PostgreSQL

Introducción

PostgreSQL es un sistema de gestión de base de datos relacional **RDBMS**, Orientada a Objetos y Libre, publicado bajo la licencia BSD.

Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales.

PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando

Historia

PostgreSQL ha tenido una larga evolución, la cual se inicia en 1982 con el proyecto Ingres en la Universidad de Berkeley. Este proyecto, liderado por Michael Stonebraker, fue uno de los primeros intentos en implementar un motor de base de datos relacional. Después de haber trabajado un largo tiempo en Ingres y de haber tenido una experiencia comercial con él mismo, Michael decidió volver a la Universidad en 1985 para trabajar en un nuevo proyecto sobre la experiencia de Ingres, dicho proyecto fue llamado post-ingres o simplemente POSTGRES.

El proyecto post-ingres pretendía resolver los problemas con el modelo de base de datos relacional que habían sido aclarados a comienzos de los años 1980. El principal de estos problemas era la incapacidad del modelo relacional de comprender "tipos", es decir, combinaciones de datos simples que conforman una única unidad. Actualmente estos son llamados objetos. Se esforzaron en introducir la menor cantidad posible de funcionalidades para completar el soporte de tipos. Estas funcionalidades incluían la habilidad de definir tipos, pero también la habilidad de describir relaciones - las cuales hasta ese momento eran ampliamente utilizadas pero mantenidas completamente por el usuario. En Postgres la base de datos «comprendía» las relaciones y podía obtener información de tablas relacionadas utilizando reglas. Postgres usó muchas ideas de Ingres pero no su código.

La siguiente lista muestra los hitos más importantes en la vida del proyecto Postgres.

- 1986: se publicaron varios papers que describían las bases del sistema.
- 1988: ya se contaba con una versión utilizable.

- 1989: el grupo publicaba la versión 1 para una pequeña comunidad de usuarios.
- 1990: se publicaba la versión 2 la cual tenía prácticamente reescrito el sistema de reglas.
- 1991: publicación de la versión 3, esta añadía la capacidad de múltiples motores de almacenamiento.
- 1993: crecimiento importante de la comunidad de usuarios, la cual demandaba más características.
- 1994: después de la publicación de la versión 4, el proyecto terminó y el grupo se disolvió.

Después de que el proyecto POSTGRES terminara, dos graduados de la universidad, Andrew Yu y Jolly Chen, comenzaron a trabajar sobre el código de POSTGRES, esto fue posible dado que POSTGRES estaba licenciado bajo la BSD, y lo primero que hicieron fue añadir soporte para el lenguaje SQL a POSTGRES, dado que anteriormente contaba con un intérprete del lenguaje de consultas QUEL (basado en Ingres), creando así el sistema al cual denominaron Postgres95.

Para el año 1996 se unieron al proyecto personas ajenas a la Universidad como Marc Fournier de Hub.Org Networking Services, Bruce Momjian y Vadim B. Mikheev quienes proporcionaron el primer servidor de desarrollo no universitario para el esfuerzo de desarrollo de código abierto y comenzaron a trabajar para estabilizar el código de Postgres95.

En el año 1996 decidieron cambiar el nombre de Postgres95 de tal modo que refleje la característica del lenguaje SQL y lo terminaron llamando PostgreSQL, cuya primera versión de código abierto fue lanzada el 1 de agosto de 1996. La primera versión formal de PostgreSQL (6.0) fue liberada en enero de 1997. Desde entonces, muchos desarrolladores entusiastas de los motores de base de datos se unieron al proyecto, coordinaron vía Internet y entre todos comenzaron a incorporar muchas características al motor. Aunque la licencia permitía la comercialización de PostgreSQL, el código no se desarrolló en principio con fines comerciales, algo sorprendente considerando las ventajas que PostgreSQL ofrecía. La principal derivación se originó cuando Paula Hawthtorn (un miembro del equipo original de Ingres que se pasó a Postgres) y Michael Stonebraker conformaron Illustra Information Technologies para comercializar Postgres.

En 2000, ex inversionistas de Red Hat crearon la empresa Great Bridge para comercializar PostgreSQL y competir contra proveedores comerciales de bases de datos. Great Bridge auspició a varios desarrolladores de PostgreSQL y donó recursos de vuelta a la comunidad, pero a fines de 2001 cerró debido a la dura competencia de compañías como Red Hat y pobres condiciones del mercado.

En 2001, Command Prompt, Inc. lanzó Mammonth PostgreSQL, la más antigua distribución comercial de PostgreSQL. Continúa brindando soporte a la comunidad PostgreSQL a través del auspicio de desarrolladores y proyectos, incluyendo PL/Perl, PL/php y el alojamiento de proyectos de comunidades como PostgreSQL Build Farm.

En enero de 2005, PostgreSQL recibió apoyo del proveedor de base de datos Pervasive Software, conocido por su producto Btrieve que se utilizaba en la plataforma Novell Netware. Pervasive anunció soporte comercial y participación comunitaria y logró algo de éxito. Sin embargo, en julio de 2006 dejó el mercado de soporte de PostgreSQL.

A mediados de 2005 otras dos compañías anunciaron planes para comercializar PostgreSQL con énfasis en nichos separados de mercados. EnterpriseDB añadió funcionalidades que le permitían a las aplicaciones escritas para trabajar con Oracle ser más fáciles de ejecutar con PostgreSQL. Greenplum contribuyó mejoras directamente orientadas a aplicaciones de Data Warehouse e Inteligencia de negocios, incluyendo el proyecto BizGres.

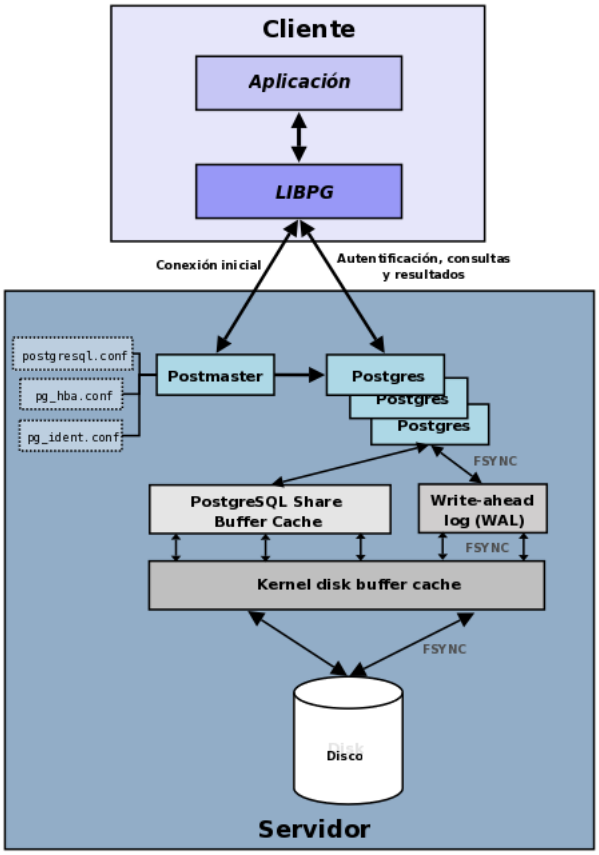
En octubre de 2005, John Loiacono, vicepresidente ejecutivo de software en Sun Microsystems comentó: "No estamos yendo tras el OEM de Microsoft pero estamos viendo a PostgreSQL ahora", aunque no se dieron especificaciones en ese momento. Para noviembre de 2005, Sun Solaris 10 (lanzamiento 6/06) incluía PostgreSQL.

En agosto de 2007 EnterpriseDB anunció el Postgres Resource Center y EnterpriseDB Postgres, diseñados para ser una completamente configurada distribución de PostgreSQL incluyendo muchos módulos contribuidos y agregados. EnterpriseDB Postgres fue renombrado Postgres Plus en marzo de 2008.

El proyecto PostgreSQL continúa haciendo lanzamientos principales anualmente y lanzamientos menores de reparación de bugs, todos disponibles bajo la licencia BSD, y basados en contribuciones de proveedores comerciales, empresas aportantes y programadores de código abierto mayormente.

Estructura

La estructura básica de los servicios de PostgreSQL está determinada en la siguiente imágen:



- Aplicación cliente: Esta es la aplicación cliente que utiliza PostgreSQL como administrador de bases de datos. La conexión puede ocurrir via TCP/IP ó sockets locales.
- Demonio #PostMaster: Este es el proceso principal de PostgreSQL. Es el encargado de escuchar por un puerto/socket por conexiones entrantes de clientes. Tambien es el encargado de crear los procesos hijos que se encargaran de autentificar estas peticiones, gestionar las consultas y mandar los resultados a las aplicaciones clientes.
- #Archivos_de_Configuracion: Los 3 ficheros principales de configuración utilizados por PostgreSQL, postgresql.conf, pg_hba.conf y pg_ident.conf.
- Procesos hijos postgres: Procesos hijos que se encargan de autentificar a los clientes, de gestionar las consultas y mandar los resultados a las aplicaciones clientes.
- #PostgreSQL_Share_Buffer_Cache: Memoria compartida usada por PostgreSQL para almacenar datos en caché.
- #Write_Ahead_Log (WAL): Componente del sistema encargado de asegurar la integridad de los datos (recuperación de tipo REDO).
- Kernel disk buffer cache: Caché de disco del sistema operativo.
- Disco: Disco Físico donde se almacenan los datos y toda la información necesaria para que PostgreSQL funcione

Limpieza (Vacuum)

Vacuum es el proceso en el cual se eliminan definitivamente tuplas marcadas para borrar (obsoletas, utilizando espacio y registros), y hay una reorganización de datos a nivel físico.

El proceso de Vacuum debe ser ejecutado regularmente, debido a:

- Para recuperar o reutilizar el espacio en disco utilizado por los registros actualizados o eliminados
- Para actualizar las estadísticas usadas por el #Planner_Query de PostgreSQL (para optimizar el acceso a la información)
- Para proteger la perdida de información debido al overflow producido por la utilización de transacciones (Frozen XID)

Se puede realizar la limpieza, utilizando el comando externo **vacuumdb** y el cual puede recibir parámetros para realizar los diferentes tipos de vaciamiento:

- **FREEZE** = Realiza el bloqueo exclusivo de las Tablas, recupera más espacio, y es muy lento
- **FULL** = (Obsoleta) Ejecuta el Vacuum con la opción vacuum_freeze_min_age en cero
- **ANALYZE** = Actualiza las estadísticas utilizadas por el #Planner_Query
- **VERBOSE** = Imprime de forma detallada las actividades por cada Tabla

Este comando es muy útil para automatizar vaciamientos a través de cualquier sincronizador de tareas, como Crontab.

Asimismo, existe la opción del #AutoVacuum, cuya funcionalidad es ir realizando de manera paulatina la mantención de nuestra base. Previamente y antes de activar esta funcionalidad, es recomendable leer sobre las consideraciones a tener en cuenta, para no degradar la performance de nuestro servidor.

Para determinar la cantidad de Transacciones que mantiene una Base de Datos, es necesario ejecutar lo siguiente

```
SELECT datname, age(datfrozenxid) FROM pg_database ORDER BY age DESC;
```

Y de forma más precisa, se puede determinar la cantidad de Transacciones por cada Tabla, conectado a la Base de Datos, ejecutando lo siguiente

```
SELECT relname, age(relfrozenxid) FROM pg_class WHERE relkind = 'r' ORDER BY age(relfrozenxid) DESC;
```

⚠ **Nota:** Es muy importante saber la cantidad de transacciones (registros de PostgreSQL) de las Bases de Datos ya que, debido a una limitante del RDBMS, no puede ser superior a 2.000.000.000 (2 billones) de registros, luego de esto, se comenzarán a reescribir los primeros registros (**overflow**) de la Base de Datos, dejando irrecuperable la información.

Ejemplo

```
regression=# VACUUM VERBOSE ANALYZE onek; INFO: vacuuming "public.onek" INFO: index "onek_unique1" now contains 1000 tuples in 14 pages DETAIL: 3000 index tuples were re
```

AutoVacuum

Comandos

Espacio de Tablas e Indices (PostgreSQL 8)

```
SELECT pg_database.datname, pg_class.relname AS "DB.Tabla", CASE WHEN pg_class.relkind = 'r' THEN 'Tabla' ELSE 'Indice' END AS "Tipo", pg_class.reltuples AS "Tuplas", pg
```

Espacio de Tablas e Indices (PostgreSQL 9)

```
SELECT pg_database.datname, pg_class.relname AS "DB.Tabla", CASE WHEN pg_class.relkind = 'r' THEN 'Tabla' ELSE 'Indice' END AS "Tipo",pg_class.reltuples AS "Tuplas", pg_
```

Tamaño de Base de Datos

```
SELECT datname, pg_size_pretty(pg_database_size(datname)) FROM pg_database;
```

Tiempo de Consultas (PostgreSQL 8)

```
SELECT query_start, datname, procpid, current_query FROM pg_stat_activity WHERE current_query != '<IDLE>' ORDER BY query_start;
```

Tiempo de Consultas (PostgreSQL 9)

```
SELECT query_start,datid,datname,pid,usename,state,client_addr,query from pg_stat_activity where state!='idle' order by query_start;
```

Cancelar Consulta

```
SELECT pg_cancel_backend(PROCPID);
```

Número de Tuplas Muertas

```
SELECT psut.relname, to_char(psut.last_vacuum, 'YYYY-MM-DD HH24:MI') as last_vacuum, to_char(psut.last_autovacuum, 'YYYY-MM-DD HH24:MI') as last_autovacuum, to_char(pg_c
```

Bloqueos a una relación (Postgres 8)

```
select pg_class.relname, pg_locks.pid,pg_locks.mode,pg_stat_activity.current_query,pg_stat_activity.query_start from pg_class join pg_locks on pg_class.oid = pg_locks.rel
```

Bloqueos a una relación (Postgres 9)

```
select pg_class.relname, pg_locks.pid,mode,pg_stat_activity.query,pg_stat_activity.query_start from pg_class join pg_locks on pg_class.oid = pg_locks.relation join pg_sta
```

Intervalo de checkpoint

```
SELECT total_checkpoints, seconds_since_start / total_checkpoints / 60 AS minutes_between_checkpoints FROM (SELECT EXTRACT(EPOCH FROM (now() - pg_postmaster_start_time()
```

Tamaño de tuplas y estadísticas para mantención (PostgreSQL 8)

```
SELECT psut.relname, pg_class.relpages * 8 / 1024 AS "Espacio", to_char(psut.last_vacuum, 'YYYY-MM-DD HH24:MI') as last_vacuum, to_char(psut.last_autovacuum, 'YYYY-MM-DD
```

Tamaño de tuplas y estadísticas para mantención (PostgreSQL 9)

```
SELECT psut.relname, pg_class.relpages * 8 / 1024 AS "Espacio", to_char(psut.last_vacuum, 'YYYY-MM-DD HH24:MI') as last_vacuum, to_char(psut.last_autovacuum, 'YYYY-MM-DD
```

Transacciones por relación

```
SELECT c.oid::regclass as table_name,greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as age FROM pg_class c LEFT JOIN pg_class t ON c.reltoastrelid = t.oid WHERE c.relk
```

Buscar índices duplicados

```
SELECT pg_size_pretty(SUM(pg_relation_size(idx))::BIGINT) AS SIZE,(array_agg(idx))[1] AS idx1, (array_agg(idx))[2] AS idx2, (array_agg(idx))[3] AS idx3, (array_agg(idx))
```

Scans por índice

```
SELECT indexname, t.tablename, c.reltuples AS num_rows,pg_size_pretty(pg_relation_size(quote_ident(indexrelname)::text)) AS index_size, CASE WHEN indisunique THEN 'Y' EL
```

Estadísticas del Query Planner

```
SELECT attname, inherited, n_distinct, array_to_string(most_common_vals, E'\n') as most_common_vals FROM pg_stats WHERE tablename = 'TABLA';
```

Uso de índices (ratio) por tabla

```
SELECT relname, 100 * idx_scan / (seq_scan + idx_scan) as index_use_ratio, n_live_tup as rows FROM pg_stat_user_tables WHERE seq_scan + idx_scan > 0 ORDER BY n_live_tup
```

Manejo de Usuarios (Roles)

Crear un usuario con permisos de logeo

```
CREATE ROLE <nombre> LOGIN;
```

Además de LOGIN, se pueden especificar opciones como:

- SUPERUSER : Da privilegios de usuarios 'postgres' al usuario creado.
- VALID UNTIL <'timestamp'> : Especificar un periodo de actividad para el usuario, luego es desactivado.
- CREATEDB : Permisos para crear bases de datos

Estas son solo algunas de las opciones más utilizadas, se pueden consultar todas en detalle en: https://www.postgresql.org/docs/9.3/static/sql-createrole.html

Dar Permisos

Se debe utilizar una opción de permisos para asignarla a un usuario específico, sobre una tabla específica:

```
GRANT [ SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER ] ON <tabla> TO <usuario>;
```

O se puede indicar que se asignen todos los privilegios sobre una tabla a un usuario:

```
GRANT ALL ON <tabla> TO <usuario>;
```

Si se requiere que un usuario tenga uno o todos los permisos sobre todas las tablas de la base de datos, se debe usar de la siguiente manera:

```
GRANT [ (PERMISO) | ALL ] ON ALL TABLES IN SCHEMA <esquema> TO <usuario>;
```

Para listar los esquemas:

```
SELECT schema_name from information_schema.schemata;
```

Quitar Permisos

```
REVOKE [ (PERMISO) | ALL ] ON [ TABLA | ALL TABLES IN SCHEMA <esquema> ] FROM <usuario> [CASCADE] ;
```

PostMaster

PostgreSQL Share Buffer Cache

Write Ahead Log

Planner Query

Configuración Básica

Archivos de Configuracion

pg_hba.conf

pg_ident.conf

postgresql.conf

Variables de Configuracion

Configuración Avanzada

Donde obtener PostgreSQL

PostgreSQL ofrece varias formas de obtener PostgreSQL, dependiendo de la forma que se vaya a instalar.

💡 **Tips:** Recomendamos realizar la instalación através del binario de PostgreSQL

- RPM
- Source
- Binario (desde EnterpriseDB)

Documentación

En internet hay un amplio espectro de lugares donde encontrar documentación de PostgreSQL, pero el lugar oficial desde donde obtener documentación el RDBMS está en los siguientes enlace:

- Documentación Oficial 8.3
- Documentación Oficial 9.1
- Manuales otras versiones