Universidad de las Ciencias Informáticas

Facultad 3

Título: Administración y optimización de un Sistema de Base de Datos Descentralizado, en PostgreSQL.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Erich Mario Gómez Pérez

Ariel Torres Gálvez

Tutor: **Miguel Ángel Martínez Acosta**

Co-tutor: **Amado Alburquerque Arias**

Consultante: **Yunieski Fábregas Santos**

Mayo de 2008

DEDICATORIA

A mis padres, que siempre me han apoyado y que nunca han dejado de confiar en mí. A ellos que siempre están aquí, en las malas y en las buenas; por la educación y consejos sabios ofrecidos en los momentos oportunos.

A mi hermana que me ha soportado siempre, por todos los momentos vividos que han formado parte de las páginas de mi vida.

A mis amigos que siempre han estado cuando les he necesitado y nunca me han dado la espalda.

A mi amiga Yanelis, por su compañía y todas los momentos vividos juntos, gracias por tu amistad.

A mis familiares y a todos aquellos que de una forma u otra han estado presentes durante el desarrollo de mi vida.

Ariel Torres Gálvez

A mi familia, que siempre ha estado apoyándome en los momentos de felicidad y amargura, en especial a mi madre por quererme siempre.

A mis amigos del proyecto y profesores por ayudarme en todo momento.

A mi amiga Jenny, por tantas veces haberla molestado para que me revisara el documento, gracias por todo.

Erich Mario Gómez Pérez

RESUMEN

En el presente documento se plasman las ideas de una propuesta para la administración y optimización del gestor PostgreSQL para un sistema que utilice bases de datos descentralizadas.

Se detallan más específicamente las características del gestor y potencialidades que presenta que lo hacen ser uno de los más utilizados.

Se hace alusión a una estrategia de seguridad para mejorar las técnicas del uso del gestor, así como la implementación de políticas para la copia de seguridad y recuperación de la base de datos como única vía de solución ante los errores de hardware y software.

Con el objetivo de mantener la información actualizada en la base de datos central se propuso una estrategia de replicación con la utilización de la herramienta Slony-I.

PALABRAS CLAVE: Administración, optimización, seguridad, monitorización, replica, tunning, vacuum, resguardo, recuperación.

ÍNDICE

INTRODUCCIÓN 1
CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA 4
1.1 Clasificaciones de las base de datos 5
1.2 Base de datos distribuida 5
1.3 Modelo de datos 6
1.3.1 Modelo de datos Jerárquico 7
1.3.2 Modelo de datos de red 7
1.3.3 Modelo de datos relacional 7
1.3.4 Modelo de datos orientado a objetos 8
1.4 Sistemas Manejadores de base de datos (DBSM) 9
1.5 Diseño de base de datos 11
1.5.1 Diseño Conceptual 11
1.5.2 Diseño lógico 12
1.5.3 Diseño físico 12
1.6 Centro de datos 13
1.6.1 Diseño de un centro de datos 13
1.6.2 Componentes de un centro de datos 14
1.6.3 Definición de Rack 14
1.6.4 RAID (Redundant Array of Independent Disk) 15
1.6.5 Definición de Storage Área Network (SAN) 15
1.7 Tendencias y tecnologías actuales 17
1.7.1 Software libre 17
1.7.2 Software código fuente abierto 18
1.7.3 PostgreSQL 19
1.7.4 MySQL 20
1.7.5 Clientes para administrar PostgreSQL 21
1.7.5.1 PgAdmin3 21
1.7.5.2 PhpPgAdmin 22
1.7.6 Herramientas para la replicación 22
1.7.6.1 Slony-I 23
1.7.6.2 PgCluster 23
Conclusiones del Capítulo 24
CAPÍTULO II: PROPUESTA DE SEGURIDAD DE UN SISTEMA DE BASE DE DATOS DESCENTRALIZADO SOBRE POSTGRESQL 25
2.1 Arquitectura de PostgreSQL 25
2.1.1 Estructura física y lógica de PostgreSQL 27
2.2 Diseño de la arquitectura de la base de datos 30
2.2.1 Arquitectura centralizada 30
2.2.2 Arquitectura descentralizada 30
2.3 Lenguaje de programación 32
2.3.1 Disparadores (Triggers) 32
2.4 Seguridad de las bases de datos en los servidores 33

2.4.1 Selección del Hardware y Software 34
2.4.2 Principios básicos de la seguridad de la base de datos 38
2.4.3 Acceso restringido desde la red 39
2.4.3.1 Seguridad en la manipulación de los ficheros de PostgreSQL 39
2.4.3.2 Seguridad en el acceso a los clientes 40
2.4.3.2.1 Conexión local y remota 42
2.4.3.2.2 Conexión con SSL 45
2.4.4 Monitoreo de la base de datos 48
2.4.5 Diseño seguro de las bases de datos 50
2.4.5.1 Integridad 51
2.4.6 Definición de los privilegios 52
2.5 Realización de copias de seguridad y recuperación 53
2.5.1 Tipos de copia de seguridad 54
2.5.1.1 Copia de seguridad en frío 54
2.5.1.2 Copia de seguridad en caliente 54
2.5.1.3 Copia de seguridad del sistema operativo (S.O) 54
2.5.2 Copias de seguridad en PostgreSQL 55
2.5.2.1 Copia de seguridad de ficheros del S.O 55
2.5.2.2 Volcado SQL usando las herramientas PostgreSQL 56
2.5.2.3 Copia de seguridad en línea y recuperación en el punto (PITR) 58
2.5.3 Estrategia de copia de seguridad 61
2.6 Base de datos central 62
2.6.1 Centro de datos 63
2.6.2 Base de datos central 65
2.7 Replicación de datos 66
2.7.1 Características de la replicación 66
2.7.2 Tipos de réplica 67
2.7.3 Técnicas de replicación 68
2.7.4 Por qué usar replicación 68
2.7.4.1 Distribución de datos a otras ubicaciones 69
2.7.4.2 Consolidación de datos desde otras ubicaciones 69
2.7.4.3 Intercambio bidireccional de datos con otras ubicaciones 69
2.7.4.4 Algunas variantes o combinaciones de los anteriores 70
2.7.5 Conflictos a la hora de utilizar replicación 70
2.7.5.1 Conflicto de actualización 70
2.7.5.2 Conflicto de unidad 71
2.7.5.3 Conflicto de supresión 71
2.7.5.4 Conflicto de orden 72
2.7.6 Estrategia de réplica entre base de datos 72
Conclusiones del capítulo: 73
CAPÍTULO III: CONFIGURACIONES DE OPTIMIZACIÓN Y ADMINISTRACIÓN DE BASE DE DATOS IMPLANTADAS EN POSTGRESQL 74
3.1 Optimización de bases de datos en PostgreSQL 74
3.1.1 Ajustes del rendimiento del servidor 74
3.1.2 Afinamiento del servidor 75

3.1.2.1 Gestión del diario (Write Ahead Logging) 75
3.1.2.1.1 Parámetros de WAL (Write Ahead Logging) 76
3.1.2.2 Buffers de diario (cache WAL) 79
3.1.2.3 Cache de la base de datos 80
3.1.2.4 Acceso a los discos 82
3.1.3 Afinamiento de consultas 83
3.1.3.1 Uso de Índices 84
3.1.3.2 Cuando crear un índice 85
3.1.3.3 Tipos de índices 86
3.1.3.4 Rendimiento de los índices 87
3.1.3.5 Reconstrucción de índices 88
3.1.3.6 Ejecución de consultas 90
3.1.3.7 Tipos de rastreo que usa PostgreSQL 91
3.1.3.8 Operadores utilizables en consultas de PostgreSQL 91
3.1.3.9 Ajuste de rendimiento de consultas 93
3.1.4 Vacuum 93
3.1.4.1 Vacuum configurado desde el sistema operativo 94
3.1.4.2 Vacuum configurado desde sentencias SQL 95
3.1.4.3 AutoVacuum 96
3.1.4.3.1 Parámetros de configuración de AutoVacuum 96
3.1.5 Tuning en general 97
3.2 Administración mediante el cliente pgAdmin3. 98
3.2.1 Administración de seguridad 99
3.2.2 Copia de seguridad, Restauración y Mantenimiento 99
3.2.3 Monitorización de recursos 101
3.3 Propuesta de configuración del archivo postgresql.conf 102
Conclusiones del capítulo 105
CONCLUSIONES GENERALES 106
RECOMENDACIONES 107
BIBLIOGRAFÍA 108
ANEXO 111

ÍNDICE DE FIGURAS

Fig 2. 1 Arquitectura de PostgreSQL 25
Fig 2. 2 Interacción entre FrondEnd y BackEnd 26
Fig 2. 3 Tabla (área) de la base de datos. 52
Fig 2. 4 Replicación maestro-esclavo 67
Fig 2. 5 Replicación Multi-maestro. 68
Fig 3. 1 Interfaz gráfica para copia de seguridad 100
Fig 3. 2 Interfaz gráfica para restauración. 100
Fig 3. 3 Interfaz gráfica para Mantenimiento 101

ÍNDICE DE TABLAS

Tabla 2. 1 Vulnerabilidad del Sistema Operativo y % que no han sacado parche. 37
Tabla 2. 2 Formas de definir un acceso limitado 40
Tabla 2. 3 Tipos de integridad 51

INTRODUCCIÓN

La cantidad de innovaciones tecnológicas que ha habido en los últimos años ha promovido un cambio en la forma de observar a los sistemas de información, las aplicaciones computacionales y el manejo de los datos. (Alarcón, 2006)

Esto posibilita que actualmente las empresas extranjeras con buen desarrollo hayan optado por la utilización de los Sistemas de Planificación de Recursos, conocidos por sus siglas en inglés ERP (Enterprise Resourse Planning).

La empresa cubana no puede estar ajena a estos cambios, por lo que el país ha llevado a cabo la política de la utilización de estos sistemas.

En la actualidad en la Universidad de las Ciencias Informáticas (UCI) existe un proyecto ERP en marcha, con el objetivo de desarrollar los sistemas empresariales que las empresas cubanas puedan utilizar para lograr una mayor planificación y control de los recursos.

Definiciones de sistemas ERP:

"El término de ERP como anacrónico de Enterprise Resources Planning fue desarrollado a comienzos de los 90 por el Gartner Group's Computer-Integrated Manufacturing Service de Stanford, estos también son conocidos como Sistemas Empresariales, Sistemas Integrales de Empresas, o Sistemas Integrados de Gestión". (Ramírez, 2004)

Arquitectura de los sistemas ERP:

Los sistemas ERP están diseñados como piezas de un mecano. Cada uno de los módulos o aplicaciones tienen una función específica. Cada organización decide qué aplicación instalar de acuerdo con sus necesidades. (Ver Anexo 1) (Ramírez, 2004)

Lo fundamental en esta arquitectura es la utilización de una base de datos centralizada que

guarda la información que se maneja en todos los módulos.

La base de datos es de suma importancia para el buen funcionamiento de cada uno de los módulos. Sus principales funciones son:

- Gestionar la información.
- Realizar reportes con información necesaria para la empresa.
- A través de la Arquitectura de Bases de Datos Centralizada y Descentralizada se garantiza el flujo de información.
- Garantizar disponibilidad de la información los 7 días de la semana durante las 24 horas (24x7).
- Garantizar un total respaldo, integridad y preservación de la información.

En el proyecto ERP se decidió utilizar PostgreSQL como sistema gestor de base de datos por ser libre y cumplir con los requerimientos necesarios para gestionar bases de datos con gran cúmulo de información, donde sea necesario realizar reportes variados y con frecuencia.

Como PostgreSQL es un Sistema Gestor de Bases de Datos relativamente nuevo en el entorno de la Universidad, no hay muchos conocimientos sobre la administración y optimización del mismo, así como una guía que explique las mejores técnicas que ayuden a los administradores en este proceso. Esto implica que se presenten problemas en la seguridad, optimización y a la hora de realizar copias de seguridad y recuperación.

Según lo planteado anteriormente se puede definir:

Problema:

¿Cómo lograr un rendimiento óptimo en una base de datos descentralizada a partir de la configuración y optimización del Sistema Gestor de Base de Datos PostgreSQL?

Objetivo General:

Realizar una propuesta de técnicas de administración de base de datos descentralizadas utilizando como Gestor de Base de Datos PostgreSQL.

Objeto de estudio:

Sistemas de base de datos

Campo de acción:

Administración de sistemas de Base de Datos Descentralizada

Hipótesis:

Si se tiene una adecuada técnica de administración sobre el Sistema Gestor de Base de Datos PostgreSQL se obtendrá un rendimiento óptimo de la base de datos.

Tareas investigativas para dar cumplimiento del objetivo:

- Investigación sobre métodos de diseño de base de datos.
- Investigación sobre el diseño de base de datos descentralizado.
- Investigación sobre técnicas y herramientas de replicación en PostgreSQL.
- Configuración de parámetros para la seguridad de base de datos en PostgreSQL.
- Estrategia de copia de seguridad y recuperación de base de datos en PostgreSQL.
- Técnicas para la optimización del sistema gestor de base de datos PostgreSQL.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se hace referencia a un grupo de conceptos que servirán de base para el resto del documento. Un breve enfoque a los principales Sistemas Gestores de Base de datos libres, en especial PostgreSQL y las herramientas que se utilizan para su administración.

Los sistemas de base de datos juegan hoy en día un papel muy importante en todos los sistemas de información, casi todas las aplicaciones y soluciones a problemas hacen uso de las base de datos. (Alburquerque, 2007)

Todo apunta a que las tecnologías Open Source cambiarán radicalmente la industria del software en 3 o 4 años. Las bases de datos son parte de esta transformación. Poco a poco, pero sin descanso, el software Open Source está adquiriendo una robustez y una potencia suficiente como para plantar cara al software comercial.

Antes del surgimiento de las bases de datos, se utilizaban ficheros secuenciales como almacenes de datos. Estos daban un acceso de forma secuencial (para acceder a una posición, tenías que recorrer entero el fichero). Más tarde aparecieron los ficheros indexados, donde el acceso ya podía ser aleatorio (acceder de una vez a la posición deseada del fichero). Presentando dificultades tales como: Separación y aislamiento de los datos, duplicación de datos, dependencia de datos, formatos de ficheros incompatibles, consultas fijas y proliferación de programas de aplicación. Debido a estas dificultades es que surge la necesidad de mejorar la forma de almacenamiento de los datos, para su mejor trabajo. (Alburquerque, 2007)

Para desarrollar un software con calidad, debe realizarse un buen diseño de la base de datos y su correcta administración donde entra a jugar su rol el administrador de la base de datos quien se encarga de su protección:

1. Administrar la base de datos
2. Optimización de la base de datos
3. Mantener la seguridad de la base de datos

1.1 Clasificaciones de las base de datos

Las bases de datos pueden ser clasificadas de diferentes formas:

"Base de datos estáticas: Son aquellas de sólo lectura, donde no se puede cambiar la información, son utilizadas preferentemente para almacenar datos históricos". (Alburquerque, 2007)

"Base de datos Dinámicas: Son aquellas donde la información puede ser cambiada con el tiempo, permitiendo operaciones como actualización y edición de datos, además de las operaciones de consultas". (Alburquerque, 2007)

Según (Herrera, 2008), una base de datos no es más que un conjunto de archivos que son almacenados en un dispositivo de almacenamiento y que son vistos como una unidad que se le da un nombre y se le definen un conjunto de propiedades.

1.2 Base de datos distribuida

"Una Base de Datos Distribuida (BDD) es el conjunto de varias bases de datos lógicamente relacionados, las cuales se encuentran distribuidas entre diferentes sitios interconectados a través de diversos medios de comunicación, tales como cables de alta velocidad o líneas telefónicas. No comparten la memoria principal ni el reloj". (Fábregas, y otros, 2007)

"Un Sistema de Base de Datos Distribuida (SBDD) es un sistema en el cual múltiples localidades de bases de datos están ligados por un sistema de comunicación que participan en la ejecución de transacciones que accedan a datos de una o varias localidades. La diferencia principal entre los Sistemas de Bases de Datos Centralizados y Distribuidos es que, en los primeros, los datos residen en una sola localidad, mientras que, en los últimos, se encuentran en varias localidades".

(Fábregas, y otros, 2007)

"Un Sistema de Manejo de Bases de datos Distribuidas (SMBDD) es aquel que se encarga del manejo de la BDD y proporciona un mecanismo de acceso que hace que la distribución sea transparente a los usuarios. El término transparente significa que la aplicación trabajaría, desde un punto de vista lógico, como si un solo SMBD ejecutado en una sola máquina, administrara esos datos". (Fábregas, y otros, 2007)

El principal objetivo de las BDD es distribuir físicamente los datos a todos los usuarios, pero manteniendo la visión del sistema como un solo componente. Para lograr esto se desarrollan diferentes algoritmos como la replicación. (Fajardo, 2007)

A modo de resumen, un Sistema de Base de Datos distribuida (SBDD) es la integración de una base de datos distribuida y el sistema para manejar la misma.

1.3 Modelo de datos

El modelado de datos puede ser descrito como el proceso de información de una aplicación de uso intensivo. Construir una representación de la aplicación que capture las propiedades estáticas y dinámicas requeridas para dar soporte a los procesos deseados.

Modelos de datos clásicos:

- Modelo Jerárquico.
- Modelo de Red.
- Modelo Relacional.
- Modelo Orientado a Objetos.

Los modelos de datos son algoritmos y conceptos matemáticos, los modelos utilizados en las bases de datos son los anteriores mencionados.

1.3.1 Modelo de datos Jerárquico

Los modelos de datos jerárquicos almacenan la información de forma jerárquica, los datos se organizan en forma similar a un árbol donde un padre tiene muchos hijos.

Este modelo es especialmente útil en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos, permitiendo crear estructuras estables y de gran rendimiento.

El modelo de datos jerárquico presenta importantes inconvenientes, que provienen principalmente de su rigidez, por la falta de capacidad de las organizaciones jerárquicas para representar sin redundancias ciertas estructuras muy difundidas en la actualidad.

La poca flexibilidad de este modelo puede obligar a la introducción de redundancias cuando es preciso instrumentar, mediante el modelo jerárquico, situaciones del mundo real que no responden a una jerarquía. (Moraga, 2001)

1.3.2 Modelo de datos de red

El modelo de red permite que un nodo tenga varios nodos padres. Esto ofrece una solución eficiente al problema de la redundancia de datos. Es un modelo poco utilizado. (Fábregas, y otros, 2007)

1.3.3 Modelo de datos relacional

En 1970 E. F. Codd introdujo el modelo relacional, todos los datos están estructurados a nivel lógico como tablas formadas por filas y columnas, aunque a nivel físico pueden tener una estructura completamente distinta. Un punto fuerte del modelo relacional es la sencillez de su estructura lógica.

Algunas de las ventajas del modelo relacional son:

- Define un álgebra, llamada álgebra relacional a partir de la cual se realizan todas las

manipulaciones posibles sobre las relaciones, que se obtienen mediante el uso de operadores.

- Garantía de independencia de los datos.
- Conectividad garantizada con los lenguajes de programación estándar.
- Compatibilidad y estandarización.
- Favorece la normalización por ser más comprensible y aplicable.
- Garantiza la integridad referencial, así al eliminar un registro elimina todos los registros relacionados dependientes.
- Garantiza herramientas para evitar la duplicidad de registros, a través de campos claves o llaves.

Este modelo presenta desventajas cuando se compara con modelos más actuales, algunas de ellas son:

- Imposibilidad de representar conocimiento en forma de reglas.
- Inexistencia de mecanismos de herencia de propiedades.
- Incompatibilidad entre los tipos de estructuras de datos que se transfieren ó desadaptación de impedancia.
- Dificultad para gestionar datos no atómicos. (Márquez, 2002)

1.3.4 Modelo de datos orientado a objetos

Las bases de datos orientadas a objetos se crearon para tratar de satisfacer las necesidades de las nuevas aplicaciones. La orientación a objetos ofrece flexibilidad para manejar algunos de estos requisitos y no está limitada por los tipos de datos y los lenguajes de consulta de los sistemas de bases de datos tradicionales. Una característica clave de las Bases de Datos

Orientadas a Objetos (BDOO) es la potencia que proporcionan al diseñador al permitirle especificar tanto la estructura de objetos complejos, como las operaciones que se pueden aplicar sobre dichos objetos.

Otro motivo de la creación de estos modelos es el creciente uso de los lenguajes orientados a objetos.

Algunas de las ventajas son:

- Combinación de los procedimientos de una entidad con sus datos, permite modelar tipos de datos complejos y definir operaciones entre ellos .
- Soporta relaciones de mucho a mucho, primer modelo que lo permite.
- La cantidad de información que puede modelarse en una base datos orientada a objetos se incrementa, y es más fácil modelar esta información.
- Los Sistemas de Bases de Datos Orientados a Objetos son capaces de tener mayores capacidades de modelado por medio de la extensibilidad.
- En una Base de Datos Orientada a Objetos, el manejo de versiones está disponible para ayudar a modelar cambios diversos a los sistemas.
- La reutilización de clases juega un rol vital en el desarrollo y mantenimiento más rápido de aplicaciones. Las clases genéricas son potentes, pero más importante es que ellas pueden ser usadas nuevamente. (Márquez, 2001)

1.4 Sistemas Manejadores de base de datos (DBSM)

Es el conjunto de programas que se encargan de manejar la creación y los accesos a las base de datos. Está compuesto por:

- Data Definition Language (DDL): Lenguaje de definición de datos.
- Data Manipulation Language (DML): Lenguaje de manipulación de datos.

- Structured Query Language (SQL): Lenguaje de consulta estructurado.

Algunos de los objetivos de los manejadores de Bases de Datos son:

- Crear una colección integrada de datos disponibles a una amplia variedad de usuarios.
- Proveer calidad e integridad de los datos.
- Asegurar la privacidad a través de medidas de seguridad.
- Permitir un control centralizado de la base de datos, para una administración más eficiente.

Los DBMS más comunes son Oracle, SqlServer, Informix, Sysbase, MySQL, PostgreSql, Magic, Firebird.

A principios de los años 80, es creado por IBM el System R, desarrollado para probar la funcionalidad del modelo relacional, proporcionando una implementación de sus estructuras de datos y sus operaciones. Esto condujo a dos grandes desarrollos:

- El desarrollo de un lenguaje de consultas estructurado denominado SQL, que se ha convertido en el lenguaje estándar de los sistemas relacionales.
- La producción de varios SGBD relacionales durante los años ochenta, como DB2 y SLQ/DS de IBM, y ORACLE de ORACLE Corporation.

Existen cientos de DBSM relacionales, tanto para microordenadores como para sistemas multiusuario, aunque muchos no son completamente fieles al modelo relacional.

Otros sistemas relacionales multiusuario son INGRES de Computer Associates, Informix de Informix Software Inc. y Sybase de Sybase Inc.

Ejemplos de sistemas relacionales de microordenadores son Paradox y dBase IV de Borland, Access de Microsoft, FoxPro y R:base de Microrim.

Dada la creciente complejidad de las aplicaciones que requieren base de datos, han surgido dos

nuevos modelos: el Modelo de Datos Orientado a Objetos y el Modelo Relacional Extendido. Sin embargo, a diferencia de los modelos que los preceden, la composición de estos modelos no está clara. Esta evolución representa la tercera generación de los DBMS. (Márquez, 2001)

1.5 Diseño de base de datos

"El diseño de una base de datos es un proceso complejo. La complejidad se controla si se divide el problema en subproblemas y se resuelven independientemente. El diseño de una base de datos se descompone en diseño conceptual, lógico y físico". (Márquez, 2001)

1.5.1 Diseño Conceptual

Un esquema conceptual es una descripción de alto nivel de la estructura de la base de datos, independientemente del Sistema Gestor de Base de Datos (SGBD) que se vaya a utilizar para manipularla.

Un modelo conceptual es un lenguaje que se utiliza para describir esquemas conceptuales.

El objetivo del diseño conceptual es describir el contenido de información de la base de datos y no las estructuras de almacenamiento que se necesitarán para manejar esta información.

Los modelos conceptuales deben ser buenas herramientas para representar la realidad, por lo que deben poseer las siguientes cualidades:

- **Expresividad:** Suficientes conceptos para expresar perfectamente la realidad.
- **Simplicidad:** Deben ser simples para que los esquemas sean fáciles de entender.
- **Minimalidad:** Cada concepto debe tener un significado distinto.
- **Formalidad:** Todos los conceptos deben tener una interpretación única, precisa y bien definida. (Márquez, 2001)

1.5.2 Diseño lógico

El diseño lógico parte del esquema conceptual y da como resultado un esquema lógico.

Un esquema lógico es una descripción de la estructura de la base de datos en términos de las estructuras de datos que puede procesar un tipo de SGBD.

Un modelo lógico es un lenguaje usado para especificar esquemas lógicos (Modelo Relacional, Modelo de Red, etc.)

El diseño lógico depende del tipo de SGBD que se vaya a utilizar, no depende del producto concreto. (Márquez, 2001)

El esquema lógico es una fuente de información para el Diseño Físico. Además, juega un papel importante durante la etapa de mantenimiento del sistema, pues permite que los futuros cambios que se realicen sobre los programas de aplicación o sobre los datos, se representen correctamente en la base de datos. (Fábregas, et al., 2007)

1.5.3 Diseño físico

El diseño físico parte del esquema lógico y da como resultado un esquema físico.

Un esquema físico es una descripción de la implementación de una base de datos en memoria secundaria: las estructuras de almacenamiento y los métodos utilizados para tener un acceso eficiente a los datos. Por ello, el diseño físico depende del SGBD concreto y el esquema físico se expresa mediante su lenguaje de definición de datos.

Para el diseño de esta etapa se tiene que haber decidido qué Sistema Gestor de Base de Datos (SGBD) se va a utilizar, porque este diseño se tiene que adaptar a él.

1.6 Centro de datos

Es un repositorio centralizado, ya sea físico o virtual, para el almacenamiento, gestión y difusión de información organizada de un determinado negocio. (Godhino, 2006)

1.6.1 Diseño de un centro de datos

Proporciona una concentración de servicios, como hospedaje de servidores, almacenamiento de datos, monitoreo permanente y otros, con todos los recursos necesarios para el procesamiento de información por parte de una organización.

Cuenta con personal técnico, servicio de vigilancia y control de acceso. Minimiza cualquier riesgo operacional.

Ventajas:

- Seguridad física (acceso restringido, sistemas de identificación biométrica o sistemas de detección temprana de incendios), redundancia de servidores, paneles eléctricos, sistemas de ventilación y software antivirus.
- Software antivirus, anti spyware y cortafuegos de alto desempeño.
- Enlaces entre el Centro de datos y los clientes con una serie de sistemas de seguridad de redes.
- Se dispone de alternativas de incremento de capacidades en forma progresiva.
- Posibilidad de evolución hacia otras plataformas si es necesario en el futuro (cambio de la base de datos, incorporación de un gestor de contenidos.)
- No hay riesgos de tecnológica obsoleta, porque la infraestructura del servicio la mantiene y actualiza.

- Posibilidad de acceder a actualizaciones y mejoras en los sistemas en forma expedita.
- Gestionar de óptima manera las redes, especialmente el ancho de banda.
- Máxima garantía de continuidad de la actividad frente a imprevistos.

Los centros de datos almacenan la información de diferentes aplicaciones entre ellas están: ERP, aplicaciones de comercio, aplicaciones Business to Business (B2B).

1.6.2 Componentes de un centro de datos

Es necesario saber el número de usuarios, aplicaciones, plataformas y unidades de rack requeridas.

Los Centros de Datos están compuestos de un sistema de comunicaciones de red de alta velocidad y alta demanda capaz de manejar el tráfico para SAN (Storage Area Networks), NAS (Network Attached Storage), granja de servidores de archivos/aplicaciones/web, y otros componentes localizados en ambiente controlado. El control de ambiente se relaciona a la humedad, inundación, electricidad, temperatura, control de fuego, y por supuesto, acceso físico. Las comunicaciones dentro y fuera del centro de datos se proveen por enlaces WAN, CAN/MAN y LAN en una variedad de configuraciones dependiendo de las necesidades particulares de cada centro.

Un centro de datos diseñado apropiadamente proporcionará disponibilidad, accesibilidad, escalabilidad, y confiabilidad 24 horas al día, 7 días a la semana, 365 días al año.

1.6.3 Definición de Rack

Los racks son muy útiles en un centro de datos, donde el espacio es escaso y se necesita alojar un gran número de dispositivos.

Estos dispositivos suelen ser:

- Servidores cuya carcasa ha sido diseñada para adaptarse al bastidor. Existen servidores de 1Unidad (U).
- 2U y 4U, y recientemente, se han popularizado los servidores blade que permiten compactar más de veinte servidores en una altura de 4U, compartiendo fuentes de alimentación y cableado.
- Conmutadores y enrutadores de comunicaciones.
- Cortafuegos.
- Sistemas de audio.

1.6.4 RAID (Redundant Array of Independent Disk)

Agrupación de dispositivos de discos estándar o particiones de discos, que junto a un controlador RAID o un software que los implemente, permita crear un sistema de almacenamiento que actúa como un solo disco y puede alcanzar un rendimiento superior al que se puede alcanzar con dispositivos individuales y garantizar niveles de seguridad de la información almacenada ante fallos de los discos. (Ver Anexo 2)

1.6.5 Definición de Storage Área Network (SAN)

La sigla SAN corresponde al concepto de red de tarea de almacenamiento (Storage Area Network). Se trata de una red que conecta servidores, "arrays" de discos y equipos de backup.

Cuando está basada en tecnología "fibre channel", permite un mejor rendimiento en la administración de los datos.

La SAN conecta múltiples servidores a un conjunto centralizado de almacenamiento en discos,

por lo que tiene un rol clave en el sistema de administración, que ahorra tiempo y recursos al no necesitar trabajar sobre cada equipo. Este sistema permite tratar todo el Storage de una empresa como un solo recurso, haciendo más simple su control y correr software sobre él. (Ver Anexo 3) (Camacho)

Ventajas que ofrece SAN

- **SAN es una red de almacenamiento de altas prestaciones:** Su función es centralizar el almacenamiento de los ficheros en una red de alta velocidad y máxima seguridad. Es una solución global que comparte todos los recursos de almacenamiento en la compañía.

- **Gran ancho de banda:** Ancho de banda actual de hasta 200 Mbyte/segundo con doble adaptador Fibre Channel. La tecnología Fibre Channel permite un incremento del ancho de banda efectivo de entre 2,5 y 10 veces la obtenida sobre una plataforma SCSI. Si en la actualidad el ancho de banda es de 1 Gb/s, el nuevo estándar especifica anchos de entre 2 a 4 Gb/s.

- **Centralización del backup/ Copia de seguridad independiente de la LAN:** El sistema de copia se conecta a la SAN, por lo que es posible realizar el backup on-line, sin afectar al trabajo de los usuarios y ejecutándose en un tiempo mínimo con un impacto prácticamente cero en el servidor.

- **Alta escalabilidad y larga distancia entre nodos de la red:** Dependiendo de la topología SAN utilizada se puede interconectar hasta 126 nodos por bucle en "Arbitrated Loop", a distancias entre ellos de 30 metros en el caso de utilizar cable de cobre y de hasta 10 Km si se emplea cable de fibra óptica. Escalabilidad es impensable en entornos SCSI y permite planificar una red SAN simplemente añadiendo dispositivos a medida que las necesidades lo requieren.

- **Alta disponibilidad:** Fibre Channel incluye soporte de conexión dual loop. Con ello se proporciona un camino alternativo a la señal en el caso de que un cable falle o sea

accidentalmente desconectado. De nada sirve un sistema RAID tolerante al fallo si el único camino para acceder a él se interrumpe, bien sea por el fallo de un componente o por la desconexión accidental de un cable.

- **Fácil administración/ Gestión centralizada:** Gestión global de todos y cada uno de los dispositivos de almacenamiento que forman parte de la red. Ello supone desde el control de conexión o desconexión de un puerto de forma remota, hasta el control de nodos o bucles, pasando por el control del estado de las cabinas, dispositivos de almacenamiento, hubs y switches, etc. Controlar y optimizar el tráfico de toda la red, diagnosticar de manera más rápida y eficiente problemas, evitando caídas del sistema con el ahorro de costes que ello supone. Además, en la SAN la centralización desde una única consola permite una gestión más eficiente de los sistemas de almacenamiento.

1.7 Tendencias y tecnologías actuales

1.7.1 Software libre

En 1984, Richard Stallman comenzó a trabajar en el proyecto GNU (es un acrónimo recursivo para "GNU No es Unix"), creando la Fundación de software Libre (FSF, Free Software Foundation) en 1985.

Software Libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De modo más preciso, se refiere a cuatro libertades de los usuarios del software:

- La libertad de usar el programa, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades (libertad 1). El acceso al código fuente es una condición previa para esto.
- La libertad de distribuir copias, con lo que puedes ayudar a tu vecino (libertad 2).
- La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie. (libertad 3). El acceso al código fuente es un requisito

previo para esto.

El software libre suele estar gratuito en Internet, a precio del coste de la distribución a través de otros medios, sin embargo no es obligatorio que sea así, aunque conserve su carácter libre puede ser comercializado.

1.7.2 Software código fuente abierto

El termino de "Software de Código Fuente Abierto" (Open Source), esto significa que se puede mirar el código fuente, es un criterio más pobre que "Software Libre". "Software de Código Fuente Abierto" incluye software libre, pero también incluye programas semi-libres. (Stallman, 2008)

La etiqueta "Open Source" nació en una reunión celebrada el 3 de febrero de 1998 en Palo Alto, California. En la cual participaron Todd Anderson, Chris Peterson, John Hall y Larry Augustin, Sam Ockman, Eric S. Raymond y Bruce Perens quien fuera el creador del primer borrador de esa definición.

La idea esencial del Open Source, es ofrecer programas con acceso al código fuente, integrando una serie de conceptos:

- **Flexibilidad:** Si el código fuente está disponible, los desarrolladores pueden modificar los programas a su antojo. Además, se produce un flujo constante de ideas que mejora la calidad de los programas.
- **Fiabilidad y Seguridad:** Con varios programadores a la vez escrutando el mismo trabajo, los errores se detectan y corrigen antes, por lo que el producto resultante posee mayor fiabilidad y eficacia con respecto al comercial.
- **Rapidez de desarrollo:** Las actualizaciones y ajustes se realizan a través de una comunicación constante vía internet.
- **Relación con el usuario:** El programador se acerca mucho más a la necesidad real de su cliente, y puede crear un producto específico para él.

1.7.3 PostgreSQL

PostgreSQL es un potente Sistema de Base de Datos Relacional libre (Open Source, su código fuente está disponible) liberado bajo licencia Berkeley software Distribución (BSD). Desarrollado en la Universidad de California, en el departamento de ciencias de la computación de Berkeley.

Posee más de 15 años de activo desarrollo y arquitectura probada que se ha ganado una muy buena reputación por su confiabilidad e integridad de datos. Funciona en todos los sistemas operativos importantes, incluyendo Linux, UNIX (AIX, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows.

El nombre como tal de PostgreSQL surge en el año 1996 para establecer una relación entre el nombre original PostgreSQL y las versiones más recientes con capacidades SQL.

Principales características

- Soporta casi toda la sintaxis SQL tiene soporte total para foreign keys, joins, views, triggers, y stored procedures (en múltiples lenguajes).
- Cliente/Servidor: PostgreSQL usa una arquitectura proceso-por-usuario cliente/servidor. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL.
- 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. además de en C, C++ y, Java.
- Interfaces con lenguajes de programación: La flexibilidad del API de PostgreSQL ha permitido a los vendedores proporcionar soporte al desarrollo fácilmente para el RDBMS PostgreSQL. Estas Interfaces incluyen Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, Pike, etc.

- Herencia de tablas.
- Incluye la mayoría de los tipos de datos SQL92 y SQL99 (INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, y TIM ESTAMP), soporta almacenamiento de objetos grandes binarios, además de tipos de datos y operaciones geométricas.
- Puntos de recuperación a un momento dado, tablespaces, replicación asincrónica, transacciones jerarquizadas (savepoints), copia de seguridad en línea.
- Un sofisticado analizador/optimizador de consultas.
- Soporta juegos de caracteres internacionales, codificación de caracteres multibyte. (Alarcón, 2006)

Ventajas

- Máximo tamaño de base de datos ilimitado.
- Máximo tamaño de tabla 32 TB.
- Máximo tamaño de tupla 1.6 TB.
- Máximo tamaño de campo 1 GB.
- Máximo tuplas por tabla ilimitado.
- Máximo columnas por tabla 250 - 1600 dependiendo de los tipos de columnas.
- Máximo de índices por tabla ilimitado. (Alarcón, 2006) (AB, 1995-2008)

1.7.4 MySQL

Es un Sistema de Gestión de Bases de Datos Relacional, fue creada por la empresa sueca MySQL AB. Surgió alrededor de la década del 90, creado por Michael Windenis.

El principal objetivo de diseño de este sistema gestor de base de datos es la velocidad, por tal motivo se sacrificaron algunas características esenciales que presentan otros SGBD más serios, su consumo de recursos como la CPU y memoria es muy baja. Está bajo la licencia Publica General de GNU (GNU GPL) a partir de la versión 3.23.19.

Ventajas:

- Mayor velocidad tanto al conectar con el servidor, como a la hora de realizar consultas.
- Mejores utilidades de administración (backup, recuperación de errores, etc).
- No suele perder información ni corromper los datos cuando hay errores.
- Mejor integración con PHP.
- No hay límites en el tamaño de los registros.
- Mejor control de acceso.
- MySQL se comporta mejor que PostgreSQL a la hora de modificar o añadir campos a una tabla "en caliente".

Inconvenientes:

- No soporta transacciones, "roll-backs" ni subselects.
- No considera las claves ajenas. Ignora la integridad referencial, dejándola en manos del programador de la aplicación.

Dado las características, ventajas y desventajas de estos dos Sistemas Gestores de Base de Datos se hace uso de PostgreSQL teniendo en cuenta los lineamientos de la Universidad de las Ciencias informáticas de la migración a Software Libre y por el gran rendimiento y estabilidad a la hora de manejar grandes cantidades de datos. (AB, 1995-2008)

1.7.5 Clientes para administrar PostgreSQL

1.7.5.1 PgAdmin3

PgAdmin3 es una herramienta cliente para la administración de PostgreSQL, desarrollado por una comunidad de expertos de todo el mundo, está disponible en más de una docena de idiomas. Es Software Libre publicado bajo la licencia artística.

PgAdmin3 es el más popular y de las características de código abierto de administración y desarrollo de plataforma de PostgreSQL. PgAdmin3 está diseñado para responder las necesidades de todos los usuarios.

La interfaz gráfica de pgAdmin3 soporta todas las características y hace fácil la administración. La aplicación también incluye un editor de sintaxis SQL, un servidor, editor de código, un SQL / lote / shell de la programación de agente de empleo, el apoyo a la replicación con Slony-I. (pgadmin)

1.7.5.2 PhpPgAdmin

Es una herramienta de administración PostgreSQL, basada en una interface Web, entre sus principales características ofrece la posibilidad de administrar varios servidores, soporte a múltiples versiones de PostgreSQL, administración de usuarios, grupos, bases de datos, esquemas, etc. Manipulación sencilla de datos, exportar los datos a diferentes formatos, importar sentencias SQL y mucho más. (phpPgAdmin, 2008)

De estos dos clientes se hace uso de pgAdmin3 para realizar la propuesta de administración debido a su fácil administración de la base de datos mediante una interfaz gráfica sencilla, también porque el mismo es el más usado en el mundo para la administración PostgreSQL.

1.7.6 Herramientas para la replicación

Existen en el mundo múltiples soluciones que permiten llevar a cabo la réplica de datos enfocadas al gestor de base de datos utilizado en cuestión. Pueden citarse entre ellos Postgres-R, PgReplicator, Usogres, ErServer, Slony-I, PgCluster y CyberCluster.

1.7.6.1 Slony-I

Slony-I es una herramienta que implementa una replicación maestro-esclavo. Se usa para la replicación en cascada, por ejemplo un nodo puede alimentar otro nodo que alimenta a otro nodo, también presenta tolerancia a fallos. (Browne, 2004-2006)

Es un sistema diseñado para su uso en centros de datos y para hacer copias de seguridad. Slony es el plural de elefante en el idioma ruso.

Slony-I permite indicar que tablas se van a replicar de un servidor a otro, implementa la replica sincrónica, usando disparadores para determinar las actualizaciones de las tablas.

Slony-I actualiza los datos exactamente igual al origen de los mismos, presenta la desventaja que no se pueden actualizar los datos cuando están ocurriendo cambios en ellos. (Fajardo, 2007)

1.7.6.2 PgCluster

PgCluster es el sistema de replicación síncrona de la multi-maestro de composición para PostgreSQL. Consta de tres tipos de servidores, un equilibrador de carga, el Clúster BD y un servidor de replicación. (Ver Anexo 4)

Clúster PP es el servidor que recibe la consulta a la base de datos de un usuario y, de hecho, los procesos. Clúster PP aplica el parche para repeticiones al back-end de PostgreSQL. Un acceso de carga se puede distribuir por el aumento del número de agrupaciones PP.

Clúster PP será ejecutado de inmediato, cuando la consulta de referencia a los datos que se reciba. La otra consulta se pasa al servidor thereplication y requiere un duplicado.

La replicación servidor envía la consulta de la base de datos de Clúster para cada grupo de PP. La replicación del servidor guarda la orden recibida y envía una consulta a cada uno de los grupos PP.

El control de la replicación del servidor es el problema de la Categoría PP en el momento de la comunicación con la base de datos de Clúster. Cuando se detecta un problema, la replicación del servidor separa de la Categoría PP posteriores repeticiones.

El balance de carga de servidor recibe la consulta del cliente, y envía una consulta a la base de datos del clúster con el más bajo índice de carga. El factor de carga a la Categoría PP se calcula con el número de sesiones. (PgCluster)

Tiene dos funciones principales:

- **Repartir la carga:** Se distribuye la demanda de carga de las sesiones, es muy útil en aplicaciones Web donde existe gran demanda.
- **Alta disponibilidad:** Al ocurrir un fallo en el Clúster BD, el equilibrador de carga y el servidor de replicación separa el Clúster BD dañado y se continúa prestando servicios. El Clúster BD puede ser reparado sin detener el servicio. Posteriormente los datos se copian automáticamente a la base de datos reparada. (Ver Anexo 5)

Conclusiones del Capítulo

En este capítulo se han abordado conceptos y definiciones que son importantes para la comprensión del proceso de diseño de bases de datos. Se plantea el Modelo de Datos Relacional como un modelo simple y eficaz para la manipulación de los datos. Sistema Gestor de Base de Datos PostgreSQL por su condición de software libre y responder a las necesidades de alta tecnología y bajo costo.

Se hace necesaria la creación de una propuesta para administrar las bases de datos para un sistema ERP cubano.

Se realiza una descripción de algunas de las tendencias del software libre, pues el país se ha trazado una política de implantarlo a nivel nacional.

CAPÍTULO II: PROPUESTA DE SEGURIDAD DE UN SISTEMA DE BASE DE DATOS DESCENTRALIZADO SOBRE POSTGRESQL.

Introducción

En el presente capítulo se hará referencia a una serie de elementos importantes para mantener una buena seguridad en Sistemas de Base de Datos Descentralizados, así como propuestas de copia de seguridad y replicación.

2.1 Arquitectura de PostgreSQL

PostgreSQL. funciona con una arquitectura Cliente/Servidor, un proceso servidor (postmaster) y una serie de aplicaciones cliente que realizan solicitudes de acciones contra la base de datos. Por cada una de estas aplicaciones cliente, el proceso postmaster crea un proceso postgres. (Ver figura 2.1)

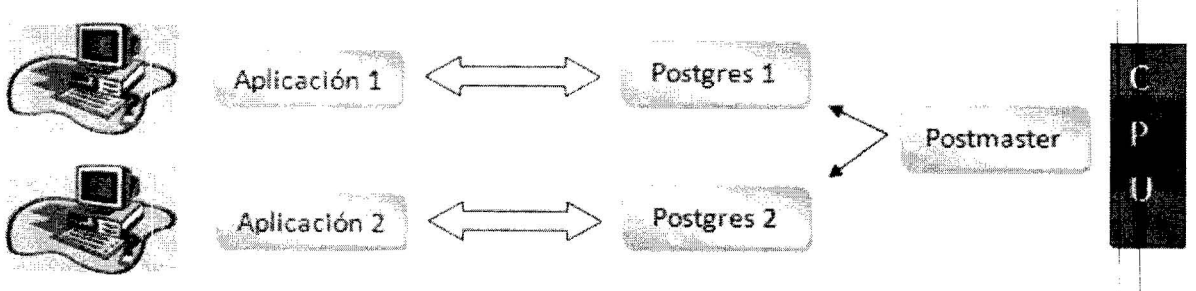


Fig 2. 1 Arquitectura de PostgreSQL

Se ejecutan una serie de aplicaciones cliente llamadas (FrontEnd) y una serie de procesos en el servidor llamados (BackEnd), de modo que la interacción entre un programa que se ejecuta en el cliente, por ejemplo pgadmin3, una aplicación de PHP y el servidor de base de datos quedaría como se muestra en la figura 2.2.

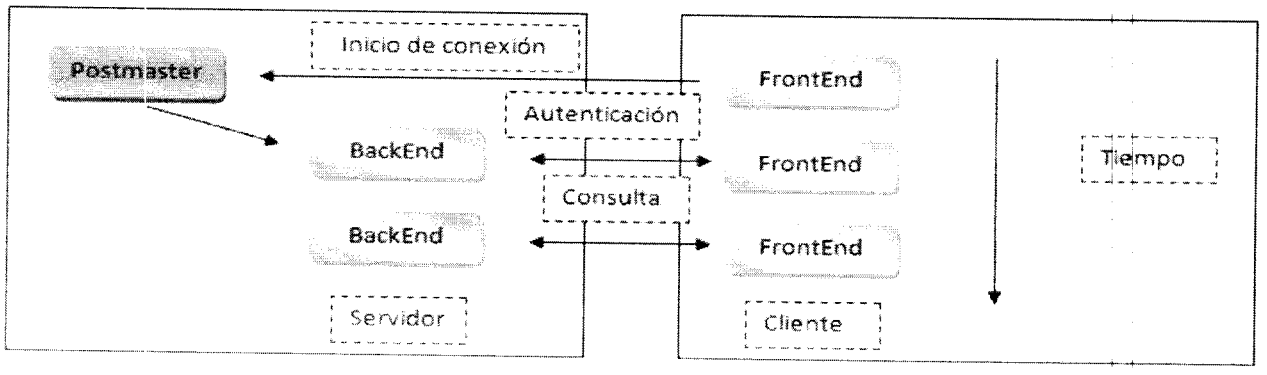


Fig 2. 2 Interacción entre FrondEnd y BackEnd

El proceso Postmaster:

- Es el proceso inicial.
- Gestiona los accesos multiusuario y multiconexión.
- Levanta la memoria compartida.
- Está al tanto de solicitudes de nuevas conexiones.
- Lanza procesos de atención de demanda, realizando las operaciones sobre la base de datos a solicitud de los clientes.
- Realiza el enlazado con los archivos de datos, gestionando estos ficheros, donde los ficheros pueden pertenecer a varias bases de datos.

La comunicación entre BackEnd/FrontEnd se realiza mediante TCP/IP, normalmente por el puerto 5432, creando un socket (/tmp/.s.PGSQL.5432).

La memoria compartida:

- Gestiona los recursos entre procesos BackEnd.
- Gestiona la caché del disco.
- Maneja otras estructuras internas.

2.1.1 Estructura física y lógica de PostgreSQL

En un servidor se pueden crear uno o varios clústeres de base de datos en los cuales la estructura física de los mismos se crea usando el comando *initdb,* cuando la instalación se realiza a través de paquetes se crea automáticamente un clúster con tres bases de datos "template0", "template1" y "postgres".

Estructura de ficheros

- **postgresql.conf:** Fichero de configuración principal, contiene la asignación a los parámetros que configuran el funcionamiento del servidor.
- **pg_hba.conf:** Fichero de configuración de la autenticación de los clientes, usuarios y del acceso a las bases de datos del clúster.
- **pg_ident.conf:** Fichero accesorio al anterior, determina como se realiza la autenticación ident que contiene la correspondencia entre usuarios del Sistema Operativo y de PostgreSQL.
- **PG_VERSION:** Fichero de texto con la versión de software Postgres que crea el clúster.

Otros ficheros:

- **postmaster.pid:** Se crea cuando el postmaster arranca, contiene el PID del proceso postmaster.
- **postmaster.opts:** Contiene las opciones con las que se ha iniciado el postmaster.
- **recovery.conf, recovery.done:** Si se quiere o se ha hecho una recuperación.

Estructura de directorios

- **base:** En ella se encuentran las plantillas y las bases de datos, contiene un directorio por cada base de datos, dentro hay un fichero por cada tabla o índice de una base de datos.
    - template0 (1): Contiene las definiciones de las tablas del sistema, vistas, funciones

y tipos estándar. Nunca se debe modificar ni intentar conectarse a él, existe por si template1 se corrompe.

- template1 (N): Base de datos plantilla para crear nuevas bases de datos, se puede modificar su estructura, añadiendo tablas, índices, funciones, etc.

➢ **global:** Posee tablas e índices del catálogo comunes a todas las bases de datos.

- catálogo compartido: *pg_shadow* (usuarios), pg_database, etc.
- *pgstat.stat:* fichero usado por el monitor de estadísticas.
- *pg_control:* fichero con parámetros del clúster, algunos inmutables (establecidos en la creación del clúster) y otros variables (establecidos en la puesta en marcha).

- **pg_log:** Ficheros de seguimiento del servidor. Se crea en la versión de Windows, en la de Linux, se debe indicar al arrancar el postmaster en qué fichero se hace el seguimiento.

- **pg_xlog:** Ficheros de diario del servidor (WAL).
  - Contiene los diarios de escritura adelantada, para usarlos en las recuperaciones.
  - Implementan un conjunto de segmentos (ficheros) de un tamaño de 16Mb y divididos en páginas de 8Kb.
  - Inicialmente se crea un fichero, y luego el sistema va creando más según las necesidades.

➢ **pg_clog:** Ficheros de diario para las transacciones (estado de cada transacción).

- Contiene los ficheros de confirmación.
- Un diario de confirmación refleja el estado de cada transacción: confirmada, en progreso o abortada.

➢ **pg_multixact:** Contiene datos sobre el estado multi-transaccional, usado para los bloqueos compartidos de filas.

➢ **pg_twophase:** Ficheros de estado para las transacciones preparadas.

- **pg_subtrans:** Para realizar los "savepoints" en medio de transacciones.
- **pg_tblspc:** Información sobre los tablespaces. En Linux/Unix contiene enlaces a los directorios donde se crean los tablespaces y hay que controlar, en caso de cambios de ficheros, que estén correctos.

Para entender la estructura lógica de PostgreSQL se debe saber qué es un clúster:

Es un conjunto de base de datos, el cual contienen objetos que se pueden guardar en distintos tablespaces, y un conjunto de usuarios que se conectan al clúster. A su vez las bases de datos están formadas por esquemas que tienen un usuario propietario. El esquema es donde se crean los objetos (tablas, índices, procedimientos, vistas, etc.).

Los tres elementos principales a nivel lógico son:

- **Bases de datos:** Agrupaciones de esquemas. Por defecto, siempre hay tres bases de datos creadas, *template0*, *template1* y *postgres*.
- **Tablespace:** Espacio reservado en disco para guardar las tablas de la base de datos.
- **Roles:** Engloba el concepto de usuarios (roles de login) y grupos de permisos (roles de grupo). Hasta la versión 8 de PostgreSQL no se podían anidar roles, ahora sí. Por defecto, si al crear el clúster no se ha indicado otro usuario, se crea el usuario postgres como superusuario.

Un aspecto a tener en cuenta es que todos los usuarios se pueden conectar a cualquiera de las bases de datos. (Ver Anexo 6)

Las bases de datos son independientes una de las otras, en un servidor pueden haber dos proyectos separados, se crearía una base de datos para cada uno. Si existen dos proyectos con recursos comunes se pueden crear distintos esquemas en una sola base de datos.

2.2 Diseño de la arquitectura de la base de datos

A partir de la necesidad de la empresa cubana de automatizar todos sus procesos, y su estructura organizativa, se proponen dos posibles diseños de la arquitectura.

2.2.1 Arquitectura centralizada

Se basa en la existencia de una máquina servidora que almacena los datos y las aplicaciones que los procesan. Los clientes se comportan como terminales y sólo sirven para introducir datos desde teclado.

Ventajas:

- Gran nivel de seguridad de la base de datos.
- Fácil de administrar.
- Eliminación de los mecanismos de replicación.
- Menores costos de Software y Hardware.

Inconvenientes:

- Alto coste.
- Máquina servidora muy cargada.

2.2.2 Arquitectura descentralizada

La arquitectura descentralizada nació como una respuesta a los altos costos y la baja flexibilidad que representaba la arquitectura centralizada. Se basa en un servidor central en el cual está la base de datos central, a la cual se conectan varios servidores que contienen las bases de datos locales.

La interacción entre la base de datos central y las locales se realiza a través de la replicación, más adelante se abordará sobre este tema.

Ventajas:

- Fiabilidad y disponibilidad, producto de que la falla de conectividad hacia una localidad no produce la desactivación del sistema local.
- Disminución de los niveles de procesamiento en la base de datos central.
- Funcionamiento offline de los sistemas locales: Si se produce el fallo de algún componente del servidor central, no demanda la paralización de los sitios de información locales.
- Disminución considerable de los tiempos de respuesta al cliente como resultado de una adecuada latencia de red local, y por consecuencia de una mejora ostensible en los tiempos de procesamiento de consultas.

Como resultado del estudio de las dos variantes de arquitectura y de los sistemas ERP a nivel mundial, para el caso de Cuba se propone hacer uso de la arquitectura descentralizada, a qué se debe esto, primero a las condiciones que presenta la empresa cubana, principalmente las condiciones de conectividad y presupuesto con que cuentan. (Ver Anexo 7)

La empresa cubana está formada por una entidad central ubicada en Ciudad de la Habana y entidades afiliadas en todas o casi todas las provincias, las cuales reportan todas las incidencias a la entidad central.

Se propone la creación de un centro de datos en la entidad central, un servidor en cada una de las provincias del país y otro en cada una de las entidades afiliadas de la empresa.

La información de cada una de las entidades afiliadas se replicará a cada uno de los servidores provinciales y estos replicarán al centro de datos.

Las ventajas que presenta esta arquitectura es que en caso de pérdida de conexión las entidades afiliadas seguirían trabajando con su servidor local y posteriormente de su establecimiento se replicaría la información, con esto se garantiza que no exista pérdida de datos, también aumentaría la seguridad en cada uno de los niveles.

2.3 Lenguaje de programación

PL/pgSQL (Procedural Language/PostgreSQL Structured Query Language) es un lenguaje previsto para el gestor de base de datos PostgreSQL. Permite ejecutar comandos SQL utilizando un lenguaje de sentencias imperativas y uso de funciones, esto da un mayor control automático que las funciones SQL básicas.

Ventajas de PL/pgSQL:

Mayor rendimiento: Con PL/pgSQL puede agrupar un grupo de computaciones y una serie de consultas dentro del servidor de base de datos, teniendo así la potencia de un lenguaje procedural y la sencillez de uso del SQL, ahorrando una gran cantidad de tiempo porque no tiene la sobrecarga de una comunicación cliente/servidor. Esto puede redundar en un considerable aumento del rendimiento.

Soporte SQL: PL/pgSQL añade a la potencia de un lenguaje procedural, la flexibilidad y sencillez del SQL. Con PL/pgSQL puede usar todos los tipos de datos, columnas, operadores y funciones de SQL.
Portabilidad: Las funciones PL/pgSQL corren dentro de PostgreSQL, estas funciones funcionarán en cualquier plataforma donde PostgreSQL corra. Así podrá rehusar el código y reducir costes de desarrollo. (Pachón)

2.3.1 Disparadores (Triggers)

Otra de las ventajas que presenta PostgreSQL es la utilización de disparadores.

Un trigger es un bloque PL/pgSQL asociado a una tabla, que se ejecuta como consecuencia de una determinada instrucción SQL (una operación DML: INSERT, UPDATE o DELETE) sobre dicha tabla. (Herrarte, 2005)

Son objetos relacionados con tablas y almacenados en la base de datos, que se ejecutan o se

muestran cuando sucede algún evento sobre sus tablas asociadas.

Los eventos pueden ser las sentencias INSERT, DELETE, UPDATE que modifican los datos de una tabla. Los triggers se pueden ejecutar antes (BEFORE) y/o después (AFTER) de que sean modificados los datos. (Bernadí)

2.4 Seguridad de las bases de datos en los servidores

Lo primero que se debe definir es qué es un servidor de base de datos: Un servidor de base de datos es un sistema bajo arquitectura cliente/servidor que proporciona servicios de gestión, administración y protección de la información (datos) a través de conexiones de red, gobernadas por protocolos definidos y a los que acceden los usuarios de modo concurrente a través de aplicaciones clientes (bien sean herramientas del propio sistema como aplicaciones de terceros).

Dichos servidores solucionan los problemas de las empresas al manejar grandes volúmenes de información de una manera estable, fiable, coherente y segura en un entorno heterogéneo de trabajo y de necesidades de información. (Menéndez, 2000)

En servidores de bases de datos se pueden mencionar 4 niveles básicos de seguridad:

- Seguridad de acceso al sistema.
- Seguridad a nivel de objetos de datos.
- Seguridad a nivel de datos.
- Seguridad en cuanto a protección de los almacenamientos físicos de los datos.

La seguridad de acceso: Se implementa de dos formas posible, a nivel de sistema operativo, en cuyo caso el Sistema Gestor de Base de Datos (SGBD) se apoya en la seguridad de entrada al sistema operativo para comprobar la validez del acceso a los datos almacenados; o bien lo que se llamará modo mixto, en el cual la seguridad de entrada a la información la llevará a cabo el propio servidor de datos a partir de la definición de cuentas de usuario

La seguridad a nivel de objetos: Detalla el acceso a nivel de creación y administración de

objetos de datos: tablas, vistas, índices, relaciones, reglas, etc. Es decir, las responsabilidades y acciones que puede hacer el usuario en el esquema de la base de datos.

La seguridad a nivel de datos: Se realiza en la capa de información, donde se indicará quién puede acceder a qué información para su consulta, actualización, inserción o borrado.

Seguridad a nivel de protección de los almacenamientos físicos de la información: Es tarea del sistema operativo, de los archivos de datos del sistema, y las políticas de copia de seguridad y restauración de los datos.

2.4.1 Selección del Hardware y Software

La selección del Hardware y el Software instalado en los servidores de base de datos son aspectos importantes en la seguridad y optimización de la base de datos, deben ofrecer soluciones de forma fiable, rentable y de alto rendimiento.

Aspectos que debe cumplir un servidor en cuanto al Hardware:

- Rendimiento
- Escalabilidad
- Disponibilidad

Rendimiento consta de tres factores:

- Procesador, el cual se debe medir por la velocidad del reloj, velocidad de acceso al Bus, tamaño de la cache L2.
- Velocidad de transferencia de datos, debe tener en cuenta el tipo de disco duro, velocidad de transferencia de los datos hacia y desde los discos duros.
- Memoria que trae, tipo de memoria y velocidad.

Escalabilidad: Es la característica que te permite adicionar más procesadores, más memorias,

más discos duros, ya sea interno o externo.

Disponibilidad: Es uno de los aspectos fundamentales que debe cumplir todo servidor estando activo 364 días al año, 7 días a la semana, 24 horas al día.

Aspectos que debe cumplir un servidor en cuanto al Software:

- Escalabilidad
- Confiabilidad
- Conectividad
- Seguridad
- Costo

Escalabilidad: Es la posibilidad de permitir crecimiento paulatino del Sistema de Cómputo aumentando las posibilidades del Sistema Operativo (S.O.), (incrementando la cantidad de usuarios e instalando nuevos paquetes de software). Lograr mantener el mismo núcleo instalado independientemente que crezca la empresa.

Confiabilidad:

- Un Sistema Operativo es confiable siempre que sea estable en su funcionamiento y en las aplicaciones que ejecuta y sea menos propenso a las caídas del sistema.

- Un Sistema Operativo es confiable en la medida en que sean confiables los Sistemas de Ficheros que implementa, los mecanismos de recuperación de información ante una caída del S.O y no tenga implementado a nivel del núcleo mecanismos que puedan incidir en su inestabilidad.

- Un Sistema Operativo es confiable en la medida en que sea menor el tiempo de respuesta a los usuarios.

Conectividad: Debido a la gran diversidad de sistemas operativos que existen en cualquier empresa un servidor debe ser capaz de prestarle servicios a un cliente que use, por ejemplo, el sistema operativo Windows y otro que utilice Linux. Compartir ficheros, datos, mensajes, etc., usando los protocolos TCP/IP.

Seguridad del Sistema Operativo (S.O): Ningún Sistema Operativo es totalmente seguro y es por ello que constantemente están bajo los ataques de personas que tratan de determinar cuáles son los huecos de seguridad y aprovecharse de ellos para poder realizar ataques directos ó indirectos.

La seguridad de un Sistema Operativo no debe ser vista solamente por su vulnerabilidad a estos ataques. Es seguridad también las posibilidades que brinda para evitar la corrupción o destrucción de los datos almacenados y los mecanismos de prevención, detección, restauración y análisis que se llevan a cabo para garantizar la seguridad del sistema. (Ver tabla 2.1)

Características que deben cumplir los sistemas operativos seguros:

- **Confidencialidad:** La información solo puede ser accedida por las entidades autorizadas.
- **Integridad:** La información sólo puede ser modificada por las entidades autorizadas. La modificación incluye escritura, cambio, borrado, creación y reenvió de los mensajes transmitidos.
- **Disponibilidad:** Requiere que los recursos del sistema informático estén disponibles para las entidades autorizadas cuando lo necesiten.
- **Confiabilidad:** Un Sistema Operativo es confiable siempre que sea estable en su funcionamiento y sea menos propenso a las caídas del sistema.

Sistemas Operativos Linux	**Vulnerabilidades**
Fedora	153
Gentoo	1258 (1%)
Mandriva	252
Debian	202
Red Hat	18
Slackware	47
Suse	111
S.O Unix	
Sun Solaris	287
HP-UX	170
S.O Windows y software	
Windows Server Enterprise 2003	150 (7%)
Windows 2000 Advenced Server	167 (13%)
Windows NT 4.0	57 (11%)
Windows XP Profesional	184 (15%)
Windows Vista	24 (4%)
Active Directory	249
Internet Explore	135 (32%)
Microsoft SQL Server 7	8 (25%)

Tabla 2. 1 Vulnerabilidad del Sistema Operativo y % que no han sacado parche.

Otro de los grandes problemas a la hora de elegir un buen servidor que mantenga nuestra base de datos segura es el costo de los mismos, algunos parámetros que hay que medir son:

- Costo del Sistema Operativo.
- Costo en cuanto a requerimientos de hardware que impone el Sistema Operativo.
- Costo que impone las actualizaciones de versiones antes las necesidades de expansión de una empresa.

- Costo por concepto de administración.
- Costo por concepto de mantenimiento.
- Costo por concepto de entrenamiento del personal.
- Costo de la información almacenada.

2.4.2 Principios básicos de la seguridad de la base de datos

Principios básicos que se deben tener en cuenta en la base de datos:

- Disgregación de responsabilidades.
- Mínimo de privilegios.
- Estándares de seguridad de contraseñas.
- Monitoreo de la base de datos.

Disgregación de responsabilidades

Este principio plantea:

- El usuario administrador de la base de datos (ABD) debe ser de total confianza, y tiene la responsabilidad de mantener la integridad de los datos.
- El ABD y el administrador del sistema operativo (ASO) no deben ser la misma persona.
- Los privilegios del usuario operador y los del ABD no deben ser los mismos.
- Los usuarios no deben compartir sus cuentas, ni contraseña.

Mínimo de privilegios

Este principio plantea:

- Sólo se debe instalar en los servidores el software requerido.
- Habilitar sólo los servicios y puertos requeridos.
- La cuenta de root y de administrador sólo la debe tener el personal requerido.
- Restringir las conexiones remotas usando el fichero *pg_hba.conf*

Estándares de seguridad y contraseñas

Este principio plantea:

- El número mínimo de caracteres de las contraseñas de los usuarios será siete caracteres.
- La contraseña de usuario no debe ser igual al nombre del usuario.
- La contraseña debe tener fortaleza requerida, con la utilización de letras, números y caracteres especiales.
- La contraseña debe cambiarse cada tres meses.

2.4.3 Acceso restringido desde la red

Es bueno usar siempre un Firewall que mantenga protegido el servidor para contrarrestar los ataques. El Firewall debe ser configurado para que sólo tenga acceso al servidor la red o subred requerida, con esto se protegen los datos.

No sólo basta con esto, también configurar los permisos de acceso a la base de datos, quiénes se pueden conectar, a qué base de datos, permisos de lectura y escritura sobre una tabla, si pueden crear roles o no.

El primer aspecto que se debe tener en cuenta después de haber instalado PostgreSQL, es el de seguridad, el cual consta de tres puntos. Esta propuesta a realizar es para versiones de la 8.1 en adelante.

- Seguridad en la manipulación de los ficheros de PostgreSQL.
- Seguridad en el acceso a los clientes.
- Definición de los privilegios para acceder a los objetos de la base de datos.

2.4.3.1 Seguridad en la manipulación de los ficheros de PostgreSQL

El directorio más crítico que presenta PostgreSQL es el PGDATA, en el que todos los ficheros que se encuentran dentro de él pertenecen al usuario postgres.

Este usuario es el único que tiene permisos de lectura y escritura sobre los directorios y ficheros dentro de PGDATA.

2.4.3.2 Seguridad en el acceso a los clientes

Este punto es importante, aquí se definen quiénes se pueden conectar al servidor, a qué base de datos, y con qué usuario.

La configuración de este punto se realiza en el fichero ***pg_hba.conf***. (Ver Anexo 8)

En este fichero se editan una serie de reglas, que se procesan en orden descendente. (Ver tabla 2.2)

TIPO	BASE DATOS	USUARIOS	DIRECCIÓN	MÉTODO
LOCAL	base_datos	usuario		método-autenticación [opción]
HOST	base_datos	usuario	direcciónCIDR	método-autenticación [opción]
HOSTSSL	base_datos	usuario	direcciónCIDR	método-autenticación [opción]
HOSTNOSSL	base_datos	usuario	direcciónCIDR	método-autenticación [opción]

Tabla 2. 2 Formas de definir un acceso limitado

Cada uno de estos campos tiene diferentes condiciones las cuales el administrador del Sistema Gestor de Base de Datos las configura según sus necesidades.

base_datos:

- **ALL:** Permite la conexión a cualquier base de datos.
- **SAMEUSER:** Permite la conexión sólo a bases de datos que su nombre sea el mismo que el usuario que se conecta.
- **SAMEROLE:** Permite la conexión solo a bases de datos que su nombre sea el mismo que

el role que se conecta.

- **nombd1, nombd2,...:** Permite la conexión a cualquiera de las bases de datos de la lista.
- **@fichero:** Permite la conexión a las bases de datos incluidas en el fichero, que debe estar en el mismo directorio que *pg_hba.conf*.

Usuario:

- **ALL:** Permite la conexión de cualquier rol.
- **role1, [+] role2,...:** Permite la conexión de los roles de la lista y además se permite la conexión de cualquier rol que sea miembro de rol2.
- **@fichero:** Permite la conexión de los roles incluidos en el fichero, que debe estar en el mismo directorio que *pg_hba.conf*.

método-autenticación:

- **TRUST:** Conexión aceptada sin condiciones.
- **REJECT:** Conexión rechazada sin condiciones.
- **PASSWORD:** Se solicita palabra de paso sin encriptar, las palabras de paso se almacenan en la tabla pg_authid y pueden estar cifradas o no según como se crea el rol.
- **CRYPT:** Palabra de paso encriptada (versiones previas a la 7.2).
- **MD5:** Palabra de paso con el método de encriptación md5, y se almacena también con este método. Es el método recomendado por PostgreSQL. Se obtiene un cifrado a partir de la ID de usuario y la palabra de paso, el cliente solicita una semilla al servidor y así se obtiene un segundo cifrado que es enviado al servidor, en el servidor se utiliza la palabra de paso almacenada, la ID del usuario (la obtiene de la conexión) y la semilla para obtener un cifrado similar y los compara.
- **KRB5:** Usa Kerberos v5 para autenticar el cliente, se habilitar en la instalación del servidor.
- **IDENT correspondencia:** A partir del usuario de la conexión cliente (se fía de la autenticación del cliente) y de la correspondencia indicada en la opción, se obtiene un rol

de PostgreSQL para realizar la conexión. Las correspondencias se obtienen del fichero *pg_ident.conf*.

La correspondencia puede ser:

- **SAMEUSER:** El usuario del sistema operativo es el mismo que se conecta a la Base de Datos (BD).
- **cambio-usuario:** El sistema mira el fichero *pg_ident.conf*, y busca una fila donde esté la correspondencia llamada 'cambio-usuario' y se corresponda con el usuario conectado al SO, haciendo la conexión a la BD con el usuario de la columna usuario-pg.

➢ **PAM servicio-pam:** Autenticación usando Pluggable Authentication Method proporcionado por el servicio PAM indicado en opción. Este método es proporcionado por el S.O. Se debe compilar el servidor con esta opción. El PAM por defecto es postgresql. Se deben configurar los métodos PAM del S.O., generalmente basta con incluir la definición del nuevo método en el directorio /etc/pam.d.

2.4.3.2.1 Conexión local y remota.

Para la configuración del fichero *pg_hba.conf* se debe tener en cuenta las políticas de cómo se podrá acceder al servidor, si permite una conexión local o una conexión remota o la combinación de las dos.

Antes de configurar el fichero *pg_hba.conf* existe otro fichero llamado *postgresql.conf,* este es el fichero de configuración de PostgreSQL, en él existe una sección llamada CONNECTIONS AND AUTHENTICATION, en la misma es donde se configuran las conexiones y cómo va a ser la seguridad.

Existen varios parámetros a configurar:

- *listen_addresses*

  listen_addresses = 'localhost' sólo permite la conexión local.

  listen_addresses = '10.32.19.12, 10.32.19.246' permite que una serie de IP específicos se conecten, separados por coma.

  listen_addresses = '*' permite que se conecte cualquier IP.

- *port*

  Se define el puerto que usara PostgreSQL, se recomienda que se use 5432 que es el que trae por defecto.

- *max_connections*

  Máximo de conexiones que permite, trae por defecto 100.

- *SSL*

  Para utilizar conexiones SSL, encriptar las consultas que viajan por la red, puede estar en true o false, más adelante se explicará con más detalles.

- *password_encryption*

  Para la encriptación de la contraseña.

  password_encryption = on permite encriptar

  password_encryption = off no permite encriptar

Este fichero consta de otra serie de parámetros que se utilizan para otras configuraciones que se irán explicando más adelante.

El fichero *pg_hba.conf* trae una configuración por defecto que no es muy segura la cual permite que cualquiera se conecte al servidor, tanto local como remoto sin contraseña.

```
# TYPE  DATABASE    USER    CIDR-ADDRESS          METHOD
# "local" is for Unix domain socket connections only
local       all           all                                 trust
# IPv4 local connections:
host        all           all       127.0.0.1/32              trust
# IPv6 local connections:
host        all           all            ::1/128              trust
```

Para tener un control más detallado se usará la siguiente configuración, sólo se puede conectar el usuario administrador de PostgreSQL, a todas las base de datos de forma local, y de forma remota se conectarán los IP que se especifiquen, los dos usando el método md5, lo demás lo hacen usando el símbolo (#). Un ejemplo de cómo quedaría es el siguiente:

```
# TYPE  DATABASE    USER          CIDR-ADDRESS          METHOD

# "local" is for Unix domain socket connections only
#local      all           all                                 trust

local       all           postgres                            md5

# IPv4 local connections:
#host       all           all       127.0.0.1/32              trust

host        SIGIA         postgres  10.32.19.12/32            md5
# IPv6 local connections:
#host       all           all            ::1/128              trust
```

En el ejemplo anterior se usa el usuario postgres como ejemplo de superusuario de la base de datos, pero se puede utilizar cualquier usuario previamente creado.

Para permitir que varios clientes se conecten al servidor usando conexión TCP/IP y los mismos están en una misma red, se especifican los permisos de la siguiente forma:

- 10.32.19.0/24 ó 10.32.19.0 255.255.255.0 : se pueden conectar todas las IPs de la red 10.32.19
- 10.32.0.0/16 ó 10.32.0.0 255.255.0.0 : todos los IPs de la red 10.32
- 10.32.19.12/32: sólo se puede conectar esa IP.
- 0..0.0.0/0 ó 0.0.0.0 0.0.0.0.: cualquier IP

Es importante ponerle una contraseña fuerte al usuario administrador para tener más seguridad, esto se realiza utilizando el terminal interactivo psql, el cual permite realizar consultas de forma interactiva a PostgreSQL, ver los resultados de las mismas, también permite la utilización de varios comandos que facilita el uso y la automatización de varias tareas.

postgres@debian:/home/erich$ psql -d template1

Una vez dentro se ejecuta

template1=# ALTER USER emgomez ENCRYPTED PASSWORD 'xxxxxxxxxx';

Con esto la contraseña del administrador de PostgreSQL ha sido actualizada y se pedirá cada vez que se quiera conectar al mismo.

De esta forma se protege la conexión de autenticación desde conexiones remota utilizando el algoritmo md5, pero las consultas realizadas dentro de psql viajan de forma plana, ¿Cómo se podría evitar esto?.

Esta dificultad se puede evitar utilizando SSL.

2.4.3.2.2 Conexión con SSL

Es un protocolo desarrollado por Netscape en 1996 que pronto se convirtió en el método elegido para asegurar las trasmisiones de datos por Internet. SSL es una parte integral de la mayoría de los exploradores y servidores web y hace uso del sistema de codificación con dos claves: una pública y una privada, desarrollado por RSA. (Thrnton)

Para poder utilizar conexiones SSL en PostgreSQL se debe tener instalado OpenSSL en ambos sistemas cliente y servidor. También se debe iniciar PostgreSQL con soporte SSL, esto se configura en el archivo *postgresql.conf*, poniendo *SSL = true* y se ejecuta:

./configure -cache_config -with-openssl.

La función de este comando es configurar internamente los archivos de PostgreSQL para cuando se inicie el servicio tenga las conexiones SSL activas, esto se realiza en el momento de la instalación.

Habilitar SSL en PostgreSQL:

PostgreSQL necesita un certificado para poder establecer un canal de comunicación seguro, lo primero es realizar una solicitud de un certificado a una autoridad certificadora válida. Pero antes hay que detener el servidor:

/etc/init.d/postgresql.8.x stop

$openssl req -new -text -out server.req

El comando anterior requiere una clave secreta, la cual no se debe olvidar, en la opción de Common Name, se pone el nombre de la computadora.

Ahora se crea una llave con el algoritmo RSA

$openssl rsa -in privkey.pem -out server.key

Se debe ingresar la clave secreta en cada opción que se pide, posteriormente se borra el archivo creado (.pem) para evitar que el servidor esté en riesgo.

$rm privkey.pem

Ahora se firma la solicitud como Autoridad Certificadora, se proporcionará la solicitud del certificado es decir el archivo **server.req.**

$openssl req -x509 -in server.req -text -key server.key -out server.crt

Una vez ejecutado el comando o la Autoridad Certificadora se regresa el certificado con el nombre de server.crt.

Se cambian los permisos a los archivos generados, y se comprueba que estén dentro del directorio de datos de PostgreSQL.

$chmod og-rwx server.key
*$chown emgomez:emgomez server.**

Creado el (.crt), sólo falta activar el SSL en el archivo *postgresql.conf*, se cambia ssl= on; y quitar el comentario.

Esto tiene una desventaja a la hora de utilizarlo porque reduce un porcentaje mínimo al servidor, pero la gran ventaja es que aumenta la confidencialidad de la información.

```
# TYPE   DATABASE      USER          CIDR-ADDRESS          METHOD

# "local" is for Unix domain socket connections only
#local      all         all                                 trust

local      all         postgres                            md5

# IPv4 local connections:
#host       all         all           127.0.0.1/32          trust

hostssl    SIGIA       postgres      10.32.19.12/32        md5
# IPv6 local connections:
#host       all         all           ::1/128               trust
```

Así quedaría la configuración del fichero *pg_hba.conf* para que permita conexiones utilizando SSL.

Para saber si el servidor tiene activado las conexiones SSL se ejecuta en consola con la ayuda del psql:

template1=# SELECT name, setting from pg_settings where name ='ssl';

La misma debe devolver:

ssl | on

Para saber si las consultas a través de la red están cifradas se utiliza la herramienta tcpdump, la cual dará información valiosa, para más detalles ejecutar:

tcpdump --help para ver las opciones de utilidad en nuestra consola.

2.4.4 Monitoreo de la base de datos

El principio de monitoreo o auditoría se basa en la posibilidad de efectuar auditorías automáticas a la base de datos PostgreSQL y los distintos tipos de actividades que se realizan en el sistema que son sujetos de auditar, entre ellas: (Fábregas, et al., 2007)

- Modificaciones al modelos de datos (comandos DDL).
- Modificación de los datos (comandos DML).
- Conexiones a la base de datos.

Las auditorías permiten mantener un historial de los cambios realizados en los datos: qué cambio, quién lo realizó y cuándo, esto permite saber e identificar cómo los datos obtuvieron su valor actual.

Las actividades pueden auditarse desde dos puntos de vistas:

1. Quién las realizó: Aquellos comandos ejecutados por cierto usuario. Ej.: Conexiones a las bases de datos.

2. Qué objeto afectó: Comandos realizados sobre cierto objeto. Ej.: Actualizaciones a objetos.

PostgreSQL presenta una característica que se puede utilizar en este caso, el uso de los disparadores (Triggers).

Se puede almacenar las actividades realizadas por los usuarios en la modificación de la información de forma inapropiada.

De esta forma se propone hacer las auditorías en la base de datos.

Un ejemplo es el siguiente:

Una tabla usuario la cual tiene una serie de atributos:

Nombre, login, pass, apellidos, fecha, correo, sueldo

En este ejemplo se registra qué cambios se realizaron en el sueldo de algún usuario y qué usuario postgres lo realizó.

1. Se crear una tabla auxiliar llamada aux_reg

```
CREATE TABLE aux_reg(
operation char(1) NOT NULL,
stamp timestamp NOT NULL,
userid integer NOT NULL,
usunombre text NOT NULL,
sueldo_anterior float not null,
 sueldo_nuevo float not null
);
```

2. Se crea el lenguaje para la base de datos, CREATE LANGUAGE 'plpgsql';

3. Función que se ejecutará en el momento de crear la tabla usuario, la cual realizará un INSERT en la tabla aux_reg.

```
CREATE OR REPLACE FUNCTION funcion_aux_reg() RETURNS TRIGGER AS E'
BEGIN
IF (TG_OP = \'DELETE\') THEN
INSERT INTO aux_reg SELECT \'D\', now(), user, OLD.idusuario,OLD.sueldo;
RETURN OLD;
ELSIF (TG_OP = \'UPDATE\') THEN
INSERT INTO aux_reg SELECT \'U\', now(), user, NEW.idusuario,NEW.sueldo;
RETURN NEW;
ELSIF (TG_OP = \'INSERT\') THEN
INSERT INTO aux_reg SELECT \'I\', now(), user, NEW.idusuario,NEW.sueldo;
RETURN NEW;
END IF;
RETURN NULL;
END;'
LANGUAGE 'plpgsql';
```

4. Crear un trigger que se ejecutara al hacer INSERT, DELETE y UPDATE a la tabla usuario.

```
CREATE TRIGGER aux_reg
AFTER INSERT OR UPDATE OR DELETE ON usuario
FOR EACH ROW EXECUTE PROCEDURE funcion_aux_reg();
```

Al actualizar los registros de la tabla usuario, se registrará en la tabla aux_reg la fecha en que se actualizó, usuario que hizo la actualización, entre otros elementos, esto permitirá mantener un control y cuidado de los datos. No es necesario realizarlo sobre todas las tablas sino sobre las que presentan información más sensible, o las que decida el Administrador de Base de Datos.

2.4.5 Diseño seguro de las bases de datos

Existen tres principios para mantener la seguridad de la base de datos, principios fundamentales integridad, disponibilidad y confidencialidad.

2.4.5.1 Integridad

El problema de la integridad se tiene en cuenta desde el mismo momento que se realiza el análisis y diseño de la base de datos. Con esto se protegen los datos contra posibles modificaciones accidentales o maliciosas, también contra inserción de datos falsos y borrados de los mismos.

Es de suma importancia los aspectos para no realizar un diseño a la ligera que después afecte la integridad de los datos. (Ver tabla 2.3)

Tipos de integridad	**Tipo de restricción**
Tabla	• Llave Primaria (Primary Key) • Valor único (UNIQUE)
Dominio	• Verificar valor NULL • Valor por default • Verificación de valor(Constraint de tipo Check)
Referencial	• Llave Coreana (Foreign Key)

Tabla 2. 3 Tipos de integridad

Integridad de tabla: Todas las tablas que se encuentran dentro de cada una de las bases de datos en el servidor deben tener una llave primaria, esto impide que haya redundancia de datos.

Integridad de dominio: En la base de datos se debe almacenar exactamente los datos correctos.

Integridad Referencial: Los registros deben integrarse de forma adecuada en las tablas, sin violar los principios básicos de las bases de datos relacionales. (González, y otros, 2008)

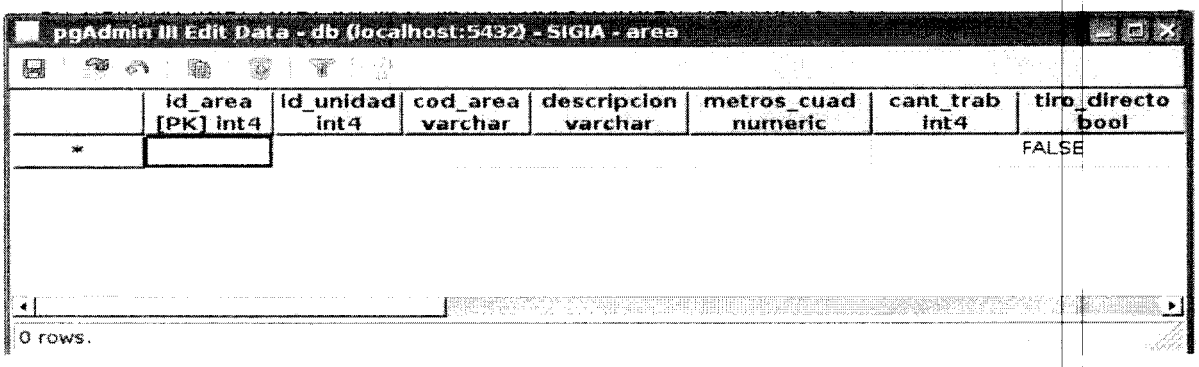


Fig 2. 3 Tabla (área) de la base de datos.

En la figura 2.3 se observa cómo la tabla debe cumplir con los requisitos de integridad para la tabla, la misma presenta una llave primaria que tiene valor *NOT NULL*, un atributo descripción que puede ser *NULL*, y un atributo *bool* que tiene valor por defecto *false.*

Haciendo un buen diseño de la base de datos, especificando los tipos de datos y sus dominios, PostgreSQL puede garantizar que se controle la redundancia de datos a la mínima expresión, así se puede lograr la integridad de los datos desde el inicio.

2.4.6 Definición de los privilegios

Para lograr una buena confidencialidad se le deben dar los permisos al usuario que lo requiera y denegárselo al que no tenga privilegios para acceder.

PostgreSQL utiliza unas sentencias llamadas REVOKE y GRANT donde se revocan y otorgan privilegios:

Ejemplo de las mismas son:

GRANT privilege [, ...] ON objecto [, ...]
TO { PUBLIC | nombre_usuario | GROUP groupname*}*

REVOKE privilege [, ...] ON object [, ...]
FROM { PUBLIC | nombre_usuario | GROUP groupname }

Otra forma de mantener la confidencialidad de los datos es haciendo uso de las vistas, para que los usuarios que tienen menos privilegios hagan las consultas sobre estas.

2.5 Realización de copias de seguridad y recuperación

El control de acceso a los usuarios de la base de datos no es el único mecanismo de seguridad, la misma se debe proteger contra los posibles errores del sistema, caídas de disco duros, desastres naturales, la pérdida de un fichero de la base de datos. Para la protección contra esto están las copias de seguridad y registro de las transacciones.

Realizar una buena política de respaldo, es la única manera de mantener el sistema contra fallos del sistema operativo, de software y otras circunstancias. (Fábregas, y otros, 2007)

Principales causas de fallos en un Sistema Gestor de Base de Datos:

- Fallas de hardware, ejemplo disco duro y CPU.
- Agujeros en el sistema operativo y en el Sistema Gestor de Base de Datos.
- Fallos debido a una mala configuración o errores introducidos a la hora de realizar copias de seguridad por el Administrador de Base de Datos.
- Fallos debido a desastres naturales, flujo de energía y mala ambientación.

La necesidad de mantener la disponibilidad de la base de dato para evitar pérdidas, se hace necesario tomar todas las medidas necesarias, y la implantación de los mecanismos de hardware.

- UPS o fuentes de corriente ininterrumpida.
- Discos en Espejo, o tecnología RAID.
- Componentes duplicados.
- Sistemas redundantes.

2.5.1 Tipos de copia de seguridad

Existen dos tipos de copia de seguridad:

Físicos: Copian físicamente los ficheros de la base de datos, entre ellos se encuentran: copia de seguridad del sistema operativo, copia de seguridad en caliente y en frío.

Lógicos: Extraen los datos de la tabla utilizando comandos SQL, se realizan con la utilidad export/import.

2.5.1.1 Copia de seguridad en frío

Implica parar la base de datos (BD) en modo normal y copiar todos los ficheros en los que se asienta. Antes de parar la BD se deben parar también todas las aplicaciones que estén utilizando la BD. Una vez terminada la copia se inicia la BD.

2.5.1.2 Copia de seguridad en caliente

Se realiza mientras la BD está abierta y funcionando el mecanismo de LOG. Este tipo de respaldo consiste en copiar todos los archivos de LOG correspondientes a un directorio determinado.

2.5.1.3 Copia de seguridad del sistema operativo (S.O)

Es el más sencillo de ejecutar, aunque consume mucho tiempo y hace inaccesible al sistema mientras se lleva a cabo. Aprovecha la copia de seguridad del S.O para almacenar también todos los ficheros de la BD. Los pasos de este tipo de copia de seguridad son los siguientes:

1. Detener la BD y S.O.
2. Iniciar en modo superusuario.
3. Realizar copia de todo sistema de ficheros.

4. Arrancar el sistema en modo normal y luego la BD.

En dependencia de cada una de las características de las empresas se debe elegir un tipo de respaldo, pero primero hay que conocer qué tipos proporciona PostgreSQL.

2.5.2 Copias de seguridad en PostgreSQL

PostgreSQL tiene tres formas principales de realizar copias de seguridad:

- Copia de seguridad de ficheros del S.O.
- Volcado y recuperación SQL usando las herramientas PostgreSQL: pg_dump, pg_dumpall y pg_restore.
- Copia de seguridad en línea y recuperación en el punto (PITR).

2.5.2.1 Copia de seguridad de ficheros del S.O

Es el método más sencillo de todos, se trata de realizar una copia de los ficheros del clúster:

su postgres

cd /tmp/backup

tar cvfz copia.tar.gz $PGDATA

Copiar este fichero a cinta.

Este método tiene desventajas:

- La base de datos debe estar parada.
- No se pueden recuperar partes del clúster (base de datos, esquemas y tablas).

La recuperación en este método consiste borrar todo el clúster y descomprimir el fichero creado anteriormente, esto trae consigo que se pierdan todos los datos que se han modificado desde la última copia.

2.5.2.2 Volcado SQL usando las herramientas PostgreSQL

Estas copias son muy flexibles y de gran utilidad, porque permiten hacer copias de toda la base de datos o parte de ella. Además sirven para la trasmisión de datos entre bases de datos.

Las herramientas a utilizar son:

pg_dump: Permite hacer una copia de seguridad de la base de datos o parte de ella en un fichero script en texto plano o en formato PostgreSQL.

Fichero en texto plano:

- Texto con instrucciones SQL.
- Se usa psql para recuperar.
- Portable para servidores SQL.

Formato propio de PostgreSQL:

- Se usa pg_restore para restaurar.
- Más flexible.
- Portable entre servidores PostgreSQL.

El proceso se debe realizar con el clúster en marcha, un ejemplo del mismo sería:

pg_dump [opciones]... [NombreBD]

Opciones Generales:

-f, --file nombre del archivo de salida

-F, --format =c|t|p formato del archivo de salida (personalizado, tar, solo texto)

Existen más opciones que se pueden visualizar en la pantalla de la consola utilizando el comando pg_dump --help.

Exportar una base de datos usando pg_dump:

pg_dump -f dbsigia.sql -h otrohost -p 5432 dbsigia -U postgres

Donde dbsigia es una base de datos hipotética.

Exportar un esquema:

pg_dump -n sigia1 > esq_sigia1.sql

pg_dumpall: Permite hacer una copia de seguridad del clúster completo incluyendo roles de grupo y roles de login.

Es similar a *pg_dump* pero presenta algunas diferencias:

1. No permite la opción de indicar el fichero de salida.
2. No tiene la opción de formatos de fichero de salida, la salida es siempre un fichero de texto plano.
3. Tiene una opción muy interesante que es "-g" que permite exportar únicamente los objetos globales, con lo que se genera un script que contenga la creación de usuarios y roles.

Exportar un clúster completo:

pg_dumpall > clustersigia.sql

Exportar sólo los datos:

pg_dumpall -a > datossigia.sql

Recuperación: Para la recuperación de la base de datos PostgreSQL utiliza dos herramientas psql y pg_restore.

Recuperación con psql:

Se recuperan los ficheros en texto plano con sentencias SQL.

psql [dbdestino] < fichero.sql

Recuperación con pg_restore:

Se recuperan ficheros en formato tar o un formato propio de PostgreSQL.

pg_restore [Opciones]... [Archivo]

Opciones generales:

-d, --dbname= nombre	Nombre de la base de datos a conectarse.
-f, --file= archivo	Nombre del archivo de salida.
-F, --format = c|t	Especifica el formato de respaldo.

Existen más opciones como son opciones que controlan la recuperación, opciones de conexión, estas se pueden ver con el siguiente comando pg_restore --help.

2.5.2.3 Copia de seguridad en línea y recuperación en el punto (PITR)

PostgreSQL presenta una gran recuperación, cuando ocurren fallos de corriente, cuelgues, etc. Para esto el sistema utiliza la recuperación mediante los ficheros de *log* (WAL).

Los ficheros **WAL** son segmentos de 16Mb que se nombran de forma secuencial en el cual el sistema va copiando los cambios en la base de datos, se encuentran en el directorio pg_xlog. (Alarcón, 2006)

Esta forma de hacer copias de seguridad se encarga de realizar respaldos periódicos, desde la última vez que se realizó el respaldo hasta el momento de hacer uno nuevo.

Se debe partir de un respaldo total para que se registren en archivos binarios log, los log almacena las transacciones hechas por cierto tiempo.

Para realizar una copia de seguridad de base se hace de la manera siguiente:

1. Activar el archivado WAL en el fichero de *postgresql.conf* con el parámetro *archive_command*, ejemplo.

archive_command = 'test ! -f /etc/postgresql/8.1/main/pgdata/pg_xlog/%f.gz && gzip -1 -c %p > /var/backups/pgsql/%f.gz'

2. Desde una consola con psql hay que ejecutar.
   Select pg_start_backup('nombre_sigia');

3. Realizar la copia de seguridad con el servidor en marcha, no es necesario pararlo.
   $ tar zhcvf /home/erich/lolo/backuptotal2.tar.gz /etc/postgresql/8.1/main/pgdata/ --exlude=/etc/postgresql/8.1/main/pgdata/pg_xlog/

4. Desde la consola con psql se ejecuta el siguiente comando para terminar.
   Select pg_stop_backup();

5. Se crea un fichero en el directorio $PGDATA/pg_xlog/archive_status y los log que se reciclan, se copian donde se indicó en *archive_command*.

El archivado WAL le permitirá restablecer las modificaciones introducidas a los datos de su base de datos PostgreSQL no va a restaurar los cambios realizados en los archivos de configuración (postgresql.conf, pg_hba.conf y pg_ident.conf).

Estos archivos pueden ser copiados y guardados en discos dedicados a salvas.

Para realizar la recuperación se siguen los siguientes pasos:

1. Detener el postmaster.
2. Copiar el clúster dañado y los tablespaces a otra ubicación.
3. Borrar todos los ficheros que hay dentro del clúster, incluyendo los tablespaces.
4. Recuperar la copia de seguridad.
   tar xvf backup_nombre_copia

5. Borrar los ficheros WAL en $PGDATA/pg_xlog, porque están obsoletos.
6. Si existen ficheros WAL sin archivar, tal como se explica en el paso 2, se debe copiar los mismos a pg_xlog.
7. Crear un fichero de comandos de recuperación, *recovery.conf* en el directorio $PGDATA. Existe una plantilla, recovery.conf.sample, en el directorio $/usr/share/postgresql/8.1/. Se copia con el nombre "recovery.conf" en $PGDATA y se edita. En principio, si se quiere recuperar todo y los logs están en su sitio, no hay que tocar nada.
8. Se arranca PostgreSQL en modo de recuperación, no se debe conectar nadie mientras el servidor esté realizando la recuperación.
9. Inspeccionar la base de datos, si todo ha ido correcto, los datos estarán recuperados hasta la última transacción confirmada y el fichero recovery.conf se renombra a recovery.done.

Para que la recuperación sea satisfactoriamente se deben tener los ficheros log en un disco distinto, donde está instalado el clúster y crear un enlace simbólico entre los dos discos, ejemplo.

mkdir /disco2/pg/

cd $PGDATA

mv pg_xlog /disco2/pg

ln –s /disco2/pg/pg_xlog pg_xlog

El fichero *recovery.conf* es el fichero que ayudará a realizar la recuperación, el mismo presenta una serie de parámetros que pueden hacer nuestra recuperación más fácil, esto se conoce como recuperación **Point-in-time**.

- **restore_command:** Lo que ejecuta esta variable es lo que PostgreSQL ejecutará antes de empezar la recuperación, por ejemplo:
  restore_command = 'cp /mnt/server/archivedir/%f %p'
- **recovery_target_time:** Hasta qué momento.

- **recovery_target_xid:** Hasta una transacción determinada.
- **recovery_target_inclusive (boolean):** Si los dos casos anteriores son inclusive o no.
- **Recovery_target_timeline (cadena)**: Especifica la recuperación en un tiempo particular.

2.5.3 Estrategia de copia de seguridad

Propuesta de estrategia de copia de seguridad en la base de datos es de la forma siguiente:

1. Respaldo automático utilizando la herramienta pg_dump de la forma siguiente:

- Creación de un script ejecutable con permisos root.

```
#!/bin/bash
clear
#Función que muestra un mensaje de Bienvenida
function bienvenida {
  echo "Script para Generar Backup de una Base de Datos en Postgres"
  echo ""
}
#función que muestra una nueva línea
function newLine {
  echo ""
}
echo El Backup se va a generar en el Directorio "$HOME"
newLine
H="127.0.0.1"
U="postgres"
BD="SIGIA"
F=backup-$BD-$(date +%Y-%m-%d)
```

```
echo pg_dump -i -h $H -p 5432 -U $U -F p -C -D -v -f "$HOME/$F.sql" "$BD"
pg_dump -i -h $H -p 5432 -U $U -F p -C -D -v -f "$HOME/$F-$(date +%Y-%m-%d).sql" "$BD"
```

- Copiar el script a la carpeta del usuario root.
- Configuración del cron del sistema operativo para ejecutar el script todos los días a las 10:00 pm.

  ```
  00 22  * * *  root   /root/backupauto.sh
  ```

- El nombre y formato de guardado: Backup-Nombre-año-mes-día.sql.
- El formato del Backup puede variar en dependencia del tamaño de información de la base de datos.
- Copiar el Backup para uno o más disco duros designados para guardar estas salvas.
- El borrado de la misma se hará después de haber pasado de 20 a 30 días.

Copia de seguridad mediante el uso de los ficheros WAL:

1. Activar el archivado WAL en el fichero *postgresql.conf*
2. Copia de seguridad de base.
   - Mediante esta copia se activará el archivado WAL, y se empezara a guardar todas las operaciones en los log.
3. Respaldo diario a las 6:00pm, de todos los ficheros WAL y fichero contenido en pgdata/pg_xlog/archive_status a discos duros destinados sólo para este uso.

2.6 Base de datos central

Debido a que la información que se maneja en las entidades afiliares debe replicarse a una base de datos central para su control y auditorías, es necesario implementar una administración centralizada que permita el monitoreo de los otros subsistemas, para esto es necesario la creación de un centro de datos en la entidad central que gestione la información de todas las entidades afiliadas.

2.6.1 Centro de datos

Contar con un centro de datos que cumpla con todas las características necesarias que se requiere, tolerancia a fallos con capacidad de mantenimiento simultáneo, escalabilidad palpable y la flexibilidad para lograr una larga vida, es de suma importancia para que el mismo pueda proporcionar una mayor disponibilidad, accesibilidad, escalabilidad, y confiabilidad las 24 horas al día, los 7 días a la semana, y los 365 días al año descontando el tiempo fuera de servicio por mantenimiento.

Componentes de un centro de dato típico:

- Infraestructura de cómputo y redes (cableado, fibra, y electrónicos).
- Network Operations Center (NOC) o comunicaciones y monitoreo NOC.
- Sistemas eléctricos de distribución, generación y acondicionamiento - UPS, generadores de control ambiental y sistemas HVAC.
- Sistemas de detección y supresión de fuego (típicamente halón u otros sistemas sin agua).
- Seguridad física y prevención de control de acceso, permisos y logging.
- Protección de circuitos (protección de iluminación en algunos casos).
- Iluminación apropiada.
- Altura mínima de techo de 8'5.
- Tierra física.
- Racks y gabinetes para equipo.
- Canalizaciones: Piso falso y bandejas en techo.
- Circuitos y equipo de carriers.

- Equipo de Telecomunicaciones.
- Separaciones alrededor del equipo, y terminaciones en paneles y racks.

Es importante predecir el número de usuarios, tipos de aplicaciones y plataformas, unidades de rack requeridas para el montaje de equipos y sobre todo, anticipar el crecimiento y los cambios tecnológicos. (Fábregas, y otros, 2007)

Se debe hacer un estimado del rendimiento vs capacidad del Centro de Datos según la estructura de la empresa donde se va implantar, cantidad de entidades afiliadas y sobre todo los 15 servidores ubicados en cada una de las provincias y municipio especial Isla de la Juventud.

Existen varios aspectos que se deben considerar los cuales son:

- Planeamiento de rendimiento.
- Planeamiento de capacidad.

Planeamiento de rendimiento:

- Transacciones de Escritura al Centro de Datos: Se debe considerar y calcular la cantidad de transacciones que se harán desde las base de datos ubicadas en cada una de las provincias.
- Transacciones de lectura al Centro de Datos: Calcular el flujo de información consultada a la base de datos central cada vez que vaya a realizar una transacción, teniendo en cuenta que varias transacciones se pueden realizar al mismo tiempo.
- Transacciones de Escritura de Imágenes Overnight: En las empresas por cada transacción realizada, casi siempre se genera un documento digitalizado, estas transacciones se consideran grandes (long running transactions).

Planeamiento de capacidad: Este parámetro es importante tenerlo en cuenta, y siempre calcularlo para saber la capacidad requerida del servidor.

- Equipo de Telecomunicaciones.
- Separaciones alrededor del equipo, y terminaciones en paneles y racks.

Es importante predecir el número de usuarios, tipos de aplicaciones y plataformas, unidades de rack requeridas para el montaje de equipos y sobre todo, anticipar el crecimiento y los cambios tecnológicos. (Fábregas, y otros, 2007)

Se debe hacer un estimado del rendimiento vs capacidad del Centro de Datos según la estructura de la empresa donde se va implantar, cantidad de entidades afiliadas y sobre todo los 15 servidores ubicados en cada una de las provincias y municipio especial Isla de la Juventud.

Existen varios aspectos que se deben considerar los cuales son:

- Planeamiento de rendimiento.
- Planeamiento de capacidad.

Planeamiento de rendimiento:

- Transacciones de Escritura al Centro de Datos: Se debe considerar y calcular la cantidad de transacciones que se harán desde las base de datos ubicadas en cada una de las provincias.
- Transacciones de lectura al Centro de Datos: Calcular el flujo de información consultada a la base de datos central cada vez que vaya a realizar una transacción, teniendo en cuenta que varias transacciones se pueden realizar al mismo tiempo.
- Transacciones de Escritura de Imágenes Overnight: En las empresas por cada transacción realizada, casi siempre se genera un documento digitalizado, estas transacciones se consideran grandes (long running transactions).

Planeamiento de capacidad: Este parámetro es importante tenerlo en cuenta, y siempre calcularlo para saber la capacidad requerida del servidor.

2.6.2 Base de datos central

La base de datos central es una agrupación de todas las bases de datos de las afiliares provinciales, su función es almacenar una copia diaria del flujo de trabajo realizado en las bases de datos locales, para mantener un control centralizado de toda la información. Por lo que en el centro de datos utilizará una estrategia similar a la de las bases de datos locales.

Almacenamiento de la base de datos central: PostgreSQL utiliza una estrategia de almacenamiento de filas llamada Control de Concurrencia Multi-Versión (Multi-Version Concurrency Control) MVCC para conseguir una mejor respuesta en ambientes de grandes volúmenes de datos. (Worsley, y otros, 2001)

MVCC: Es una tecnología que PostgreSQL usa para evitar bloqueos innecesarios. Los SGBD con capacidades SQL, tales como MySQL o Access, no la usan, probablemente habrá notado que hay ocasiones en las que cuando se hace una lectura tiene que esperar para acceder a información de la base de datos. La espera está provocada por usuarios que están escribiendo en la base de datos. Resumiendo, el lector está bloqueado por los escritores que están actualizando registros.

Mediante el uso de MVCC, PostgreSQL evita este problema por completo. MVCC está considerado mejor que el bloqueo a nivel de fila porque un lector nunca es bloqueado por un escritor. En su lugar, PostgreSQL mantiene una ruta a todas las transacciones realizadas por los usuarios de la base de datos. PostgreSQL es capaz de manejar los registros, sin necesidad de que los usuarios tengan que esperar que los registros estén disponibles.

Monitorización del uso de discos: PostgreSQL es quien controla este tema

Formas de obtener el uso del disco:

- Consultando vistas del catálogo
- Usando las extensiones $PGSRC/contrib: *dbsize:* base de datos y sus tamaños, oid2name: correspondencia entre nombre de tablas y ficheros

2.7 Replicación de datos

En esta se sección abordará sobre características, tipos de réplica y estrategia que se propone.

La replicación de datos permite copiar y distribuir idénticamente las tablas de una base de datos en múltiples bases de datos. Esto tiene la ventaja de que en todas las bases de datos van a estar siempre disponible los datos correctos.

2.7.1 Características de la replicación

Según (Fajardo, 2007) estas son algunas de las características:

Efectividad: Depende de la forma que los datos sean distribuidos y almacenados. A mayor efectividad, mayor será la disponibilidad de datos para ejecutar procesos paralelos.

Alta Disponibilidad: Es la razón de tiempo prudente que un servicio puede ser accedido. En el mejor de los casos puede ser de un 100%, a pesar de los fallos que se puedan presentar en el servidor, debe existir un servidor adicional que posea alguna técnica de replicación que lo pueda suplantar en caso de ser necesario.

Tolerancia a fallos: Garantiza un comportamiento correcto, donde efectivamente pueden existir un número finito de fallos y tipos de fallos.

Coordinación: Cada una de las partes que conforman la base de datos distribuida, acuerda un consenso para realizar las invocaciones de los servicios a los objetos, que al final de la transacción debe realizarse tal y como fue solicitada, para lo cual debe utilizar algún tipo de ordenamiento.

2.7.2 Tipos de réplica

Existen dos tipos de réplica, Maestro-esclavo (master-slave) y Maestro-Maestro (multi-máster).

Maestro-esclavo: Sólo el maestro permite recibir consultas de lectura y escritura, mientras que el esclavo sólo de lectura. (Ver figura 2.4)

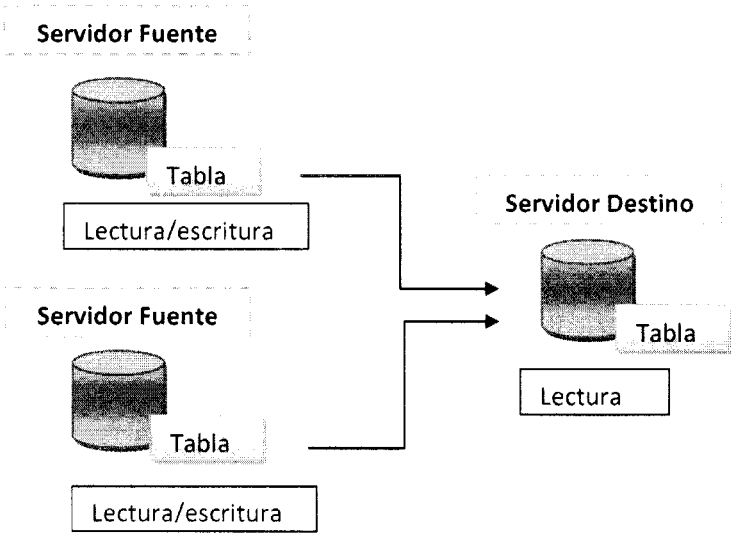


Fig 2. 4 Replicación maestro-esclavo.

Maestro-Maestro: Son varios sitios comunicándose entre sí y a la vez van actualizando sus datos, este es muy usado para equilibrar la carga de consultas, en sitios donde se conecten muchos usuarios. (Ver figura 2.5)

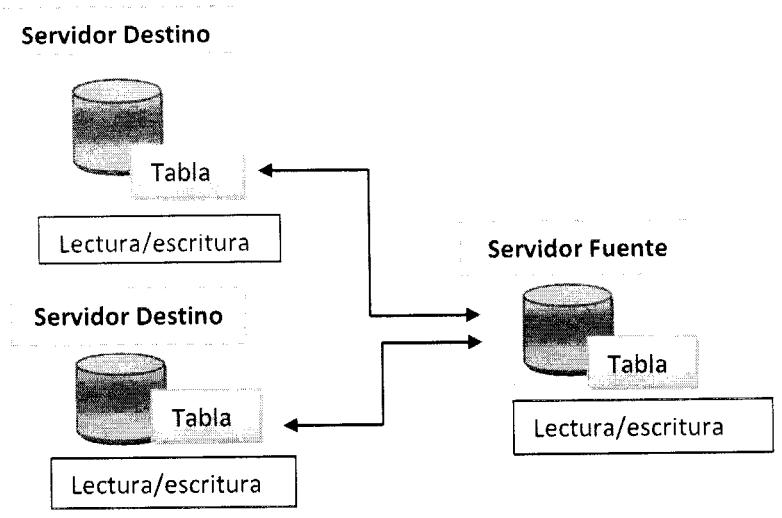


Fig 2. 5 Replicación Multi-maestro.

2.7.3 Técnicas de replicación

Sincrónica (Confirmación en dos fases): Sólo acepta la transacción cuando todos los sistemas implicados están conectados y listos para recibirla, si alguna falla, se anula el proceso.

Asincrónica (Descarga y Recarga): Copia los datos para un dispositivo de almacenamiento y luego esta salva a los demás servidores. Tiene la desventaja que el proceso se hace manual, y que se consultaban datos que ya estaban desactualizados.

Instantánea: Consiste en hacer una instantánea a una tabla en un momento dado, y esto es lo que se va a replicar. Convenientemente para la réplica de datos pequeños. (Fajardo, 2007)

2.7.4 Por qué usar replicación

Las empresas usan replicación por varias razones, las cuales pueden ser categorizadas de la siguiente forma:

1. Distribución de datos a otras ubicaciones
2. Consolidación de datos desde otras ubicaciones
3. Intercambio bidireccional de datos con otras ubicaciones
4. Algunas variantes o combinaciones de los anteriores

2.7.4.1 Distribución de datos a otras ubicaciones

Involucra el movimiento de datos o subconjuntos de datos de una o más ubicaciones.

La distribución también es usada para proveer datos a aplicaciones desarrolladas en plataformas similares o diferentes. Es tan simple como poder tener una copia de los datos en otro sistema similar. También puede ser usada para proveer coexistencia de dos sistemas, en la instancia de migración de uno a otro.

2.7.4.2 Consolidación de datos desde otras ubicaciones

Una empresa puede tener sus datos en distintos sistemas distribuidos. Por ejemplo una empresa puede tener sus datos en cada local, o en cada piso, o en cada oficina, incluso en cada uno de los computadores personales de sus trabajadores. La replicación puede copiar los cambios de cada sitio distribuido al sitio central, para analizar, hacer reportes, o para el procesamiento central de datos.

2.7.4.3 Intercambio bidireccional de datos con otras ubicaciones

Los datos se pueden modificar en múltiples ubicaciones, esto trae consigo que la replicación debe procesar los datos en cada uno de los sitios de forma coordinada. Uno de los servidores es visto como maestro, el cual se encarga de distribuir los cambios a todos los sitios. Estos cambios realizados en los destinos influyen sobre otros sitios a través del servidor maestro.

Esta forma de replicar es conocida como "Maestro-Esclavo".

Otro tipo de replicación bidireccional es que no tiene designada un servidor maestro. Cada ubicación copia los cambios desde los otros sitios directamente, la misma se conoce como "multi-máster".

2.7.4.4 Algunas variantes o combinaciones de los anteriores

Algunas aplicaciones podrían no requerir una copia completa de los datos pero sí necesitar un registro de todos los cambios que han ocurrido. Esto es útil para mantener una auditoría del sistema y también puede ser útil para proveer los cambios para procesos especiales.

Los cambios también pueden ser utilizados en escenarios de replicación multi-capas, donde son copiados desde una fuente central hacia un área de almacenamiento en otro sistema y finalmente desde esta área a los sitios destino. (González, y otros, 2003)

2.7.5 Conflictos a la hora de utilizar replicación

Existen diferentes conflictos a la hora de realizar la replicación, se explicará brevemente algunos de ellos.

Los conflictos pueden ocurrir cuando se trabaja en un entorno de replicación que permite actualizaciones concurrentes de los datos en diferentes sitios.

2.7.5.1 Conflicto de actualización

Un conflicto de actualización ocurre cuando se produce la replicación de un UPDATE sobre un registro con otro UPDATE sobre el mismo registro. Este conflicto ocurre cuando dos transacciones originadas desde distintos sitios actualizan el mismo registro.

Para estos casos existen tres formas aceptadas de resolverlos:

- **Prioridad:** Cada servidor obtiene una prioridad única, y el servidor de mayor prioridad

"gana", respecto a aquellos con prioridad menor.

- **Timestamp:** La más nueva o la más antigua de las modificaciones es la considerada correcta, y por defecto, sino se eligió ninguno de los criterios "gana" la más nueva.
- **Particionamiento de datos:** Garantiza que cada registro sea manipulado por un único servidor, lo que simplifica la arquitectura.

2.7.5.2 Conflicto de unidad

El conflicto de unicidad sucede cuando la replicación de un registro intenta violar una restricción de integridad, ya sea por Primary Key o Unique. Por ejemplo, cuando dos transacciones originadas de diferentes sitios, quieren insertar un registro en la tabla replicada con el mismo valor de clave primaria.

Para resolver este problema existen tres formas:

1. Para cada servidor brindar un rango distinto de números para los generadores de clave (secuencias).
2. Agregar el identificador del servidor a la clave primaria.
3. Replicar en tablas separadas, y acceder a los datos a través de una vista formada por la unión de ellas. Para resolver el conflicto de potenciales claves duplicadas en la unión se usará una pseudo columna que representa la base de datos fuente.

2.7.5.3 Conflicto de supresión

Ocurre cuando dos transacciones originadas de diferentes sitios, una de ellas quiere borrar un registro y la otra actualizar o borrar el mismo registro.

Una posible solución es que los sitios marquen lógicamente los registros borrados y que periódicamente el sitio maestro corra un proceso que realice el "DELETE" físico de los datos, es decir desde los sitios replicados no se puede ejecutar una sentencia DELETE.

2.7.5.4 Conflicto de orden

Los conflictos de orden pueden ocurrir en ambientes de replicación con tres o más sitios maestros. Si la propagación al sitio maestro X, está bloqueada por alguna razón, entonces la replicación de modificaciones en datos puede seguir siendo propagada a través de los otros sitios maestros; al finalizar la propagación estas modificaciones debieron ser propagadas al sitio X en un orden diferente a como ocurrieron en los demás sitios maestros, pudiendo producirse un conflicto.

Este tipo de conflictos se suele resolver asignándole distintas prioridades a los sitios maestros, de forma que se ordenen las transacciones de acuerdo a esta. (González, y otros, 2003)

2.7.6 Estrategia de réplica entre base de datos

Para la propuesta de replicación entre bases de datos se hace uso de la herramienta de Slony-I, que implementa la réplica asincrónica (Maestro-Esclavo), esta es una herramienta provechosa pero no resuelve el problema planteado en su totalidad. (Ver Anexo 9)

La propuesta se realizará de la forma siguiente:

1. Creación de dos tablas auxiliares en cada una de las bases de datos (BD)
2. Creación de un trigger en la BD (máster) escrito en pl/pgsql que se dispare después de haber hecho un INSERT, DELETE, UPDATE en cualquier tabla, el mismo reconstruirá la sentencia SQL y la insertará en la tabla auxiliar1 (máster).
3. La herramienta Slony con su capacidad para crear espejos replicará cada dato insertado en la tabla auxiliar1 (máster), a la tabla auxiliar1 (esclava). Esta operación se realizará inmediatamente después de haber insertado la tupla en la tabla auxiliar1 (máster).
4. Creación de un trigger en la BD (esclava) implementado en pl/pgsql que se dispare después de un INSERT en la tabla auxiliar1 (esclava), esto posibilitará la ejecución de las sentencias SQL recibidas.

5. Si la operación anterior se realiza correctamente el identificador de la tupla se guardará en otra tabla auxiliar2 (máster) destinada para esto.
6. Las tuplas almacenadas en esta entidad auxiliar2 (máster) serán replicadas para otra entidad auxiliar2 (esclava) ubicada en la otra BD.
7. Creación de un trigger (pl/pgsql) que tendrá la función del borrado de las tuplas en la tabla auxiliar1 (máster) a medida que lleguen a él los identificadores, de esta misma forma se hace el borrado de la tabla auxiliar1 (esclava).

Para su mejor comprensión la réplica de Slony-I se hace a nivel de tabla, una misma BD puede tener varias tablas máster y varias esclavas.

Mediante el uso de esta propuesta se podrá tener la información actualizada para uso de los usuarios en el momento preciso.

Conclusiones del capítulo:

En el capítulo se han abordado un grupo de aspectos esenciales para el desarrollo de un Sistema de Planificación de Recursos, sus siglas en ingles ERP.

Se dirigió la atención a la seguridad de las base de datos, así como distintos elementos que deben cumplir los servidores para mantener la seguridad en ellas y que la información permanezca intacta.

La arquitectura descentralizada como forma de mantener la disponibilidad en caso de interrupciones en la red.

Como consecuencia de la arquitectura a utilizar en el documento se propone el uso de mecanismos de gestión y replicación de la información.

Queda plasmada la necesidad de la creación de un centro de datos que cumpla con las características de disponibilidad, accesibilidad y escalabilidad.

CAPÍTULO III: CONFIGURACIONES DE OPTIMIZACIÓN Y ADMINISTRACIÓN DE BASE DE DATOS IMPLANTADAS EN POSTGRESQL

Introducción

En el actual capítulo se presentarán soluciones de optimización y administración que pueden servir como guía para administradores y desarrolladores de bases de datos sobre el gestor PostgreSQL.

3.1 Optimización de bases de datos en PostgreSQL

La idea central de la optimización no es más que realizar las mismas tareas en menos tiempo, con menos requerimientos, o en general empleando los recursos en forma óptima.

La optimización de PostgreSQL tiene distintos niveles, desde los más bajos como la implementación o diseño de algoritmos sobre las consultas a la base de datos, hasta la configuración o afinamiento del servidor.

3.1.1 Ajustes del rendimiento del servidor

Existen dos aspectos fundamentales con respecto al rendimiento de las bases de datos. La primera es la mejora de las bases de datos en el uso de la CPU, la memoria y las unidades de discos montadas en el servidor. La segunda es la optimización de las consultas enviadas a la bases de datos; la optimización de las consultas se realiza a través de comandos SQL como CREATE INDEX, VACUUM, VACUUM FULL, ANALYZE, CLUSTER y EXPLAIN.

Para poder comprender mejor el rendimiento del servidor es necesario entender cómo funciona en su interior. En resumen, el servidor presenta una unidad central de procesamiento de datos (CPU) la cual maneja los datos, en el mismo chip hay integrado varios registros de información junto con varios punteros y contadores, y a su vez cuenta con la cache que es la contiene la información más reciente utilizada por la CPU. También se puede contar con la memoria de

acceso aleatorio (RAM), que está compuesta de uno o más chips y se utiliza como memoria de trabajo para programas y datos, conteniendo la ejecución de los mismos. Lo que quedaría después de la memoria principal sería las unidades de disco, que pueden almacenar grandes cantidades de información, puesto que son la única zona del computador para el almacenamiento. Ver (Anexo 10)

Una correcta afinación implicaría mantener a la mano la mayor cantidad de información posible en la RAM, mientras que no afecte a los demás programas fundamentales del sistema.

3.1.2 Afinamiento del servidor

¿Cómo saber que operaciones realizar para afinar y mejorar el rendimiento del servidor?

Para resolver esta interrogante lo primero que hay que hacer es recordar la arquitectura de PostgreSQL. (Ver Anexo 11)

Entonces viendo el anexo de la estructura de PostgreSQL se puede decir que se puede actuar sobre:

- Gestión del diario (WAL)
- Buffers de diario (cache WAL)
- Cache de la base de datos
- Acceso a los discos

3.1.2.1 Gestión del diario (Write Ahead Logging)

Todo sistema que utilice WAL presenta técnicas de prestación de átomos y durabilidad (dos de las propiedades de ACID en sistemas de bases de datos), todas las modificaciones se escriben en un registro antes de que se apliquen a las bases de datos, por lo general rehacer y deshacer información almacenada. Dicho sistema permite actualizaciones de las bases de datos,

reduciendo la necesidad de modificar las tablas y los índices.

Beneficios:

- Reduce significativamente el número de escrituras a disco. Sólo es necesario escribir el buffer de diario en el momento de la confirmación y se puede aprovechar una escritura para llevar información de varias transacciones.
- Consistencia de las páginas de datos: antes del WAL, PostgreSQL no podía asegurar la consistencia de las páginas en caso de fallo: entradas de índices apuntando a filas inexistentes, pérdidas de entradas de índices en operaciones de división (árbol -B), páginas de tablas o índices totalmente corrompidas por escrituras parciales de las mismas.

3.1.2.1.1 Parámetros de WAL (Write Ahead Logging)

Todos los parámetros que contiene WAL afectan directamente en el rendimiento de la base de datos, como son los que se presentan a continuación y que sólo se puede configurar en el servidor de inicio o en el archivo postgresql.conf.

Parámetros de configuración (Settings)

- **Fsync (Boolean)**

  Parámetro que cuando se encuentra activado permite al servidor tratar de garantizar que las escrituras sean físicamente en el disco, o sea esto asegura que pueda recuperarse o volver a un estado anterior después de un error del sistema o un accidente de hardware.

  En caso de estar desactivado, el sistema puede hacer todos los cambios posibles en el buffer sin tener que hacerlo directamente el disco, esto resulta ser una opción para mejorar el rendimiento significativamente, pero si en el sistema operativo ocurre algún error o falla, las últimas transacciones realizadas se pueden perder en parte o total, en el peor de los casos irrecuperable, y puede traer consigo corrupción de datos.

- **Wal_sync_method (string)**

  Método utilizado por WAL que obliga las actualizaciones en el disco, pero en caso, que el parámetro **Fsync** esté desactivado entonces no servirían para nada los ajustes realizados en este parámetro, en caso contrario los valores posibles son:

  - Open_datasync (WAL escribir archivos con open() opción O_DSYNC)
  - Fdatasync (llamada fdatasync() en cada commit)
  - Fsync_writethrough (llamada fsync() en cada commit, obligando a escribir a través de cualquier disco caché de escritura)
  - Fsync (llamada fsync() en cada commit)
  - Open_sync (WAL escribir archivos con open() opción O_SYNC)

- **Full_page_writes (Boolean)**

  Este parámetro activado, le permite al servidor escribir todo el contenido de cada página WAL a disco, durante la primera modificación de página después de un punto de control, es de suma importancia si ocurre un fallo en el sistema operativo y se encuentre en proceso la escritura de una página, entonces sólo podría ser completado en parte, y recuperan totalmente los datos después del fallo. Una forma de garantizar la recuperación de la página correctamente es mediante el aumento de la cantidad de datos que deben ser escritos en WAL, porque siempre WAL replay parte de un puesto de control, es suficiente con hacer esto durante el primer cambio de cada una de las páginas después de un puesto de control, por lo tanto una manera de reducir el costo de la página es aumentando el intervalo de los parámetros de control.

  Cuando esta opción se encuentra desactivada la velocidad de todas las operaciones serán normales, pero podría tener como consecuencia una base de datos corrupta después de la caída del sistema operativo o falta de alimentación. Es seguro desactivar esta opción

cuando se tenga hardware con un respaldo de batería controlador de disco o un software de sistema de archivo que reduzca el riesgo de la escritura parcial de la página a un nivel aceptablemente bajo.

- **Wal_buffers (integer)**

  Se le pone el número de páginas de disco en buffers de memoria compartida asignados para los datos WAL. El valor que presenta por defecto es de ocho. El aumento de este valor provoca que el gestor solicite más memoria compartida del sistema operativo que tiene configurado por defecto.

- **Commit_delay (integer)**

  Valor de tiempo en microsegundos que esperará el servidor para llevar las entradas del buffer de diario a los diarios después de que una transacción se confirme. Permite que otros procesos aprovechen esta operación, pero el retraso es desperdiciado si no hay otras transacciones en espera. El valor por defecto es el de cero, o sea sin demora.

- **Commit_siblings (integer)**

  Número mínimo de transacciones simultáneas abiertas a pedir antes de realizar la demora commit_delay.

Parámetros de los puestos de control (Checkpoint)

- **Checkpoint_segments (integer)**

  Valor de distancia máxima entre los puestos de control automático WAL, en el archivo de registro de los segmentos (cada segmento es normalmente 16 megabytes). El valor por defecto es de tres. Cuando se han llenado el número de segmentos de este parámetro, se realiza un punto de verificación y así el sistema puede reutilizar los segmentos.

- **Checkpoint_timeout (integer)**

  Valor del tiempo máximo en segundos entre los puestos de control automático WAL. El valor por defecto es 300 segundos.

- **Checkpoint_warning (integer)**

  Escribir un mensaje en el registro del servidor si un punto de verificación se lanza antes del intervalo dado en este parámetro. El valor por defecto es de 30 segundos y en caso de desactivar la alerta se le pasa el valor cero.

3.1.2.2 Buffers de diario (cache WAL)

También influyen los buffers de diario WAL, los cuales realizan las siguientes operaciones:

1. LogInsert: Un proceso servidor escribe una entrada de una operación.
2. LogFlush: Una confirmación provoca la escritura de los buffers de diario en disco.

Ejemplos de problemas más comunes que se presentan a diario y solución propuesta para resolverlos:

Problema: Cuando se intenta guardar un nuevo buffer, no existe espacio suficiente en la cache y el proceso del servidor debe esperar, las operaciones que necesita generar la nueva entrada se pararon y se quedaron en espera.

Cuando los sistemas están sobrecargados, no se escribe y no libera espacio en el buffers WAL lo suficientemente rápido.

Solución: Para estos tipos de problemas lo más común es aumentar el tamaño de la cache de WAL, incrementando el número de WAL_BUFFERS.

Problema: Que la escritura de los buffers de diario al disco no sea real porque se usan los buffers intermedios del sistema.

Solución: Esto se resolvería utilizando el método de PostgreSQL para forzar al sistema operativo que escriba en disco los buffers quedando de esta manera (parámetros FSYNC=true y WAL_SYNC_METHOD=método).

Problema: Cada commit requiere una escritura de diario

Solución: Si el sistema puede llevarse toda la información de varias transacciones en una sola escritura de diario sería lo ideal, para dicho problema se cuenta con lo siguientes parámetros a configurar:

- **Commit_delay (integer)**
  Este valor tomando un número por debajo de los 10 000 no es seguro que mejore el rendimiento del servidor, más información en el epígrafe 3.1.2.1.1.
- **Commit_Sibling (integer)**
  Información en el epígrafe 3.1.2.1.1.
- **Fsync (Boolean)**

  Información en el epígrafe 3.1.2.1.1.

3.1.2.3 Cache de la base de datos

En este aspecto lo que se quiere es minimizar el acceso a los disco, cualquier instrucción que se realice debe primero referirse a la cache en busca de los datos a utilizar y si no se encuentran se extraen de los disco y se trabajan en la cache.

Algunos de los parámetros de los cuales depende la memoria compartida de PostgreSQL se pueden visualizar en la memoria compartida de la computadora:

```
# /sbin/sysctl -A I grep shm
```

```
kernel.shmmni = 4096       #El tamaño mínimo de segmento de memoria
                             compartida (en bytes)

kernel.shmall = 2097152    #Monto total de la memoria compartida
                             disponible (bytes o páginas)

kernel.shmmax = 33554432   #Tamaño máximo del segmento de memoria
                             compartida (en bytes)
```

Para cambiar alguno de estos valores se utiliza el comando:

```
sysctl -w kernel.shmmax = nnnn
```

"nnnn": Número nuevo a asignar, el cual depende de la cantidad de *shared_buffers.* El cual se explica en el *epígrafe 3.1.5.*

Si se configura el parámetro *shared_buffers* con un valor muy alto y cuando se reinicia el servidor, el sistema muestra un error como el siguiente:

```
2008-04-22 15:02:04 CDT FATAL: no puede crear segmento de memoria compartida: argumento no
válido.

2008-04-22 15:02:04 CDT Detalle: no fue llamada al sistema shmget (key = 5432001, size = 112009216,
03600).
2008-04-22 15:02:04 CDT SUGERENCIA: Este error usualmente significa que la petición de PostgreSQL
para un segmento de memoria compartida excedió el núcleo del parámetro SHMMAX. Puede reducir el
tamaño de la petición o reconfigurar el núcleo con mayor SHMMAX. Para reducir el tamaño de la petición
(en la actualidad, 112009216 bytes), PostgreSQL reducir el parámetro shared_buffers (actualmente
13107) y / o su parámetro max_connections (actualmente 100).
Si la petición ya es pequeño de tamaño, es posible que éste es menor que el de su núcleo del SHMMIN
parámetro, en cuyo caso el aumento de la solicitud o el tamaño de la reconfiguración SHMMIN es
necesario.
La documentación de PostgreSQL contiene más información acerca de la configuración de memoria
compartida.
 No!
```

Se cambia el valor de *shmmax* por el valor del tamaño requerido por el *shared_buffer* configurado

y se reinicia el servidor.

Los desarrolladores de PostgreSQL pueden usar las vistas de rendimiento *pg_statio_** que indican para cada tabla, índice y secuencia los bloques leídos de disco y de la cache. Permite analizar y cambiar el tamaño o la forma de consultas.

Problemas más comunes en este aspecto:

- Un problema que ocurre con bastante frecuencia es que el número de bloques de información encontrado en la cache es bajo o muy bajo, y esto quiere decir que el tamaño configurado para la cache es insuficiente.
- Otro problema común es que se realizan consultas secuenciales sobre tablas muy grandes, por lo que no caben e impiden la utilización de la misma.

Soluciones para resolver dichos problemas son:

- En cuanto a la insuficiencia del tamaño de la cache, lo recomendado es aumentar o modificar el parámetro Shared_buffers, claro esto sin tener que sobrepasar los valores recomendados para la misma.
- Respecto a las consultas, lo mejor sería modificar las consultas e introducir en el sistema el uso de los índices para que pueda pasar a rastreo indexado.

3.1.2.4 Acceso a los discos

Lo primero es evitar la competición por su uso, o sea separar operaciones diferentes en discos diferentes; las operaciones de manipulación (sobre ficheros de datos y ficheros de índice) en un disco y la gestión de diarios en otro.

Segundo, realizar las tareas de mantenimiento rutinarias: REINDEX, VACUUM o mejor AUTOVACUUM.

Tercero, como debe estar organizados los datos:

La forma en que se organice el clúster también influye en el rendimiento.

La versión 8.1 en adelante usa TABLESPACE que permite dividir físicamente las bases de datos, mejorando el rendimiento.

Ejemplo:

```
$ cd /mnt/
$ mkdirhier /media/datos/datos_postgresql/data
$ psql
postgres#= CREATE TABLESPACE datos LOCATION
    '/media/datos/datos_postgresql/data';
```

Así se puede crear varias localizaciones diferentes para los datos, aunque parezca dentro del mismo clúster, porque el directorio pg_tblspc contiene enlaces simbólicos a estos directorios.

Si una fila no cabe en una página, PostgreSQL la divide y escribe una parte en una tabla TOAST (The oversized attribute storage technique). Esta tabla actúa como una extensión, manteniéndose los valores que son demasiado grandes para que estén en línea. Esta misma técnica se aplica a los índices.

Hay una tabla TOAST para cada tabla que tenga alguna fila que no quepa: pg_toast_nnnn, donde nnnn es el OID de la tabla.

3.1.3 Afinamiento de consultas

Una de las formas de hacer nuestras consultas más rápidas, es mediante el uso de vistas, por ejemplo:

Se quiere hacer una consulta que implica tres tablas, es decir se tendría que hacer un Join sobre

las mismas, de esta forma sería ineficaz. Lo más óptimo sería crear una vista, "CREATE VIEW nombre" donde se guarde en ella la consulta de las tres tablas y después sólo se tendría que consultar la vista.

Select * FROM nombrevista

Para verificar que la consulta SQL sea eficiente, PostgreSQL hace uso del EXPLAIN, este permitirá ver el plan de ejecución de una consulta y ver si se puede rediseñar de un modo más eficiente. (Ver Anexo 12)

El uso de EXPLAIN permitirá:

- Ver el plan de ejecución de consulta.
- Ver si se están usando índices.
- Usa estadísticas para generar plan óptimo.
- EXPLAIN ANALYZE muestra costo real de ejecución. (Ver Anexo 13)

3.1.3.1 Uso de Índices

Los índices son estructuras opcionales asociadas a las tablas que permiten incrementar notablemente la velocidad de ejecución de las sentencias. (Fábregas, y otros, 2007)

PostgreSQL te permite crear y borrar Índices sin necesidad de reescribir el código y es el mismo gestor el encargado del mantenimiento de los mismos.

La presencia de Índices en las tablas mejora considerablemente las operaciones de los datos. Sin embargo las operaciones de actualización e inserción se pueden ver perjudicadas con el hecho de que el índice se tiene que actualizar. Es por ello que no debe existir más de dos o tres Índices.

3.1.3.2 Cuando crear un índice

Los Índices pueden crearse sobre uno o varios campos de una tabla, la creación de un Índice único permite que no se creen duplicaciones de campos.

Cómo saber cuando hacer un Índice en una columna:

- No existe duplicidad de valores (el mejor caso). Por ejemplo, llave primaria.
- El campo toma un amplio rango de valores. Por ejemplo el código de cliente en una tabla de pedidos. Una mala elección sería el campo sexo que sólo toma dos valores.
- El campo es usualmente nulo y con frecuencia se seleccionan aquellas filas que tienen un valor. (Fábregas, y otros, 2007)

PostgreSQL antes de crear el índice, hace un poco de heurística para determinar qué tipo de estructura de datos utilizará y que sea óptima para esa columna. Si la información de la base de datos se consultará con frecuencia sobre un atributo, ese es el candidato ideal para ser indexado. En el caso de una llave primaria, PostgreSQL automáticamente crea el índice.

La mejor manera de saber en qué atributos conviene tener índices es correr un VACUUM VERBOSE ANALYZE sobre la base de datos, para actualizar las estadísticas de las tablas y después utilizar la instrucción EXPLAIN sobre las consultas que serán las más frecuentes:

```
SIGIA=> VACUUM VERBOSE ANALYZE SIGIA;
SIGIA=> EXPLAIN SELECT cod_producto FROM producto WHERE marca = 'adidas';
```

De esta manera saber cuál es el costo de cuando se realiza una consulta sobre un campo indexado y uno que no esté indexado.

A continuación se muestra un ejemplo de cómo crear un Índice:

```
CREATE UNIQUE INDEX index_ejemplo
  ON artículo
  USING btree
  (id_art);
```

Para eliminar un índice:

```
DROP INDEX index_ejemplo;
```

Cambiar la definición de un índice:

```
ALTER INDEX nombre
RENAME A new_name
ALTER INDEX nombre SET TABLESPACE tablespace_name
```

ALTER INDEX: Cambios a la definición de un índice.

RENOMBRAR: La forma RENAME cambia el nombre del índice.
SET TABLESPACE: Realiza cambios en el índice de tablas y mueve el archivo de datos asociado con el índice a las nuevas tablas.
Nombre: El nombre de un índice a alterar.
New_name: Nuevo nombre para el índice.
Tablespace_name: Las tablas a las que el índice se moverá.

3.1.3.3 Tipos de índices

El definir índices acelerará el rendimiento de las búsquedas siempre que la consulta tenga condiciones asociadas a los índices. Existen varios tipos de índice:

- árbol-B (BTREE): Valor por defecto, admiten varios predicados (>, <, =, LIKE, etc), multicolumna y unicidad.

- árbol-R (RTREE): Indicado para datos espaciales.
- Hash: Sólo admite "=", mal rendimiento.
- GiST: Índice btree que puede ser extendido mediante la definición de nuevos predicados de consulta, también es multicolumna.

```
// un btree
CREATE INDEX mitabla_idx ON tabla (edad);
// un HASH
CREATE INDEX mitabla_idx ON tabla USING HASH (edad);
```

3.1.3.4 Rendimiento de los índices

Se trata de agilizar el acceso a los datos:

- En las búsquedas: Cuando en la condición de la instrucción interviene alguna columna, que es utilizada en algún índice, el sistema agiliza la búsqueda de la información cambiando a rastreo indexado sobre este índice.
- En las ordenaciones: Cuando la instrucción exige ordenar la información, el sistema comprueba si existe algún índice que le permite devolver la información ya ordenada.

En las búsquedas, conforme la condición tenga más columnas de la clave del índice y su uso coincida con su orden, su utilización será más eficiente, es decir, las mejoras consisten en modificar la condición y/o crear nuevos índices.

Para las ordenaciones: Si el orden físico de almacenamiento de las filas de la tabla coincide con el orden del índice, el rendimiento sube mucho debido al mayor éxito de localizaciones en la cache, así las soluciones podrían ser:

- Recrear la tabla con ese orden usando "CREATE TABLE... AS SELECT ... ORDER BY",

"DROP TABLE", "ALTER TABLE" ... "RENAME".

- Crear un clúster de tabla con "CLUSTER [[índice ON] tabla]" que reordena físicamente las filas de la tabla según el orden del índice (bloqueo exclusivo en lectura y escritura) y el sistema recuerda la forma de agrupar la tabla.

3.1.3.5 Reconstrucción de índices

Otra tarea importante es la reconstrucción de los índices, debido que puede haber índices incorrectos por problemas en el software o hardware, o porque haya que optimizar el índice debido a que tiene páginas muertas que son necesarias eliminar.

Usando el comando REINDEX se reconstruye un índice utilizando los datos almacenados en el índice de la tabla.

Existen varios escenarios en los que se puede usar:

- Un índice se ha corrompido, y no contiene datos válidos debido a errores de software o hardware. REINDEX proporciona un método de recuperación.
- Un índice se ha convertido en "hinchado", cuando contiene muchos vacíos esto puede ocurrir con índices B-tree, en determinadas modalidades de acceso poco frecuente. REINDEX proporciona una manera de reducir el espacio de consumo del índice.
- Se han alterado los parámetros de almacenamiento por un índice, y se desea saber que cambio ha tenido vigor.

Desde el SO:

```
reindexdb [connection-option...] [--table | -t table ] [--index | -i index ]
[dbname]
reindexdb [connection-option...] [--all | -a]
reindexdb [connection-option...] [--system | -s] [dbname]
```

Desde SQL:

```
REINDEX {INDEX | TABLE | DATABASE | SYSTEM} name [FORCE]
```

Parámetros

INDEX: Recrear el índice especificado.

TABLE CUADRO: Recrear todos los índices de la tabla especificada.

BASE DE DATOS: Recrear todos los índices dentro de la base de datos actual.

SISTEMA: Recrear todos los índices del sistema de catálogos dentro de la actual base de datos.

Nombre: El nombre del índice específico en el cuadro, o la base de datos que se reindexa.

Se pueden reindexar las bases de datos completas, tablas, índices o el catálogo del sistema.

Este operador bloquea las tablas en modo exclusivo cuando las reindexa.

Ejemplos:

```
-- reparar un índice:
reindex index nomindice;
-- reparar todos los índices de una tabla:
reindex table nomtabla;
-- reparar los índices del catálogo de base de datos, para esto
-- hay que levantar postmaster sin que use los índices y luego reindexar
$ export PGOPTIONS="-P"        // significa lo mismo que $ postmaster –P
$ psql bd
bd=# reindex database bd;
-- reparar los índices del catálogo compartido (pg_database, pg_group, pg_shadow,
-- etc.). No se debe levantar postmaster si no que hay que arrancar un proceso
-- autónomo:
$ postgres –D $PGDATA –P prueba
backend> reindex index indice_catalogo_compartido
```

Se puede introducir en el cron que realice ciertas operaciones de mantenimiento:

```
$ crontab –e
// se abre el fichero de cron y se añaden estas líneas:
0 3 * * * psql -c 'VACUUM FULL;' test
0 3 * * * vacuumdb -a –f
0 30 * * * reindexdb bd
```

Nota: Cuando se reindexa, se rehacen muchos ficheros, por lo que hay que tener presente si el archivado WAL está activado, pues una reindexación de toda la base de datos implica que se puede duplicar el tamaño de los ficheros y llenar el disco duro.

3.1.3.6 Ejecución de consultas

En la ejecución de consultas participan tres elementos:

Analizador:

- Análisis sintáctico y semántico.
- Generación del árbol de análisis.

Planificador:

- Generación de planes de ejecución.
- Plan de operadores: Rastreos secuenciales, rastreos indexados, métodos de concatenación de tablas y ordenaciones.

Optimizador:

- Búsqueda del plan más económico.
- Estimación del coste de cada operador.
- Se puede influir en los costes asignados a cada operador.

Mediante EXPLAIN se puede ver el plan de ejecución de una consulta, y ver si se puede

rediseñar de un modo mejor, al ejecutar explain sobre una consulta permitirá:

- Ejecutando comandos "SET enable_* TO off", permite averiguar qué haría en otro caso.

3.1.3.7 Tipos de rastreo que usa PostgreSQL

Rastreo secuencial (SEQ SCAN): Revisa la tabla de principio a fin, evalúa la condición de la consulta para decidir si incorporar la fila al resultado.

Rastreo indexado (INDEX SCAN): Utiliza un índice de la tabla implicada, no tiene que revisar todas las filas de la tabla, Pero algunos bloques (páginas) pueden ir y venir varias veces, si la tabla está muy desordenada según el índice (no se ha hecho un CLUSTER recientemente), el resultado es ordenado según el orden del índice.

Sólo se utiliza en índices árbol -B, árbol -R y GiST. El planificador/optimizador utiliza este rastreo cuando puede reducir el número de filas a revisar o la necesidad de ordenar el resultado.

Rastreo por TID (identificador de tupla) (TID SCAN): TID (corresponde a la pseudo columna **ctid**) está compuesto del bloque donde se encuentra la tupla y el número de la misma dentro del bloque. Sólo se puede utilizar dentro de una transacción. Es la forma más rápida de localizar una tupla.

3.1.3.8 Operadores utilizables en consultas de PostgreSQL

Operadores LIMIT. y OFFSET (LIMIT):

```
SELECT ... OFFSET x LIMIT y;
```

Añadir (APPEND): Este operador se usa para implementar UNION. Devuelve todas las filas del primer conjunto, luego las del segundo, y así hasta las del último conjunto. Su coste es la suma de los costes de obtener los conjuntos orígenes. Interrelación con la herencia.

Resultado (RESULT): Para la obtención de cálculos intermedios no relacionados con la

consulta, evalúa partes o condiciones constantes llamadas a funciones numéricas (no agregadas).

Bucle anidado (NESTED LOOP): Se utiliza para la realización de la concatenación entre tablas. Para cada fila de la tabla externa busca las filas de la tabla interna que cumplan la condición de concatenación.

Concatenación por fusión (MERGE JOIN): Para la realización de la concatenación entre tablas. Los dos conjuntos deben estar ordenados por la columna de la concatenación. Utiliza el hecho de estar ordenados los conjuntos para realizar la concatenación. Se usa para todo tipo de joins y para la unión.

Hash y concatenación hash (HASH y HASH JOIN): Se usan combinadamente para la realización de la concatenación entre tablas. Empieza utilizando el operador hash sobre la tabla interna que crea un índice hash temporal. La concatenación comienza tomando una fila de la tabla externa y mediante el uso del índice hash alguna fila de la tabla interna que combine. Se usa para joins internos, joins por la izquierda y unión.

Agrupar (GROUP): Se usa para realizar la agrupación en las consultas, dependiendo de si se van a realizar cálculos agregados, se añade una fila con valores nulos a los grupos formados como separador.

Materializar (MATERIALIZE): Se usa con algunas operaciones de subconsultas. El planificador/optimizador puede decidir que es más económico materializar la subconsulta, que repetirla varias veces (suele pasar en la concatenación por fusión).

Operadores conjuntistas (SETOP): Intersección, intersección total, diferencia total. Requieren dos conjuntos, que combinan en una lista ordenada y luego identifican grupos de filas idénticas (se contabilizan cuántas y de qué conjunto provienen). Posteriormente, dependiendo del operador se determina cuantas filas pasan al resultado final. Se usa para INTERSECT, INTERSECT ALL, EXCEPT y EXCEPT ALL.

3.1.3.9 Ajuste de rendimiento de consultas

Parámetros que son configurables en el archivo *postgresql.conf*, permitiendo el uso de los operadores antes mencionados.

Método de planificación y optimización:

- **enable_bitmapscan:** Habilita o deshabilita el uso de tipos de planes de escaneo en bitmaps.
- **enable_hashagg:** Habilita o deshabilita el uso de tipos de planes de agregación por dispersión.
- **enable_hashjoin:** Habilita o deshabilita el uso de tipos de planes concatenación.
- **enable_indexscan:** Habilita o deshabilita el uso de planes de recorrido en índice.
- **enable_mergejoin:** Habilita o deshabilita el uso de tipos de planes de concatenación por fusión.
- **enable_nestloop:** Habilita o deshabilita el uso de tipos de planes de concatenación de bucles anidados.
- **enable_seqscan:** Habilita o deshabilita el uso de tipos de planes de recorrido secuencial (full scan).
- **enable_sort:** Habilita o deshabilita el uso de pasos de ordenación explícitos.
- **enable_tidscan:** Habilita o deshabilita el uso de planes del rastreo TID en el planificador. Se usa un rastreo TID cuando la pseudo-columna CTID aparece en un WHERE CTID es la localización física de una fila.

3.1.4 Vacuum

Para optimización de las Bases de Datos en PostgreSQL mediante el uso del proceso vacuum, lo primero es saber qué significa y para qué se emplea, para tal propósito se dan las siguientes

respuestas a continuación:

El proceso que realiza la limpieza de las bases de datos en PostgreSQL se denomina vacuum.

Vacuum es un proceso de limpieza que se encarga de:

- Recuperar el espacio de disco ocupado por filas modificadas o borradas.
- Actualizar las estadísticas usadas por el planificador / optimizador.

La frecuencia con que se ejecute este proceso depende de cada instalación y, además, se puede ejecutar en paralelo con operaciones de actualización pero no de definición, se puede ejecutar desde el SO o desde SQL. (Ver Anexo 14)

3.1.4.1 Vacuum configurado desde el sistema operativo

```
vacuumdb [OPCIÓN]... [BASE-DE-DATOS]
Opciones:
 -a, --all                       limpia todas las bases de datos
 -d, --dbname=BASE               base de datos a limpiar
 -t, --table='TABLE [(COLUMNS)]' limpiar sólo esta tabla
 -f, --full                      usar «vacuum full»
 -z, --analyze                   actualizar las estadísticas
 -e, --echo                      mostrar los comandos enviados al servidor
 -q, --quiet                     no desplegar mensajes
 -v, --verbose                   desplegar varios mensajes informativos
 --help                          mostrar esta ayuda y salir
 --version                       mostrar el número de versión y salir
Opciones de conexión:
 -h, --host=ANFITRIÓN            nombre del servidor o directorio del socket
```

-p, --port=PUERTO	puerto del servidor
-U, --username=USUARIO	nombre de usuario para la conexión
-W, --password	preguntar la contraseña

3.1.4.2 Vacuum configurado desde sentencias SQL

```
VACUUM [FULL | FREEZE] [VERBOSE] [table]
VACUUM [ FULL | FREEZE ] [ VERBOSE ] ANALYZE [ table [ (column [, ...] ) ] ]
```

Para obtener detalladamente cómo funciona este, consultar "man vacuum":

- Desde el sistema operativo se puede hacer un VACUUM de todas las bases de datos.
- Sin tabla: Se realiza para toda la Base de Datos.
- Sin FULL: sólo reclama espacio y lo deja libre para nuevas filas, lo agrega al FSM (Free Space Map)
- FULL: mueve filas entre bloques para compactar la tabla. Se exige bloquear exclusivamente la tabla.
- ANALYZE: es una limpieza junto con una actualización de las estadísticas.
- FREEZE: tarea de bajo nivel, se hace para evitar el problema del rehúso del identificador de transacción, sólo se hace en bases de datos que se limpiaron hace mucho tiempo.

Recomendaciones:

- Usar VACUUM FULL si alguna tabla ha cambiado mucho y no va a crecer en el futuro.
- Ejecutar VACUUM diariamente en todas las bases de datos.
- Hacerlo con más frecuencia sobre tablas con mucha carga de actualización.
- Determinar las tablas grandes con pocas actualizaciones para eliminarlas del VACUUM.

Dentro del VACUUM, se pueden actualizar las estadísticas con la opción ANALYZE, él cual proporciona:

- Información usada por el optimizador.
- No es muy pesado el proceso porque se realiza sobre una muestra al azar.
- Necesario si la distribución de los datos cambia mucho.

3.1.4.3 AutoVacuum

Para evitar los problemas con la ejecución de vacuum, PostgreSQL 8.1 incorpora un proceso de fondo AUTOVACUUM que se ejecuta periódicamente cuando está habilitado, él mismo comprueba las tablas con alta carga de actualizaciones. Hay una serie de parámetros en *postgresql.conf* para modificar el comportamiento de este proceso.

3.1.4.3.1 Parámetros de configuración de AutoVacuum

Los parámetros a continuación se configuran manualmente desde el archivo *posgresql.conf* y en el servidor de inicio.

- **Autovacuum (Boolean):** Controla si el servidor debe comenzar el subproceso autovacuum. Stats_start_collector y stats_row_level también debe ser para esto en principio.
- **Autovacuum_naptime (integer):** Valor que especifica el retraso en segundos entre la actividad de las rondas del subproceso autovacuum. En cada ronda el subproceso examina una Base de Datos y las cuestiones Vacuum y comandos Analyze como sea necesario para las tablas en la base de datos, y el valor por defecto es 60.
- **Autovacuum_vacuum_threshold (integer):** Valor que especifica el número mínimo de tuplas a actualizarse antes de hacer Vacuum. El valor por defecto es de 1000. Esta configuración puede ser anulada por los cuadros individuales de las entradas en pg_autovacuum.
- **Autovacuum_analyze_threshold (integer):** Valor que especifica el número mínimo de tuplas a actualizar antes de hacer un ANALYZE en las tablas. El valor por defecto es 500.

Esta configuración puede ser anulada por los cuadros individuales de las entradas en pg_autovacuum.

- **Autovacuum_vacuum_scale_factor (floating point):** Valor que especifica una fracción del tamaño de la tabla para añadir a la hora de decidir si Autovacuum_vacuum_threshold para provocar una Vacuum. El valor por defecto es de 0,4. Esta configuración puede ser anulada por los cuadros individuales de las entradas en pg_autovacuum.
- **Autovacuum_analyze_scale_factor (floating point):** Valor que especifica una fracción del tamaño de la tabla para añadir a la hora de decidir si Autovacuum_analyze_threshold a desencadenar una Analyze. El valor por defecto es de 0,2. Esta configuración puede ser anulada por los cuadros individuales de las entradas en pg_autovacuum.
- **Autovacuum_vacuum_cost_delay (integer):** Valor que especifica el costo demora valor que se utilizará en las operaciones automáticas Vacuum. Si se especifica -1 (que es el valor por defecto), se utilizará Vacuum_cost_delay. Esta configuración puede ser anulada por los cuadros individuales de las entradas en pg_autovacuum.
- **Autovacuum_vacuum_cost_limit (integer):** Valor que especifica el número límite de los costos que se utilizarán en las operaciones automáticas Vacuum. Si se especifica -1 (que es el valor por defecto), se utilizará Vacuum_cost_limit. Esta configuración puede ser anulada por los cuadros individuales de las entradas en pg_autovacuum. (Ver Anexo 15)

3.1.5 Tuning en general

Existen una serie de parámetros que afectan directamente el rendimiento del servidor, a continuación se muestra un resumen de los más importantes en la optimización y que dependiendo de los valores que se les imponga se vera el resultado para bien o para mal.

- **Max_connections (integer):** Valor del número máximo de conexiones simultáneas a la Base de Datos. El valor por defecto es de 100.

- **Shared_buffers (integer):** Valor del número de buffers de memoria utilizada por el servidor de datos. El valor por defecto es de 1 000, el mínimo es de 2 x max_connections. Esta opción es una de las formas más fácil de mejorar el rendimiento en el servidor, es aconsejable estar entre el rango de 6-15% de la memoria RAM disponible en el sistema, y no excederse de los 20 000 *shared_buffers* porque puede degradar el rendimiento. Este valor se puede calcular de la siguiente forma:
  (% de la RAM)/8, este valor debe darse en KB.

- **Work_men (integer):** Valor de la cantidad de memoria total que se usará para operaciones de ordenación antes de usar ficheros temporales en disco. El valor por defecto es de 1024kb. Se debe usar entre el 2-4% de la memoria total con que cuenta el sistema.
- **Fsync (Boolean):** Este parámetro está explicado anteriormente en el *epígrafe 3.1.2.1.1*

- **Commit_delay (integer) y Commit_siblings (integer):** Estas opciones son utilizadas en conjunto para ayudar a mejorar el rendimiento. Más información en el *epígrafe 3.1.2.1.1.*

- **Efecctive_cache_size (integer):** Valor del tamaño efectivo de la cache de disco disponible que ayuda el optimizador de PostgreSQL para realizar un rastreo con índice. El valor por defecto es de 1 000, al aumentar este valor, elevaría la probabilidad de rastreo por índice, en caso contrario será secuencial. Esto se mide en páginas de disco, que normalmente son 8kb cada uno.

Es importante no dejar pasar por alto que muchas de estas opciones consumen memoria compartida y que lo más seguro es que sea necesario aumentar la cantidad que trae por defecto el sistema para que se pueda obtener un optimo rendimiento de la configuración efectuada en el archivo de configuración de PostgreSQL.

3.2 Administración mediante el cliente pgAdmin3.

El máximo exponente de administración para PostgreSQL es el cliente gráfico pgAdmin3. En

pgAdmin3 se puede ver y trabajar con casi todos los objetos de la Base de Datos, examinar sus propiedades y realizar tareas administrativas.

Una característica interesante de pgAdmin3 es que, cuando se realiza alguna modificación en un objeto, escribe las sentencias SQL correspondientes, lo que da la ventaja de ser una herramienta muy útil y a la vez didáctica. (Ver Anexo 16)

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

Todas estas características hacen de pgAdmin3 una de las mejores herramientas gráfica para trabajar con PostgreSQL, tanto desde el punto de vista del usuario como del administrador.

3.2.1 Administración de seguridad

Desde pgadmin3 los administradores autorizados pueden dar de alta, borrar roles y grupos de roles, se puede cambiar la contraseña y permisos mediante una interfaz visual sencilla y fácil de entender. (Ver Anexo 17)

La sección de seguridad tiene dos apartados:

Grupos de roles: Permite la creación de grupos mediante la interfaz gráfica.

Roles de usuarios: Permite la creación de roles y asignación de privilegios utilizando una interfaz gráfica.

3.2.2 Copia de seguridad, Restauración y Mantenimiento

Las copias de seguridad y restauración de la Base de Datos se pueden realizar de forma manual mediante el uso de una interfaz gráfica. (Ver figura 3.1, 3.2, 3.3)

Copia de seguridad

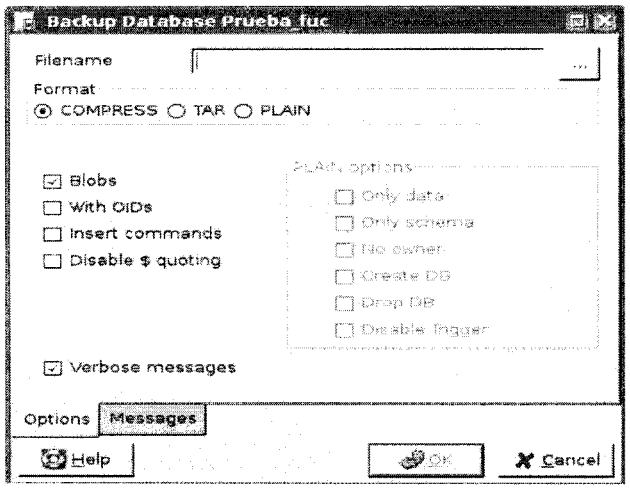


Fig 3. 1 Interfaz gráfica para copia de seguridad.

Restauración

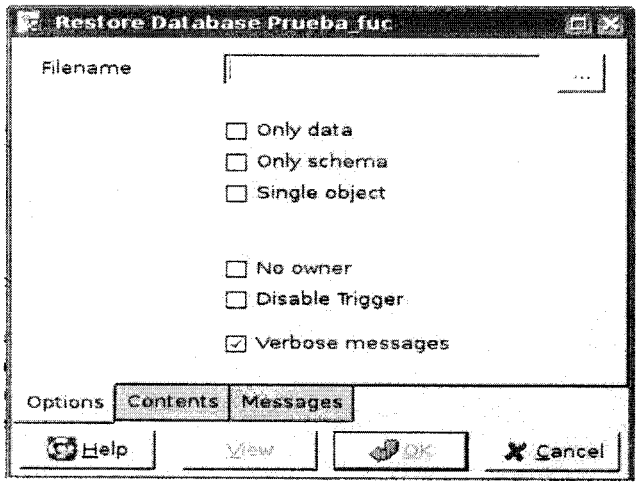


Fig 3. 2 Interfaz gráfica para restauración.

Mantenimiento

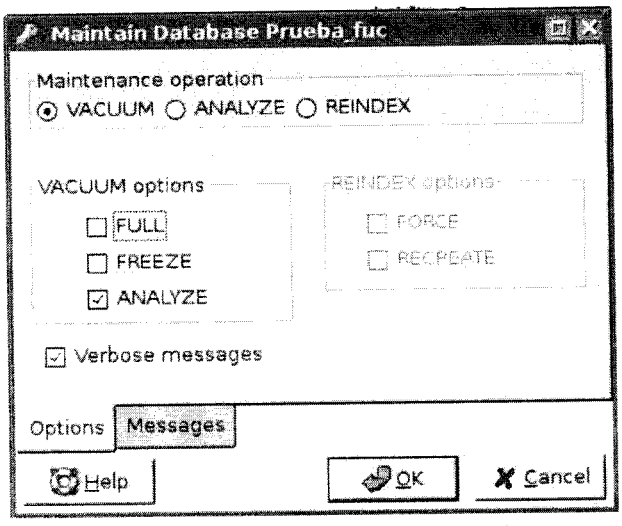


Fig 3. 3 Interfaz gráfica para Mantenimiento

PgAdmin3 trae una herramienta integrada llamada pgAdmin3 Query, la cual tiene gran ventaja, que realiza tanto consultas SQL como la creación de funciones en el lenguaje plpgsql (lenguaje interno de PostgreSQL). (Ver Anexo 18)

Lo más importante de pgadmin3 es que evita el uso de lenguajes SQL pudiendo realizar las tareas mediante una interfaz gráfica.

3.2.3 Monitorización de recursos

Los recursos, como son la CPU y memoria deben ser monitorizados constantemente para evitar que se llene el segmento de memoria compartida que tiene PostgreSQL en la RAM. Con el uso del comando EXPLAIN aplicado a las consultas se puede ver el rendimiento.

Por tanto, se puede llegar a la conclusión de que el cliente pgAdmin3 ofrece una herramienta con grandes ventajas para la administración y monitorización, proporciona una mayor calidad y de

una manera fácil la administración del Sistema Gestor de Base de Datos PostgreSQL.

PostgreSQL posee algunas funciones muy útiles para determinar el tamaño en disco que ocupa una tabla o una Base de Datos. Estas son:

pg_start_backup(etiqueta)
pg_stop_backup()
pg_column_size(nombre_columna)
pg_tablespace_size(nombre)
pg_database_size(nombre)
pg_relation_size(nombre)
pg_total_relation_size(nombre)
pg_size_pretty(bigint)
pg_ls_dir(diretorio)
pg_read_file(archivo texto, offset bigint, tamaño bigint)
pg_stat_file(archivo texto)

3.3 Propuesta de configuración del archivo postgresql.conf

Para tener una información más detallada (Ver Anexo 19)

```
# CONNECTIONS AND AUTHENTICATION
#- Connection Settings -
max_connections = 100
superuser_reserved_connections = 3
unix_socket_directory = '/var/run/postgresql'
# - Security & Authentication -
authentication_timeout = 15
# RESOURCE USAGE (except WAL)
```

```
# -Memory-
shared_buffers = 13107                  # min 16 o max_connections*2,

                                        #recomendable usar e/ el 6-15% de la
                                        #RAM

work_mem = 41943                        #4% de la RAM

maintenance_work_mem = 65536

max_stack_depth = 4090                  #no puede excederse del total del límite
                                        #del kernel, que se puede ver usando #(ulimit -s)

# -Cost-Based Vacuum Delay-
vacuum_cost_delay = 0
vacuum_cost_limit = 1000
# WRITE AHEAD LOG
# -Settings-
fsync = on
wal_sync_method = fsync
full_page_writes = on
wal_buffers = 64
commit_delay = 10000
commit_siblings = 5
# -Checkpoints-
checkpoint_segments = 3
checkpoint_timeout = 600
checkpoint_warning = 60
 QUERY TUNING
-Planner Method Configuration-
enable_hashagg = on
```

enable_hashjoin = on

enable_indexscan = on

enable_mergejoin = on

enable_nestloop = on

enable_seqscan = on

enable_sort = on

-Planner Cost Constants-

effective_cache_size = (50% RAM) #recomendable usar un número menor al 2/3 de la RAM

-Genetic Query Optimizer-

geqo = on

geqo_threshold = 16

geqo_effort = 5

geqo_pool_size = 0

geqo_generations = 0

geqo_selection_bias = 2.0

RUNTIME STATISTICS

-Query/Index Statistics Collector-

stats_start_collector = on

stats_row_level = on

AUTOVACUUM PARAMETERS

autovacuum = on

autovacuum_naptime = 120

autovacuum_vacuum_threshold = 2000

autovacuum_analyze_threshold = 1000

autovacuum_vacuum_scale_factor = 0.4

autovacuum_analyze_scale_factor = 0.2

autovacuum_vacuum_cost_delay = -1

autovacuum_vacuum_cost_limit = -1

Conclusiones del capítulo

En este capítulo se ha abordado dos temas fundamentales como la optimización y administración desde un tipo de cliente como pgadmin3 del Gestor de Base de Datos PostgreSQL.

A partir de la investigación realizada para resolver el problema que acechaba constantemente se percató de la mala utilización que se le daba a PostgreSQL, como la no utilización de las potencialidades que brinda con respecto a la seguridad, administración de la información, al mantenimiento del gestor.

CONCLUSIONES GENERALES

Con la terminación de este trabajo se puede plantear que se le ha dado cumplimiento a todas las tareas investigativas:

Se decidió por la Arquitectura de Base de Datos Descentralizada como solución a la implantación del ERP cubano.

Se realizó una propuesta de seguridad, resguardo y recuperación de la información en la Base de Datos para un sistema descentralizado con el objetivo de proteger la información contra errores del sistema operativo.

Como Sistema Gestor de Base de Datos (SGBD) para una arquitectura descentralizada se describió el empleo de PostgreSQL. Lograndose responder a las necesidades de alta disponibilidad y bajo costo.

Como cliente para la administración se propuso pgAdmin3. Se expusieron sus principales características y funcionalidades, las cuales lo ponen como mayor exponente para la administración de PostgreSQL.

Se propone una solución de replicación de cada información que se gestiona en las entidades afiliadas a la Base de Datos Central, que permiten mantener la información actualizada en todos los niveles.

Para un mejor rendimiento y optimización del SGBD se realizó una propuesta de optimización para así lograr que el mismo este al máximo nivel de prestaciones.

RECOMENDACIONES

En el desarrollo del trabajo existen algunos temas que no han sido analizados con la profundidad requerida a pesar de poseer marcada importancia. Por lo que se recomienda un conjunto de ideas para darle continuidad al trabajo:

1. Extender la propuesta utilizando un servidor dedicado con prestaciones superiores a las utilizadas en el trabajo.
2. Implementar la propuesta en el proyecto ERP.
3. Investigar y proponer una solución de replicación con la herramienta PgCluster, para con ello realizar réplicas multi-master y mejorar el rendimiento de actualización de los datos a la Base de Datos Central.
4. Diseñar e implementar un herramienta que se integre con PostgreSQL, para llevar el control y monitoreo de los datos, lo que conllevará a un incremento en la eficiencia de la administración y monitoreo de los mismos.
5. Diseñar e implementar una herramienta con una interfaz gráfica amigable que realice las configuraciones de seguridad, optimización, copia de seguridad y restauración de la Base de Datos.

BIBLIOGRAFÍA

1. **AB, MySQL. 1995-2008 .** Sitio Oficial de MySQL . [En línea] 1995-2008 . http://www.mySQL.com.

2. **Alarcón, Jose Manuel. 2006.** Administración de SGBD PostgreSQL. [En línea] Octubre-Noviembre de 2006. [Citado el: 4 de Enero de 2008.] http://www.google.com.cu/search?hl=es&q=administracion+sgbd+postgresql&btnG=Buscar&meta=.

3. **Alburquerque, Amado. 2007.** *Sistema Integrador de Gestión Estadística(SIGE).* Universidad de las Ciencias Informaticas. Ciudad Habana : Universidad de las Ciencias Informaticas, 2007.

4. **Bernadí, Xavier.** ¿Que son los Triggers y como usarlos en MySQL 5.0? [En línea] http://www.webtaller.com/construccion/lenguajes/mysql/lecciones/que-son-los-triggers-como-usarlos-mysql-5.0.php.

5. **Browne, Christopher. 2004-2006.** Slony-I, 1.2.13 Documentación . *El grupo de desarrollo global de PostgreSQL.* [En línea] 2004-2006. http://slony.info/.

6. **Camacho, M. Ávila.** Solución para Centros de Datos Siemon 10G ip™. *Sitio de SIEMON.* [En línea] http://www.siemon.com/la/white_papers/SD-03-06-Centros-de-Datos.asp.

7. **Correa, Ramírez. 2004.** *Rol y Contribución de los Sistemas de Planificación de los Recursos de la Empresa ERP.* Universidad de Sevilla. Sevilla : Universidad de Sevilla, 2004.

8. **CyberCluster, Grupo. 2008.** PostgreSQL productos - Cybercluster. [En línea] 2008.

9. **de Lorenzo, Jorge.** Tecnologías para empresas. [En línea] http://www.microsoft.com/spain/empresas/tecnologia/base_datos_empresa.mspx.

10. **Fábregas, Yuniesky y Fernández, Daniel. 2007.** *Administración, configuración y optimización de un Sistema de Base de Datos Descentralizado en Oracle Database 10g release 2* . Ciudad Habana : Universidad de las Ciencias Informaticas, 2007.

11. **Fajardo, Norge. 2007.** *Sistema de replicación para base de datos distribuida en PostgreSQL.* Universidad de las Ciencias Informaticas. Ciudad Habana : s.n., 2007.

12. **Fonseca, Roberto Andrea. 2002.** Programación de funciones en PL/pgSQL para PostgreSQL. *ABL Consultores S.A de C.V.* [En línea] 8 de Febrero de 2002. http://foobar.cl/~chepito/software/PL_pgSQL.pdf.

13. **Giraldo, Luis y Zapata, Yuliana. 2005.** Herramienta de desarrollo de ingenieria de sw para linux. [En línea] 24 de Septiembre de 2005. http://hugolopez.phi.com.co/docs/download/file=Giraldo-Zapata-Herramientas%20de%20ISW.pdf,_id=17.

14. **Godhino, Rogel. 2006.** Definición de Centro de Datos . *DataCenter.com .* [En línea] 20 de Abril de 2006. http://searchdatacenter.techtarget.com/sDefinition/0,,sid80_gci332661,00.html.

15. **González, Gustavo, y otros. 2003.** Técnicas avanzadas para la gestión de sistemas de información. *Universidad de la Republica Oriental de Uruguay.* [En línea] 2003. http://www.fing.edu.uy/inco/cursos/tagsi/Trabajos/2003/TAGSI03-TareaTecnSI-grupo4.pdf.

16. **Herrarte, Sánchez. 2005.** Triggers en PL/SQL. [En línea] 28 de Junio de 2005. http://www.devjoker.com/asp/ver_contenidos.aspx?co_contenido=54.

17. **Herrera, Juan Manuel. 2008.** La labor del adminidtrador de base de datos. *Informatizate.* [En línea] 10 de Enero de 2008. http://www.informatizate.net/articulos/la_labor_del_administrador_de_base_de_datos_parte_01_14062004.html.

18. **Lemus, Juan Manuel.** Modelado de datos e implementación de la base de datos. *Maestro del Web.* [En línea] http://www.maestrodelweb.com/editorial/modelado-de-dato-implementacion-de-la-base-de-datos-primer-nivel-ls/.

19. **Márquez, Marche. 2002.** Base de datos Orientado a Objetos. *Diseño de Base de Datos.* [En línea] 12 de abril de 2002. http://www3.uji.es/~mmarquez/e16/teoria/cap2.pdf .

20. **Márquez, María Mercedes. 2001.** Historia de los Sistemas de Base de Datos. [En línea] 12 de febrero de 2001. http://www3.uji.es/~mmarquez/f47/apvn/node6.html.

21. **Martoq, aldrin. 2000.** ¿Porque linux?, Una visión global de linux. [En línea] 2000. http://2000.encuentrolinux.cl/documentacion/amartoq/index.html.

22. **Moraga, M. Angeles. 2001.** El Modelo de datos Jerárquico. *Escuela Superior de Informática.* [En línea] 26 de abril de 2001. http://alarcos.inf-cr.uclm.es/doc/trab/T0001_mamoraga.pdf.

23. **Oracle.** Sistema ERP, Oracle. *Sitios oficial de Oracle.* [En línea] http://www.oracle.org.

24. **Pachón, Adolfo.** PL/pgSQL SQL Procedural Language. [En línea] http://www.org/traducciones/postgresql-devldoc/node3.htnl.

25. **pgadmin, Grupo.** PgAdmin PostgreSQL Tools. [En línea] http://www.pgadmin.org/.

26. **PgCluster, Grupo de.** Sitio de PgCluster. [En línea] http://pgcluster.projects.postgresql.org/.

27. **phpPgAdmin, Grupo. 2008.** phpPgAdmin, versión 4.0. *Sitio Linuxparty Group* . [En línea] 17 de Noviembre de 2008. http://www.linux-party.com/modules.php?name=News&file=article&sid=19.

28. **Racciatti, Hernán Marcelo. 2005.** Seguridad base de datos. [En línea] 26 de Octubre de 2005.

29. **Ramírez, Patricio.** [En línea]. **2004.** *Rol y contribución de los sistemas de planificación de los recursos de la empresa ERP* . Departamento de administración de empresas y comercialización e investigaciones del mercado, Universidad de Sevilla . Sevilla : s.n., 2004.

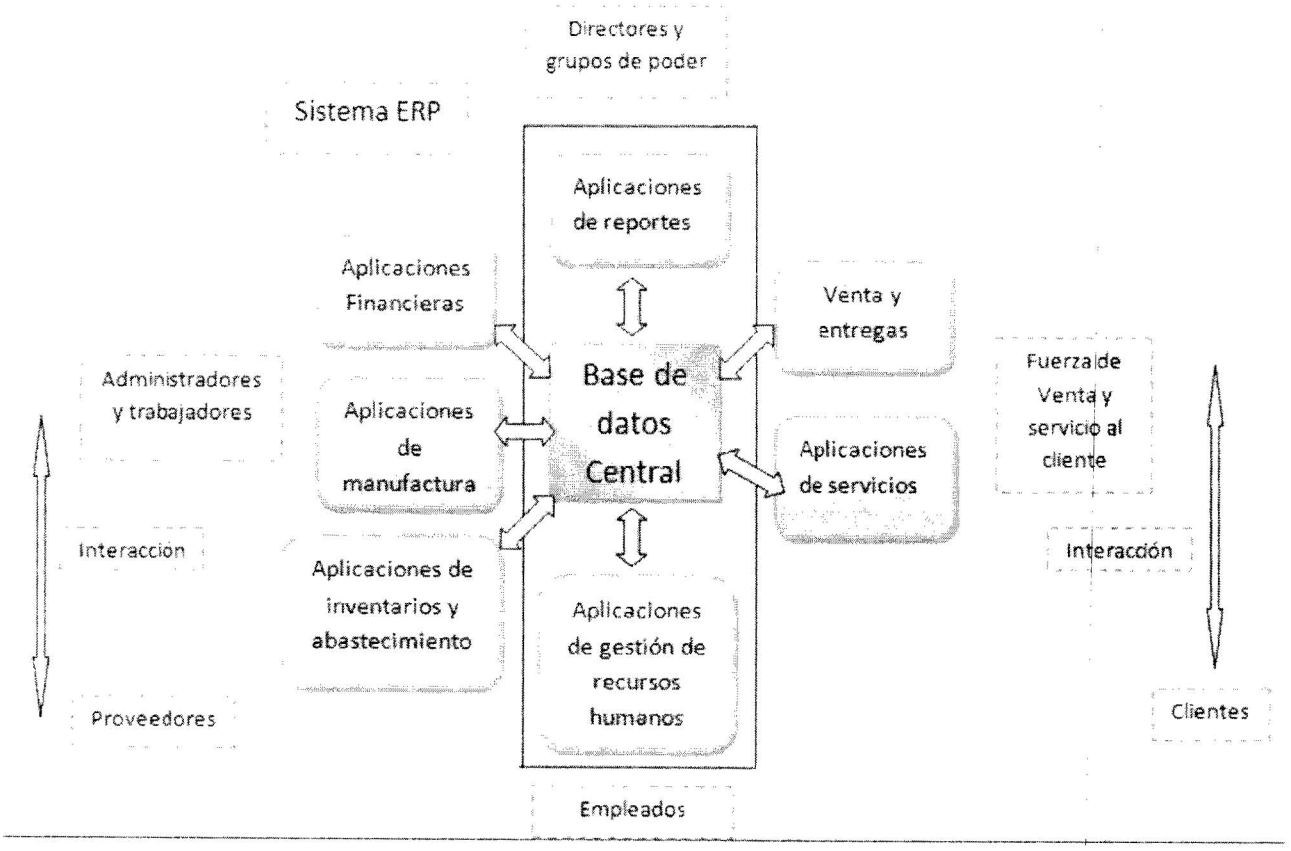
30. **Rojo, Iñaki. 1999.** Open Source: Los programas integros. [En línea] 16 de Octubre de 1999. [Citado el: 2 de Febrero de 2008.] http://www.baquia.com/com/legacy/8512.html.

31. **2005.** Software libre. *Hispalinux.* [En línea] 19 de Marzo de 2005. http://www.hispalinux.es/softwarelibre.

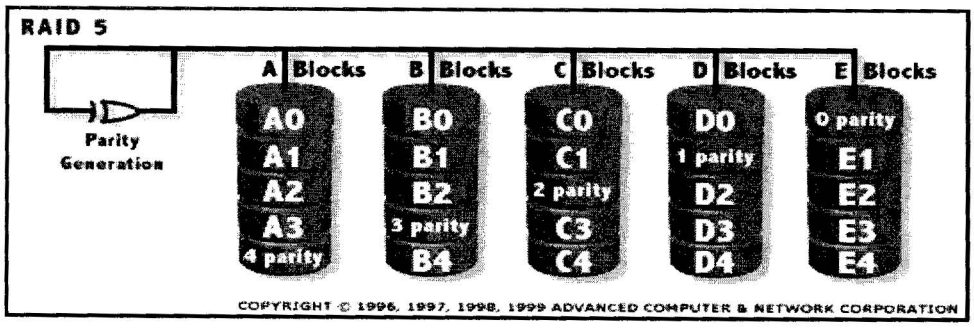
32. **Stallman, Richard. 2008.** Porque el software libre es mejor que software de código fuente abierto. *Pagina principal de GNU.* [En línea] 1 de Febrero de 2008. http://www.gnu.org/philosophy/free-software-for-freedom.es.html.

ANEXO

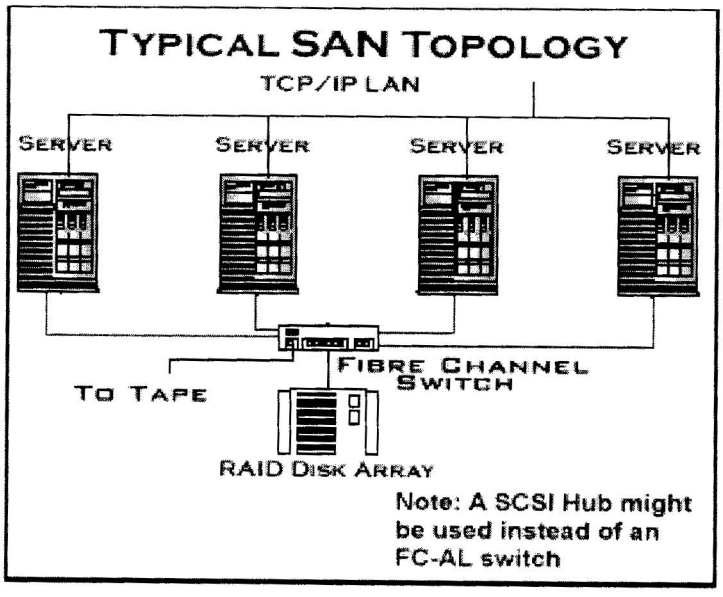
Anexo # 1 Arquitectura de un ERP.



Anexo # 2 RAID nivel 5.

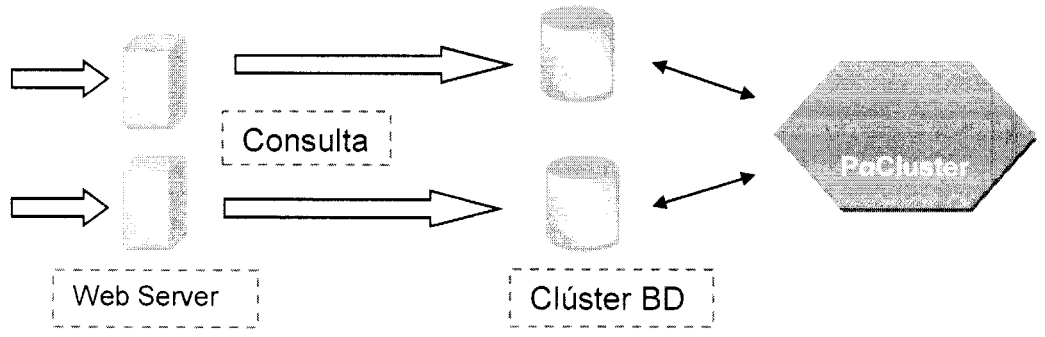


Anexo # 3 Topología típica de SAN.



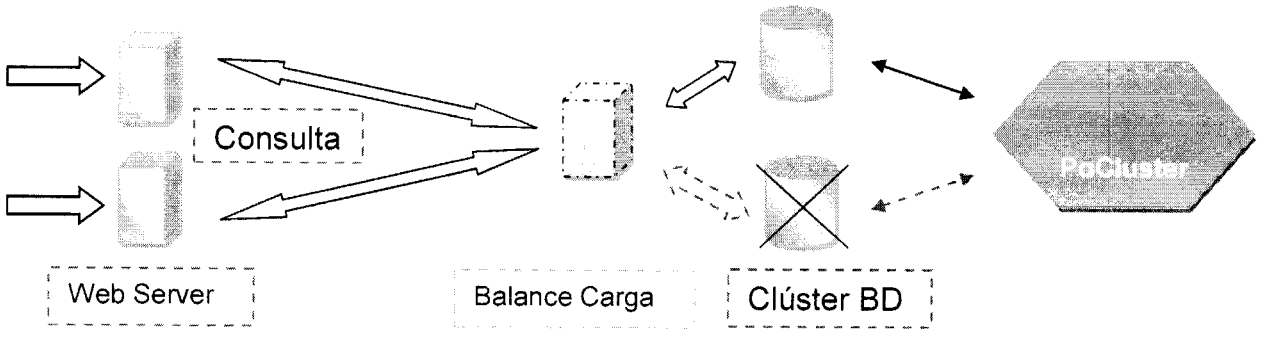
Anexo # 4 PgCluster.

Con la combinación de la Base de Datos Clúster y una copiadora, PgCluster pueden distribuir la carga de acceso:

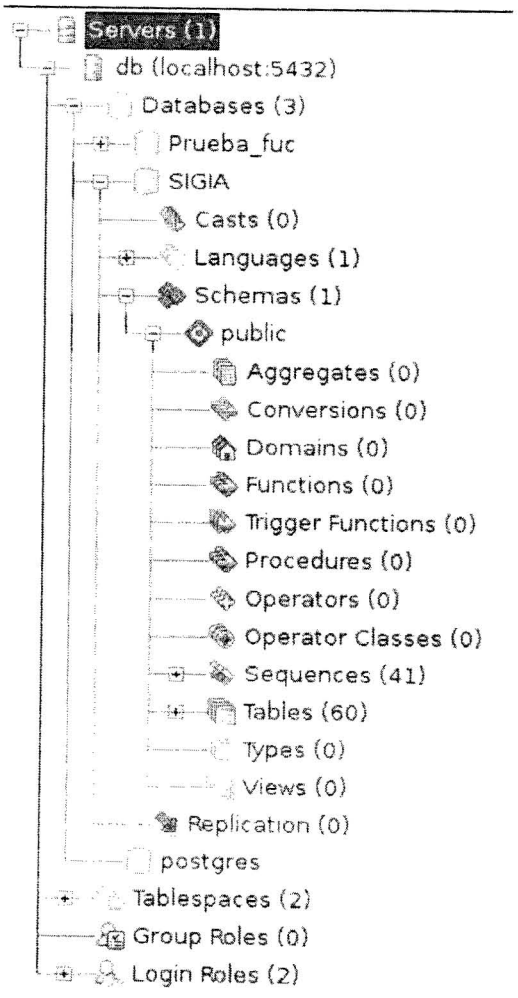


Anexo # 5 PgCluster.

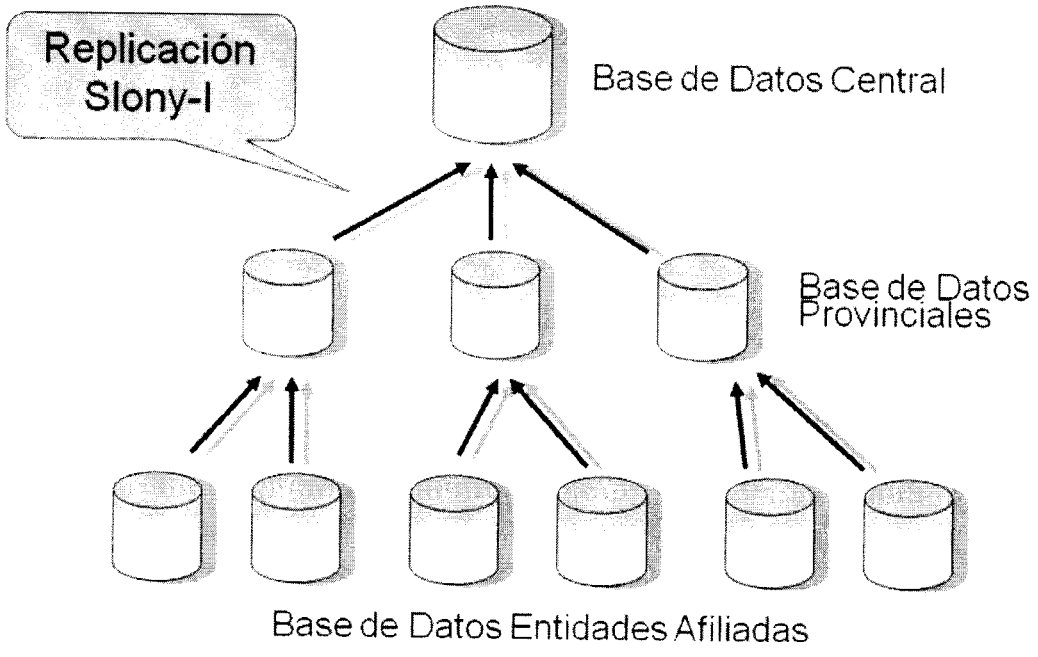
PgCluster puede hacer un sistema de alta disponibilidad:



Anexo # 6 Estructura de objeto a nivel lógico.



Anexo # 7 Arquitectura descentralizada.



Anexo # 8 Propuesta de seguridad en la Base de Datos.

Configuración de un servidor de 1GB de RAM y microprocesador Intel Pentium 4 a 3.00GHz, sistema operativo instalado Debian kernel 2.6.18

Conexión local:

Solo se conectara el administrador de la Base de Datos de forma superusuario.

local *all* *emgomez* *md5*

Conexiones remotas:

Para las conexiones remotas al servidor de Base de Datos:

Ir al fichero de configuración *postgresql.conf* y configurarlo de la siguiente forma:

listen_addresses = '*'

port = 5432

max_connections = 100

ssl = true

password_encryption = on

- Ir al fichero de configuración *pg_hba.conf*

hostssl SIGIA atgalvez 10.32.19.247/32 md5

El usuario atgalvez es un usuario que solo tiene permisos para hacer cambios sobre los objetos de la Base de Datos SIGIA.

1. Para que se conecten un grupo de clientes a una Base de Datos por un usuario en especifico:

hostssl	SIGIA	atgalvez	10.32.19.0/24	md5

- Para que las conexiones SSL seguras se activen deben seguir los pasos explicados en el epígrafe **2.4.3.2.2 Conexión con SSL**

Inclusión de un password fuerte al usuario administrador:

Vamos a la consola y mediante el usuario postgres ejecutamos

postgres@debian:/home/erich$ psql -d template1

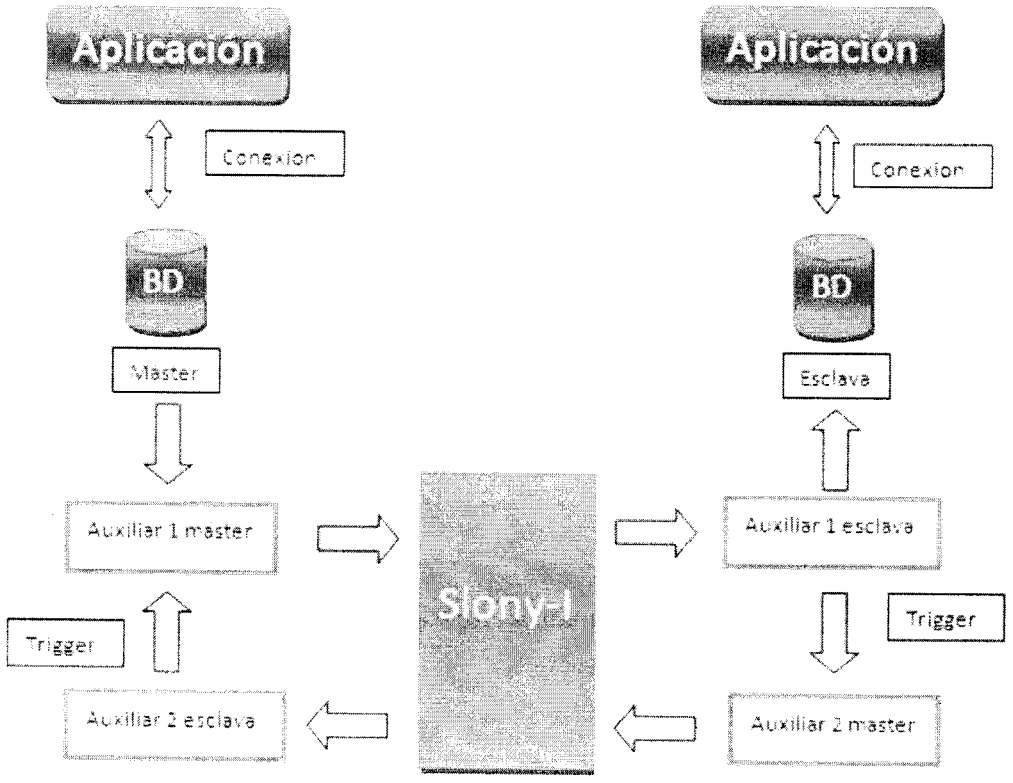
template1=# ALTER USER emgomez ENCRYPTED PASSWORD 'xxxxxxxxxx';

Política de privilegio a los usuarios:

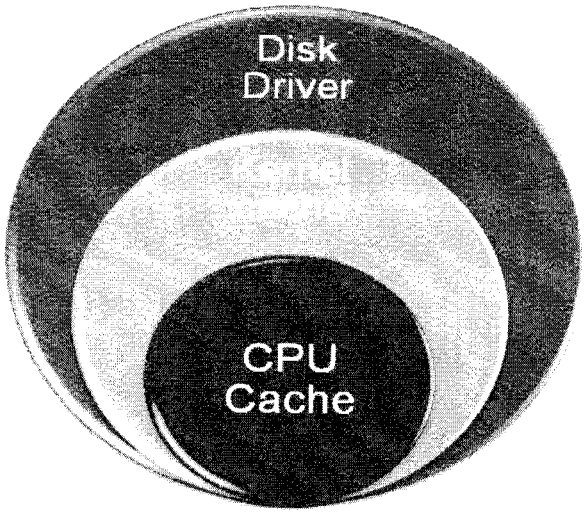
Los usuarios se gestionaran mediante el cliente pgAdmin3 de forma gráfica y sencilla.

- Solo tendrá permiso de superusuario el administrador de la Base de Datos y se conectara de forma local.
- Los usuarios clientes solo tendrán permisos para modificar los objetos de la Base de Datos con la cual trabajaran.

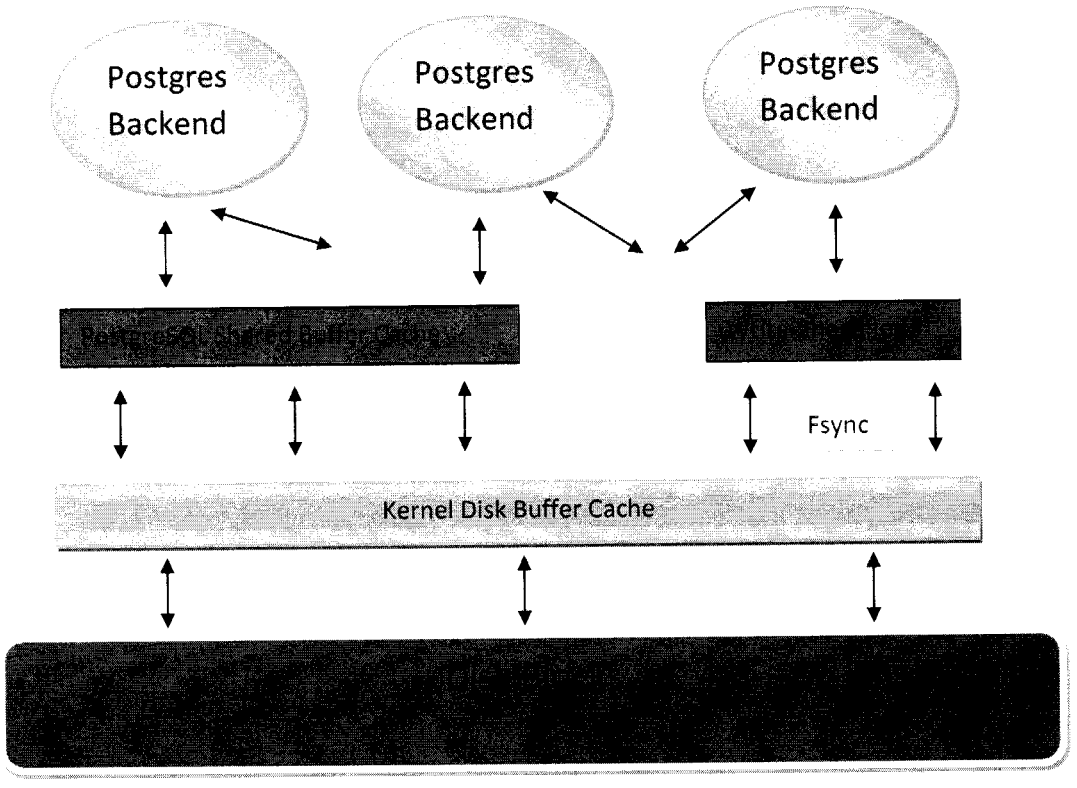
Anexo # 9 Propuesta replicación haciendo uso del Slony-I



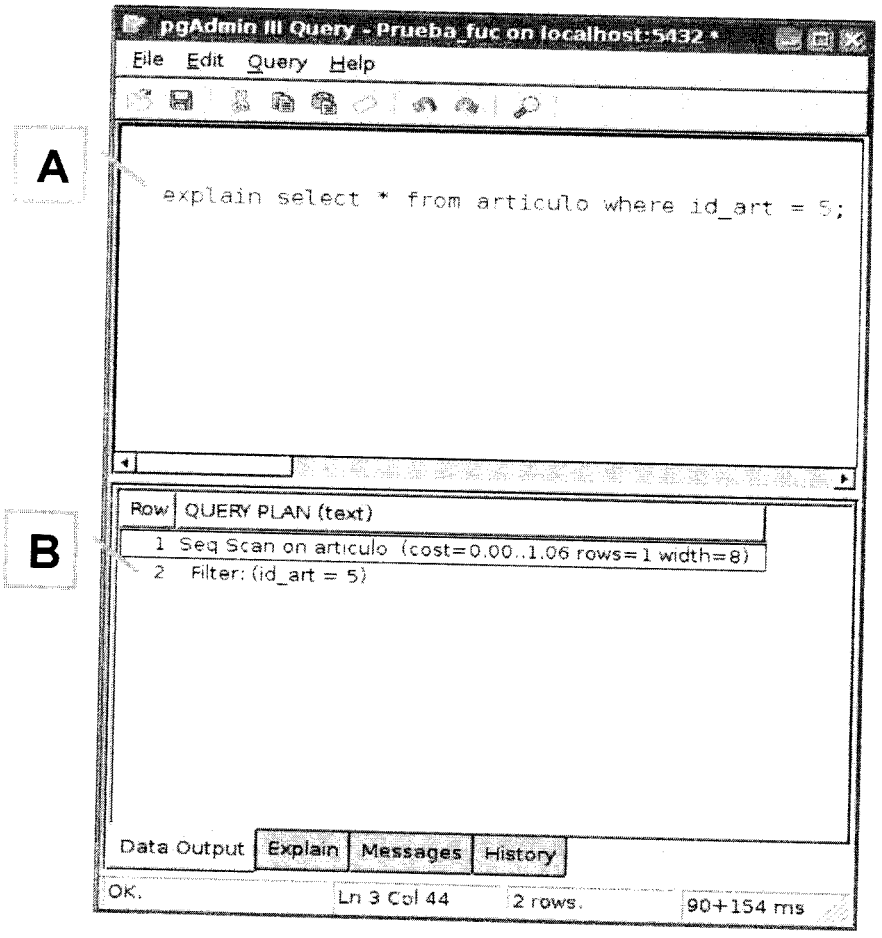
Anexo # 10 Zonas por la que está compuesta la computadora.



Anexo # 11 Arquitectura interna de PostgreSQL.



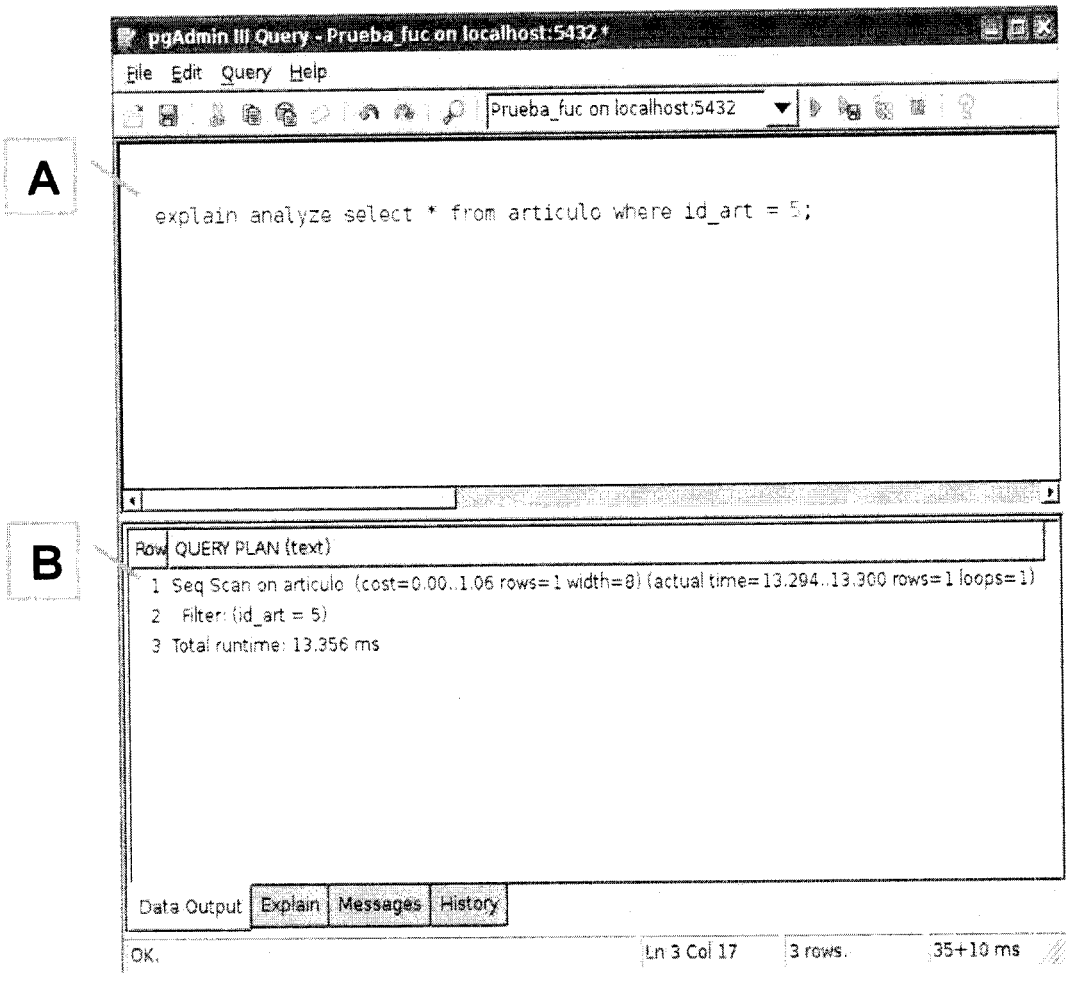
Anexo # 12 Uso del EXPLAIN



A: Comando SQL para ver la optimización de la consulta.

B: Resultado estadístico, en este ejemplo podemos apreciar que da un dato menos que en el anexo anterior.

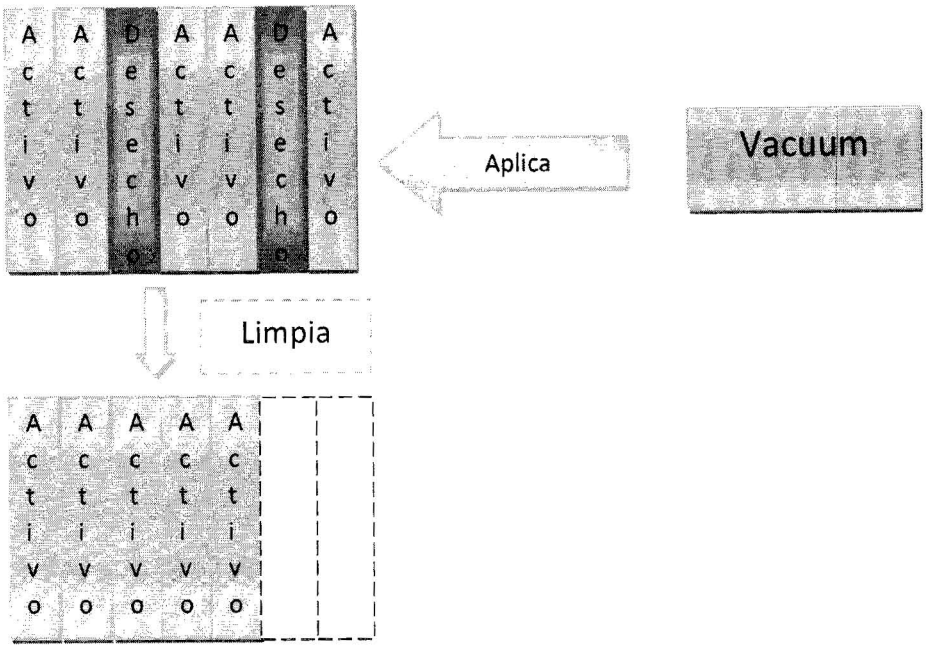
Anexo # 13 Uso de le EXPLAIN ANALYZE



A: Comando SQL para ver la optimización de la consulta.

B: Resultado estadístico del costo, tiempo de duración y demás datos.

Anexo # 14 Vacuum a la Base de Datos.



Anexo # 15 Realización de autovacuum

La realización de un autovacuum a la Base de Datos o a todas la base de datos se hará de la siguiente forma:

Creación de un script con permisos de ejecución llamado autovacuum

```
clear

#Funcion que muestra un mensaje de Bienvenida

function bienvenida {

  echo "Script para Generar autovacuum de una Base de Datos en Postgres"

  echo ""

}

H="127.0.0.1"

U="postgres"

echo vacuumdb -i -a -f -v -z -h $H -U $U -p 5432

vacuumdb -i -a -f -v -z -h $H -U $U -p 5432
```

En este caso se realizará para todas las Bases de Datos de nuestro servidor, en caso de una sola Base de Datos, se cambia el comando -a por -d y el nombre de la base de datos.

- Configuración del fichero postgresql.conf

```
autovacuum = on

autovacuum_naptime = 60

autovacuum_vacuum_threshold = 1000

autovacuum_analyze_threshold = 500
```

autovacuum_vacuum_scale_factor = 0.4

autovacuum_analyze_scale_factor = 0.2

autovacuum_vacuum_cost_delay = -1

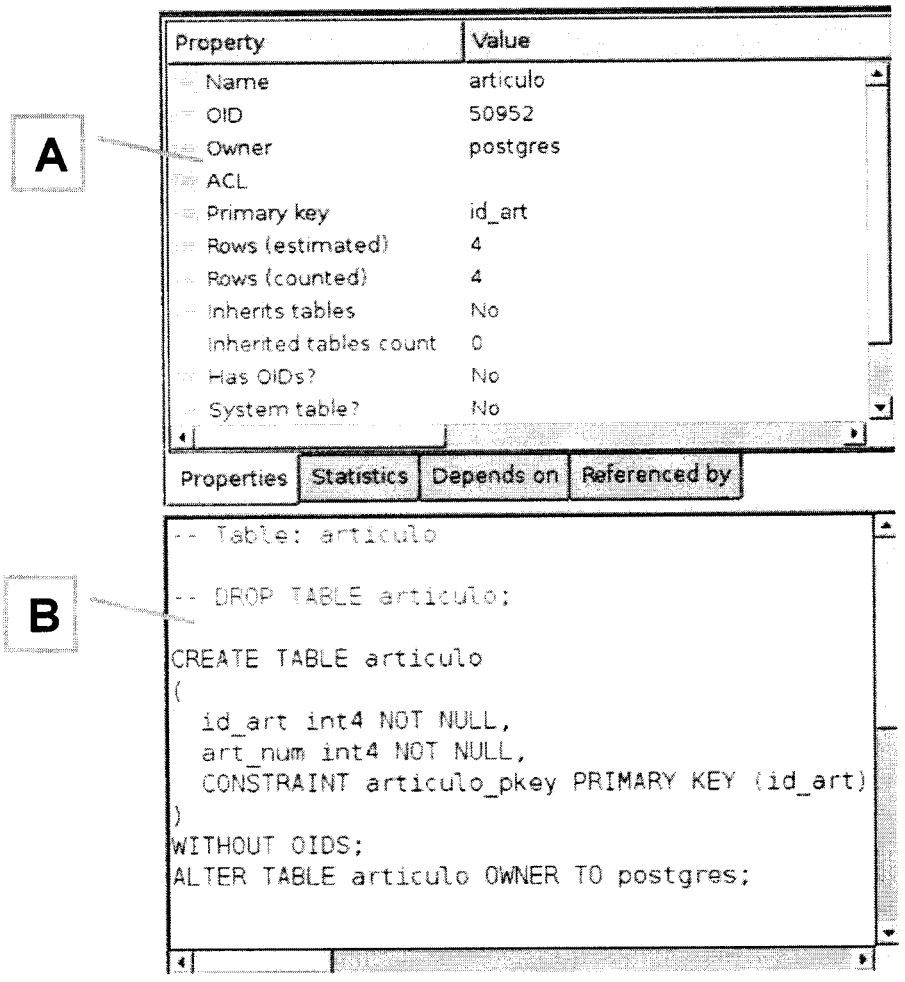
autovacuum_vacuum_cost_limit = -1

2. Copiar el script para el directorio de /root
3. Configuración del /etc/crontab del sistema operativo para que se ejecute todos los días 9:00 pm

00 21 * * * root /root/autovacuum.sh

Esto eliminara los desechos realizados en el resto del día por los comando SQL (INSERT, DELETE, UPDATE).

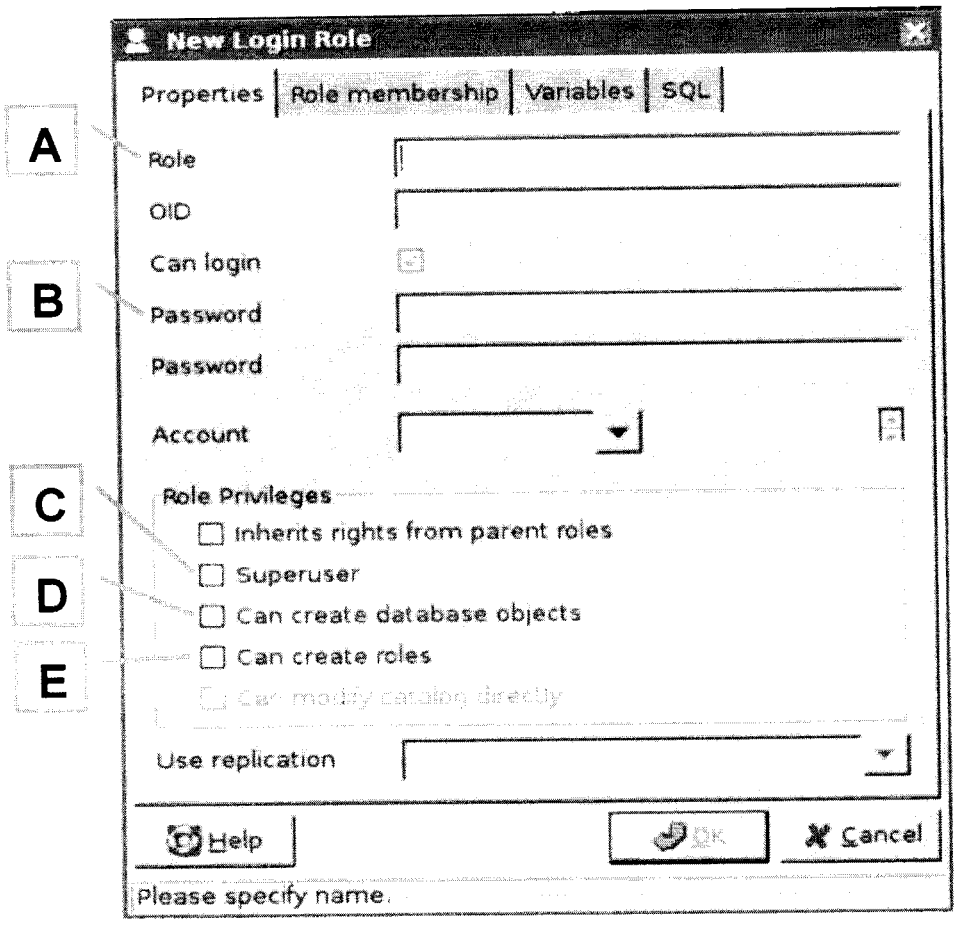
Anexo # 16 Sentencias SQL en pgAdmin3



A: Muestra las propiedades de un objeto previamente seleccionado.

B: Muestra las sentencias SQL del objeto seleccionado.

Anexo # 17 Creación de roles



A: Nombre del rol que desea crear.

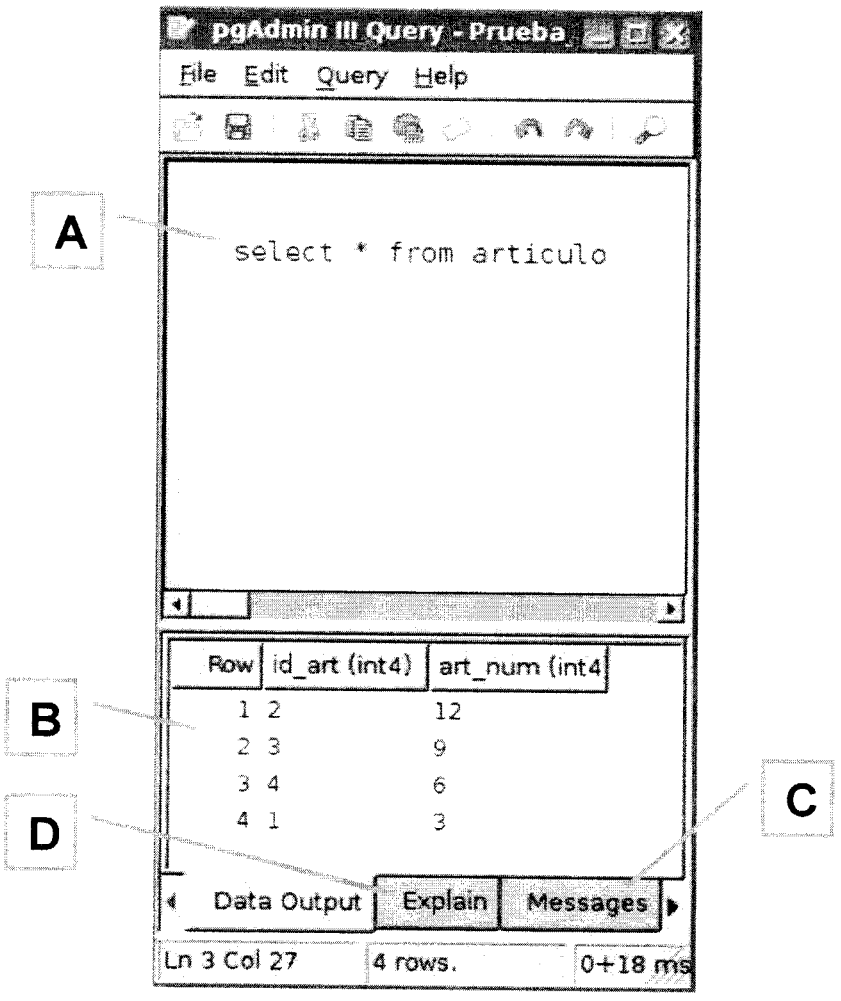
B: Contraseña del rol, debe tener más de 7 caracteres.

C: Permisos de superusuario, podrá acceder tanto a los objetos, como al los directorios de la Base de Datos.

D: Solo tendrá permisos para crear objetos, esquemas, tablas, etc.

E: Permisos para crear otros roles.

Anexo # 18 Imagen del Query de PostgreSQL



A: Código SQL, plpgsql.

B: Muestra los datos pedidos en la consulta.

C: Mostrar los mensajes de error.

D: Para mostrar el resultado cuando se ejecuta el comando EXPLAIN.

Anexo # 19 Guía de administración

Para realizar una buena instalación del Sistema Gestor PostgreSQL, se deben seguir los siguientes pasos, la misma se realizará sobre el sistema operativo Debian.

1. No se aconseja que se instale PostgreSQL desde el repositorio, sino obtener el paquete de instalación y compilarlo con las configuraciones deseadas, también se puede instalar desde el repositorio si estamos en una red local segura y no queremos hacerle las configuraciones siguientes.
2. Para que PostgreSQL soporte conexiones SSL se debe instalar la herramienta OpenSSL en el cliente y servidor, vamos a la consola.
   $aptitude install openssl

   Se configuran los paquetes internos de PostgreSQL con el comando:

   ./configure -cache_config -with-openssl

3. Posteriormente de haber instalado iniciamos el servidor.
   /etc/init.d/postgresql.8.x start

4. Para lograr un canal de comunicación seguro, PostgreSQL necesita un certificado, para esto paramos el servidor.
   /etc/init.d/postgresql.8.x stop

   Ponemos el siguiente comando en consola:

   $openssl req -new -text -out server.req

   El comando anterior requiere una clave secreta, la cual no se debe olvidar, en la opción de Common Name, se pone el nombre de la computadora.

   Ahora se crea una llave con el algoritmo RSA

   $openssl rsa -in privkey.pem -out server.key

   Cuando se ejecute la instrucción anterior se pedirá una serie de parámetros en los cuales se debe

ingresar la clave secreta. Posteriormente borramos el archivo creado (.pem) para evitar que el servidor esté en riesgo.

$rm privkey.pem

Ahora vamos a firmar la solicitud como Autoridad Certificadora, la que le proporcionaremos la solicitud del certificado es decir el archivo **server.req.**

$openssl req -x509 -in server.req -text -key server.key -out server.crt

Una vez ejecutado el comando o la Autoridad Certificadora nos regresa el certificado con el nombre de server.crt.

Cambiamos los permisos a los archivos generados, así como le cambiamos el dueño al usuario administrador y nos aseguramos que estén dentro del directorio de datos de PostgreSQL.

$chmod og-rwx server.key
*$chown emgomez:emgomez server.**

Ya tenemos creado el (.crt), sólo falta activar el SSL en el archivo *postgresql.conf*, se cambia ssl= on; y quitar el comentario.

5. Para saber si nuestro servidor tiene activado las conexiones SSL ejecutamos en consola con la ayuda del psql:
   template1=# SELECT name, setting from pg_settings where name ='ssl';

   La misma debe devolver:

   ssl | on

   Para saber si las consultas a través de la red están cifradas se utiliza la herramienta tcpdump, el cual nos dará información valiosa, para más detalles ejecutar:

   tcpdump --help para ver las opciones de utilidad en nuestra consola.

6. Para darle conexiones seguras SSL a un usuario se debe configurar en el fichero pg_hba.conf

   hostssl SIGIA postgres 10.32.19.12/32 md5

7. Para la configuración del fichero *postgresql.conf* se realizará los siguientes pasos.

   Abrir el fichero con el comando.

   $nano /etc/postgresql/8.1/main/postgresql.conf

   listen_addresses = '*' permite que se conecte cualquier IP

   port = 5432

   max_connections = 100 Este valor viene por defecto

   ssl = true

   password_encryption = on

8. Una vez configurado el fichero anterior se pasa a configurar el fichero *pg_hba.conf.*

   $nano /etc/postgresql/8.1/main/pg_hba.conf

   El mismo queda de la forma siguiente:

   # TYPE DATABASE USER CIDR-ADDRESS METHOD

   # "local" is for Unix domain socket connections only

   #local all all trust

   local all postgres md5

   # IPv4 local connections:

   #host all all 127.0.0.1/32 trust

   host SIGIA postgres 10.32.19.12/32 md5

   # IPv6 local connections:

   #host all all ::1/128 trust

   Se debe tener en cuenta que las conexiones pueden ser de un cliente o varios, esto se configura

de la siguiente forma:

- 10.32.19.0/24 ó 10.32.19.0 255.255.255.0 : se pueden conectar todas las IPs de la red 10.32.19
- 10.32.0.0/16 ó 10.32.0.0 255.255.0.0 : todos los IPs de la red 10.32
- 10.32.19.12/32: sólo se puede conectar esa IP.
- 0..0.0.0/0 ó 0.0.0.0 0.0.0.0.: cualquier IP

9. Para que el usuario administrador tenga una contraseña fuerte que cumpla con los requerimientos se realiza con el siguiente comando:

   postgres@debian:/home/erich$ psql -d template1

   Una vez dentro se ejecuta

   template1=# ALTER USER emgomez ENCRYPTED PASSWORD 'xxxxxxxxxx';

   Con esto la contraseña del administrador de PostgreSQL ha sido actualizada y se pedirá cada vez que se quiera conectar al mismo.

Las copia de seguridad y recuperación como vía para mantener la Base de Datos segura se realizarán de la siguiente forma:

2. Respaldo automático utilizando la herramienta pg_dump de la forma siguiente:

- Creación de un script ejecutable con permisos root.

```
#!/bin/bash
clear
#Función que muestra un mensaje de Bienvenida
function bienvenida {
  echo "Script para Generar Backup de una Base de Datos en Postgres"
  echo ""
}
#función que muestra una nueva línea
```

```
function newLine {
  echo ""
}
echo El Backup se va a generar en el Directorio "$HOME"
newLine
H="127.0.0.1"
U="postgres"
BD="SIGIA"
F=backup-$BD-$(date +%Y-%m-%d)
echo pg_dump -i -h $H -p 5432 -U $U -F p -C -D -v -f "$HOME/$F.sql" "$BD"
pg_dump -i -h $H -p 5432 -U $U -F p -C -D -v -f "$HOME/$F-$(date +%Y-%m-%d).sql" "$BD"
```

- Copiar el script a la carpeta del usuario root.
- Configuración del cron del sistema operativo para ejecutar el script todos los días a las 10:00 pm.
  00 22 * * * root /root/backupauto.sh
- El nombre y formato de guardado: Backup-Nombre-año-mes-día.sql
- El formato del Backup puede variar en dependencia del tamaño de información de la Base de Datos.
- Copiar el Backup para uno o más disco duros designados para guardar estas salvas.
- El borrado de la misma se hará después de haber pasado de 20 a 30 días.

Para realizar las copias de seguridad mediante los log de PostgreSQL se debe seguir los siguientes pasos:

1. Activar el archivado WAL en el fichero de *postgresql.conf* con el parámetro *archive_command*, ejemplo.

archive_command = 'test ! -f /etc/postgresql/8.1/main/pgdata/pg_xlog/%f.gz && gzip -1 -c %p >

/var/backups/pgsql/%f.gz'

3. Desde una consola con psql hay que ejecutar.
   Select pg_start_backup('nombre_sigia');

4. Realizar la copia de seguridad con el servidor en marcha, no es necesario pararlo.
   $ tar zhcvf /home/erich/lolo/backuptotal2.tar.gz /etc/postgresql/8.1/main/pgdata/ --exlude=/etc/postgresql/8.1/main/pgdata/pg_xlog/

5. Desde la consola con psql se ejecuta el siguiente comando para terminar.
   Select pg_stop_backup();

6. Se crea un fichero en el directorio $PGDATA/pg_xlog/archive_status y los log que se reciclan, se copian donde se indicó en *archive_command*.

El archivado WAL le permitirá restablecer las modificaciones introducidas a los datos de su Base de Datos PostgreSQL no va a restaurar los cambios realizados en los archivos de configuración (postgresql.conf, pg_hba.conf y pg_ident.conf).

Estos archivos pueden ser copiados y guardados en discos dedicados a salvas.

Para realizar la recuperación se siguen los siguientes pasos:

5. Parar el postmaster
6. Copiar el clúster dañado y los tablespaces a otra ubicación
7. Borrar todos los ficheros que hay dentro del clúster, incluyendo los tablespaces
8. Recuperar la copia de seguridad
   tar xvf backup_nombre_copia

10. Borrar los ficheros WAL en $PGDATA/pg_xlog, porque están obsoletos
11. Si existen ficheros WAL sin archivar, tal como se explica en el paso 2, se debe copiar los mismos a pg_xlog.
12. Crear un fichero de comandos de recuperación, *recovery.conf* en el directorio $PGDATA. Existe una plantilla, recovery.conf.sample, en el directorio $/usr/share/postgresql/8.1/. Se copia con el

nombre "recovery.conf" en $PGDATA y se edita. En principio, si se quiere recuperar todo y los logs están en su sitio, no hay que tocar nada.

13. Se arranca PostgreSQL en modo de recuperación, no se debe conectar nadie mientras el servidor esté realizando la recuperación.
14. Inspeccionar la Base de Datos, si todo ha ido correcto, los datos estarán recuperados hasta la última transacción confirmada y el fichero recovery.conf se renombra a recovery.done.

Para que la recuperación sea satisfactoriamente se deben tener los ficheros log en un disco distinto, donde está instalado el clúster y crear un enlace simbólico entre los dos discos, ejemplo.

```
mkdir /disco2/pg/

cd $PGDATA

mv pg_xlog /disco2/pg

ln –s /disco2/pg/pg_xlog pg_xlog
```

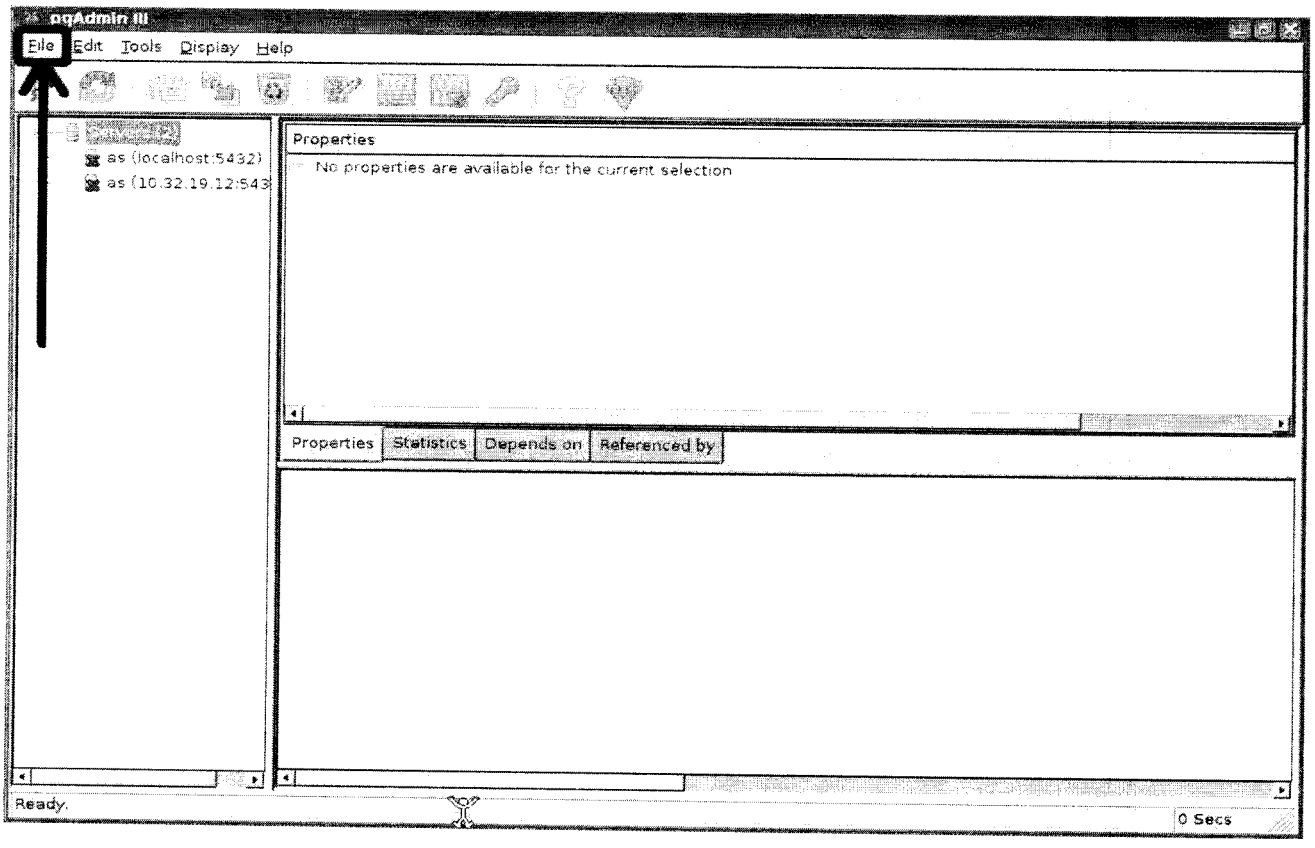
El fichero *recovery.conf* es el fichero que ayudará a realizar la recuperación, el mismo presenta una serie de parámetros que pueden hacer nuestra recuperación más fácil, esto se conoce como recuperación **Point-in-time**.

- **restore_command:** Lo que ejecuta esta variable es lo que PostgreSQL ejecutará antes de empezar la recuperación, por ejemplo:
  restore_command = 'cp /mnt/server/archivedir/%f %p'
- **recovery_target_time:** Hasta qué momento.
- **recovery_target_xid:** Hasta una transacción determinada.
- **recovery_target_inclusive (boolean):** Si los dos casos anteriores son inclusive o no.
- **Recovery_target_timeline (cadena)**: Especifica la recuperación en un tiempo particular.

3. Para la creación de roles se hace uso del cliente de administración pgAdmin3, el cual permitirá mediante una interfaz gráfica gestionar los mismos.

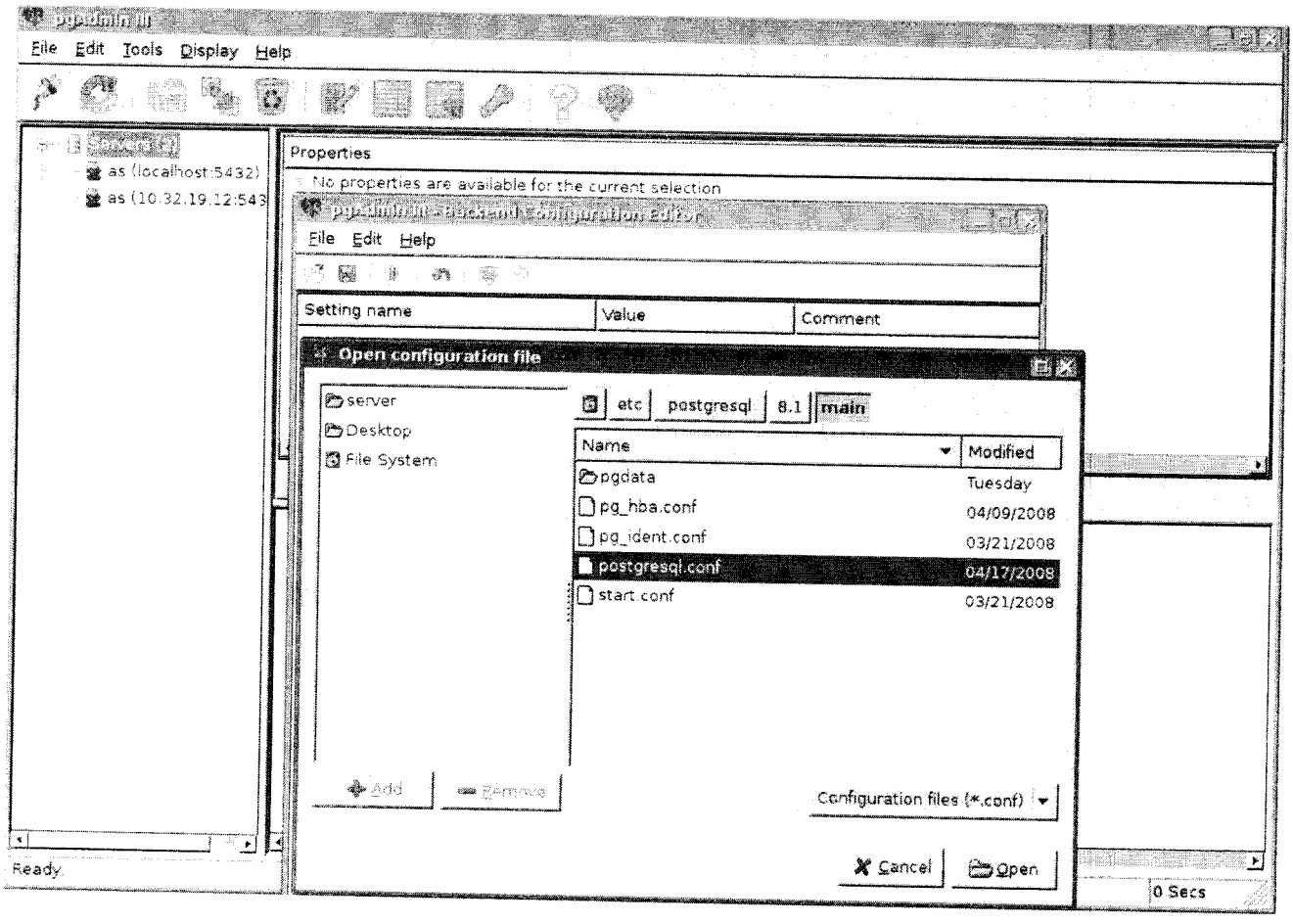
La optimización del sistema gestor PostgreSQL se puede llevar cabo mediante la configuración del archivo postgresql.conf se puede hacer de dos formas distintas, la primera es mediante uso del cliente pgAdmin3 pero no puedes cambiar los valores de los distintos parámetros que contiene, la única opción es seleccionarlo para que funcione con el valor por defecto que trae.

Primer paso abrir el cliente pgAdmin3 y seleccionar file->postgresql.conf

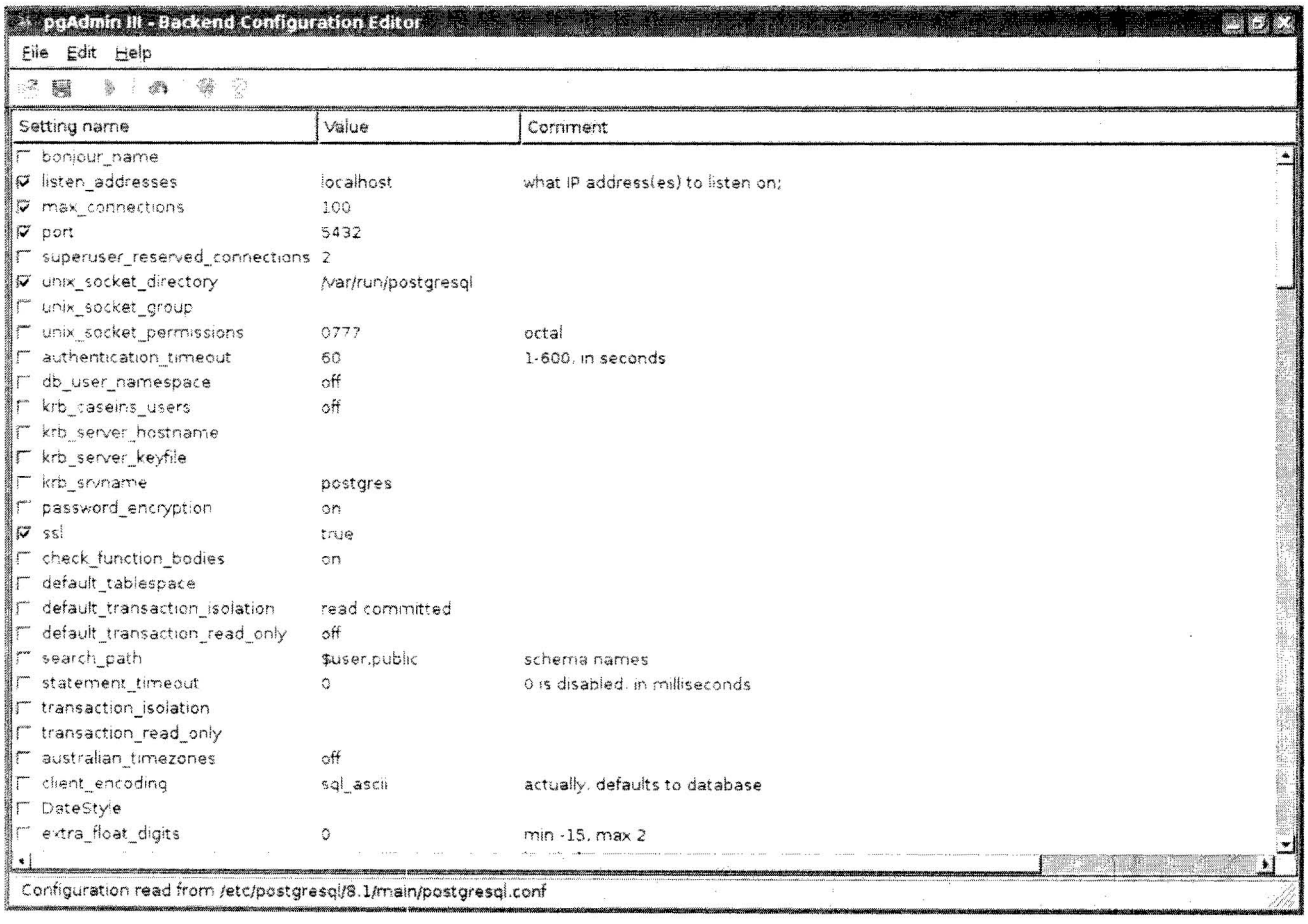


Segundo paso buscar el archivo en la siguiente dirección:

/etc/postgresql/8.1/main/postgresql.conf



Y por ultimo selección de los parámetros a activar.

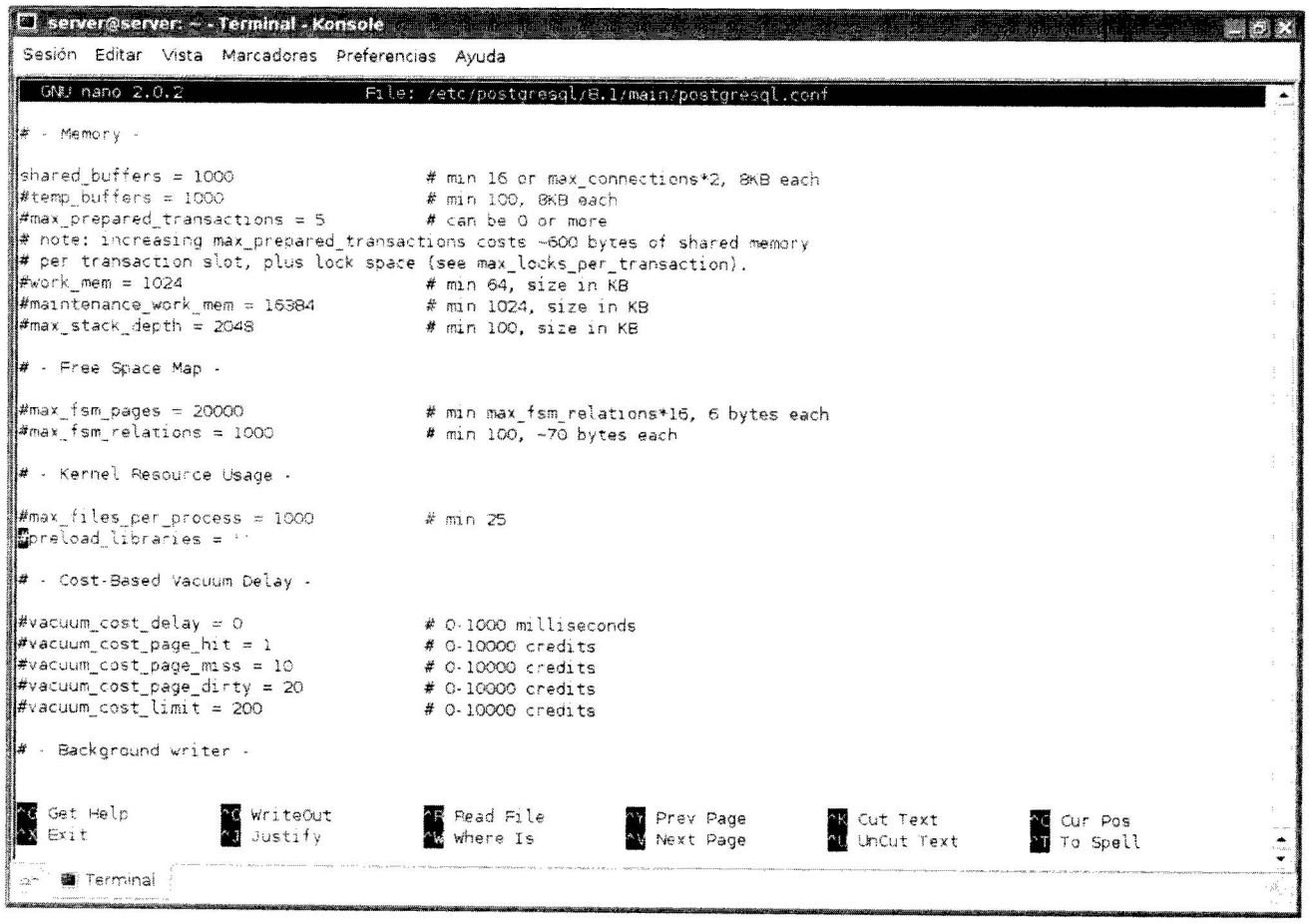


La segunda forma y mucho mejor puesto que te permite cambiar todos los valores que traen los parámetros en el archivo.

Primer paso abrir la consola, introducir la contraseña del administrador del sistema, y después por esa dirección que se muestra en la figura siguiente acceder al archivo:

```
server@server:~$ su
Password:
server:/home/server# nano /etc/postgresql/8.1/main/postgresql.conf
```

Luego se hace todas las configuraciones que el usuario determine y se presiona las teclas control->o y Enter para sobrescribir el archivo de configuración y por último se oprime control->x para salir, como muestra la figura siguiente.



El calculo de los diferentes valores que se deben poner en cada parámetro están explicados en el capítulo 3, así como la creación de índices para la optimización de las consultas.