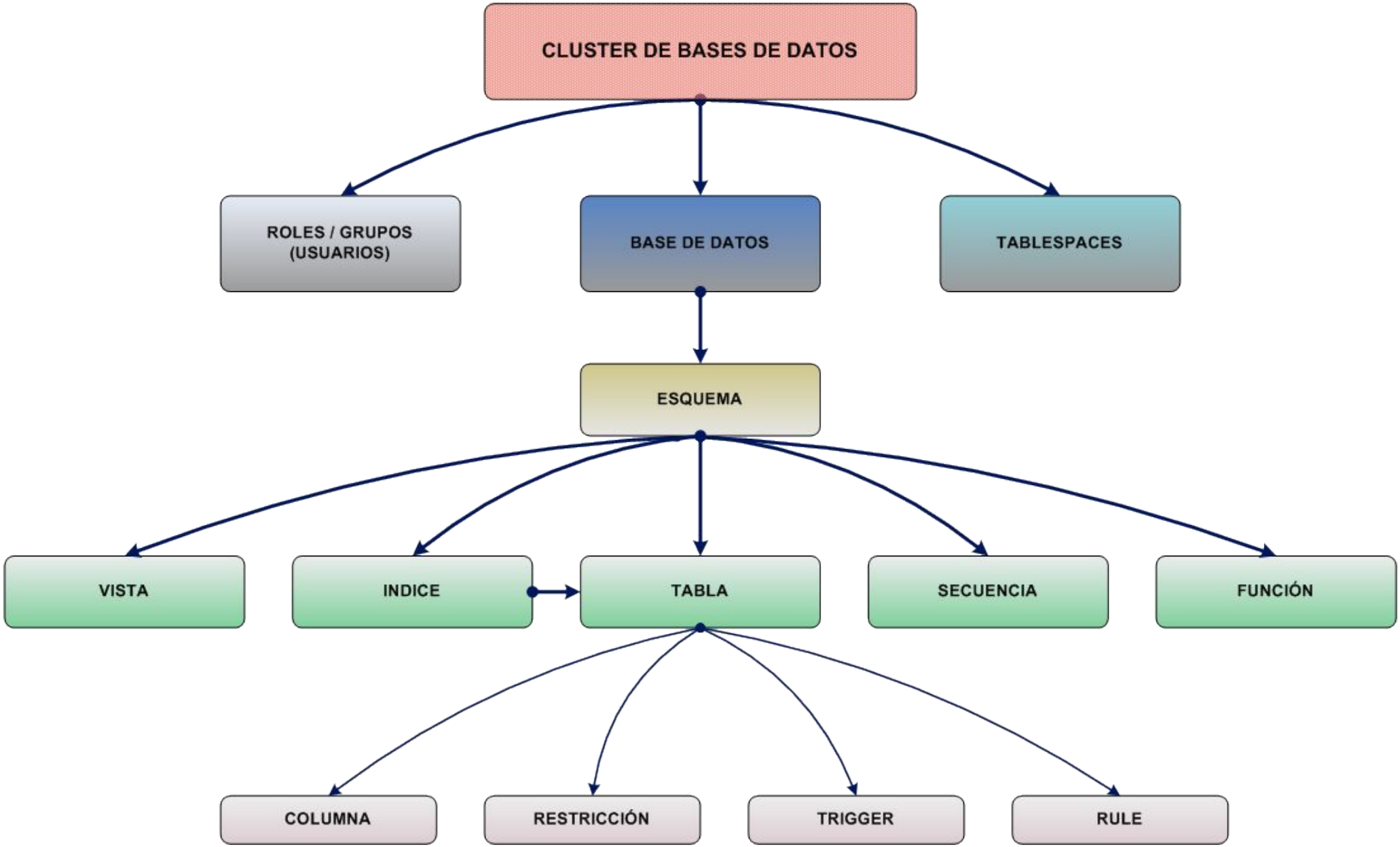
Más opciones: NoSQL, Extensiones JSON.

Jerarquía de Objetos

- CLUSTER = instancia de servidor
  - Roles
  - Lenguajes Procedurales (PL)
  - Base de Datos ~ Aplicación
    - Extensiones
    - Esquemas

Jerarquía de Objetos

- Esquema
  - Tabla
  - Secuencia
  - Vista
  - Tipos de datos definidos por el usuario
  - Funciones y triggers



Todo PostgreSQL
by Abatic

Bases de Datos

- Una base de datos es una **colección de objetos de SQL con nombre**.
- Contiene una **colección de esquemas** y éstos contienen tablas, vistas, funciones, etc.

- Se crean con el comando CREATE DATABASE
- Se eliminan con el comando DROP DATABASE

Bases de Datos

Sintaxis:

```
CREATE DATABASE name [ [ WITH ]
    [ OWNER [=] user_name ]
    [ TEMPLATE [=] template ]
    [ ENCODING [=] encoding ]
    [ TABLESPACE [=] tablespace_name ]
    [ CONNECTION LIMIT [=] connlimit ]
    [ IS_TEMPLATE [=] is_template ] ]
```

Bases de Datos

```
CREATE DATABASE todopostgresql;

CREATE DATABASE midb OWNER propietario;

CREATE DATABASE miplantilla IS_TEMPLATE true;

CREATE DATABASE midb TABLESPACE mispace;
```

Bases de Datos

Se pueden “clonar” bases de datos, usándolas como plantillas.

Existen dos BB.DD. “template” por defecto.

- Template1→ base de datos “normal”.
- Template0 → base de datos “vacía”.

Bases de Datos

- Las nuevas bases de datos se copian desde template1. Por ello, las extensiones y PL se instalan por defecto en template1.
- Desde v.9.0, no hace falta instalar el lenguaje procedural PL/pgSQL.

Esquemas

- Es una colección de **objetos SQL designada mediante un nombre único** (dentro de la base de datos).
- Una base de datos contiene uno o más esquemas señalados por nombre que a su vez contienen tablas.
- Los esquemas también contienen otros objetos con nombre incluyendo tipos de datos, funciones y operadores.

Esquemas

Hay varias razones para utilizar esquemas:

- Permitir que muchos usuarios utilicen una base de datos sin interferir unos con otros.
- Organizar los objetos de base de datos en grupos lógicos para hacerlos más manejables.
- Aplicaciones de terceros pueden ser colocadas dentro de esquemas separados de modo que no colisionen con los nombres de otros objetos.

Esquemas

Sintaxis:

```
CREATE SCHEMA schemaname [ AUTHORIZATION role ]
[ schema_element [ ... ] ]
CREATE SCHEMA AUTHORIZATION role [
schema_element [ ... ] ]
CREATE SCHEMA IF NOT EXISTS schemaname [
AUTHORIZATION role ] [ schema_element [ ... ] ]
```

Esquemas

Sintaxis:

Donde en “role” puede ser:

- Nombre_user
- CURRENT_USER
- SESSION_USER

Esquemas

```
CREATE SCHEMA training AUTHORIZATION
postgre_dba;

CREATE SCHEMA training_materialized_views;

CREATE SCHEMA IF NOT EXISTS AUTHORIZATION
todopostgresql;
```

Esquemas

```
CREATE SCHEMA terror

    CREATE TABLE peliculas (titulo text, anio date,
    premios text[])

    CREATE VIEW ganadores AS SELECT titulo, anio FROM
    peliculas WHERE premios IS NOT NULL;
```

Esquemas

Por defecto los objetos se crearán en un esquema llamado public.

Para crearlo o referenciarlo en otro esquema:
NOMBRE_ESQUEMA.NOMBRE_OBJETO

Esquemas - search_path

- Los nombres absolutos (esquema.tabla) son difíciles de escribir así que habitualmente usamos sólo los nombres de las tablas en las consultas.
- Si no se provee un nombre de esquema, Se usa la variable de entorno de la conexión "**search_path**" para determinar en qué esquemas se buscan las tablas coincidentes.

Esquemas - search_path

- El primer esquema, hace referencia a un esquema llamado igual que el usuario. En caso de que exista se utiliza como esquema actual, en caso contrario se omite dicho esquema.
- El segundo esquema se llama “public”.
- En el esquema actual serán creados los objetos que no especifique un nombre de esquema.

Esquemas - search_path

```
postgres=# show search_path;
   search_path
-----------------
 "$user", public
(1 fila)
```

Esquemas - search_path

```
SELECT * FROM trading;
```

Esta sentencia encontrará la primera tabla llamada “trading” según la lista de esquemas que contiene la ruta de búsqueda (search_path).

Es una “variable de entorno de la conexión” y se puede cambiar mediante **SET**.

```
SET search_path TO trading, public;
```

Roles

Un rol puede ser:

- Un usuario de una base de datos (user).
- Un conjunto de usuarios de base de datos (grupo).
- Un usuario sin login (rol).

Roles

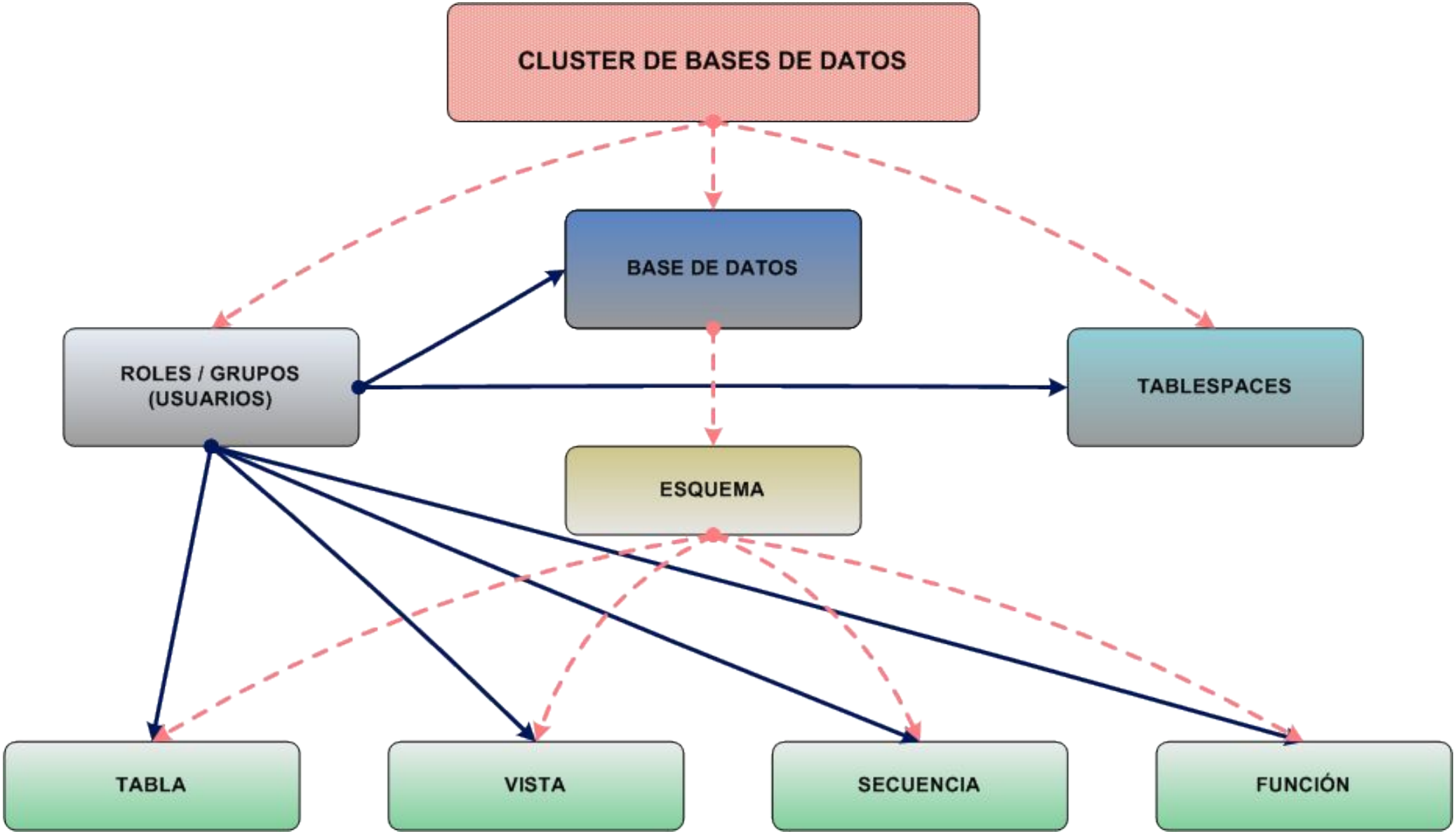
Diferencias:

- Un **Rol** es un usuario sin privilegios de login.
- Un **Usuario** es un rol que puede acceder a una base de datos.
- Un **Grupo** es un rol que no puede ser utilizado para acceder a ninguna base de datos.

Roles

Los Usuarios, Grupos y Roles:

- Son globales en toda la instancia del servidor (cluster) de base de datos.
- Pueden anidarse.
- En PostgreSQL son roles con ciertas diferencias.



Todo PostgreSQL
by Abatic

Roles

- CREATE ROLE equivale a CREATE USER, con la diferencia que al crear un usuario, éste tiene el privilegio de **LOGIN** (acceso).
- Se pueden añadir Usuarios a ROLES de manera similar a la forma en que se los puede añadir a GRUPOS.

Roles: Usuarios

- Los **Usuarios** de la Base de Datos son diferentes a los usuarios del Sistema Operativo. Es posible que coincida en el nombre.
- El “superusuario” es aquél bajo cuya identidad corre el postmaster. Es decir, es el usuario del S.O. que crea el cluster.
- Se determinan equivalencias mediante el método “peer” (sólo UN*X).

Roles: Usuarios

Desde SQL, utilizando el comando CREATE USER

```
CREATE USER usuario1 PASSWORD 'postgresql';
```

Desde línea de comandos, con la utilidad createuser

```
createuser [OPTION]... [ROLENAME]
```

```
createuser --interactive jose
```

Roles: Grupos

- Pueden ser propietarios de objetos de base de datos.
- Pueden asignar privilegios sobre estos objetos a otros roles para controlar quién tiene acceso a qué objetos.
- Además, es posible otorgar pertenencia en un grupo a otro grupo permitiendo así que el rol miembro use privilegios asignados al rol del que es miembro.
- Parecidos a los grupos en Unix.

Roles: Grupos

```
CREATE GROUP grupo_dbas WITH USER dba1;
CREATE GROUP becarios;
ALTER GROUP becarios ADD USER usuario1;
DROP GROUP becarios;
```

Control de Acceso

Se otorga y se quita permisos a un objeto usando los comandos de SQL GRANT y REVOKE.

GRANT **UPDATE DELETE** ON curso TO usuario1;

GRANT **ALL** ON dept TO **GROUP** becarios;

REVOKE **UPDATE DELETE** ON curso FROM usuario1;

GRANT **USAGE** ON **SCHEMA** formacion TO instructor;

Cursos de PostgreSQL para DBA y Developers

Todo PostgreSQL
by Abatic

TODOPOSTGRESQL.COM