Universidad de las Ciencias Informáticas

Facultad 2

Migración de la solución de base de datos del Sistema de Gestión de Policial (SIGEPOL) hacia un gestor de base de datos libre.

Trabajo de Diploma

Presentado para optar por el título de

Ingeniero en Ciencias Informáticas

Autor (es): Nathyara Pérez Navarro

Manuel De Jesús Abascal Matos

Tutor (es): Ing. Henry Góngora Columbié

"Año 52 de la Revolución"

Ciudad de La Habana, Cuba.

Declaración de Autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.
Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____ _____

Firma de Autor **Firma de Autor**

Nathyara Pérez Navarro **Manuel de Jesús Abascal Matos**

_____

Firma del Tutor

Ing. Henry Góngora Columbié

A Nuestros Padres…

Agradecimientos

Nathyara

A mis padres: Por guiarme, confiar en mí, quererme mucho, apoyarme siempre en todo y por ser el principal motivo de inspiración para la realización de este trabajo.

A mi hermana: Por quererme mucho, por acompañar a mis padres y por hacer que cada día me esfuerce para darle un buen ejemplo.

A mi esposo y tutor: Por confiar en mí, apoyarme y ayudarme en todo momento, tanto en el plano personal como profesional.

A mis suegros: Por su contribución en mi formación como profesional y por su apoyo en todo.

A mi familia, a mi compañero de tesis, y a todo aquel, que durante todo este tiempo ha contribuido de una forma u otra en mi formación tanto personal como profesional.

Manuel

A mi madre: Por ser la luz de mi camino en todos los momentos de mi vida. A ella le debo todo lo que soy.

A mi gran familia: Por apoyarme y confiar en mí siempre. A mi tía Muchy otra madre para mi, a mi tío Joaquín por todos esos consejos que me ha dado, a mi padre y a mi hermano que ojalá estuviesen aquí ahora y a mi vieja Hilda por malcriarme toda una vida.

A todos mis compañeros de aula: Por ser una familia para mí en estos años de estudio.

A mis compañeros de proyecto, a mi tutor y a mi compañera de tesis.

Resumen

Numerosos países a nivel mundial han determinado alcanzar la soberanía tecnológica y consideran el software libre como la solución más rápida y efectiva a los problemas que ocasiona el uso de software propietario. Incitados por el libre uso, la modificación y redistribución que posibilita, se han trazado planes concretos para realizar un proceso de migración hacia software libre, haciendo del mismo la alternativa de futuro, con la cual cada país alcance la independencia tecnológica deseada.

Venezuela es uno de los países con interés en el proceso de migración al Software Libre, el Decreto N° 3.390 emitido por el presidente de la República Bolivariana de Venezuela, Hugo Rafael Chávez Frías, publicado en la Gaceta Oficial Nº 38.095 de fecha 28/12/2004 busca incentivar a las instituciones gubernamentales del país a encontrar alternativas libres, migrando los sistemas, proyectos y servicios informáticos a esta nueva plataforma de desarrollo.

Una de las medidas a considerar dentro del proceso de migración hacia software libre es la referente a los Sistemas Gestores de Bases de Datos (SGBD), muy utilizados en las soluciones de software propuestas a diferentes problemas dentro de la sociedad venezolana. Dentro de los sistemas que hoy en día gestionan sus datos con el SGBD propietario Oracle, se encuentra el Sistema de Gestión Policial (SIGEPOL).

En el presente trabajo de diploma se exponen aspectos relacionados con la migración a software libre y las licencias de los mismos, se hace referencia a conceptos importantes dentro del mundo de las bases de datos y temas relacionados con algunos de los SGBD Libres; al mismo tiempo se indica como realizar la migración de la solución de BD de SIGEPOL hacia un SGDB libre, permitiendo de esta forma que SIGEPOL gestione sus datos con PostgreSQL como SGBD y por otra parte se explican los elementos que permitieron realizar la validación práctica de la solución propuesta.

Índice

Índice

Introducción

La globalización, y en especial la generalización del uso de Internet en la sociedad desarrollada y subdesarrollada han facilitado el advenimiento de grandes compañías en el mundo del software, algunas de ellas como Microsoft, HP, Oracle, IBM y Cisco, son corporaciones transnacionales de origen estadounidense cuyos productos a nivel de mercado poseen un alto precio de uso, los mismos son comercializados como manufacturados, pero con condiciones diferentes a otros que se ofrecen en el mercado; no se puede modificar, ni adaptar ya que el productor impone las condiciones de usabilidad a través de licencias restrictivas que imposibilitan en ocasiones el buen desempeño de una organización.

Muchos países a nivel mundial se han propuesto alcanzar la soberanía tecnológica y ven en el software libre la solución más rápida y efectiva a los problemas que causa el uso de software propietario. Motivados por el libre uso, la modificación, así como la redistribución que este posibilita, se han trazado planes concretos para realizar un proceso de migración hacia software libre, haciendo del mismo la alternativa de futuro con la cual cada país adquiera la independencia tecnológica deseada. Cuba, Chile, Perú, Brasil y Venezuela son algunos de los gobiernos que con mayor fuerza han tomado el tema, ejemplo de ello son: El Proyecto de Software Libre – RS lanzado en Brasil el día 30 de julio de 1999 y que tiene entre sus principales iniciativas la implantación de una red de laboratorios en empresas y universidades para el estudio del GNU / Linux y demás software libres (1). Años más tarde, el 18 de septiembre de 2003 fue presentado un proyecto que propone emplear en todas las instituciones del Estado peruano programas de software libre en sus sistemas y equipamientos de informática (2).

Venezuela es otro de los países con interés en el proceso de migración al Software Libre, el Decreto N° 3.390 emitido por el presidente de la República Bolivariana de Venezuela, Hugo Rafael Chávez Frías, publicado en la Gaceta Oficial Nº 38.095 de fecha 28/12/2004, busca incentivar a las instituciones gubernamentales del país a encontrar alternativas libres migrando los sistemas, proyectos y servicios informáticos a esta nueva plataforma de desarrollo, instando a la Institución Pública Nacional a tomar cartas en el asunto; además de que establece que es prioridad del Estado incentivar y fomentar la producción de bienes y servicios para satisfacer las necesidades de la población, mediante el uso de herramientas desarrolladas con estándares abiertos para robustecer la industria nacional, aumentando y aprovechando sus capacidades y fortaleciendo su soberanía (3).

Además, el artículo 110 de la Constitución de la República Bolivariana de Venezuela, reconoce como de interés público: la ciencia, la tecnología, el conocimiento, la innovación y los servicios de información, con el objeto de lograr el desarrollo económico, social y político del país. Esta disposición constitucional se expresa con fuerza en los artículos 1° de la Ley de Telecomunicaciones y 12° de la Ley Orgánica de la Administración Pública. Con el Decreto N° 825, emitido el 10 de mayo de 2000, en los cuales se establece el acceso y el uso de Internet como política prioritaria para el desarrollo cultural, económico, social y político del Estado.

Actualmente en Venezuela existen alrededor de 500 empresas dedicadas a la integración de sistemas, el desarrollo y la comercialización de software propio o de terceros. Entre las aplicaciones comerciales, no desarrolladas en base a requerimientos, las producidas en mayor volumen por las compañías de software son las destinadas a actividades financieras, de inventario, facturación, inteligencia empresarial, gestión de recursos humanos y servicios de Internet. Por otro lado, encontramos a la industria de software libre, la cual se encuentra en pleno surgimiento gracias al auge mundial que se le ha dado como alternativa a las plataformas soportadas por sistemas de operación como UNIX y Windows. (4)

Otro elemento que ha ayudado a que inicie el crecimiento del software libre en Venezuela, es el apoyo recibido desde el Gobierno Nacional con un marco regulatorio que promueve la utilización de software libre, principalmente en la Administración Pública Nacional (APN). Si bien es cierto que la industria del software libre en Venezuela está creciendo, aún no se encuentra al nivel de la industria de software propietario, por lo tanto, se requiere seguir potenciando esta industria así como el fortalecimiento del Plan de Migración al Software Libre de la APN para alcanzar a mediano plazo una Empresa de Software Libre de alta calidad.

Una de las medidas a considerar dentro del proceso de migración hacia software libre es la concerniente a los Sistemas Gestores de Bases de Datos (SGBD), muy utilizados en las soluciones de software propuestas a determinados problemas dentro de la sociedad venezolana, destacando que aunque en Venezuela existe una sucursal del SGBD más usado a nivel empresarial (Oracle), el país enfrenta un gran problema de usabilidad del mismo, este problema es sin lugar a dudas las grandes sumas de dinero que se debe pagar por concepto de licencia, trayendo esto como consecuencia que muchas dependencias del gobierno esparcidas por todo el país en ocasiones no cuentan con el presupuesto suficiente para financiar el pago de dicha licencia, elemento que hace compleja la creación de soluciones

informáticas para dichas dependencias que utilicen Oracle como SGBD, por lo que se quiere llevar a cabo la migración a SGBD libres.

En la República Bolivariana de Venezuela actualmente uno de los sistemas informáticos que sus datos están gestionados por el SGBD Oracle es el Sistema de Gestión de Policial (SIGEPOL), en el cual la base de datos es utilizada para:

- Almacenar la información que se recoge en los módulos de Reseña, Denuncia, Operativos Policiales y Administración de todas las dependencias policiales.
- Garantizar un punto de intercambio de información entre los diferentes módulos del sistema.
- Servir de fuente para los reportes policiales así como para los del CTAISC.
- Mantener la integridad y disponibilidad de la información.
- Servir como fuente de datos para otros sistemas informáticos del Ministerio de Interiores y Justicia.
- Garantizar la confidencialidad y seguridad de la información.

De esta forma, las aplicaciones buscan información en el servidor de base de datos la mayor parte del tiempo para su funcionamiento, datos que asincrónicamente han sido insertados, actualizados o eliminados por estas mismas aplicaciones, estableciendo así un medio de comunicación entre ellas.

Dándole cumplimiento a lo planteado en el Decreto N° 3.390, se decide por parte de la dirección de SIGEPOL migrar el SGBD utilizado en la solución (Oracle) a un SGBD libre.

Problema Científico

Como resultado de lo analizado anteriormente surge el siguiente **problema científico** ¿Cómo realizar la migración de la solución de base de datos del Sistema de Gestión Policial (SIGEPOL) hacia un Sistema Gestor de Bases de Datos libre?

El **objeto de estudio** son los Sistemas Gestores de Base de Datos. Siendo el **campo de acción** los Sistemas Gestores de Bases de Datos Libres.

Introducción

Para dar solución al problema planteado anteriormente se propone como **objetivo general** de la investigación:

Desarrollar la migración de la solución de base de datos del Sistema de Gestión Policial (SIGEPOL) hacia un Sistema Gestor de Bases de Datos libre.

Para desarrollar satisfactoriamente la investigación se definieron un conjunto de **tareas de investigación** que permiten cumplir el objetivo general propuesto, estas son:

- Realizar un estudio de los principales SGBD libres.
- Seleccionar el SGBD libre hacia el cual se realizará la migración.
- Analizar la estructura y compatibilidad de la solución de la base de datos de SIGEPOL con el SGDB libre seleccionado.
- Definir la estrategia de migración de cada uno de los elementos de base de datos utilizados en la solución de la base de datos de SIGEPOL hacia el SGDB libre seleccionado.
- Migrar la solución de base de datos de SIGEPOL hacia el SGBD libre seleccionado.
- Validar la estrategia de migración en uno de los módulos del sistema.

El presente documento está compuesto por las siguientes partes: resumen, introducción, desarrollo, conclusiones, recomendaciones, referencias bibliográficas, bibliografía, anexos y glosario de términos. El desarrollo está estructurado en 3 capítulos:

- Capítulo I: “Fundamentación Teórica”, incluye los aspectos teóricos que soportan la investigación que se llevará a cabo. Se describen los principales SGBD libres. Se realiza la selección del gestor de base de datos hacia el cual se realizará la migración.

- Capítulo II: “Migración de la Solución de BD de SIGEPOL”, se describe el proceso a seguir para lograr la migración de cada uno de los objetos de la base de datos de SIGEPOL hacia el SGBD seleccionado.

- Capítulo III: “Validación de la Solución”, incluye elementos que permiten realizar la validación práctica de la solución.

Capítulo I

Fundamentación Teórica

1.1 Introducción del Capítulo I.

En este capítulo se presentan un grupo de elementos teóricos necesarios para llevar a cabo la investigación y a los que se harán referencia en el resto del documento. Se abordan aspectos relacionados con la migración a software libre, las licencias de software libre así como otros elementos relacionados con el uso de las nuevas tecnologías de la informática y las comunicaciones. Se hace referencia a conceptos importantes dentro del mundo de las bases de datos y temas relacionados con algunos de los SGBD Libres.

1.2 Proceso de Migración a Software Libre.

La migración a Software Libre se refiere a un conjunto de actuaciones cuya finalidad es la sustitución de infraestructuras tecnológicas apoyadas en software propietario por otras con funciones equivalentes basadas en Software Libre (5).

La misma proporciona a nivel técnico y operativo, una serie de ventajas para el cliente; algunos ejemplos claros de las típicas ventajas son los bajos costes al eliminar los costos derivados de licencias, la elevada fiabilidad al ser testeados, usados y corregidos continuamente en diferentes entornos, la mayor seguridad que presenta al ser pública su especificación y estar sometida a constante revisión, la gran flexibilidad que da el poder modificar y adaptar el código a los requerimientos particulares del cliente, la inter-operatividad que proporciona al estar basado generalmente en estándares abiertos, la fácil integración al permitir la modificación del código y ser software usado y nacido de las necesidades de los usuarios, siendo el mantenimiento por otra parte flexible y competitivo al no estar monopolizado por una determinada compañía (5).

Estas ventajas prácticas de la migración a software libre son consecuencia de la definición del mismo, que defiende los derechos de los usuarios, empresas y desarrolladores. Así pues, con migración a software libre ahorras por concepto de licencias, que aunque para grandes empresas puede ser un coste insignificante del total, para pequeñas y un gran número de medianas empresas puede significar un ahorro considerable (6).

1.3 Licencias de Software.

La licencia de software es una especie de contrato, en donde se especifican todas las normas y cláusulas que rigen el uso de un determinado programa, principalmente se estipulan los alcances de uso, instalación, reproducción y copia de estos productos (7).

Las licencias de uso de software generalmente caen en alguno de estos tipos:

- Licencia propietaria. Uso en una computadora por el pago de un precio.
- Shareware. Uso limitado en tiempo o capacidades, después pagar un precio.
- Freeware. Usar y copiar sin límites, precio es cero.
- Software libre. Usar, copiar, estudiar, modificar, redistribuir. Código fuente incluido.

Es posible dividir las **licencias de software libre** en dos grandes familias, una de ellas está compuesta por las licencias que no imponen condiciones especiales, sólo especifican que el software se puede redistribuir o modificar, estas son las llamadas **licencias permisivas** y la otra familia, denominadas **licencias robustas** o **licencias copyleft**, imponen condiciones en caso de que se quiera redistribuir el software, condiciones que van en la línea de forzar a que se sigan cumpliendo las condiciones de la licencia después de la primera redistribución (7).

Independientemente de la existencia de estos dos grandes grupos, las licencias pueden ser clasificadas de la siguiente forma:

1.3.1 Licencia GPL

La Licencia Pública General (inglés: General Public License o GPL) otorga al usuario la libertad de compartir el software licenciado bajo ella, así como realizar cambios en él. Es decir, el usuario tiene derecho a usar un programa licenciado bajo GPL, modificarlo y distribuir las versiones modificadas de este.

La licencia GPL adopta el principio de la no ocultación, respaldando el concepto moral que establece que todo software desarrollado con el uso de material licenciado bajo GPL debe estar disponible para ser compartido con el resto de la humanidad.

GPL fue creada para mantener la libertad del software y evitar que alguien quisiera apropiarse de la autoría intelectual de un determinado programa. La licencia advierte que el software debe ser gratuito y que el paquete final también debe ser gratuito, asegurándose siempre de mantener los nombres y créditos de los autores originales.

Como aspecto curioso, se debe considerar que si se reutiliza un programa "A", licenciado bajo GPL, y se reutiliza un programa "B", bajo otro tipo de licencia libre, el programa final "C", debe de estar bajo la licencia GPL. Este concepto se introduce con el denominado copyleft a fin de garantizar que cualquier aprovechamiento de un programa bajo licencia GPL redunde sobre la comunidad (8).

1.3.2 Licencia BSD

La Licencia de Distribución de Software de Berkeley (inglés: Berkeley Software Distribution o BSD) no impone ninguna restricción a los desarrolladores de software en lo referente a la utilización posterior del código en derivados y licencias de estos programas. Este tipo de licencia permite a los programadores utilizar, modificar y distribuir a terceros el código fuente y el código binario del programa de software original con o sin modificaciones. Los trabajos derivados pueden optar por licencias de código abierto o comercial.

La licencia BSD es un buen ejemplo de una licencia permisiva, que casi no impone condiciones sobre lo que un usuario puede hacer con el software, la misma permite la redistribución, uso y modificación del software.

Esta licencia permite el uso del código fuente en software no libre, lo que la hace muy similar a la GPL descrita anteriormente. La diferencia consiste en que en la licencia BSD no es obligatorio mencionar a los autores ni proporcionar el código fuente.

El autor, bajo esta licencia, mantiene la protección de copyright únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación (7).

1.3.3 Licencia MPL

La Licencia Pública de Mozilla (inglés: Mozilla Public License o MPL) es una licencia de código abierto y software libre utilizada por el navegador de Internet Mozilla y sus productos derivados. Dicha licencia

cumple completamente con los postulados del open source y del software libre; sin embargo, la MPL deja abierto el camino a una posible reutilización comercial y no libre del software si el usuario así lo desea sin restringir la reutilización del código ni el re-licenciamiento bajo la misma licencia.

Aunque el uso principal de la MPL es servir como licencia de control para el navegador Mozilla y el software relacionado con él, esta licencia es ampliamente utilizada por desarrolladores y programadores que quieren liberar su código (7).

1.3.4 Licencia DUAL

El uso de una licencia dual o doble licenciamiento es la práctica de conceder dos o más licencias para el mismo producto intelectual. Este tipo de licenciamiento es común en el caso del software donde un producto liberado como copyleft (que implica que los trabajos derivados deben ser también libres) puede ser licenciado a mayores para permitir su uso comercial de una forma no libre (9).

1.4 Base de Datos (BD).

Una Base de Datos (BD) es un conjunto de datos interrelacionados entre sí, almacenados con carácter más o menos permanente en la computadora. O sea, que una BD puede considerarse una colección de datos variables en el tiempo (10).

1.4.1 Clasificación de las BD.

Las bases de datos pueden clasificarse de varias maneras, de acuerdo con el contexto que se esté manejando, o la utilidad de la misma (11):

Según la variabilidad de los datos almacenados:

1.4.1.1 Bases de datos estáticas

Estas son bases de datos de solo lectura, utilizadas primordialmente para almacenar datos históricos que posteriormente se pueden utilizar para estudiar el comportamiento de un conjunto de datos a través del tiempo, realizar proyecciones y tomar decisiones.

1.4.1.2 Bases de datos dinámicas

Estas son bases de datos donde la información almacenada se modifica con el tiempo, permitiendo operaciones como actualización y adición de datos, además de las operaciones fundamentales de

consulta. Un ejemplo de esto puede ser la base de datos utilizada en un sistema de información de una tienda de abarrotes, una farmacia o un videoclub.

Según el contenido:

1.4.1.3 Bases de datos bibliográficas

Sólo contienen un representante de la fuente primaria, que permite localizarla. Un registro típico de una base de datos bibliográfica contiene información sobre el autor, fecha de publicación, editorial, título o edición de una determinada publicación. Puede contener un resumen o extracto de la publicación original, pero nunca el texto completo, porque sino se estaría en presencia de una base de datos de texto completo. Como su nombre lo indica, el contenido son cifras o números. Por ejemplo, una colección de resultados de análisis de laboratorio.

1.4.1.4 Bases de datos de texto completo

Almacenan las fuentes primarias, como por ejemplo, todo el contenido de todas las ediciones de una colección de revistas científicas.

1.4.1.5 Directorios

Un ejemplo son las guías telefónicas en formato electrónico.

1.4.1.6 Bases de datos o "bibliotecas" de información Biológica

Son bases de datos que almacenan diferentes tipos de información proveniente de las ciencias de la vida o médicas. Se pueden considerar en varios subtipos:

- Aquellas que almacenan secuencias de nucleótidos o proteínas.
- Las bases de datos de rutas metabólicas.
- Bases de datos de estructura, comprende los registros de datos experimentales sobre estructuras 3D de biomoléculas.
- Bases de datos clínicas.
- Bases de datos bibliográficas (biológicas).

Además de la clasificación por la función de las bases de datos, estas también se pueden clasificar de acuerdo con su modelo de administración de datos.

1.4.2 Modelos de BD.

Un modelo de datos es básicamente una "descripción" de algo conocido como contenedor de datos (algo en donde se guarda la información), así como de los métodos para almacenar y recuperar información de esos contenedores. Los modelos de datos son abstracciones que permiten la implementación de un sistema eficiente de base de datos; por lo general se refieren a algoritmos y conceptos matemáticos (12).

Algunos modelos con frecuencia utilizados en las bases de datos son:

1.4.2.1 Bases de datos jerárquicas

Estas son bases de datos que, como su nombre indica, almacenan su información en una estructura jerárquica. En este modelo los datos se organizan en una forma similar a un árbol, en donde un nodo padre de información puede tener varios hijos. El nodo que no tiene padres es llamado raíz, y a los nodos que no tienen hijos se les conoce como hojas. Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos permitiendo crear estructuras estables y de gran rendimiento (12).

Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos.

1.4.2.2 Base de datos de red

Este es un modelo ligeramente distinto del jerárquico; su diferencia fundamental es la modificación del concepto de nodo: se permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico). Fue una gran mejora con respecto al modelo jerárquico, ya que ofrecía una solución eficiente al problema de redundancia de datos; pero, aún así, la dificultad que significa administrar la información en una base de datos de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales (12).

1.4.2.3 Base de datos relacional

Una base de datos relacional es una base de datos basada en un modelo relacional. El término se refiere a una colección específica de datos, pero a menudo es usado como sinónimo del software utilizado para

gestionar esa colección de datos. Ese software se conoce como sistema gestor de base de datos relacional o RDBMS (Relational Database Management System) (12).

Este es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postulados sus fundamentos en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos. Su idea fundamental es el uso de "relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas". Pese a que esta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar. Esto es pensando en cada relación como si fuese una tabla que está compuesta por registros (las filas de una tabla), que representarían las tuplas, y campos (las columnas de una tabla) (12).

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia. Esto tiene la considerable ventaja que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información.

El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL (Structured Query Language o Lenguaje Estructurado de Consultas), un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

Durante su diseño, una base de datos relacional pasa por un proceso al que se le conoce como normalización de una base de datos, él cual elimina posibles errores de diseño y posibilita que no existan problemas tales como la redundancia de datos y otros elementos importantes a la hora de diseñar e implementar una base de datos relacional.

1.4.2.4 Bases de datos orientadas a objetos

Este modelo, bastante reciente, y propio de los modelos informáticos orientados a objetos, trata de almacenar en la base de datos los objetos completos (estado y comportamiento).

Una base de datos orientada a objetos es una base de datos que incorpora todos los conceptos importantes del paradigma de objetos:

- Encapsulación - Propiedad que permite ocultar la información al resto de los objetos, impidiendo así accesos incorrectos o conflictos.
- Herencia - Propiedad a través de la cual los objetos heredan comportamientos dentro de una jerarquía de clases.
- Polimorfismo - Propiedad de una operación mediante la cual puede ser aplicada a distintos tipos de objetos.

En bases de datos orientadas a objetos los usuarios pueden definir operaciones sobre los datos como parte de la definición de la base de datos. Una operación (llamada función) se especifica en dos partes: la interfaz (o signatura) de una operación que incluye el nombre de la operación y los tipos de datos de sus argumentos (o parámetros) y la implementación (o método) de la operación, ambas especificadas separadamente pudiendo modificarse sin afectar la interfaz. Los programas de aplicación de los usuarios pueden operar sobre los datos invocando a dichas operaciones a través de sus nombres y argumentos, sea cual sea la forma en la que se han implementado. Esto podría denominarse independencia entre programas y operaciones (12).

Se está trabajando en SQL3, que es el estándar de SQL92 ampliado, que soportará los nuevos conceptos orientados a objetos y mantendría compatibilidad con SQL92.

1.4.2.5 Bases de datos deductivas

Un sistema de base de datos deductivo permite hacer deducciones a través de inferencias. Se basa principalmente en reglas y hechos que son almacenados en la base de datos. También las bases de datos deductivas son llamadas bases de datos lógicas, a raíz de que se basan en lógica matemática (12).

1.4.2.6 Bases de datos distribuidas

En este modelo la base de datos está almacenada en varias computadoras conectadas en red, las mismas surgen debido a la existencia física de organismos descentralizados, lo que le da la capacidad de unir las bases de datos de cada localidad y acceder así a distintas universidades, sucursales de tiendas, etc. (12).

1.5 Sistemas Gestores de Base de Datos (SGBD).

Los Sistemas Gestores de Base de Datos (SGBD) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta (13).

Dentro de los SGBD existen los **comerciales** y **los libres**, la principal diferencia entre ambos radica en sus licencias, ya que los libres como su nombre indica poseen una licencia libre.

Los primeros SGBD surgen por la necesidad de sustituir los métodos de almacenamiento basados en Sistemas de Archivos los cuales se limitaban a la estructuración física de los datos. Los SGBD pueden verse como una sola estructura de almacenamiento mediante restricciones de integridad permitiendo manipular la información a través de esquemas que establecen métodos de acceso (13).

Los segundos surgen por la necesidad imperiosa de ahorrarle a pequeñas y medianas empresas el dinero correspondiente al pago de las licencias así como por las numerosas ventajas que implica utilizar SGBD cuyos principios estén basados en el Software Libre.

Dentro de los SGBD libres se encuentran un conjunto que incluye los que se detallan a continuación:

1.5.1 SQLite

SQLite es un sistema gestor de bases de datos (BD) relacional compatible con ACID, y que está contenido en una relativamente pequeña (~275 Kb) biblioteca en C. El mismo es un proyecto de dominio público creado por D. Richard Hipp (14). Es un motor de BD SQL embebido, el cual a diferencia de la mayoría de bases de datos SQL, no tiene un proceso servidor independiente, lee y escribe directamente en los archivos de disco normal producto a que una BD SQL completa con varias tablas, índices, triggers y vistas, está contenida en un único archivo de disco.

A diferencia de los sistemas de gestión de BD cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica, en lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo, dicho programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones, esto reduce la latencia en el acceso a la

BD, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la BD (definiciones, tablas, índices, y los propios datos), son guardados como un solo fichero estándar en la máquina host; este simple diseño se logra bloqueando todo el fichero de base de datos al principio de cada transacción (15).

La biblioteca implementa la mayor parte del estándar SQL-92, incluyendo transacciones de BD atómicas, consistencia de BD, aislamiento y durabilidad (ACID), triggers y la mayor parte de las consultas complejas. SQLite usa un sistema de tipos inusual, en lugar de asignar un tipo a una columna como en la mayor parte de los sistemas de bases de datos SQL, los tipos se asignan a los valores individuales. Por ejemplo, se puede insertar un string en una columna de tipo entero; a pesar de que SQLite tratará en primera instancia de convertir la cadena en un entero (14).

Varios procesos o hilos pueden acceder a la misma base de datos sin problemas, varios accesos de lectura pueden ser servidos en paralelo; un acceso de escritura sólo puede ser servido si no se está sirviendo ningún otro acceso concurrentemente, en caso contrario, el acceso de escritura falla devolviendo un código de error (o puede automáticamente reintentarse hasta que expira un timeout configurable). Esta situación de acceso concurrente podría cambiar cuando se está trabajando con tablas temporales. Sin embargo, podría producirse un deadlock debido al multithread.

Algunas de las características específicas de SQLite se presentan a continuación:

1.5.1.1 Extensiones del Lenguaje SQL

SQLite proporciona una serie de mejoras en el lenguaje SQL que no se encuentran normalmente en otros Motores de BD sean comerciales o no, provee declaraciones como REPLACE y ON CONFLICT, cláusulas que permiten controlar las limitaciones de resolución de conflictos. Además soporta ATTACH y DETACH, comandos que permiten que múltiples bases de datos independientes sean utilizadas en una misma consulta. Por último, define las API que permiten al usuario añadir nuevas funciones SQL y el cotejo de las secuencias (16).

1.5.1.2 De dominio Público

El código fuente de SQLite es de dominio público, ninguna reivindicación se hace de los derechos de autor en cualquier parte del núcleo del código fuente. Esto significa que nadie es capaz de hacer lo que quiera con el código fuente de SQLite. Existen otros motores de BD basados en SQL liberados de licencias que permiten al código ser amplia y libremente utilizados pero estos motores se siguen rigiendo por la ley de derechos de autor. SQLite es diferente ya que en el derecho de autor la ley simplemente no se aplica (16).

1.5.1.3 De código Legible

El código fuente de SQLite está diseñado y codificado para ser legible y accesible hasta para un programador medio. Todos sus procedimientos, estructuras de datos y variables están cuidadosamente comentadas con información útil e integral para que de esa forma se pueda comprender su estructura y el significado de cada una de las partes de la aplicación (16).

1.5.1.4 Registros de Longitud Variable

La mayoría de los motores de BD SQL, asignan una cantidad fija de espacio en disco para cada fila en la mayoría de los campos de una determinada tabla. Tomemos un clásico ejemplo para entender este concepto sencillo, tomemos un campo de tipo VARCHAR (100), esto significa que el motor de BD asignará 100 Bytes de espacio en disco fijo, independientemente de la cantidad de información que se almacene en ese campo.

SQLite por el contrario, utiliza únicamente la cantidad de espacio de disco necesario para almacenar realmente la información en ese campo. Tomando el ejemplo anterior, si se quisiera almacenar un solo caracter en un campo definido como VARCHAR(100), entonces sólo un único byte de espacio de disco se consume, realmente una característica especial para ahorrar espacios en memorias (16).

El uso de registros de longitud variable por SQLite, tiene una serie de ventajas, entre ellas el resultado de un pequeño archivo de BD, también hace que la BD funcione más rápidamente, debido a que hay menos información desperdiciada que leer y recorrer.

1.5.1.5 Tipado Dinámico

La mayoría de los Motores de BD SQL utilizan un tipado convencional de datos. ¿Qué significa esto?, un dato se asocia con cada columna de una tabla de valores y sólo ese tipo de dato se permite almacenar en

esa columna. SQLite, por el contrario, rompe con este pensamiento, imponiendo el concepto del Tipado Dinámico, diciendo que el dato es un valor de la propiedad en sí, no de la columna en la que se va a almacenar el valor, por lo tanto permite al usuario almacenar cualquier valor de cualquier dato en cualquier columna, independientemente de la declaración del tipo de la columna (16).

1.5.1.6 Único archivo de Base de Datos

Una BD SQLite es un único archivo de disco ordinario y que además puede estar situado en cualquier parte del directorio dentro de las jerarquías de directorios. Esto trae como ventaja que el archivo de BD puede ser fácilmente copiado en algún dispositivo de memoria por ejemplo en USB o por correo electrónico. Otros motores de BD tienden por lo general a almacenar gran cantidad o colección de archivos, que sólo el motor de la BD puede llegar a tener acceso a las mismas, esto hace que los datos sean más seguros y más difíciles de acceder (16).

Algunos motores de BD SQL proporcionan la opción de escribir directamente en el disco y pasar por el sistema de ficheros todos juntos. Esto aporta un mayor rendimiento, pero a costa de una considerable complejidad de instalación y mantenimiento.

1.5.1.7 Tamaño máximo de las cadenas o los BLOB

El número máximo de una cadena o un BLOB está definido en la variable **SQLITE_MAX_LENGTH** y su valor por defecto es de 1 billón, pudiendo ser modificado en el proceso de compilación mediante el comando:

-DSQLITE_MAX_LENGTH

La implementación actual sólo apoyará una cadena o BLOB de longitud de hasta 2147483647, y algunas funciones incorporadas, tales como HEX () puede fallar mucho antes de ese punto. En la seguridad de las aplicaciones sensibles se considera que es mejor no tratar de aumentar la longitud máxima de la cadena o los blob; de hecho sería conveniente reducir el tamaño máximo de la cadena o blob a un número en el rango de unos pocos millones de ser posible (17).

1.5.1.8 Número máximo de tablas en un JOIN

SQLite no soporta JOIN´s que contengan más de 64 tablas, este límite se deriva producto a que el generador de códigos de dicho gestor utiliza mapas de bit con un bit por cada tabla en el join en el

optimizador de consulta. Utiliza un muy eficiente algoritmo voraz, cuya complejidad es O (N^2) para determinar el orden de las tablas en el join por lo que una gran operación de este tipo puede ser preparada rápidamente; por lo tanto, no existe ningún mecanismo para elevar o disminuir el límite en el número de tablas en un join en este gestor (17).

1.5.1.9 Número máximo de términos en una sentencia SELECT compuesta

Se denomina una sentencia SELECT compuesta a dos o más sentencias SELECT conectadas mediante los operadores UNION, UNION ALL, EXCEPT o INTERSECT, bajo estas condiciones se denomina término a una sola sentencia SELECT (17).

El generador de código en SQLite en una sentencia SELECT compuesta, usa un algoritmo recursivo con el fin de limitar el tamaño de la pila de ejecución en dicha operación estableciendo el número máximo de términos en 500, pudiendo ser modificado en tiempo de compilación realizando cambios a la variable **SQLITE_MAX_COMPOUND_SELECT**, la cual es la encargada de definir este valor (17).

1.5.1.10 Multiplataforma

SQLite es multiplataforma, funcionando en Sistemas Operativos (SO) como Windows, Linux y Unix.

1.5.2 DB2 Express-C

DB2 Express-C es un miembro de la familia IBM DB2 de poderosas aplicaciones de servidores de datos para manejar tanto datos relacionales como XML. DB2 Express-C es una edición de DB2 libre, sin límites y fácil de usar. La ‘C’ en DB2 Express-C significa “Comunidad”, una comunidad de usuarios que se juntan para ayudarse unos a otros, tanto en línea como fuera de ella. La comunidad DB2 Express-C consiste en una variedad de personas y compañías que diseñan, desarrollan, implementan o utilizan soluciones de base de datos, ejemplos de ellos son (18):

- Desarrolladores de aplicaciones que requieren un software de base de datos de estándar abierto para construir aplicaciones cliente-servidor, web y empresariales.
- Estudiantes, profesores y otros usuarios académicos que quieran un servidor de datos altamente versátil para enseñanza, proyectos e investigaciones.

Las ideas centrales de DB2 Express-C desde el punto de vista de usabilidad son (18):

- **Libre para desarrollar**: Para un desarrollador de aplicaciones que necesite una base de datos para su aplicación.
- **Libre para implementar**: Si se está trabajando en un ambiente de producción y se necesita una base de datos para almacenar los registros vitales.
- **Libre para distribuir**: Si se está desarrollando una aplicación o herramienta que requiera un servidor de datos embebido, se puede incluir DB2 Express-C, aún si dicho gestor se encuentra bajo estas circunstancias en una aplicación desarrollada y se desea distribuir, cada vez que se comparte o vende la aplicación, este sigue siendo libre. Es requerido que se registre con IBM para poder redistribuir DB2 Express-C; sin embargo este registro también es libre.

Dentro de las características más específicas de DB2 Express-C se encuentran:

1.5.2.1 Tamaño de la BD

DB2 Express-C no establece límites en cuanto a tamaño de la BD (18).

1.5.2.2 Utilización de recursos

En términos de recursos de hardware, DB2 Express-C puede ser instalado sobre sistemas con cualquier número de núcleos de CPU y memoria, sin embargo, este sólo utilizará hasta 2 núcleos y 2GB de memoria; los sistemas pueden ser físicos o virtuales creados por particionamiento o controlando el software de máquina virtual, posibilitando correr sobre sistemas más pequeños si se prefiere, por ejemplo un sistema de un solo procesador con 1GB de memoria (18).

1.5.2.3 Particionamiento de la Base de Datos

La característica de particionamiento de BD (DPF) permite a la base de datos ser extendida a través de múltiples particiones, las cuales pueden estar alojadas en varias computadoras. DPF está basado en una arquitectura shared-nothing que plantea que cada computadora, como es añadida al grupo de partición, trae el poder adicional de procesamiento de datos con sus propias CPUs y memoria. DPF es en particular útil en ambientes de servidor de datos grandes como data warehouses donde las sentencias de los sistemas de apoyo de decisión (DSS) son controlados (18); dicha característica no está disponible para DB2 Express-C.

1.5.2.4 Geodetic Extender

Permite desarrollos para aplicaciones de inteligencia de negocios que requieren un análisis de localización geográfica mucho más sencillo. DB2 Geodetic Extender puede construir un globo mundial virtual a cualquier escala. La mayoría de información de localización es tomada usando sistemas worldwide, por ejemplo el sistema de satélites de posicionamiento global (GPS), y puede ser representado en coordenadas de latitud/longitud (geocódigo), esta característica sólo se encuentra disponible en versiones propietarias de DB2 por lo que la versión libre de este producto (DB2 Express-C) no cuenta con esta característica (18).

1.5.2.5 Workload Management (WLM)

El manejador de carga es el encargado de manejar cargas de trabajo en una base de datos en base a prioridades de un usuario y de aplicaciones combinadas con la disponibilidad de recursos y límites. Permite regular la carga de trabajo de la base de datos y consultas, de tal manera que consultas importantes y de alta prioridad pueden correr inmediatamente, y además previene consultas espontáneas que no tienen buen rendimiento evitando el uso excesivo de los recursos del sistema de manera que el mismo puede correr eficientemente, esta característica no se encuentra disponible en DB2 Express-C lo que trae como consecuencia que el balance de carga llevado a cabo en el servidor no sea lo suficientemente óptimo (18).

1.5.2.6 Arquitectura Cliente-Servidor

DB2 Express-C está diseñado para soportar la arquitectura cliente-servidor elemento que posibilita entre otras cosas la implementación de aplicaciones que utilizan este tipo de arquitectura de DB.

1.5.2.7 Réplica de Datos

En DB2 Express-C los servicios de replicación de BD no están disponibles (19).

1.5.2.8 Paralelismo de Consultas

Esta característica solamente está disponible para versiones propietarias de DB2 por lo tanto la versión DB2 Express-C no cuenta con esta característica (19).

1.5.2.9 Multiplataforma

Se puede considerar que DB2 Express-C no es multiplataforma puesto que solamente está disponible en Linux, Solaris (x64), y Windows 2003, 2000, XP, y Vista (18).

1.5.3 PostgreSQL

La implementación del SGBD PostgreSQL comenzó en 1986, los conceptos iniciales para el sistema fueron presentados en “The Design of Postgres”, la definición del modelo de datos inicial apareció en “The Postgres Data Model”, el diseño del sistema de reglas fue descrito en ese momento en “The Design of The Postgres Rules System” y la lógica y arquitectura del gestor de almacenamiento fueron detalladas en “The Postgres Storage System” todas estas publicaciones sobre el SGBD PostgreSQL (20).

PostgreSQL ha pasado por varias revisiones importantes desde entonces; el primer sistema de pruebas fue operacional en 1987 y fue mostrado en la Conferencia ACM-SIGMOD de 1988. Se lanzó la versión número 1, descrita en “The Implementation of Postgres”, a unos pocos usuarios externos en Junio de 1989, en respuesta a una crítica del primer sistema de reglas (“A Commentary on The Postgres Rules System”), este fue rediseñado (“On Rules, Procedures, Caching and Views in Database Systems”) y la versión 2, que salió en Junio de 1990, lo incorporaba. La versión 3 apareció en 1991 y añadió una implementación para múltiples gestores de almacenamiento, un ejecutor de consultas mejorado y un nuevo sistema de reescritura de reglas. En su mayor parte, las siguientes versiones hasta el lanzamiento de Postgres95 se centraron en mejorar la portabilidad y la fiabilidad (20).

En 1996, el proyecto cambia su concepto al mundo del código abierto e inicia su versión 6.0. En el año 2000 se comienza la implementación del soporte para Ipv6. Corre el año 2004, y ya PostgreSQL es reconocido como uno de los mejores motores de bases de datos del mundo, y es el 5to SGBD más popular en Estados Unidos ese mismo año. En el año 2005, PostgreSQL pasa la prueba de Coverty Inspected, en la cual encontraron sólo 20 errores en 775,000 líneas de código, lo cual constituyó un orgullo y un compromiso para el proyecto (21).

PostgreSQL proporciona un gran número de características que normalmente solo se encontraban en las bases de datos comerciales tales como Oracle o SQL Server. A continuación se presentan un conjunto de

características, a partir de PostgreSQL 7.1.x y hasta la versión 8.3 que es la última versión estable del producto.

1.5.3.1 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 (22).

1.5.3.2 SGBD Objeto-Relacional

PostgreSQL aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas. Ejemplos de su avanzada funcionalidad son consultas SQL declarativas, control de concurrencia multi-versión, soporte multi-usuario, transacciones, optimización de consultas, herencia, y arrays (22).

1.5.3.3 Arquitectura Cliente-Servidor

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 (22).

1.5.3.4 Alta concurrencia

Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés), el cual permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo **commit**. Esta estrategia es superior al uso de bloqueos por tabla o por filas comunes en otras bases, eliminando la necesidad del uso de bloqueos explícitos (23).

1.5.3.5 Amplia variedad de tipos nativos

PostgreSQL provee nativamente soporte para:

- Números de precisión arbitraria.
- Texto de largo ilimitado.
- Figuras geométricas (con una variedad de funciones asociadas)
- Direcciones IP (IPv4 e IPv6).

- Bloques de direcciones estilo CIDR.
- Direcciones MAC.
- Arrays.

Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS (23).

1.5.3.6 Licencia de PostgreSQL

Está bajo la licencia BSD (24).

1.5.3.7 Altamente Extensible

Soporta operadores, funcionales métodos de acceso y tipos de datos definidos por el usuario (22).

1.5.3.8 Soporte SQL Comprensivo

PostgreSQL soporta la especificación SQL99 e incluye características avanzadas tales como las uniones (joins) SQL92 (22).

1.5.3.9 Integridad Referencial

Soporta integridad referencial, la cual es utilizada para garantizar la validez de los datos de la BD (22).

1.5.3.10 Multiplataforma

PostgreSQL es un SGBD multiplataforma (21).

1.5.4 MySQL

MySQL es un sistema de gestión de bases de datos relacional, fue creado por la empresa sueca MySQL AB, la cual tiene el copyright del código fuente del servidor SQL, así como también de la marca. MySQL es un software de código abierto, licenciado bajo la GPL de la GNU, aunque MySQL AB distribuye una versión comercial, en lo único que se diferencia de la versión libre, es en el soporte técnico que se ofrece, y la posibilidad de integrar este gestor en un software propietario, ya que de otra manera, se vulneraría la licencia GPL. El lenguaje de programación que utiliza MySQL es Structured Query Language (**SQL**) que

fue desarrollado por IBM en 1981 y desde entonces es utilizado de forma generalizada en las bases de datos relacionales (25).

Dicho gestor surgió alrededor de la década del 90, Michael Windenis comenzó a usar mSQL para conectar tablas usando sus propias rutinas de bajo nivel (ISAM). Tras unas primeras pruebas, llegó a la conclusión de que mSQL no era lo bastante flexible ni rápido para lo que necesitaba, por lo que tuvo que desarrollar nuevas funciones. Esto resultó en una interfaz SQL a su base de datos, totalmente compatible a mSQL (25).

MySQL es muy utilizado en desarrollo de aplicaciones web junto a Drupal o phpBB, en plataformas (Linux/Windows-Apache-MySQL-PHP/Perl/Python), y por herramientas de seguimiento de errores como Bugzilla. Su popularidad está muy ligada a PHP, que a menudo aparece en combinación con MySQL. Es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional **MyISAM**, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a dicho gestor ideal para este tipo de aplicaciones pero utilizando el motor transaccional **InnoDB** (26).

InnoDB es una tecnología de almacenamiento de datos de fuente abierta para MySQL, incluido como formato de tabla estándar en todas las distribuciones de MySQL AB a partir de las versiones 4.0. Su característica principal es que soporta transacciones de tipo ACID y bloqueo de registros e integridad referencial. Ofrece una fiabilidad y consistencia muy superior a MyISAM, la anterior tecnología de tablas de MySQL, el mejor rendimiento de uno u otro formato dependerán de la aplicación específica (26).

Algunas de las características de MySQL se presentan a continuación:

1.5.4.1 Seguridad

MySQL cuenta con un sistema de privilegios y contraseñas que es muy flexible y seguro, y que permite verificación basada en el host. Las contraseñas son seguras porque todo el tráfico de contraseñas está encriptado cuando se conecta con un servidor (27).

1.5.4.2 Interioridades y portabilidad

- Escrito en C y en C++
- Probado con un amplio rango de compiladores diferentes.

- APIs disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl.
- Uso completo de multi-threaded mediante threads del kernel. Pueden usarse fácilmente múltiples CPUs si están disponibles.
- Proporciona sistemas de almacenamiento transaccional y no transaccional.
- Usa tablas en disco B-tree (MyISAM) muy rápidas con compresión de índice.
- Un sistema de reserva de memoria muy rápido basado en threads.
- Tablas hash en memoria, que son usadas como tablas temporales.
- Las funciones SQL están implementadas usando una librería altamente optimizada y deben ser tan rápidas como sea posible. Normalmente, no hay reserva de memoria tras toda la inicialización para consultas.

1.5.4.3 Arquitectura Cliente-Servidor

MySQL está diseñado para soportar arquitectura cliente-servidor, el servidor está disponible como un programa separado para usar en un entorno de red cliente/servidor. También está disponible como biblioteca y puede ser incrustado en aplicaciones autónomas. Dichas aplicaciones pueden usarse por sí mismas o en entornos donde no hay red disponible (27).

1.5.4.4 Conectividad

Los clientes pueden conectarse con el servidor MySQL usando sockets TCP/IP en cualquier plataforma. En sistemas Windows de la familia NT (NT, 2000, XP, o 2003), los clientes pueden usar named pipes para la conexión y en sistemas Unix, los clientes pueden conectar usando ficheros socket Unix. La interfaz para el conector ODBC (MyODBC Open Database Connectivity) proporciona a MySQL soporte para programas clientes que usen conexiones ODBC (27).

1.5.4.5 Requerimientos

MySQL tiene un bajo costo en requerimientos para la elaboración de bases de datos, debido a su bajo consumo puede ser ejecutado en una máquina con escasos recursos sin ningún problema (25).

1.5.4.6 Clientes y herramientas

MySQL server tiene soporte de comandos SQL para chequear, optimizar, y reparar tablas. Estos comandos están disponibles a través de la línea de comandos y el cliente **mysqlcheck**, también incluye

myisamchk, una utilidad de línea de comandos muy rápida para efectuar estas operaciones en tablas MyISAM (27).

1.5.4.7 Licencia de MySQL

Licenciado bajo la GPL de la GNU (25).

1.5.4.8 Multiplataforma

MySQL es un SGBD multiplataforma (25).

1.6 Sistemas Gestores de Base de Datos Libres: Comparación.

En el siguiente apéndice se mostrarán las características más importantes de los Sistemas Gestores de Base de Datos Libres PostgreSQL y MySQL. En el mismo no se analizará SQLite debido a que no está recomendado para sistemas de software que utilicen grandes bases de datos, utiliza el tipado dinámico característica que afecta el buen funcionamiento de una aplicación cuya implementación o propósito sea el de utilizar el tipado convencional y además no soporta la arquitectura cliente-servidor, elemento muy utilizado en las aplicaciones en la actualidad; tampoco será analizado BD2 Express-C pues aunque no presenta limitaciones en cuanto tamaño de la base de datos, no es un SGDB libre multiplataforma, no soporta la replicación de los datos y no se puede desarrollar particionamiento de tabla ni de base de datos, todos estos, elementos importantes a la hora de desarrollar aplicaciones actuales ya sea web o desktop.

1.6.1 PostgreSQL vs MySQL

Cuando a sistemas gestores de bases de datos relacionales libres se refiere, realizar una elección entre MySQL y PostgreSQL es una decisión difícil a la cual se deben enfrentar todos los desarrolladores de aplicaciones de software que utilicen bases de datos. Ambos servidores son soluciones comprobadas al paso del tiempo y que compiten fuertemente con las bases de datos de software propietario. **MySQL** ha sido durante mucho tiempo la más rápida, pero de los dos gestores de base de datos es el que cuenta con menos funciones, mientras que **PostgreSQL** es un sistema de base de datos más denso, el cual a menudo se describe como la versión de código abierto de Oracle.

Como la innovación en estas bases de datos ha progresado, cada comunidad de desarrollo ha participado activamente realizando cambios para hacer frente a sus respectivos conjuntos de desventajas. El

resultado de este trabajo ha hecho más difícil determinar objetivamente qué BD puede ser más adecuada para una aplicación determinada (28).

MySQL y PostgreSQL tienen un impresionante conjunto de características que aumentan la integridad de los datos, funcionalidad y rendimiento. Las características incluidas en una base de datos pueden ayudar a mejorar el rendimiento, la funcionalidad, o la estabilidad. Algunos elementos a tener en cuenta a la hora de seleccionar uno u otro gestor se presentan a continuación:

Llaves Foráneas

La correcta aplicación de las técnicas de diseño de BD como la Normalización se basa en la capacidad de las BD de utilizar las llaves foráneas para mapear las relaciones existentes entre las tablas, en MySQL las llaves foráneas únicamente son compatibles utilizando el motor transaccional InnoDB (y algunos motores de almacenamiento más recientes, como PBXT y Falcón, que están en desarrollo temprano y generalmente no se consideran preparados para su uso en producción). La filosofía básica de diseño de base de PostgreSQL es producir errores o advertencias en situaciones similares cuando una operación es ambigua o no soportada por el gestor (28).

Transacciones DLL

En PostgreSQL cuando se está dentro de una transacción las operaciones como actualizaciones en una tabla pueden deshacerse utilizando las cláusulas **ROLLBACK**, aunque hay que tener en cuenta que algunas son irreversibles (como crear o eliminar una base de datos o una tabla). En el caso de MySQL, dicho gestor no soporta el retroceso de transacciones utilizando el motor MyISAM e incluso utilizando el motor InnoDB las transacciones DLL no son soportadas ya que las operaciones DLL causan un COMMIT que confirman la transacción que se está ejecutando (28).

Rendimiento

Los SGBD pueden ser optimizados de acuerdo con el entorno en el que corren. Por lo tanto, es muy difícil dar una comparación precisa en el rendimiento, sin prestar atención a la configuración y el medio ambiente. PostgreSQL así como MySQL emplean diversas tecnologías para mejorar el rendimiento en el nivel básico:

MySQL comenzó su desarrollo con un enfoque en la velocidad, mientras que PostgreSQL comenzó a desarrollarse con un enfoque sobre las características y normas. Por lo tanto, MySQL es a menudo considerado como el más rápido de los dos. La configuración por defecto de PostgreSQL fue diseñada para ejecutarse en sistemas con poca memoria. El motor MyISAM de MySQL funciona más rápido que PostgreSQL sobre las consultas simples y cuando la concurrencia es baja o sigue ciertos patrones (por ejemplo, las inserciones que se realizan en tablas optimizadas y sin bloqueos). El costo de la velocidad del motor MyISAM viene de no brindar soporte a las transacciones, **llaves foráneas,** y no ofrece durabilidad garantizada en los datos (26).

Compresión de los datos

PostgreSQL puede comprimir y descomprimir sus datos sobre la marcha con un rápido sistema de compresión para encajar más datos en un espacio de disco asignado. La ventaja de los **datos comprimidos**, además de ahorrar espacio en disco, es que la lectura de datos tarda menos, por lo que lee datos con mayor rapidez (26).

Hasta la versión 5.1 de MySQL, sus motores de almacenamiento de alto rendimiento no soportan compresión sobre la marcha.

La versión de MySQL 6.0 tendrá soporte de compresión sobre la marcha con su nuevo motor de almacenamiento Falcón, los datos almacenados en las tablas del mismo están comprimidos en el disco, pero se almacenan en un formato sin comprimir en la memoria. La compresión se produce automáticamente cuando los datos se guardan al disco. Con InnoDB instalado, MySQL 5.1 soporta compresión sobre la marcha de tablas InnoDB (26).

Concurrencia

PostgreSQL escala mucho mejor, tanto en términos de la utilización de un hardware de alto rendimiento, como al hacer frente a la concurrencia. MySQL, por otra parte, se centra en tecnologías comunes de bajo rendimiento y el uso de hardware básico (28).

Algunas Limitaciones de ambos gestores:

Disparadores

Tanto MySQL como PostgreSQL soportan disparadores. Un disparador PostgreSQL puede ejecutar cualquier función definida por el usuario desde cualquiera de sus lenguajes de procedimiento, no sólo PL/pgSQL.

Los disparadores de MySQL son activados solamente por comandos SQL. Estos no son activados por cambios en las tablas realizados por APIs que no transmiten por las declaraciones de SQL al servidor MySQL, en particular, no son activadas por las actualizaciones hechas utilizando el NDB API.

PostgreSQL también soporta las “reglas”, que permiten operar en el árbol de sintaxis de la consulta, y puede hacer algunas operaciones más simplificadas que tradicionalmente son realizadas por disparadores (29).

La sintaxis para la definición de los disparadores en PostgreSQL no es tan sencilla como en MySQL. En PostgreSQL se requiere una definición de una función con la devolución del tipo de datos específico.

Replicación y Alta Disponibilidad (HA)

La replicación es la capacidad de un sistema de gestión de base de datos de duplicar sus datos almacenados a efectos de brindar copias de seguridad y una manera de prevenir la inactividad de la base de datos (30). Ambos PostgreSQL y MySQL soportan replicación:

PostgreSQL es modular por su diseño, y la replicación no está en el núcleo. Hay varios paquetes que permiten la replicación en PostgreSQL:

- PGCluster

- Slony-I
- DBBalancer
- pgpool
- PostgreSQL table comparator
- SkyTools
- Sequoia
- Bucardo

Es un error común pensar que estos “paquetes de terceros” de alguna manera son menos integrados. Slony, por ejemplo, fue diseñado y construido por Jan Weick, un miembro del equipo del núcleo de PostgreSQL, y tiene otros miembros de la comunidad que participan en su continuo diseño y mantenimiento.

Sin embargo, Slony es considerablemente más lento y utiliza más recursos que MySQL y su replicación incorporada, ya que utiliza SQL y disparadores en lugar de un registro binario de envío para replicar los datos a través de los servidores, esto lo puede hacer menos adecuado para grandes instalaciones de clústers con necesidades de alto rendimiento. Recientemente, el equipo del núcleo de PostgreSQL anunció que la replicación básica se ha previsto como parte de la liberación 8.4.

MySQL brinda soporte para replicación. A partir de la versión 5.1, MySQL soporta dos formas de replicación; replicación basada en declaración (SBR) y replicación basada en la fila (RBR). SBR, recolecta las consultas SQL que realizan cambios a la base de datos en un registro binario a los cuales los servidores esclavos se conectan para copiar sus cambios (31).

A diferencia RBR registra los cambios incrementales a las filas en el registro binario que luego son aplicados a los esclavos. Algunos motores de almacenamiento, tales como NDB y Falcón sólo soportan la replicación usando este nuevo formato basado en la fila.

Subconsultas

MySQL y PostgreSQL soportan subconsultas, pero en MySQL algunas subconsultas, dependiendo de la forma en que se estructuren pueden causar un gran impacto en el rendimiento, esto será corregido en la versión 6.0.

Desarrollo

PostgreSQL no es controlado por una sola empresa, sino que se basa en una comunidad global de desarrolladores y empresas para desarrollarlo.

MySQL es propiedad y está patrocinado por una sola empresa con fines de lucro, la empresa sueca MySQL AB, que posee los derechos de autor a la mayoría del código.

- ✓ MySQL es PRODUCTO de código abierto.
- ✓ PostgreSQL es proyecto de código abierto.

A continuación se presenta una comparación mucho más abarcadora entre estos dos SGBD en cuanto a otras características que son usadas por aplicaciones de mediana o gran envergadura, y para uso empresarial, lo que ilustrará como se encuentran estos SGBD en cuanto a la implementación de funcionalidades que poseen sus competidores comerciales.

Leyenda para Tabla 1:

🟢 Elementos Positivos 🔴 Elementos Negativos

Característica	MySQL	PostgreSQL
Soporte Técnico	Sun MicroSystems	Grupo Global de Desarrollo PostgreSQL
Licencia de Software	GPL	BSD
Soporte de Sistemas Operativos	Multiplataforma	Multiplataforma
Soporte para Transacciones ACID	Si	Si

Soporte de Integridad Referencial	Si	Si
Soporte Transacciones	Si	Si
Soporte Unicode	Parcial	Si
Lenguaje	SQL	SQL
Tamaño máximo de la BD	Ilimitado	Ilimitado
Tamaño máximo de tabla	Ilimitado	Ilimitado
Máximo de filas en tabla	Ilimitado	Ilimitado
Tablas Temporales	Si	Si
Máximo de columnas por fila	4096[1]	250-1600 (dependiendo del tipo de dato)
MVCC	Si	Si
Tamaño máximo de los BLOB/CLOB	4 GB	1 GB o 2GB
Tamaño máximo de los CHAR	64 KB (text)	1 GB
Tamaño máximo de los NUMBER	64 bits	Ilimitado
Tamaño mínimo de los DATE	1000	-4713
Tamaño máximo de los DATE	9999	5874897
Tamaño Máximo de caracteres en el nombre de una columna	64	63
Vistas Materializadas	No[2]	No[3]

[1] InnoDB está limitado a 1000 columnas.

[2] Las vistas materializadas en MySQL pueden ser emuladas usando procedimientos almacenados y triggers.

Cantidad de Índices por Tablas	Ilimitado	Ilimitado
Índices	Figura	Figura
Herencia entre Tablas	No	Si
Capacidades de Base de Datos	Figura	Figura
Dominio de Datos	No	Si
Cursores	Si	Si
Triggers	Si	Si
Funciones	Si	Si
Procedimientos Almacenados	No	Si
Rutinas Externas	Si	Si
Encriptación Nativa de Red	Si (con SSL 4.0)	Si
Protección contra Fuerza Bruta	No	No
Control de Acceso	Parcial	Si
Certificación de Seguridad	No	Si (EAL1[4])
Compatibilidad de Directorios Empresariales	No	Si (LDAP)
Particionamiento de Tabla	No	Si (básico)
Manejo de Excepciones en las funciones y los procedimientos	Si (declarando	Si

[3] Las vistas materializadas en PostgreSQL pueden ser emuladas con funciones y triggers usando los lenguajes procedurales PL/pgSQL, PL/Perl, PL/Python u otros lenguajes.

[4] Las transmisiones en el tráfico de red son transmitidas por un canal seguro (no en texto plano, en general encriptado usando SSL).

	manejadores para cada tipo de excepción)	
Múltiples métodos de autenticación	No	Si
Creación de operadores por los usuarios	No	Si
Escribir antes de registrar los logs **(importante para la recuperación y seguimiento de los logs)**	Si	Si
Tablespaces	No	Si
Software para replicación asincrónica de código abierto	Si	Si
ACID	Si (InnoDB)	Si
Soporte para UTF-8	Si	Si
Verificación de Restricciones	Si (InnoDB)	Si
Monitoreo de BD	Si	Si
Capacidad de consultar otras BD en otros servidores locales o remotos	Si	Si (DBLink)
Online/Hot Backups	Si	Si
Soporte nativo para GIS[5]	Si	Si (PostGIS)

[5] Sistema de Información Geográfica son herramientas que permiten a los usuarios crear consultas interactivas, analizar la información espacial, editar datos, mapas y presentar los resultados de todas estas operaciones

Actualizaciones regulares de software	Si	Si

Tabla 1: Tabla comparativa de MySQL y PostgreSQL

La tabla comparativa de MySQL y PostgreSQL demuestra que en cuanto a funcionalidades el SGBD PostgreSQL es superior a MySQL. Por la amplia gama de funcionalidades que implementa PostgreSQL es reconocido como uno de los mejores motores de bases de datos libres del mundo.

A continuación se presentan una serie de funcionalidades presentes en Oracle y que SIGEPOL utiliza, a través de estas se determinará qué SGBD seleccionar, mostrando si PostgreSQL o MySQL presentan dichas funcionalidades.

Leyenda para la Tabla 2:

🟢 Elementos Positivos 🔴 Elementos Negativos

Característica de Oracle	MySQL	PostgreSQL
Múltiple Control de Acceso	No	Si
Secuencias	No	Si
Lenguaje Procedural	Si	Si
Actualizaciones Regulares	Si	Si
Soporte para Múltiples Codificaciones	Si	Si
Soporte para objetos binarios grandes	Si	Si
Transaccional	Si (InnoDB)	Si
Paquetes	No	No

Esquemas	No	Si
Índices Únicos	Si	Si
Roles	No	Si
Triggers	Si	Si
Tablespaces	No	Si
Tratamiento de Excepciones	Si	Si

Tabla 2: Tabla comparativa de MySQL y PostgreSQL teniendo en cuenta elementos de ambos gestores que son utilizados por SIGEPOL

Muchas son las características que ofrece PostgreSQL, su estabilidad, escalabilidad, variedad de tipos de datos soportados, rendimiento excelente unido a que posee una licencia que permite su uso para cualquier fin, lo convierte en un poderoso sistema gestor de bases de datos.
Teniendo en cuenta cada uno de los elementos presentados anteriormente a lo largo de esta amplia comparación entre ambos SGBD libres, se decide la utilización de **PostgreSQL** como el sistema gestor de base de datos hacia el cual se realizará la migración de la solución de base de datos de SIGEPOL.

Conclusiones del Capítulo I

En este capítulo se realizó un estudio exhaustivo sobre temas relacionados con la migración hacia software libre, las licencias de software, los diferentes modelos de BD y los 4 SGBD libres seleccionados para analizar durante el desarrollo de la investigación, después de realizado el estudio de cada uno de ellos se determinó que la migración de la solución de BD de SIGEPOL se desarrollará utilizando PostgreSQL por ser el más compatible y el de mejores resultados durante el análisis realizado.

2.1 Introducción del Capítulo II

EL proceso de migración de un SGBD como Oracle a otro (PostgreSQL) en ocasiones es complicado producto a la diferencia tanto en terminología como en las formas de tratar los diferentes objetos de BD. Las palabras reservadas, las formas de declaración así como la manera de describir la arquitectura de Oracle a menudo tienen un significado totalmente diferente en PostgreSQL. Desde el punto de vista de los desarrolladores de aplicaciones que utilizan BD para gestionar su información, Oracle y PostgreSQL gestionan los datos de forma similar, teniendo en cuenta que las diferencias internas entre ambos gestores son significativas, pero si son manejadas de forma apropiada pueden causar un impacto mínimo en la migración que se desarrolle.

En este capítulo se analizarán cada uno de los elementos y objetos de base de datos propios de Oracle que son utilizados por la aplicación de SIGEPOL. Se identificarán y clasificarán los elementos a migrar y además se mostrará como realizar la migración de cada uno de estos hacia el SGBD PostgreSQL, gestor seleccionado y hacia el cual se llevará a cabo la migración.

2.2 ¿Qué es SIGEPOL?

La Aplicación Informática SIGEPOL será la encargada de la captura de la información sobre la gestión policial, a través de la cual las dependencias policiales tendrán acceso a información nacional y podrán describir los resultados de sus actuaciones. Se adaptará a las nuevas condiciones, basada en las necesidades del Centro Regional.

Dicha aplicación cuenta con 5 módulos los cuales serán descritos a continuación:

2.2.1 Módulos de SIGEPOL

Módulo de Reseña: Este módulo permite la gestión de los casos policiales y la información relacionada con los mismos. Dentro de sus principales funcionalidades se encuentran las siguientes:

- **Gestionar Caso Policial**: Permite registrar, modificar y eliminar los casos policiales que se reportan, facilitando el acceso a dicha información.

- **Gestionar Detención**: Permite gestionar la detención de un acusado y toda la información referente a la misma.
- **Gestionar Ciudadano Reseñado**: Permite gestionar la información de un ciudadano que ha sido previamente reseñado, asociando dicha información a un caso policial.
- **Generar Reporte**: Permite generar múltiples reportes de las reseñas, filtrándolas por diferentes criterios.

Módulo de Denuncia: Este módulo permite la gestión de las denuncias realizadas. Dentro de sus principales funcionalidades se encuentran las siguientes:

- **Gestionar Denuncia**: Permite gestionar la información de las denuncias de los casos policiales.
- **Gestionar Datos de Denunciante, Víctimas y Testigos**: Permite registrar, modificar y eliminar los denunciantes, víctimas y testigos de las denuncias.
- **Generar Reportes**: Permite generar múltiples reportes de las denuncias, filtrándolas por diferentes criterios.

Módulo de Operativos Policiales: Este módulo permite la gestión de los operativos policiales que se planifican y ejecutan. Dentro de sus principales funcionalidades se encuentran las siguientes:

- **Gestionar Operativo Policial**: Permite la gestión de los operativos policiales que se planifican y ejecutan.
- **Gestionar Cierre de Operativo Policial**: Permite cerrar el operativo policial introduciendo la información del cierre del mismo.
- **Generar Reportes**: Permite generar múltiples reportes de los operativos policiales, filtrándolos por diferentes criterios.

Módulo de Administración: Este módulo permite la administración y configuración de la Aplicación Informática. Dentro de sus principales funcionalidades se encuentran las siguientes:

- **Gestionar Usuario**: Permite adicionar, eliminar y modificar los usuarios de la Aplicación Informática.
- **Gestionar Punto**: Permite gestionar los puntos desde los cuales se puede acceder a la Aplicación Informática.
- **Gestionar Tiempos de Configuración**: Permite modificar los tiempos asociados a la configuración de la aplicación.
- **Generar Reportes**: Permite generar múltiples reportes de los elementos de administración filtrándolos por diferentes criterios.

Módulo de Dependencia: Este módulo permite la gestión de las dependencias y la información relacionada con la misma. Dentro de sus principales funcionalidades se encuentran las siguientes:

- **Gestionar Dependencia Policial**: Permite gestionar las dependencias policiales.
- **Gestionar Funcionarios**: Permite gestionar los funcionarios de la dependencia policial y la información de los mismos.
- **Gestionar Arsenal Policial**: Permite mantener un control del arsenal con que cuenta la dependencia policial.
- **Entregar y Devolver el Arsenal Policial**: Permite gestionar el proceso de entrega y devolución del equipamiento de cada funcionario de la dependencia policial.

Cada uno de estos módulos y funcionalidades tienen asociado un conjunto de elementos y objetos de BD que están modelados e implementados utilizando el SGBD Oracle y que deberán ser migrados al SGBD PostgreSQL, gestor seleccionado para realizar la migración.

En los próximos epígrafes se presentarán un conjunto de pasos y procedimientos para realizar dicha migración y que permitirán que la aplicación SIGEPOL siga prestando los mismos servicios para los que fue creada.

2.3 Pasos para realizar la migración de la solución de BD de SIGEPOL

Para realizar la migración de la solución de base de datos de SIGEPOL se deberán seguir los siguientes pasos:

- Creación de las Estructuras Físicas y Lógicas de la BD de SIGEPOL. Creación de Usuarios. Creación de la BD.
- Creación de los Objetos de BD.
- Creación de las políticas de salvas y recuperación de la BD de SIGEPOL.

A continuación se presentan un conjunto de elementos que permitirán desarrollar la migración de la solución de BD de SIGEPOL hacia PostgreSQL:

2.4 Creación de las Estructuras Físicas y Lógicas de la BD de SIGEPOL. Creación de Usuarios. Creación de la BD.

2.4.1 Diseño de BD de SIGEPOL

Desde el punto de vista del diseño de DB de la aplicación informática SIGEPOL no existe ningún cambio significativo que se deba realizar, solamente se debe evaluar que en Oracle los identificadores son manejados en mayúscula sin embargo en PostgreSQL son manejados en minúscula por lo que el único cambio que se debe realizar desde el punto de vista de diseño será el de sustituir las mayúsculas por minúsculas en cada uno de los elementos y objetos de BD para tener una mayor compatibilidad con lo que propone PostgreSQL con respecto al manejo de los identificadores (ver epígrafe 2.5.1: Identificadores de los objetos de BD).

Es importante aclarar que el diseño original de BD de SIGEPOL funcionaría en PostgreSQL, pero por temas de compatibilidad con el gestor, se deberá tener en cuenta lo mencionado anteriormente para un mejor funcionamiento de la aplicación.

2.4.2 Espacios de Tablas (Tablespaces)

En Oracle toda base de datos tiene uno o más archivos de datos físicos asociados. En estos archivos de datos es donde se almacenan físicamente todos los datos de la base de datos. Para contribuir a una mejor administración de estos archivos de datos, se agrupan en unos objetos de base de datos denominados tablespaces (espacios de tablas). Un tablespace es una colección de uno o más archivos de datos. Las tablas, índices y otros objetos de base de datos se colocan dentro de estos tablespaces (32).

Los tablespaces en PostgreSQL permiten al Administrador de Base de Datos (DBA) definir localizaciones en el sistema de ficheros donde los archivos que representan objetos en la base de datos pueden ser almacenados. Mediante el uso de tablespaces, un DBA puede controlar la estructura en el disco de la instalación de PostgreSQL. Esto es útil en al menos dos formas:

- Si la partición o volumen en el que la base de datos se ha inicializado se queda sin espacio y no puede extenderse, se puede crear un tablespace en una partición diferente y se utiliza hasta que el sistema pueda ser reconfigurado (26).

- Los tablespaces permiten la utilización de los patrones de uso de los objetos de la base de datos para optimizar el rendimiento. Por ejemplo, un índice que es muy usado puede ser colocado en un disco altamente disponible, al mismo tiempo, una tabla que almacena datos que rara vez se utiliza o no participa en operaciones críticas, podría estar almacenada en un disco más lento y menos costoso (26).

Existen varias razones que justifican este modo de organización de las tablas en tablespaces:

- Permiten distribuir a nivel físico los distintos objetos de las aplicaciones.

- Un tablespace puede quedarse *offline* debido a un fallo de disco, permitiendo que el SGBD continúe funcionando con el resto.

- Como son una estructura lógica de almacenamiento, pueden usarse para aislar completamente los datos de diferentes aplicaciones.

Para la creación de un tablespace en PostgreSQL se utiliza en comando ***create tablespace***, por ejemplo:

create tablespace <nombre_tablespace> **owner** <usuario> **location** <direccion_fisica>;

La localización del tablespace debe existir y el propietario de ese directorio debe ser el usuario administrador del sistema (por defecto: postgres). Posteriormente, todos los objetos creados dentro del tablespace se almacenarán en archivos bajo ese directorio.

Tablas, índices y bases de datos enteras pueden ser asignadas a un tablespace. Para hacer esto, un usuario con el privilegio de CREATE sobre un tablespace dado debe pasar el nombre del tablespace como un parámetro al comando. Por ejemplo:

```
create table <ntabla>(<atrib td>) tablespace<nom_tsp>;
```

donde:

ntabla: nombre de tabla.

atrib: nombre de atributo.

td: tipo de dato.

nom_tsp: nombre del tablespace.

Para determinar el conjunto de tablespaces existentes en el caso de un servidor PostgreSQL, se consulta uno de los catálogos del sistema (**pg_tablespace)**, por ejemplo:

```
select spcname from pg_tablespace;
```

Definición de los tablespace de SIGEPOL:

En el caso particular de SIGEPOL, almacenar la información del sistema en espacios de tablas diferentes de los utilizados por el gestor de BD para su funcionamiento facilita:

- Una mejor organización lógica y física de los objetos en la base de datos de la aplicación.

- De ocurrir un error en un tablespaces de datos de SIGEPOL, la instancia PostgreSQL continúa su ejecución sin problemas.
- Si se tiene clasificada la información por tablespaces se puede realizar salvas (backup) y recuperación (recovery) al tipo de información que se desee sin tener que afectar a otras.

Teniendo en cuenta las ventajas de tener la solución almacenada en tablespaces y las características de los datos a almacenar, se decide migrar íntegramente los tablespace que están definidos para SIGEPOL utilizando el SGBD PostgreSQL. A continuación se presentan los tablespace y una breve descripción de su uso en la aplicación, presentándose además la forma en la que quedaría cada uno migrado al gestor seleccionado:

ts_nomencladores: este tablespace se creó con el propósito de almacenar los datos que son accedidos con mayor frecuencia por el sistema, con el fin de lograr un mayor rendimiento, en este caso se tendrá almacenada la información correspondiente a los nomencladores (13).

Dicho tablespace se mantiene en la solución propuesta y la forma en que se deberá migrar se presenta a continuación:

```
create tablespace "ts_nomencladores" owner "sigepol" location
'mnt/sigepol/nomencladores';
```

ts_auditoria: creado con el propósito de almacenar la información correspondiente a la auditoría, como las acciones que realizan los usuarios en el sistema y las excepciones que puedan ocurrir durante la ejecución de alguna instrucción, esta información tiene un crecimiento independiente de los datos que se almacenan como resultado de un proceso policial y es por eso que se separa en una estructura lógica (13). Este tablespace quedaría de la siguiente manera:

```
create tablespace "ts_auditoria" owner "sigepol" location
'mnt/sigepol/auditoria';
```

ts_usuarios: creado con el propósito de almacenar la información relativa a los usuarios del sistema, los roles que existen para la aplicación así como los puntos de red desde donde puede ser ejecutada. Es

necesario tener esto separado ya que es una información que debe mantenerse independiente de la almacenada como resultado de un proceso policial, de forma tal que si ocurre algún error en otra estructura de almacenamiento, la información necesaria para la ejecución del sistema no se vea afectada (13). Quedaría de la siguiente forma:

```
create tablespace "ts_usuarios" owner "sigepol" location
'mnt/sigepol/usuarios';
```

ts_indices: creado para separar los índices de sus tablas. Si bien los índices pertenecen lógicamente a una tabla, estos son estructuras físicas con crecimiento independiente de los objetos sobre los cuales actúan. Unas de las ventajas que tiene la creación de un tablespaces para los índices es que si el mismo sale de línea como producto de la falta de capacidad en disco, la información de las tablas puede seguir siendo consultada (13). Quedaría de la siguiente manera:

```
create tablespace "ts_indices" owner "sigepol" location
'mnt/sigepol/indices';
```

ts_historial: creado con el propósito de guardar la información correspondiente al historial policial eliminado de los ciudadanos reseñados, con el fin de aislar estos datos que exclusivamente pueden ser consultados por un grupo específico de usuarios del sistema (13). Quedaría de la siguiente forma:

```
create tablespace "ts_historial" owner "sigepol" location
'mnt/sigepol/historiapolicial';
```

ts_datos: tablespaces para almacenar el resto de la información que se recoge en el sistema, aquí estarían fundamentalmente las tablas que tienen relación con los módulos del proceso policial como reseña, denuncias y operativos policiales (13). Quedaría de la siguiente manera:

```
create tablespace "ts_datos" owner "sigepol" location
'mnt/sigepol/datosgenerales';
```

Luego de esta operación quedarían migrados los 6 tablespace que estaban propuestos en la solución de BD de SIGEPOL utilizando el SGBD PostgreSQL.

La siguiente figura muestra como quedaría la BD y los tablespace:

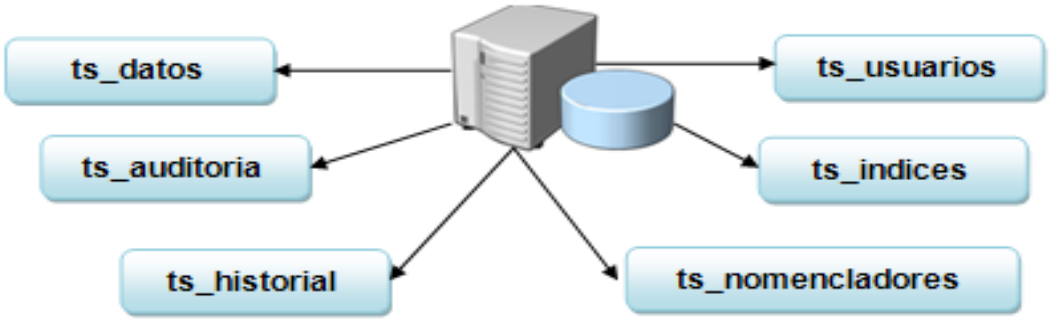

Figura 3: Estructura de tablespace de SIGEPOL.

2.4.3 Creación de Usuarios

El objetivo de la creación de usuarios es establecer una cuenta segura y útil, que tenga los privilegios adecuados y los valores por defecto apropiados. Tanto en Oracle como en PostgreSQL se puede especificar todo lo necesario para abrir una cuenta con el comando ***create user***. Esto es muy importante para el sistema ya que se puede gestionar los niveles de acceso a la BD en dependencia del objetivo con que se desee acceder, ya sea para administrar la información o para modificarla, en el caso del acceso por parte del sistema.

Los parámetros que se deben tener en cuenta para crear un usuario en el caso de Oracle son (13):

- *Username*: Nombre del Usuario.
- *Password*: Contraseña para el Usuario.
- *Default Tablespace*: Espacio de tablas por defecto en el que los objetos de este usuario serán creados. Con esto el usuario no tiene derecho de crear objetos.
- *Temporary Tablespace*: El espacio de tablas en el que se almacenarán los segmentos temporales de las ordenaciones.

- *Profile*: Asigna un perfil al usuario. Los perfiles se utilizan para restringir el uso de recursos como el tiempo de CPU.

Dado estos elementos un usuario en Oracle quedaría de la siguiente forma:

```
create user <nombre de usuario (esquema)>
identified by <clave>
default tablespace user
temporary tablespace temp;
```

Para el caso de PostgreSQL los elementos a tener en cuenta para la creación de los usuarios son diferentes, a continuación se muestra como realizar la creación de un usuario en dicho gestor:

```
create user name [ [with] option [...] ]

Donde las opciones (option) pueden ser las siguientes:

     superuser | nosuperuser

     | createdb | nocreatedb

     | createrole | nocreaterole

     | createuser | nocreateuser

     | inherit | noinherit

     | login | nologin

     | connection limit connlimit

     | [encrypted | unencrypted] password 'password'

     | valid until 'timestamp'
```

```
| in role rolename [, ...]

| in group rolename [, ...]

| role rolename [, ...]

| admin rolename [, ...]

| user rolename [, ...]

| sysid uid
```

Para el caso de SIGEPOL se crearon dos usuarios, el primero con el objetivo de administrar la BD y el segundo para acceder a la información desde el sistema, cada uno de estos con los privilegios específicos para realizar la tarea por la que fueron creados, estos usuarios fueron creados con el nombre de ***sigepol*** y ***sigepol_sistema*** (13).

A continuación se muestra como quedaría la sentencia de creación de uno de estos usuarios para de esta manera migrar los usuarios que fueron creados y así dar cumplimiento a otro de los elementos que se deben transformar en la solución de BD de SIGEPOL:

```
create user "sigepol" superuser createdb createrole login password 'password' role "postgres";
```

2.4.4 Creación de la BD

La creación de la BD es otro de los elementos a tener en cuenta a la hora de realizar la migración de la solución de BD de SIGEPOL hacia PostgreSQL, en dicho gestor como en todos los demás, una base de datos es una colección de objetos SQL (objetos de BD). Generalmente, cada objeto de base de datos (tablas, funciones, etc.) pertenece a una y solo una base de datos.

A continuación se presenta la definición de la creación de la BD en dicho gestor, así como la manera en la que quedaría la definición de la BD de la aplicación y de esta forma dejar planteado como realizar la creación de la BD para dar cumplimiento a otro de los elementos que se deben tener en cuenta a la hora de llevar a cabo la migración.

Las bases de datos son creadas con el comando ***create database***:

```
create database <name>
    [ [ whit ] [ owner [=] dbowner ]
            [ template [=] template ]

            [ encoding [=] encoding ]
            [ tablespace [=] tablespace ]
            [ connection limit [=] connlimit ] ]
```

name: Nombre de la nueva base de datos.

dbowner: El nombre del usuario de base de datos que será propietario de la nueva base de datos. Si es omitido se tomará el nombre del usuario que ejecuta el comando.

template: El nombre de la base de datos plantilla desde la cual se creará la nueva base de datos. Si es omitido, se usará la base de datos plantilla por defecto (template1).

tablespace: El nombre del tablespace que será asociado a la nueva base de datos. Si es omitido, se usará el tablespace de la base de datos plantilla. Este tablespace será el tablespace por defecto de los objetos creados en la base de datos.

encoding: Es la codificación del conjunto de caracteres que se utilizará en la nueva base de datos. Se puede especificar mediante una cadena (ejemplo: '**utf-8**'). Si se omite, se utilizará por defecto la codificación de la base de datos plantilla.

connlimit: Cantidad de conexiones concurrentes que pueden ser realizadas a la nueva base de datos.

Para el caso específico de la BD de SIGEPOL quedaría de la siguiente manera:

```
create database dbsigepol

  with encoding= 'UTF-8'

  owner=sigepol
```

El nombre que se utilizó para la base de datos de SIGEPOL fue **dbsigepol** y la codificación utilizada en el servidor Oracle fue **UTF-8**, la cual se debe mantener en la creación de la base de datos en el servidor PostgreSQL; además de esto el usuario propietario de la base de datos y de todos los objetos creados en ella será el usuario **sigepol**.

2.4.5 Esquemas de BD

Las bases de datos organizan los objetos relacionados dentro de un esquema. Por ejemplo, es normal organizar dentro de un esquema de base de datos sencillo todas las tablas y los objetos de base de datos necesarios para soportar una aplicación. La figura 4 muestra cómo podría ser un esquema de una aplicación.

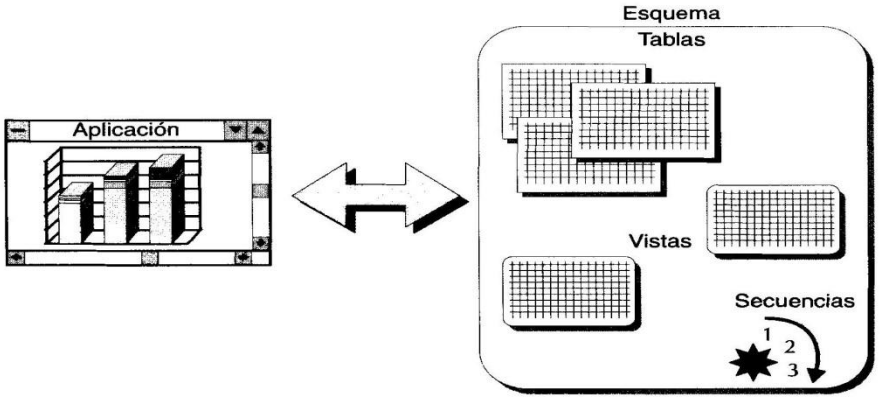

Figura 4: Ejemplo de esquema.

Tanto en Oracle como en PostgreSQL los esquemas no organizan físicamente el almacenamiento de los objetos, por el contrario, los esquemas organizan lógicamente los objetos de base de datos relacionados (26).

Una base de datos en PostgreSQL contiene uno o más esquemas, el cual contiene tablas. Los esquemas pueden contener además otros tipos de objetos, incluyendo tipos definidos por el usuario, funciones y operadores.

Existen varias razones por las cuales se recomienda el uso de los esquemas:

- Permiten a muchos usuarios usar una base de datos sin interferir unos con otros.
- Organizar los objetos de la base de datos en grupos lógicos para hacerlos más manejables.
- Los datos y objetos de terceras aplicaciones pueden ubicarse en esquemas separados, por lo que los nombres de dichos objetos no entrarían en conflicto con los de otra base de datos.

La sentencia de creación de esquemas en PostgreSQL tiene la siguiente estructura:

```
create schema <nombre_esquema> [authorization propietario]
```

nombre_esquema: Nombre del esquema que será creado. Si el nombre es omitido, será utilizado el nombre del usuario como nombre del esquema.

propietario: Nombre del usuario que será propietario del esquema. Si es omitido, será por defecto el usuario que ejecuta el comando.

La creación o acceso a los objetos en un esquema, se realiza mediante un nombre, compuesto por el nombre del esquema y el nombre del objeto separado por un punto, ejemplo:

```
create esquema.tabla(i integer);
```

Cada nueva base de datos en PostgreSQL se crea con un esquema especial nombrado "*public*". Por defecto, todos los objetos que se creen en la base de datos irán para ese esquema si no existe un esquema creado con el mismo nombre del usuario que se conecta a la base de datos.

Por defecto, un usuario no puede acceder a los objetos en los esquemas de los que no es propietario. Para ello, el propietario debe conceder el permiso de uso (USAGE) sobre el esquema.

A un usuario se le puede permitir crear objetos en cualquier esquema. Para ello, se le debe conceder el privilegio de ***create***. Por defecto, todos los usuarios tienen los privilegios de ***usage*** y ***create*** sobre el esquema public. Esto permite que todos los usuarios sean capaces de conectarse a la base de datos para crear objetos en el esquema public. Si no se desea esto, se puede revocar ese privilegio:

```
revoke create on schema public from public;
```

El primer "*public*" es el esquema, el segundo "*public*" significa "todos los usuarios".

Definición de los esquemas de SIGEPOL

La BD de SIGEPOL contará con varios esquemas producto a que PostgreSQL no tiene soporte para paquetes (ver epígrafe 2.5.8: Paquetes), se propone la creación de un esquema por cada paquete que existía en la aplicación cuando se usaba Oracle como SGBD, esto posibilita la migración de cada una de las funciones y procedimientos de cada paquete por una similar pero utilizando PostgreSQL y guardándola en un esquema con nombre igual al del paquete donde se encontraba. Esto posibilita además que los cambios desde el punto de vista de la capa de acceso a datos sean mínimos pues la llamada a las funciones y procedimientos sería similar (**paquete.procedimiento** o **paquete.función** se invocarían utilizando PostgreSQL de la siguiente forma: **esquema.función**).

Producto a esto se crearon 21 esquemas (incluyendo el esquema por defecto *public* serían 22) de los cuales se presenta a continuación la manera de crearlos:

```
--Creación de esquemas necesarios para SIGEPOL:

create schema pkactualizardenuncia authorization sigepol;

create schema pkactualizarnomenclador authorization sigepol;

create schema pkactualizaroperativopolicial authorization sigepol;

create schema pkactualizarresenna authorization sigepol;
```

```
create schema pkadministracion authorization sigepol;

create schema pkauditoria authorization sigepol;

create schema pkcasopolicial authorization sigepol;

create schema pkcomunes authorization sigepol;

create schema pkdepenfunc authorization sigepol;

create schema pkeliminardenuncia authorization sigepol;

create schema pkeliminaroperativopolicial authorization sigepol;

create schema pkhistorialpolicial authorization sigepol;

create schema pkinsertardenuncia authorization sigepol;

create schema pkinsertarnomenclador authorization sigepol;

create schema pkinsertaroperativopolicial authorization sigepol;

create schema pkinsertarresenna authorization sigepol;

create schema pkseleccionardenuncia authorization sigepol;

create schema pkseleccionarnomenclador authorization sigepol;

create schema pkseleccionaroperativopolicial authorization sigepol;

create schema pkseleccionarresenna authorization sigepol;

create schema pkservicioweb authorization sigepol;
```

De esta forma se realizaría la creación de los esquemas que se proponen para SIGEPOL utilizándolos para dar solución al problema de incompatibilidad de PostgreSQL con los paquetes.

2.5 Creación de los Objetos de BD.

2.5.1 Identificadores de los objetos de BD

Los identificadores en Oracle son manejados en mayúsculas mientras que PostgreSQL los maneja en minúscula, la siguiente tabla muestra de una forma más generalizada cómo manejan ambos SGBD los identificadores en cada uno de los casos (26):

Oracle	PostgreSQL
Identificadores: de 1-30 caracteres. Nombre de bases de datos: > 8 caracteres.	Identificadores: de 1-63 caracteres. Nombre de bases de datos: 1-63 caracteres.
Los identificadores deben comenzar con una letra y contener caracteres alfanuméricos, o los caracteres _, #, y $.	Los identificadores deben comenzar con una letra y contener caracteres alfanuméricos, o los caracteres _, #, y $.
Almacena los identificadores en mayúsculas.	Almacena los identificadores en minúsculas.
Los nombres de los tablespaces deben ser únicos.	Los nombres de los tablespaces deben ser únicos.
Los nombres de las columnas deben ser únicos en las tablas y las vistas.	Los nombres de las columnas deben ser únicos en las tablas y las vistas.
Los nombres de los índices tienen que ser únicos en el Esquema del usuario.	Los nombres de los índices tienen que ser únicos dentro de cada Esquema de la base de datos.

Tabla 3: Manejo de identificadores de objetos en Oracle y PostgreSQL

Teniendo en cuenta la tabla anterior se realizó la migración de cada uno de los identificadores utilizados en la solución de BD de SIGEPOL, pero se hace necesario aclarar que los cambios realizados no fueron significativos como para ser descritos en este epígrafe.

2.5.2 Tipos de datos

Los tipos de datos es otro de los elementos a tener en cuenta a la hora de realizar la migración, alguna de las conversiones de tipos de datos pueden ser sencillas pero en ocasiones dicha operación exigirá de una determinada evaluación entre varias opciones, a continuación se muestra una tabla con las posibles variantes que se pueden presentar y cuál deberá ser la equivalencia para desarrollar la migración adecuadamente:

Oracle	PostgreSQL
VARCHAR2	varchar, text
CLOB, LONG	varchar, text
NCHAR, NVARCHAR2, NCLOB	varchar, text
NUMBER	numeric, bigint, int, smallint, double precision, real
BINARY_FLOAT/BINARY_DOUBLE	real/double precision
BLOB, RAW, LONG RAW	bytea
DATE	date, timestamp

Tabla 4: Tabla de equivalencia de tipos de datos (Oracle a PostgreSQL)

2.5.3 Secuencias

Una secuencia es un objeto de BD que genera una serie de números enteros únicos, y es apropiada sólo para las tablas que utilizan columnas numéricas sencillas como por ejemplo claves o llaves primarias (33). Cuando una aplicación inserta una nueva fila en una tabla, la aplicación solicita simplemente una secuencia para proporcionar el siguiente valor disponible como clave primaria de la nueva fila, por consiguiente, la aplicación puede volver a utilizar posteriormente un número de secuencia generada para coordinar los valores de clave ajena de las filas secundarias relacionadas.

Tanto Oracle como PostgreSQL gestionan la generación de secuencias con una cantidad insignificante de actividad, para crear una secuencia, se utiliza el comando SQL ***CREATE SEQUENCE*** como se muestra en la siguiente tabla (26):

Oracle	PostgreSQL
CREATE SEQUENCE [esquema.] secuencia [START WITH entero] [INCREMENT BY entero] [MAXVALUE entero | NOMAXVALUE] [MINVALUE entero | NOMINVALUE] [CYCLE I NOCYCLE] [CACHE entero I NOCACHE] [ORDERI NOORDER]	CREATE [TEMPORARY | TEMP] SEQUENCE [*esquema.*] *seuencia* [INCREMENT [BY] *entero*] [MINVALUE *entero* | NO MINVALUE] [MAXVALUE *entero* | NO MAXVALUE] [START [WITH] *entero*] [CACHE *entero*] [[NO] CYCLE] [OWNED BY { *table.column* | NONE }]

Tabla 5: Sentencia de creación de secuencias (Oracle a PostgreSQL)

La principal diferencia que existe entre las secuencias en Oracle y PostgreSQL está en el alcance (***número máximo*** que se puede generar: PostgreSQL = 9223372036854775807 y Oracle=9999999999999999999999999) que ambas secuencias pueden tener y además de esto la manera en la que son invocadas o llamadas a la hora de ser utilizadas dentro del código SQL de una determinada función o procedimiento(Oracle: **nombre_secuencia.nextval** y en PostgreSQL: **nextval('nombre_secuencia')**); este elemento no constituye un problema para realizar la migración de las secuencias. A continuación se presenta una de las secuencias que existían en Oracle utilizando el SGBD seleccionado y además cómo debería ser llamada a la hora de su utilización en cualquier entorno de la BD de SIGEPOL:

```
create sequence secmodusoperandis
increment by 1
start whit 6
no maxvalue
minvalue 6
no cycle
--- Llamada a la secuencia:
      nextval('secmodusoperandis');
```

2.5.4 Tablas

Para el caso de las tablas los elementos a tener en cuenta para la migración no son relevantes puesto que la sentencia SQL de creación de la tabla cumple con el estándar de SQL92 (26), a continuación se presenta cual es la sentencia de creación de una tabla en ambos gestores y posteriormente la sentencia de creación de una de las tablas que componen la solución de BD de SIGEPOL:

Oracle	PostgreSQL
CREATE [GLOBAL TEMPORARY] TABLE [schema.]table (column datatype [DEFAULT expr] [column_constraint(s)[,…]] [,column datatype [,…]]) [table_constraint [,…]] [table_ref_constraint [,…]] [ON COMMIT {DELETE|PRESERVE} ROWS] storage_options [COMPRESS int|NOCOMPRESS])	CREATE [[GLOBAL | LOCAL] { TEMPORARY | TEMP }] TABLE [schema.]table_name([{ column_name data_type [DEFAULT default_expr] [column_constraint [...]] | table_constraint | LIKE parent_table [{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES }] [, ...]]) storage_options

Tabla 6: Sentencia de creación de tablas (Oracle a PostgreSQL)

```
--- TABLA: dciudadanoresennado

create table dciudadanoresennado(

    idciudadano          varchar(36)      default sys_guid() not null,

    esidentificado       numeric(1, 0)    not null

                         check (esidentificado = 1 or esidentificado = 0),

    sexo                 char(1)          not null

                         check (sexo = 'F' or sexo = 'M'),

    primerapellido       varchar(20)      NOT NULL,

    segundoapellido      varchar(20),

    primernombre         varchar(20)      NOT NULL,

    segundonombre        varchar(20),
```

```
    fechadenacimiento    date              NOT NULL,
    lugarnacimiento      varchar(200),
    CONSTRAINT ind_ciudadanoresennado1 PRIMARY KEY (idciudadano)
)
```

2.5.5 Funciones y Procedimientos Almacenados

Sin lugar a dudas el mayor problema que puede existir a la hora de realizar la migración que se debe desarrollar está en el proceso de migrar las funciones y procedimientos que fueron implementados en la BD de SIGEPOL utilizando Oracle hacia PostgreSQL, producto a las diferencias que existen en cuanto a la declaración e implementación de ambos objetos en uno y otro gestor. A continuación se presenta un estudio realizado sobre cómo migrar cada uno de estos objetos y la manera de hacerlo:

Migración de Funciones:

En el caso de las funciones, el primer elemento que se debe tener en cuenta es la declaración de la misma:

```
--- Declaración de la función en Oracle:
      CREATE FUNCTION ... RETURN type
--- Declaración de la función en PostgreSQL
      create function ... returns type
```

Como se puede observar la diferencia está en la palabra reservada RETURN que se coloca para indicar el tipo que devuelve la función que será implementada, para el caso de PostgreSQL se le agrega una **S** al final.

Otro de los elementos que se deben tener en cuenta es que en el cuerpo de la función para el caso de la declaración de las variables la palabra reservada **DECLARE**, en el caso de Oracle es opcional, sin embargo, en el caso de PostgreSQL es obligatoria, por lo que una función en dicho gestor quedaría de la siguiente manera:

```
      create function <nombre_funcion> returns type
      as $function$
```

```
declare
      <declaracion_variables>;
begin
      <cuerpo de la función (código SQL)>
end; $function$
language plpgsql;
```

Siguiendo este estándar de codificación se desarrolló la migración de las funciones que se implementaron en la solución de BD de SIGEPOL, una de ellas se presenta a continuación como muestra de la migración realizada con respecto a las funciones:

```
--Dado un ip y un entero q representa la posición del octeto devuelve ese
octeto.
create or replace function pkadministracion.compararip(in ip1 varchar,
in ip2 varchar) returns numeric
as $function$
declare
  varparteip1 varchar(3);
  varparteip2 varchar(3);
  parte integer;
begin
  parte := 1;
  loop
    varparteip1 := pkadministracion.parteip(ip1, parte);
    varparteip2 := pkadministracion.parteip(ip2, parte);
    if (varparteip1 <> '*' and varparteip2 <> '*') then
      if (varparteip1 <> varparteip2) then
        return 0;
      end if;
```

```
    end if;
    parte := parte + 1;
    if (parte = 5) then
      exit;
    end if;
  end loop;
  return 1;
end; $function$
language plpgsql;
```

El llamado a las funciones dentro de otras funciones es otro elemento a considerar dentro de la migración, para ello existen diferentes variantes, dos de las más usadas son la llamada a una función que no devuelve ningún valor específico (retorna VOID) y las que devuelven un valor específico, estas 2 variantes fueron las utilizadas en la solución de BD de SIGEPOL y a continuación se presenta cómo realizar ambas invocaciones:

```
--- Llamada a funciones que no devuelven valor (returns VOID):
PERFORM <nombre_funcion>(<parámetros de la función>);
--- Llamada a funciones que si devuelven valor específico:
var_declarada:= <nombre_funcion>(<parámetros de la función>);
```

nota: la variable (***var_declarada***) debe ser del mismo tipo que el que se especificó en el returns de la función.

De esta manera quedarían aclarados todos los elementos necesarios para realizar una correcta migración de las funciones de la solución de BD de SIGEPOL hacia PostgreSQL.

Procedimientos Almacenados:

En el caso de los procedimientos almacenados el análisis realizado es similar, teniendo en cuenta que aunque PostgreSQL tiene soporte para este tipo de objeto de BD lo que crea es una función cuando se ejecuta el comando de creación de un procedimiento. Basado en esto se realizó un estudio de cómo realizar la migración de los mismos, a continuación se presentan los resultados obtenidos y cómo migrar cada unos de los procedimientos que fueron implementados fuera de los paquetes y dentro de ellos.

Para la declaración de los procedimientos se debe tener en cuenta lo mencionado anteriormente en el caso de las funciones, a continuación se presenta cómo se declara un procedimiento en PostgreSQL.

```
--- Declaración de un procedimiento en PostgreSQL
      create function ... returns type
```

Luego de la declaración es importante aclarar que como mismo se analizó para las funciones, en la parte del cuerpo de la misma, la palabra reservada en el caso de la declaración de variables (**DECLARE**) es obligatoria por lo que quedaría de la siguiente manera:

```
create function <nombre_funcion> returns type
as $procedure$
declare
      <declaracion_variables>;
begin
      <cuerpo de la función (código SQL)>
end; $procedure$
language plpgsql;
```

Se puede observar que entre los símbolos de $$ está la palabra reservada **procedure**, lo que indica que es un procedimiento lo que se está declarando.

Luego de este análisis un procedimiento quedaría de la siguiente manera:

```
create or replace function pkadministracion.insertarUsuario(in
p_nombreusuario dusuariosistema.nombreusuario%type,
in p_contrasenna dusuariosistema.contrasenna%type,
in p_idfuncionario dusuariosistema.idfuncionario%type,
out p_idusuariosistema dusuariosistema.idusuariosistema%type)

as $procedure$

begin

      insert into dusuariosistema(idusuariosistema, contrasenna,
      nombreusuario,activado,idfuncionario,fechamodificacioncontrasenna)

      values (default, p_contrasenna, p_nombreusuario, 1, p_idfuncionario,
      current_date)

      returning idusuariosistema into p_idusuariosistema;

end; $procedure$

language plpgsql;
```

De esta manera, quedarían aclarados todos los elementos que hay que tener en cuenta para migrar los procedimientos de la solución de BD de SIGEPOL hacia PostgreSQL.

2.5.6 Triggers

Tanto en Oracle como en PostgreSQL los triggers o disparadores son un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación de inserción (INSERT), actualización (UPDATE) o eliminación (DELETE). Los mismos son usados para mejorar la administración de la BD, sin necesidad de contar con que el usuario ejecute la sentencia SQL, además de que pueden generar valores de columnas, prevenir errores de datos, sincronizar tablas o modificar valores de una vista.

Independientemente de esto, la manera de implementar este tipo de objeto en ambos gestores difiere en gran medida debido a las particularidades de estos SGBD para desarrollar la declaración e implementación de algunos de sus componentes u operaciones, en Oracle se crea un trigger y en su cuerpo se programan las acciones que se desean realizar, mientras en PostgreSQL la manera de

implementar es muy diferente, en dicho gestor se debe implementar una función que retorne un trigger, dentro de esta se programan las acciones que se desean ejecutar y posteriormente se creará un trigger, en el cual se especificará que se debe ejecutar la función anteriormente implementada. A continuación se presenta un ejemplo de un trigger existente en la BD de SIGEPOL y cómo quedaría después de realizada la migración:

--- Trigger **T_DFOTOPERSONANAT** para ORACLE:

```
CREATE or REPLACE TRIGGER T_DFOTOPERSONANAT
  BEFORE UPDATE on DALBUMDEFOTO FOR EACH ROW
BEGIN
  UPDATE DFOTOPERSONANATURAL  SET IDCIUDADANO = :NEW.IDCIUDADANO
  WHERE DFOTOPERSONANATURAL  = :OLD.IDCIUDADANO;
END T_DFOTOPERSONANAT;
```

--- Trigger **T_DFOTOPERSONANAT** para PostgreSQL:

Definición de la Función:

```
create or replace function t_dfotopersonanat_trfunc()
returns trigger
   as $trigger_function$
begin
   update dfotopersonanatural
   set idcuidadano = new.idcuidadano
   WHERE dfotopersonanatural.idcuidadano = old.idcuidadano;
   return new;
end; $trigger_function$
language plpgsql;
```

Definición del Trigger:

```
create trigger t_dfotopersonanat
before update on dalbumdefoto for each row
   execute procedure t_dfotopersonanat_trfunc();;
```

Luego de esto quedaría realizar una pequeña aclaración, y es que cuando se va a hacer referencia al valor viejo o nuevo de una columna en la implementación de un trigger para el caso de Oracle se realiza utilizando los dos puntos delante de la sentencia (ej→ :old.columna o :new.columna), sin embargo, en PostgreSQL los dos puntos son omitidos (ej→ old.columna o new.columna). De esta manera se realizaría la migración de todos los triggers de SIGEPOL hacia PostgreSQL.

2.5.7 Vistas

En el caso de las vistas que fueron implementadas utilizando Oracle como SGBD en la aplicación SIGEPOL, no se debe realizar ningún cambio, producto a que la compatibilidad entre ambos gestores con respecto a este tipo de objeto de BD es completa, solamente se debe evaluar como en el caso del diseño el uso de los identificadores, los cuales deben ser declarados y utilizados en minúscula, de esta manera, se da cumplimiento a otro de los elementos de la migración de la solución de BD de SIGEPOL hacia el gestor de BD libre seleccionado.

2.5.8 Paquetes

Para la separación de grupo de funciones relacionadas, en la base de datos de SIGEPOL, se utilizó un recurso de Oracle llamado “paquete”. Un paquete es un conjunto de procedimientos, funciones, variables y comandos SQL creado como una única unidad. Se utiliza para almacenar juntos objetos relacionados.

Entre las ventajas que ofrecen los paquetes en Oracle se encuentran:

- Permiten agrupar los temas relacionados, tipos y procedimientos en PL/SQL como un módulo.
- Cuando un procedimiento en un paquete se invoca, todo el paquete se carga, aunque pasa a ser costosa la primera vez, la respuesta es más rápida en las próximas invocaciones.

- Los paquetes permiten crear tipos, variables y procedimientos que son privados o públicos.

Las llamadas a los procedimientos que se encuentran dentro de los paquetes tienen la siguiente estructura:

```
paquete.procedimiento([param1, param2, ….])
```

Dado que PostgreSQL no implementa la estructura paquete que implementa Oracle, se hizo necesario utilizar una estructura que sirva para agrupar conjuntos de funcionalidades. Para lograrlo, se utilizaron los esquemas (Ver sección 2.4.5 Esquemas de DB).

Por cada paquete existente en la base de datos de SIGEPOL, se creó en PostgreSQL un esquema, los cuales contienen todos los procedimientos y funciones del paquete que representan en la solución de BD de SIGEPOL anterior, de esta manera, se logra migrar completamente la estructura de paquetes que existía anteriormente en dicha solución de BD.

2.5.9 Elementos Generales

Luego de haber realizado la migración de cada uno de los objetos de BD de SIGEPOL, quedaría analizar algunos elementos que son considerados generales puesto que se refieren a funciones propias del sistema, formas de declaración de algunos elementos como por ejemplo los cursores, que son invocados o implementados en algunas de las funciones que fueron migradas.

Funciones propias de Oracle utilizadas en la solución:

- Para el caso de la función **TRUNC** la cual es utilizada para realizar el truncado de las fechas, en PostgreSQL es utilizada la función **date_trunc** la cual realiza la misma operación sobre una determinada fecha.

- Para obtener la fecha actual del sistema en Oracle es utilizada la función **SYSDATE** la cual en el caso de PostgreSQL no existe con este nombre sino que pueden ser utilizadas las funciones **current_date**, **current_timestamp** o **localtimestamp**.

- En Oracle es común utilizar la sentencia **from dual** cuando se necesita obtener por ejemplo la fecha del sistema (**select sysdate from dual**), en el caso de PostgreSQL esta sentencia no es utilizada por lo que la anterior consulta quedaría de la siguiente manera: **select current_date**.

- Otro de los elementos muy utilizados en Oracle es el **not null** cuando se desea asignar un valor nulo a una variable, en el caso de PostgreSQL esta sentencia es también utilizada solo que de la siguiente manera **notnull**, como puede apreciarse la diferencia está dada en que uno de los gestores separa ambas palabras y en el otro no.

- Para el caso que se desee realizar una función en la cual se necesite utilizar el operador **MINUS** de Oracle debe ser sustituido por el **except** el cual es el equivalente en el SGBD PostgreSQL.

- Cuando en Oracle se desea recorrer un cursor dentro de una función o procedimiento y se utiliza para ello una de las tantas formas de realizar un ciclo dentro de un conjunto de información, es necesario establecer una condición de parada, el recurso utilizado por dicho gestor no es más que la sentencia **exit when <nombre_cursor>%notfound,** para el caso de PostgreSQL el recurso utilizado para realizar este tipo de operación es el siguiente: **exit when not found**, dicho gestor es capaz de determinar que está dentro de un ciclo y sin especificar que cursor se está recorriendo él realiza las operaciones de forma satisfactoria.

- Para las excepciones en Oracle es utilizada la sentencia SQL **NO_DATA_FOUND** en el caso de que se disparen durante inserciones, actualizaciones, eliminaciones y selecciones; para el caso de

PostgreSQL esta sólo funciona cuando se dispara una excepción en una selección, para los demás casos se debe utilizar la sentencia **if not found**.

- Para el caso de la declaración de los cursores en Oracle se realiza de la siguiente manera: **CURSOR <nombre_cursor> IS <consulta_SQL>** sin embargo para el caso del SGBD PostgreSQL la declaración de este tipo de objeto es de la siguiente forma: **<nombre_cursor> cursor is <consulta_SQL>**.

- El último elemento a mencionar es el utilizado para realizar el paginado de una determinada consulta, para estos casos era muy utilizada la declaración de una variable (**nombre_variable: fila**) a la cual se le asignaba una variable propia de Oracle nombrada **rownum** y esta era utilizada posteriormente para establecer el límite de elementos a devolver por cada página; para el caso de PostgreSQL no se hace necesaria la declaración de la variable **fila** puesto que **rownum** no existe, el paginado se realiza utilizando las palabras reservadas **limit** y **offset** (ej: **limit** posicionFinal **offset** posicionInicial) y de esta manera es realizado el paginado de las consultas SQL en el caso de la migración de este tipo de operación en la solución de BD de SIGEPOL.

2.6 Creación de las políticas de salvas (backup) y recuperación (recovery) de la BD de SIGEPOL.

La solución de BD de SIGEPOL en su versión original (utilizando Oracle) no tenía definida una política para realizar **backup** y **recovery** de la BD, sin embargo como parte de la migración se ha definido cómo se deben realizar ambas operaciones y de esta manera lograr que la información registrada en la aplicación pueda ser salvada o recuperada en el momento definido por la dirección del proyecto o por el DBA de SIGEPOL; a continuación se describirá cómo realizar ambas operaciones:

2.6.1 Copias de Seguridad (backup):

La capacidad de mantener un sistema constante de datos es probablemente la parte más importante del trabajo de los administradores de la base de datos. En caso de un fallo del sistema, la necesidad de la recuperación de los datos se convierte en una prioridad superior. Sin un procedimiento de reserva definido, la capacidad de restaurar datos a un estado constante es obstaculizada seriamente o imposible.

Un backup válido es una copia de la información sobre la base de datos necesaria para reconstruirla a partir de un estado no utilizable de la misma. Normalmente, si la estrategia de backup se basa en la copia de los ficheros de datos y en el archivado de los ficheros de log (WAL), se han de tener copias de los ficheros de datos, y también de los ficheros log archivados. Si se pierde uno de los ficheros de log se dice que se tiene un agujero en la secuencia de ficheros, esto invalida el backup, pero permite a la base de datos ser llevada hasta el principio del agujero realizando una recuperación incompleta (26).

Antes que nada, es muy importante entender ciertas reglas que determinan la situación de los ficheros y otras consideraciones que afectarán al esquema de backup (26):

- Es recomendable archivar los ficheros de log en disco, y luego copiarlos a cinta, pero siempre en un disco diferente del que soporta los ficheros de datos.
- Los ficheros copias no deben estar en el mismo dispositivo que los originales. No siempre hay que pasar las copias a cinta, ya que si se dejan en disco se acelera la recuperación. Además, si se copian las copias a cinta y se mantienen en el disco, se puede sobrevivir a diversos fallos de dispositivo.

Teniendo en cuenta las reglas anteriores, los siguientes puntos pueden considerarse un ejemplo de estrategia de backup:

- Activar el archivado de log.
- Realizar un backup al menos una vez a la semana si la BD se puede parar, en otro caso, realizar backups en caliente cada día.
- Copiar todos los ficheros log archivados. El tamaño y el número de ellos dependerá de la tasa de transacciones.
- Efectuar un export de la BD semanalmente.

Existen tres formas principales de realizar copias de seguridad (26):

- Copia de seguridad de ficheros del SO.

- Volcado SQL usando las herramientas PostgreSQL: pg_dump (pg_dumpall) y pg_restore.
- Volcado en línea y recuperación en el punto (PITR).

Para el caso específico de la solución de BD de SIGEPOL se escogió para realizar los backup la segunda de las formas, la cual se realiza utilizando herramientas propias de PostgreSQL, es más flexible, de gran utilidad, permite hacer copias de toda o parte de la BD y luego dada una copia de seguridad se podrá restaurar lo que se desee.

Además, estas herramientas sirven para la transmisión de datos entre bases de datos. Las herramientas que se van a utilizar son (26):

pg_dump: Vuelca una base de datos o parte de ella a un fichero, bien en texto plano o en un formato propio de PostgreSQL. Se puede recuperar cualquier objeto que esté en el fichero aisladamente. El servidor debe estar en marcha.

pg_restore: Recupera los objetos volcados en una copia de seguridad que no se realizó en texto plano sino en un formato propio de PostgreSQL.

psql: se usa para recuperar los volcados en texto plano.

Para realizar las copias de seguridad de SIGEPOL se propone realizar un backup completo los fines de semana, para su preparación y realización se proponen los siguientes pasos (26):

1. Se establece el parámetro de configuración **`archive_mode`** a **`on`**, y se crea un comando para la opción **`archive_command`** que realice el archivado de los archivos de log. Por ejemplo.

   ```
   archive_command = 'test ! -f /var/lib/pgsql/backup_in_progress || cp -i
   %p /var/lib/pgsql/archive/%f < /dev/null'
   ```

Este comando realizará el archivado cuando exista el fichero /var/lib/pgsql/backup_in_progress que funciona como archivo interruptor, si no existe este archivo, silenciosamente retornará cero como código de salida (permitiendo a PostgreSQL reciclar el fichero de log).

2. Luego de esta preparación, se puede realizar el backup usando un script como el siguiente:

```
touch /var/lib/pgsql/backup_in_progress
psql -c "select pg_start_backup('hot_backup');"
tar -cf /var/lib/pgsql/backup.tar /var/lib/pgsql/data/
psql -c "select pg_stop_backup();"
sleep 20
rm /var/lib/pgsql/backup_in_progress
tar -rf /var/lib/pgsql/backup.tar /var/lib/pgsql/archive/
```

Primeramente es creado el fichero /var/lib/pgsql/backup_in_progress, permitiendo el archivado de todos los ficheros de log que se puedan crear en el tiempo que dure el proceso de creación del backup. Luego de crearse el backup se borra el archivo interruptor. Por último los archivos de log archivados se añaden al backup.

2.6.2 Recuperación de la BD (recovery):

La recuperación de los datos es el proceso de revertir tanto la estructura como los datos de la base de datos en cualquier momento hacia un punto anterior en el tiempo. Teniendo así la posibilidad de recuperarse frente a los distintos problemas que puedan provocar la caída del servidor.

Una vez caído el servidor, se siguen los siguientes pasos para restaurar la BD con la última copia de la misma (26):

1. Se detiene el servicio del PostgreSQL, si está en ejecución.
2. Borrar todo los ficheros y subdirectorios existentes bajo el directorio de datos de la BD y de la carpeta raíz de los tablespaces.
3. Restaurar los ficheros de la base de datos desde el último archivo de salva.
4. Crear un fichero de recuperación recovery.conf en el directorio de datos de la BD y agregarle unas líneas como las siguientes:

   ```
   restore_command = 'cp /var/lib/pgsql/%f "%p"'
   ```

5. Iniciar el servidor. El servidor arrancará en modo recuperación y procederá a leer los ficheros de log archivados que necesita. Una vez completado el proceso de recuperación, el servidor renombrará el fichero recovery.conf a recovery.done (para prevenir que accidentalmente vuelva a entrar en modo recuperación si vuelve a caer) y comenzará las operaciones normales sobre la base de datos.

6. Inspeccionar el contenido de la base de datos para asegurar que se haya restaurado lo que se desea.

Los planes de recuperación están sujetos a problemas en un entorno de producción real, ya que sólo son aplicables cuando ha surgido un problema, mientras tanto se deben realizar pruebas sistemáticas de las salvas que se hayan realizado para verificar la integridad de las mismas y prevenir cualquier incidencia sobre estos ficheros, como pueden ser corrupción de los ficheros o deterioro de los dispositivos de almacenamiento.

Conclusiones del Capítulo II

En este capítulo se realizó un estudio exhaustivo sobre cómo realizar la migración hacia el SGBD libre PostgreSQL de cada uno de los objetos de BD utilizados en SIGEPOL, en el mismo se define cómo realizar dicha migración obteniéndose como resultado un conjunto de scripts de BD que permiten obtener de forma rápida y funcionalmente la BD de SIGEPOL totalmente migrada a PostgreSQL.

3.1 Introducción del Capítulo III.

En este capítulo se presentan un grupo de elementos necesarios para llevar a cabo la validación de la solución propuesta en el capítulo II, de esta manera queda concluido el presente trabajo de diploma el cual posibilitará la migración de la solución de BD de SIGEPOL, lo que constituyó el objetivo general de dicha investigación, para mostrar cada uno de los objetos migrados se utilizará uno de los innumerables clientes de BD que existen para interactuar con las BD implementadas con PostgreSQL, el cliente SQL Manager 2007 para PostgreSQL.

3.2 Validación de paso número 1: Creación de las Estructuras Físicas y Lógicas de la BD de SIGEPOL. Creación de Usuarios. Creación de la BD.

Como parte de este paso se desarrollaron las siguientes operaciones, las mismas llevadas a cabo con éxito, posibilitando dar los primeros pasos hacia la migración de la solución de BD de SIGEPOL hacia PostgreSQL:

- Se crearon los usuarios descritos en el capítulo II (sigepol y sigepol_sistema) de la siguiente manera:

```
create user "sigepol" superuser createdb createrole login password 'password' role "postgres";

create user "sigepol_sistema" noinherit login;
```

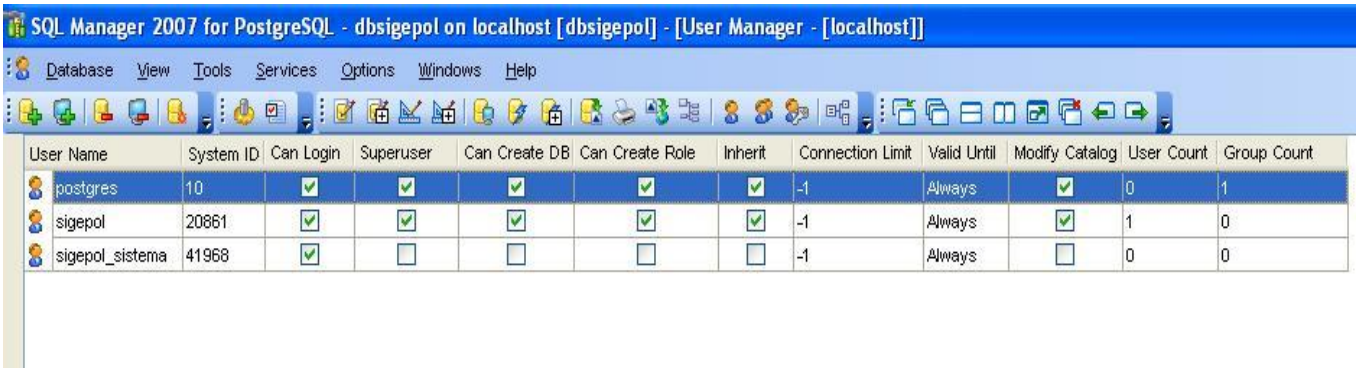

Figura 5: Imagen de los usuarios creados.

- Se creó satisfactoriamente la BD con nombre **dbsigepol** y como propietario de la misma se puso al usuario **sigepol** creado anteriormente, BD en la cual se montarían más adelante los demás objetos, la misma fue creada de la siguiente manera:

```
create database dbsigepol

with encoding= 'UTF-8'

owner=sigepol
```

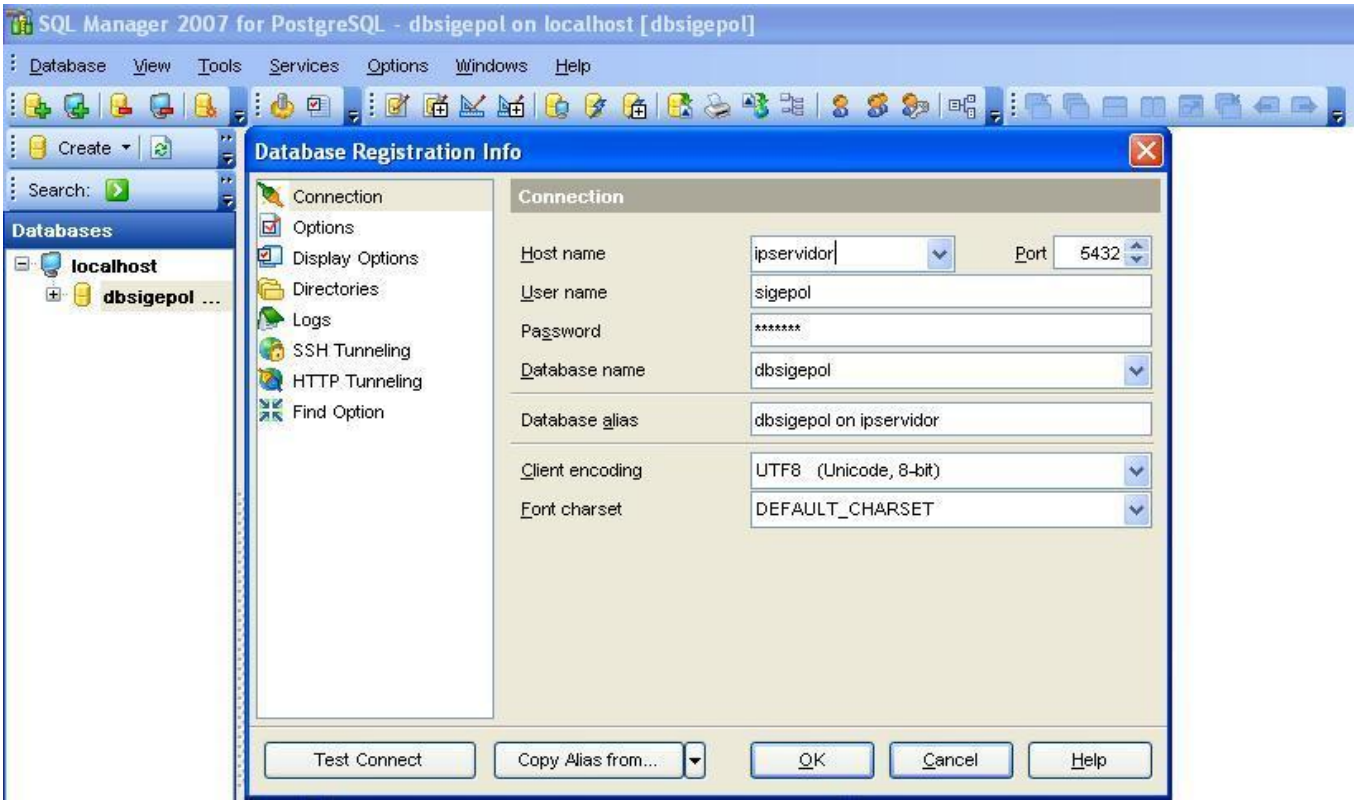

Figura 6: Imagen de la creación de la BD utilizando el cliente SQL Manager 2007 para PostgreSQL.

- Se crearon todos los esquemas necesarios para migrar la solución, de ellos solamente se mostrará el código SQL de cómo crear uno, así como una imagen de cómo quedó la BD una vez creados todos los esquemas:

```
create schema pkadministracion authorization sigepol;
```

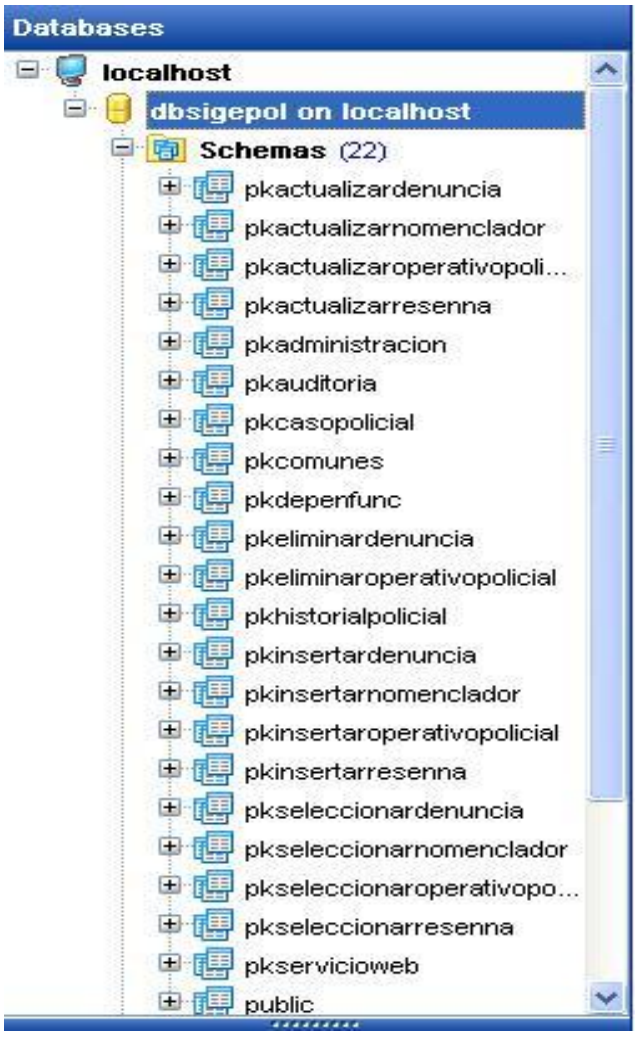

Figura 7: Imagen de los esquemas creados.

Una vez terminada la creación de los esquemas, se puede argumentar que el primero de los pasos para realizar la migración de la solución de BD de SIGEPOL se cumplió totalmente por lo que se pasará a realizar la validación de los siguientes pasos.

3.3 Validación de paso número 2: Creación de los Objetos de BD.

Como parte de este paso se puede resaltar que fueron migrados en su totalidad todos los objetos que anteriormente existían en Oracle como parte de la solución de BD de SIGEPOL, a continuación se presentarán un conjunto de objetos pertenecientes al módulo de administración y otros generales de la

aplicación para probar la veracidad de la solución planteada; como resultado de esta validación el módulo de administración debe quedar funcionando utilizando PostgreSQL como SGBD para gestionar su información:

- **Secuencias:**

```
create sequence "public"."secnbanda"
increment 21  minvalue 1
maxvalue 9223372036854775807  start 21
cache 1;
```

- **Procedimientos Almacenados:**

```
--Mostrar Roles
create or replace function pkadministracion.mostrarRoles()returns refcursor as $procedure$
declare
p_Resultado refCursor;
begin
  open p_Resultado for
   select rol.idrol, rol.rol
    from nrol rol
   order by rol.rol;
return  p_Resultado;
end; $procedure$
languaje plpgsql;
```

- **Funciones:**

```
--Dado un ip y un entero q representa la pos del octeto te devuelve ese octeto.
```

```
create or replace function pkadministracion.CompararIP(in ip1 varchar, in ip2 varchar)returns numeric
as $function$
declare
  varparteip1 varchar(3);
  varparteip2 varchar(3);
  parte integer;
begin
  parte := 1;
  loop
    varparteip1 := pkadministracion.ParteIP(ip1, parte);
    varparteip2 := pkadministracion.ParteIP(ip2, parte);
      if (varparteip1 <> '*' and varparteip2 <> '*') then
      if (varparteip1 <> varparteip2) then
        return 0;
      end if;
    end if;
    parte := parte + 1;
    if (parte = 5) then
      exit;
    end if;
  end loop;
  return 1;
end; $function$
language plpgsql;
```

- **Triggers:**

```
--script para definir trigger T_DFOTOPERSONANAT para PostgreSQL:
create or replace function t_dfotopersonanat_trfunc()
returns trigger
   as $trigger_function$
begin
   update dfotopersonanatural
   set idciudadano = new.idciudadano
   where dfotopersonanatural.idciudadano = old.idciudadano;
   return new;
end; $trigger_function$
language plpgsql;
--Creación del trigger:
create trigger t_dfotopersonanat
before update on dalbumfoto
for each row
   execute procedure t_dfotopersonanat_trfunc();
```

- **Vistas:**

```
-- Definition for view vultimostelefonosc
create view public.vultimostelefonosc as
select ciu1.idnumerotelefonico, ciu1.idciudadano, ciu1.fecha,
tel.numero,tel.ubicacion as idubicacion, ubicacion.descripcion as
ubicacion,tel.tipo as idtipotelefono, tipo.descripcion as tipotelefono
from (((rciudadanonumerotelef ciu1 join dnumerotelefonico tel on
    (((ciu1.idnumerotelefonico)::text=(tel.idnumerotelefonico)::text)))
    join ntipotelefono tipo on ((tel.tipo = tipo.id))) join
```

```
    nubicaciontelefono ubicacion on ((tel.ubicacion = ubicacion.id)))
where (ciu1.fecha = ( select max(ciu2.fecha) as max
                      from rciudadanonumerotelef ciu2
                      where((ciu1.idciudadano)::text=  ciu2.idciudadano)::text)
                      ))
order by ciu1.idciudadano;
```

- **Paquetes**

Los paquetes fueron migrados a esquemas como se había planteado en el capítulo II y anteriormente fue presentada la forma en que quedó la estructura de dichos esquemas (ver figura 7). Luego de la creación de cada uno de los objetos se muestra una figura de cómo quedó la BD completa luego de la migración.

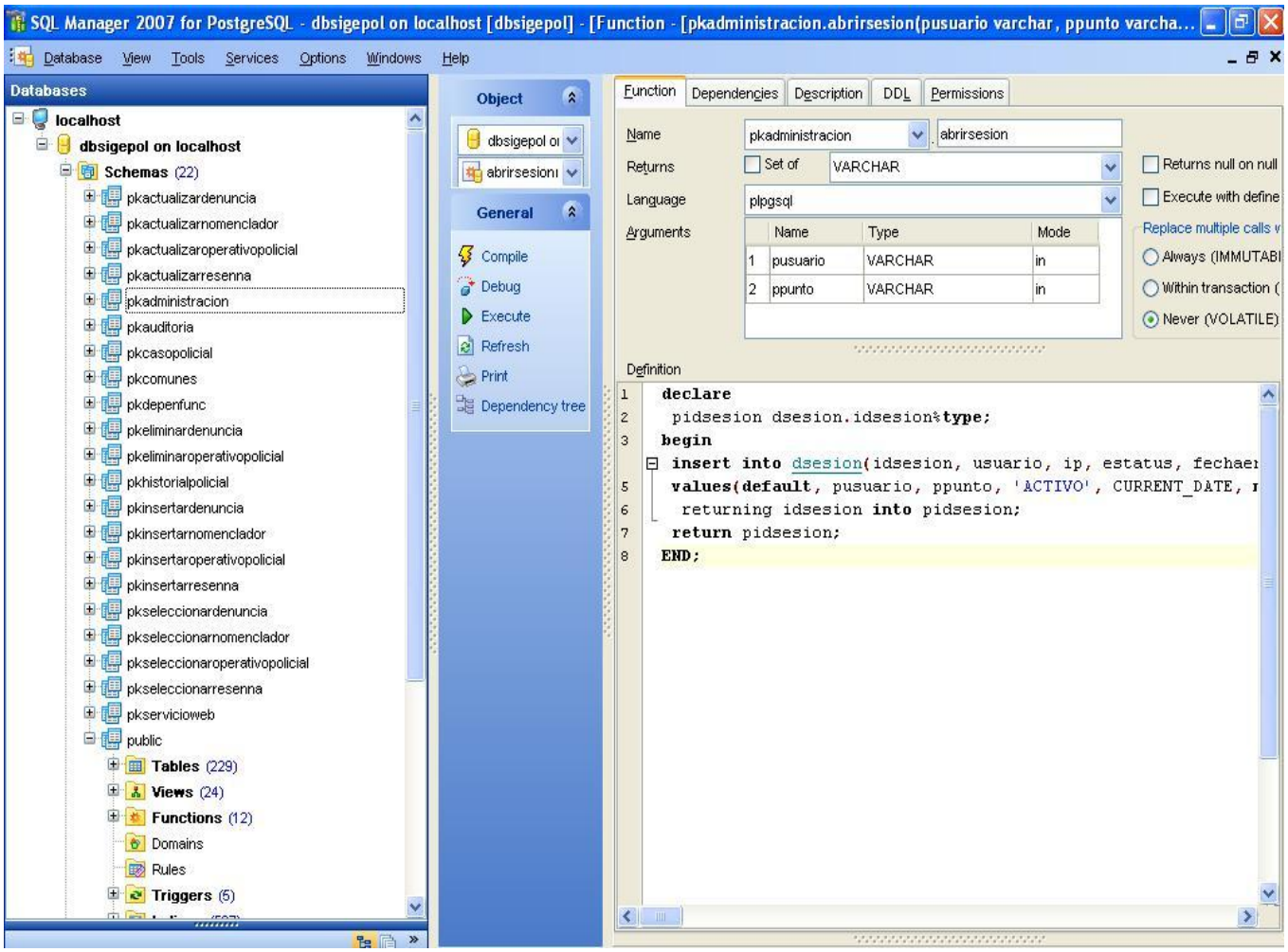

Figura 8: Imagen de la BD de SIGEPOL.

3.4 Validación de paso número 3: Creación de las políticas de salvas (backup) y recuperación (recovery).

Como se planteó en el capítulo 2, se decidió implementar como parte de la solución una política para realizar salvas y recuperación de BD, la cual no se encontraba implementada dentro de la solución de BD de SIGEPOL, como parte de la validación se puede afirmar que la misma fue implementada en su

totalidad, permitiendo obtener un backup semanal de la BD de SIGEPOL, el cual servirá para restaurar el sistema en caso de que este falle en algún momento.

3.5 Validación práctica utilizando el Módulo de Administración de SIGEPOL:

Para realizar la validación fue escogido el módulo de administración el cual contaba en la parte de la BD con un paquete llamado PKADMINISTRACION, el cual fue migrado en su totalidad hacia PostgreSQL creando un esquema cuyo nombre es **pkadministracion** que contiene las 59 funciones equivalentes a los 59 procedimientos almacenados con que contaba el paquete, a continuación se muestra una figura donde se evidencia como quedó dicho esquema (ver Anexos: Figura 3).

Como parte también de la propia validación desde el punto de vista del acceso a datos de la aplicación, se realizaron algunos cambios que permitieran a la aplicación funcionar con PostgreSQL tal y como lo hacia anteriormente con Oracle, los principales cambios que se produjeron son comentados a continuación:

- Se realizó la configuración del driver de PostgreSQL para que la aplicación pudiera funcionar correctamente con este SGBD, la siguiente figura muestra cómo quedó dicha configuración:

```
dbcp.connection.url= jdbc:postgresql://10.31.19.19:5432/dbsigepol
dbcp.connection.driver_class= org.postgresql.Driver
dbcp.connection.username= sigepol
dbcp.connection.password= sigepol
```

Figura 9: Configuración del driver de PostgreSQL.

- Se realizaron cambios en los mapeos que tenían que ver con los llamados a funciones de selección, no siendo necesario realizar cambios en los mapeos de funciones de inserción, actualización y eliminación; el principal cambio que se realizó en los mapeos fue en el **jdbcType** de los **parameterMap**, que cuando se utilizaba Oracle se utilizaba **oracleCursor** y para el caso de PostgreSQL se utiliza **OTHER**; la siguiente figura muestra un mapeo donde se evidencia el cambio mencionado:

```
<!-- MOSTRAR FALTAS  -->
<resultMap class="sigepol.administracion.domain.Falta" id="resultMostrarFalta">
    <result property="idFalta" column="iddelitofalta"/>
    <result property="nombreFalta" column="delitofalta"/>
    <result property="esGrave" column="esgrave"/>
    <result property="referenciaLegal" column="referencia"/>
    <result property="descripcion" column="articuloresumen"/>
    <result property="estado.descripcion" column="estado"/>
    <result property="municipio.descripcion" column="municipio"/>
</resultMap>

<parameterMap class="map" id="paramMostrarFalta">
    <parameter property="municipio" jdbcType="VARCHAR" mode="IN"/>
    <parameter property="estado" jdbcType="VARCHAR" mode="IN"/>
    <parameter property="posicionInicial" jdbcType="INTEGER" mode="IN"/>
    <parameter property="cantidad" jdbcType="INTEGER" mode="IN"/>
    <parameter property="total" jdbcType="INTEGER" mode="OUT"/>
    <parameter property="Faltas" mode="OUT" javaType="java.sql.ResultSet" jdbcType="OTHER"
    resultMap="Falta.resultMostrarFalta"/>
</parameterMap>

<procedure id="MostrarFalta" parameterMap="paramMostrarFalta">
     {call dbsigepol.PKSeleccionarResenna.mostrarFaltas(?,?,?,?,?,?)}
</procedure>
```

Figura 10: Mapeo del método -- Mostrar Faltas.

- Se realizó la implementación de un nuevo método en la aplicación de SIGEPOL, ubicado en la clase abstracta **AbstractIbatisDAO**, producto a que Ibatis presenta una propiedad cuyo nombre es **defaultAutoCommit**, la cual se inicializó con un valor por defecto a **false**(ver figura 11). En general, se recomienda esta configuración ya que brinda más flexibilidad y potencia a las aplicaciones, posibilitando utilizar transacciones complejas, además de que se puede decidir cuándo la transacción se inicia y cuándo termina. Esta configuración provoca que las operaciones de inserción, actualización y eliminación no puedan ser realizadas y si se cambia su valor a **true** no permite realizar el resto de las transacciones (dígase búsquedas o consultas de selección), por este motivo se implementa este método que permite manejar todas las transacciones del tipo inserción, actualización y eliminación, pudiendo controlar en todo momento el estado de la misma. La siguiente figura muestra la

configuración del **datasource** donde se encuentra inicializada la propiedad **defaultAutoCommit** y la implementación de dicho método:

```
<bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="url">
            <value>${dbcp.connection.url}</value>
        </property>
        <property name="driverClassName">
            <value>${dbcp.connection.driver_class}</value>
        </property>
        <property name="username">
            <value>${dbcp.connection.username}</value>
        </property>
        <property name="password">
            <value>${dbcp.connection.password}</value>
        </property>
        <property name="initialSize">
            <value>${dbcp.initialSize}</value>
        </property>
        <property name="maxActive">
            <value>${dbcp.maxActive}</value>
        </property>
        <property name="maxIdle">
            <value>${dbcp.maxIdle}</value>
        </property>
        <property name="minIdle">
            <value>${dbcp.minIdle}</value>
        </property>
        <property name="maxWait">
            <value>${dbcp.maxWait}</value>
        </property>
        <property name="defaultAutoCommit" value="false"/>
    </bean>
```

Figura 11: Configuración del datasource.

```
public void executeQueryIssue(String query, Object object) throws Exception{

        SqlMapClient sqlMap = sqlMapClientTemplate.getSqlMapClient();
        BasicDataSource bdt = (BasicDataSource)sqlMapClientTemplate.getDataSource();
        Connection conn = null;
        SqlMapSession session = null;
        try {

            conn = bdt.getConnection();
            session = sqlMap.openSession(conn);
            session.queryForObject(query, object);
            conn.commit();

        } catch (Exception e) {
            e.getCause();
        } finally {
            if (session != null)
                session.close();
            if (conn != null)
                conn.close();
        }
    }
```

Figura 12: Método para cambiar valor de variable **defaultAutoCommit**.

Luego de realizar estos cambios en el módulo escogido para realizar la validación, quedó funcionando totalmente y muestra de ello son las siguientes ventanas del mismo funcionando con PostgreSQL una vez aplicados cada uno de los pasos que en el trabajo de diploma fueron presentados.

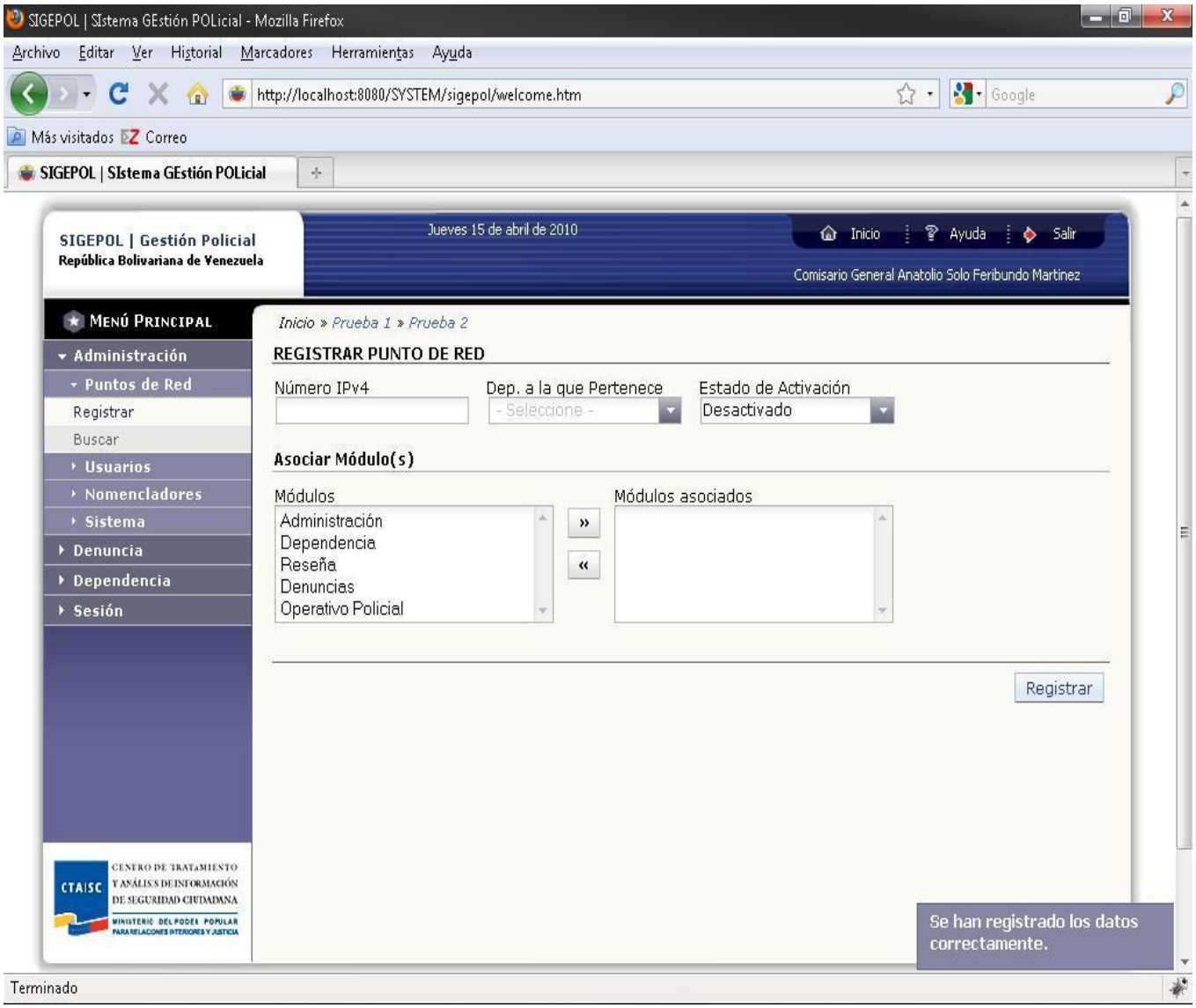

Figura 12: Interfaz para registrar punto.

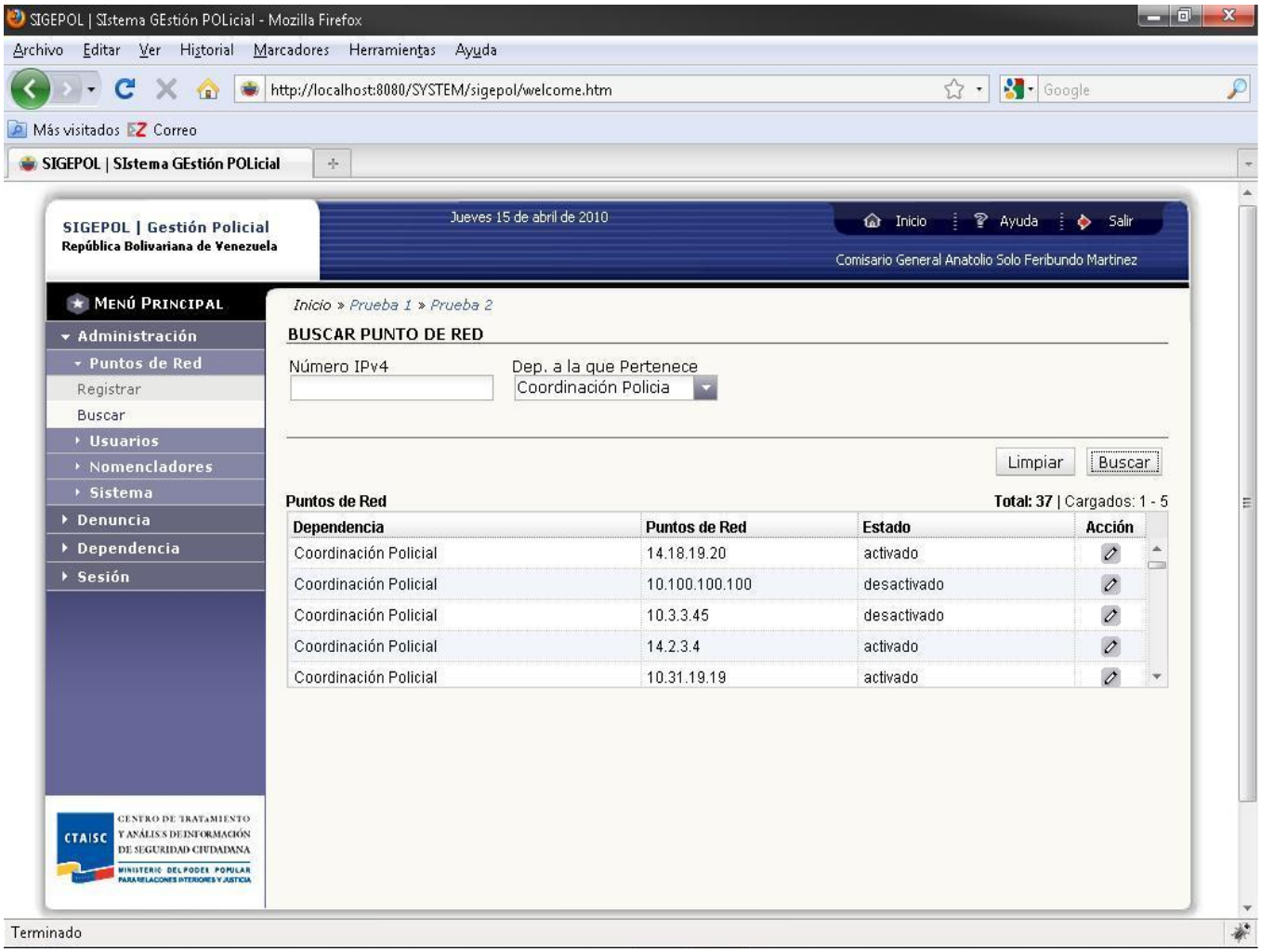

Figura 13: Interfaz para buscar punto.

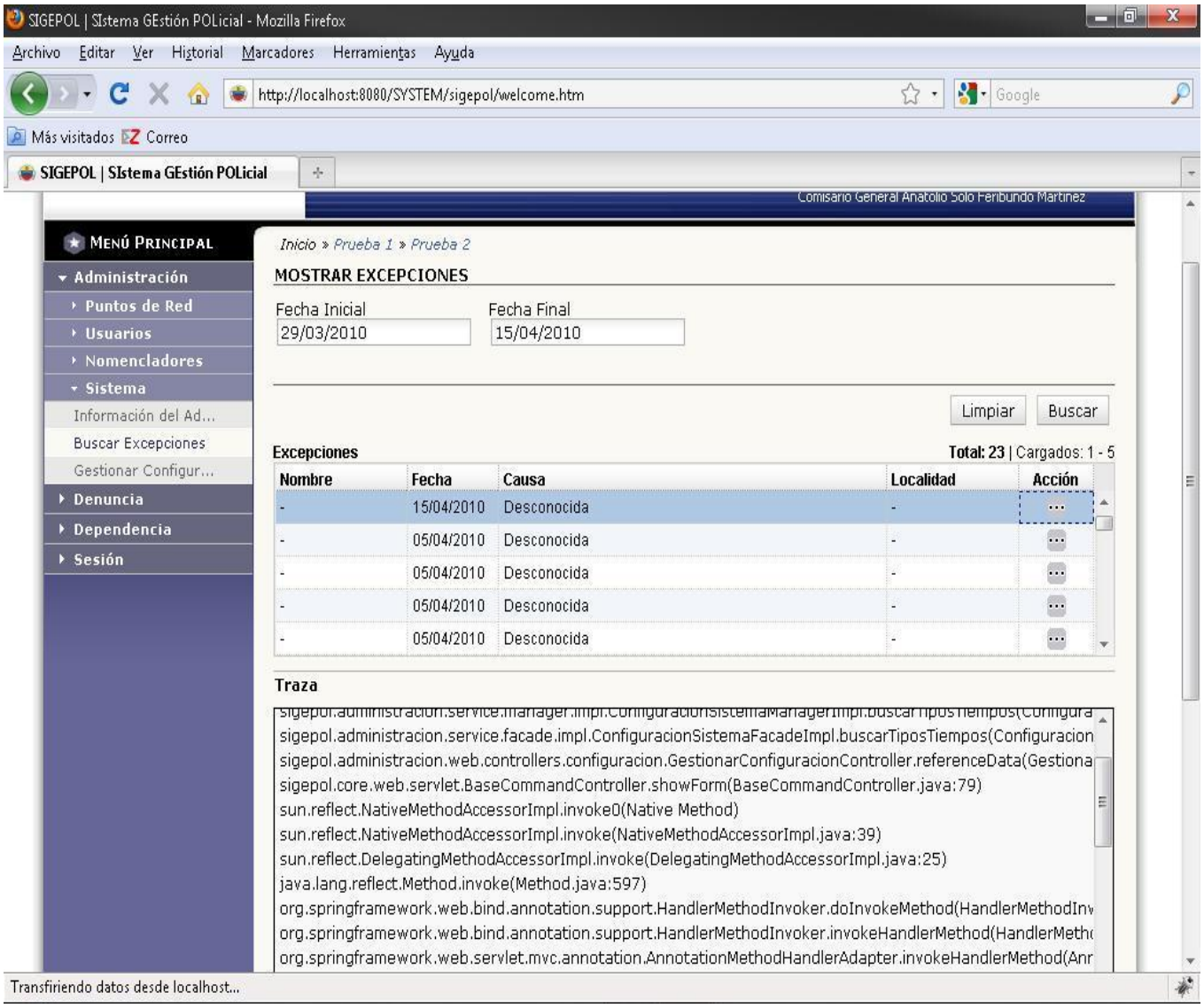

Figura 14: Interfaz para mostrar las excepciones.

Como parte de la validación se tuvieron en cuenta algunos elementos que son básicos a la hora de establecer comparaciones o validaciones de determinadas soluciones, por ejemplo el tema del rendimiento, la capacidad de respuesta y la extensibilidad de la solución propuesta, en todos los casos la solución propuesta cumple a cabalidad los estándares con los que fue pensada la aplicación inicialmente, logrando que el nivel de respuesta de la aplicación utilizando PostgreSQL sea el mismo que cuando usaba Oracle, que las prestaciones del módulo de administración (módulo escogido para probar la solución

propuesta) sean las mismas, que la capacidad de respuesta sea igual a la obtenida anteriormente y que pese a que la solución brindada está directamente enfocada a la solución de BD de SIGEPOL, se considera que puede ser extensible a otros proyectos o aplicaciones que deseen realizar migraciones similares.

Conclusiones del Capítulo III

En este capítulo se presentaron los elementos necesarios para realizar la validación práctica de la solución propuesta y de esta manera formalizar la migración de la solución de BD de SIGEPOL quedando validada dicha solución y cumpliéndose así el objetivo general de la investigación.

Al concluir el trabajo de diploma, “Migración de la solución de base de datos del Sistema de Gestión Policial (SIGEPOL) hacia un gestor de base de datos libre”, se ha dado cumplimiento a las tareas de investigación y al objetivo general de la investigación propuestos, desarrollándose la migración a PostgreSQL de todos los objetos de base de datos utilizados por la versión inicial de la aplicación.

Se realizó un estudio de los principales SGBD libres, que permitió la selección de PostgreSQL para realizar la migración de la solución de BD de SIGEPOL. Se analizó la estructura y compatibilidad de la solución de la BD de SIGEPOL con el SGDB libre seleccionado, lo que posibilitó definir la estrategia de migración para cada uno de los elementos de BD que conformaban dicha solución. Se realizó la migración de cada uno de los objetos de base de datos definidos en la solución original, posibilitando que SIGEPOL quedara funcionando completamente utilizando PostgreSQL como gestor de base de datos para gestionar la información que antes era gestionada utilizando Oracle. Además, se realizó la validación de la estrategia de migración en uno de los módulos del sistema lo que posibilitó dar veracidad a la solución propuesta en el trabajo de diploma.

Se realizó la confección de un documento, el cual constituye una guía que servirá de ayuda para realizar la migración concerniente a la aplicación de SIGEPOL, donde se explica detalladamente cada uno de los cambios que deben ser realizados para lograr un correcto despliegue usando PostgreSQL como SGBD. Dicho documento, “**Guía de cambios en el ADO de SIGEPOL utilizando PostgreSQL**” forma parte de los anexos contenidos en el trabajo de diploma.

Con dicha migración se aportará un granito de arena en el logro de la soberanía tecnológica en Venezuela, además muchas dependencias de la República Bolivariana de Venezuela podrán utilizar el Sistema de Gestión Policial sin necesidad de pagar el monto asociado a la licencia del SGBD propietario Oracle, ya que SIGEPOL podrá gestionar su información con PostgreSQL como SGBD libre seleccionado, facilitando esto la contribución a solucionar determinados problemas que afectan a la sociedad venezolana.

En el desarrollo de este trabajo se ha profundizado en un conjunto de aspectos estratégicos que se consideran importantes para el desarrollo del mismo, a continuación se presentan un conjunto de elementos que se recomienda realizar para dar continuidad al trabajo presentado:

- Utilizar la **"Guía para la Migración de datos en Oracle"** desarrollada por el Centro de Desarrollo de Software UCI en Villa Clara para realizar la migración de los datos que puedan existir como historial de la aplicación.
- Implementar una herramienta que se encargue del proceso de generar los XML de mapeos, teniendo en cuenta los cambios propuestos en este trabajo de diploma.
- Realizar la implementación de una herramienta que permita la conversión de un script generado para Oracle hacia PostgreSQL utilizando los elementos descritos en la solución propuesta.
- Estudiar la arquitectura de servidores para garantizar desde el punto de vista del hardware la seguridad y disponibilidad de los datos, un rendimiento óptimo del servidor y flexibilidad del mismo a la hora de realizar los backup sin detener el servicio.

Referencias Bibliográficas

1. Portal Informativo de la Casa de las Américas: Primera ley estatal de software libre en Latinoamérica. [Citado el: 25 de septiembre de 2009.] http://laventana.casa.cult.cu/modules.php?name=News&file=article&sid=738.

2. Somos Libres(Activismo por el software libre).Proyecto de ley de uso de software libre en el Estado. [Citado el: 28 de septiembre de 2009.] http://www.somoslibres.org/modules.php?name=News&file=article&sid=11.

3. Constitución de la República Bolivariana de Venezuela. [Citado el: 15 de octubre de 2009.] http://www.tsj.gov.ve/legislacion/constitucion1999.htm.

4. Ministerio del Poder Popular para las Telecomunicaciones y la Infomatica, Guía para el plan de migración a Software Libre en la Administración Pública Nacional República Bolivariana de Venezuela. [Citado el: 20 de octubre de 2009.] http://www.scribd.com/doc/6442474/Guia-para-el-plan-de-migracion-a-Software-Libre-en-la-Administracion-Publica-Nacional

5. ¿Qué es un proceso de migración a Software Libre? [En línea] Investic (Software Libre: Tecnologías de la Colaboración). [Citado el: 30 de octubre de 2009.] http://www.investic.net/node/105.

6. Migración a Software Libre. [En línea] Kopialan. [Citado el: 5 de noviembre de 2009.] http://kopialan.hezi.eu/es/kopialan/migracion-software-libre.html.

7. Licencias de Software. [Citado el: 15 de noviembre de 2009.] http://www.monografias.com/trabajos55/licencias-de-software/licencias-de-software.shtml.

8. Categorías de software libre y no libre. [Citado el: 25 de noviembre de 2009.] http://www.gnu.org/philosophy/categories.es.html#FreeSoftware.

9. Concesión de licencias dual. [Citado el: 25 de noviembre de 2009.] http://producingoss.com/es/dual-licensing.html.

10. **GARCIA, LIC. ROSA MARIA MATO.** *DISEÑO de BASES DE DATOS*.

11. Base de datos. [Citado el: 5 de diciembre de 2009.] http://www.monografias.com/trabajos55/base-de-datos/base-de-datos.shtml#tipos.

12. Modelos de bases de datos. [Citado el: 10 de diciembre de 2009.] http://www.monografias.com/trabajos55/base-de-datos/base-de-datos2.shtml#modelos.

13. **Ross, Elieser Bello.** *Sistema de Gestión Policial (SIGEPOL). Diseño de Base de Datos*.

14. http://es.wikipedia.org. [Citado el: 5 de enero de 2010.] http://es.wikipedia.org/wiki/Sqlite.

Referencias Bibliográficas

15. http://www.sqlite.org. [Citado el: 5 de enero de 2010.] http://www.sqlite.org/about.html.

16. http://www.sqlite.org. [Citado el: 5 de enero de 2010.] http://www.sqlite.org/different.html.

17. http://www.sqlite.org. [En línea] 2016. [Citado el: 5 de enero de 2010.] http://www.sqlite.org/limits.html.

18. **R A U L C H O N G, I A N H A K E S, R A V A H U J A,D R . A R V I N D K R I S H N A.** *Conociendo el DB2 Express-C.*

19. **Portal de Comunidad deDesarrollado Java y Linux.** DB2 en su versión gratuita. [Citado el: 15 de enero de 2010.] http://www.codejava.org/v2_vernota.htm?idxnota=92285&destacada=1.

20. **Lockhart, Thomas.** *Manual del usuario de PostgreSQL.* s.l. : Equipo de desarrollo de PostgreSQL.

21. **Comunidad de PostgreSQL en español.** Acerca de PostgreSQL. [Citado el: 17 de enero de 2010.] http://www.postgresql-es.org/sobre_postgresql.

22. Base de Datos PostgreSQL, SQL avanzado y PHPurl. [Citado el: 17 de enero de 2010.] http://eaprende.com/gestor-de-basededatos-mysql-postresql-sqlite.html.

23. http://monografias.com/PostgreSQL. [Citado el: 20 de enero de 2010.] http://monografias.com.

24. http://www.postgre.org/Licencia. [Citado el: 21 de enero de 2010.] http://www.postgre.org.

25. http://www.uaem.mx. [Citado el: 21 de enero de 2009.] http://www.uaem.mx/posgrado/mcruz/cursos/miic/MySQL.pdf.

26. **Solís, Daymel Bonne.** Migración del diseño y administración de la base de datos del Sistema de Gestión de Emergencias y Seguridad Ciudadana (171).

27. http://dev.mysql.com. [Citado el: 21 de enero de 2010.] http://dev.mysql.com/doc/refman/5.0/es/features.html.

28. http://wiki.postgresql.org/. [Citado el: 22 de enero de 2010.] http://wiki.postgresql.org/wiki/Why_PostgreSQL_Instead_of_MySQL_2009.

29. Reglas en PostgreSQL. Sitio Oficial de PostgreSQL. [Citado el: 23 de enero de 2010.] http://www.postgresql.org/docs/8.3/interactive/rules-update.html.

30. PostgreSQL vs. MySQL vs. Commercial Databases: It's All About What You Need. DevX. [Citado el: 23 de enero de 2010.] http://www.devx.com/dbzone/Article/20743.

31. Replicación en MySQL. MySQL 5.0 Reference Manual. [Citado el: 24 de enero de 2010.] http://dev.mysql.com/doc/refman/5.0/es/replication.html.

32. Tablespaces. Sitio Oficial de PostgreSQL. [Citado el: 5 de febrero de 2010.] http://www.postgresql.org/docs/8.3/static/manage-ag-tablespaces.html.

33. Secuencias en PostgreSQL. Sitio Oficial de PostgreSQL. [Citado el: 25 de febrero de 2010.] http://www.postgresql.org/docs/8.3/static/functions-sequence.html.

Disponible en: http://www.uaem.mx/posgrado/mcruz/cursos/miic/MySQL.pdf.

Disponible en: http://dev.mysql.com/doc/refman/5.0/es/features.html.

Base de Datos PostgreSQL, SQL avanzado y PHPurl. . Disponible en: http://eaprende.com/gestor-de-basededatos-mysql-postresql-sqlite.html.

Base de datos. . Disponible en: http://www.monografias.com/trabajos55/base-de-datos/base-de-datos.shtml#tipos.

Categorías de software libre y no libre. . Disponible en: http://www.gnu.org/philosophy/categories.es.html#FreeSoftware.

Comunidad de PostgreSQL en español. Acerca de PostgreSQL. . Disponible en: http://www.postgresql-es.org/sobre_postgresql.

Concesión de licencias dual. . Disponible en: http://producingoss.com/es/dual-licensing.html.

Constitución de la República Bolivariana de Venezuela. . Disponible en: http://www.tsj.gov.ve/legislacion/constitucion1999.htm.

GARCIA, L. R. M. M. DISEÑO de BASES DE DATOS. Editado por: Editorial Pueblo Y Educación. 2005,

Licencias de Software. . Disponible en: http://www.monografias.com/trabajos55/licencias-de-software/licencias-de-software.shtml.

Migración a Software Libre. . Disponible en: http://kopialan.hezi.eu/es/kopialan/migracion-software-libre.html.

Ministerio del Poder Popular para las Telecomunicaciones y la Infomatica, Guía para el plan de migración a Software Libre en la Administración Pública Nacional República Bolivariana de Venezuela. . Disponible en: http://www.scribd.com/doc/6442474/Guia-para-el-plan-de-migracion-a-Software-Libre-en-la-Administracion-Publica-Nacional.

Modelos de bases de datos. . Disponible en: http://www.monografias.com/trabajos55/base-de-datos/base-de-datos2.shtml#modelos.

Portal de Comunidad deDesarrollado Java y Linux. DB2 en su versión gratuita. . Disponible en: http://www.codejava.org/v2_vernota.htm?idxnota=92285&destacada=1.

Portal Informativo de la Casa de las Américas: Primera ley estatal de software libre en Latinoamérica. . Disponible en: http://laventana.casa.cult.cu/modules.php?name=News&file=article&sid=738.

Bibliografía

POSTGRESQL, E. E. D. D. D. Manual del usuario de PostgreSQL. s.l. : Equipo de desarrollo de PostgreSQL. Editado por: Lockhart, T. Disponible en: http://linuxwall32.wordpress.com/2009/06/02/manual-del-usuario-de-postgresql/.

PostgreSQL vs. MySQL vs. Commercial Databases. Disponible en: http://www.devx.com/dbzone/Article/20743.

¿Qué es un proceso de migración a Software Libre? . Disponible en: http://www.investic.net/node/105.

R A U L C H O N G, I. A. N. H. A. K. E. S., R A V A H U J A Y DR. ARVIND KRISHNA. Conociendo el DB2 Express-C. s.l. SEGUNDA EDI CI ON ed. Disponible en: http://www.scribd.com/doc/16300655/Conociendo-DB2-Express-v95.

Reglas en PostgreSQL. Sitio Oficial de PostgreSQL. . Disponible en: http://www.postgresql.org/docs/8.3/interactive/rules-update.html.

Replicación en MySQL. MySQL 5.0 Reference Manual. Disponible en: http://dev.mysql.com/doc/refman/5.0/es/replication.html.

ROSS, E. B. Sistema de Gestión Policial (SIGEPOL). Diseño de Base de Datos. 2008.

Secuencias en PostgreSQL. Sitio Oficial de PostgreSQL.. Disponible en: http://www.postgresql.org/docs/8.3/static/functions-sequence.html.

SOLÍS, D. B. Migración del diseño y administración de la base de datos del Sistema de Gestión de Emergencias y Seguridad Ciudadana (171). 2009.

Somos Libres(Activismo por el software libre).Proyecto de ley de uso de software libre en el Estado. Disponible en: http://www.somoslibres.org/modules.php?name=News&file=article&sid=11.

Tablespaces. Sitio Oficial de PostgreSQL. . Disponible en: http://www.postgresql.org/docs/8.3/static/manage-ag-tablespaces.html.

	R-/R+ tree	Hash	Expression	Partial	Reverse	Bitmap	GiST	GIN
MySQL	MyISAM tables only	MEMORY, Cluster (NDB), InnoDB, tables only	No	No	No	No	No	No
PostgreSQL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Figura 1: Tipos de índices y soporte por parte de MySQL y PostgreSQL.

	Union	Intersect	Except	Inner joins	Outer joins	Inner selects	Merge joins	Blobs and Clobs	Common Table Expressions	Windowing Functions
MySQL	Yes	No	No	Yes	Yes	Yes	Yes	Yes	No	No
PostgreSQL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Figura 2 Capacidades de los gestores de base de datos MySQL y PostgreSQL ante diversas operaciones.

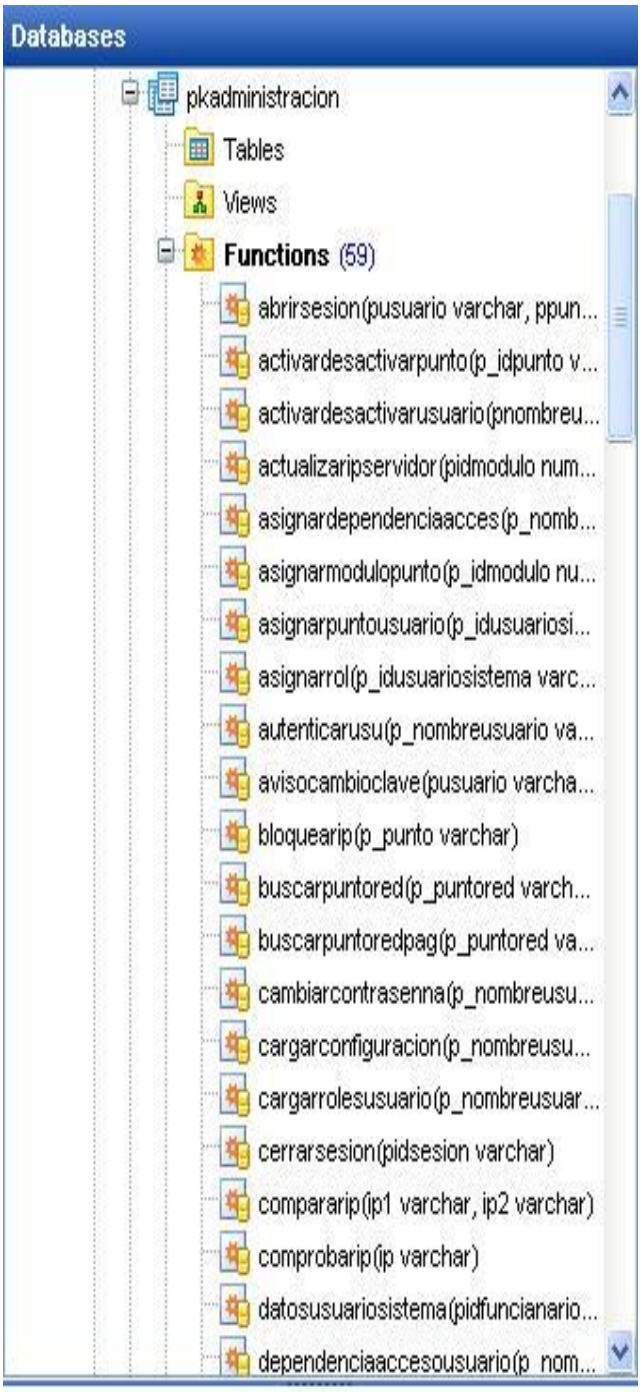

Figura 3: Esquema pkadministracion.

AVAL DEL TRABAJO DE DIPLOMA

El Trabajo de Diploma, titulado **"Migración de la solución de base de datos del Sistema de Gestión Policial (SIGEPOL) hacia un Sistema Gestor de Bases de Datos Libre"**, fue realizado en la **Universidad de las Ciencias Informáticas (UCI)**. Esta entidad así como la dirección del proyecto SIGEPOL considera que, en correspondencia con los objetivos trazados, el trabajo realizado le satisface

_X_ Totalmente

__ Parcialmente en un ____ %

Con el desarrollo del trabajo de diploma se obtiene una guía de migración de la solución de base de datos del Sistema de Gestión Policial (SIGEPOL), la cual usa como gestor de base de datos Oracle, hacia un gestor libre, en este caso PostgreSQL. Esto surge por la necesidad de cumplir con las políticas de la República Bolivariana de Venezuela de alcanzar una soberanía tecnológica a partir del uso de herramientas no propietarias para la creación de sistemas informáticos. El trabajo cuenta con los elementos teóricos que apoyan la correcta transferencia hacia el gestor de base de datos PostgreSQL de las funcionalidades, estructura y datos de la base de datos del SIGEPOL. Como parte de la validación se implementó la migración del módulo de Administración Informática logrando que todas las funcionalidades se ejecuten correctamente y dando por válida la propuesta para la migración de la Base de Datos del Sistema.

Y para que así conste, se firma la presente a los ____ días del mes de _____ del año _____.

Elieser Bello Ross	_Líder de Software del SIGEPOL
Representante de la entidad	Cargo
_____	_____
Firma	Cuño

Guía de cambios en el acceso a datos (ADO) de SIGEPOL utilizando PostgreSQL

La Aplicación Informática SIGEPOL es la encargada de la captura de la información sobre la gestión policial, a través de la cual las dependencias policiales tendrán acceso a información nacional y podrán describir los resultados de sus actuaciones.

En la implementación de la misma se utilizó Ibatis en su versión 2.3.0.677, este framework se ocupa de la capa de persistencia y está situado entre la lógica de Negocio y la capa de la BD. A la hora de realizar la migración de la solución de BD hacia PostgreSQL se deben tener en cuenta algunos aspectos de la aplicación para que la misma pueda funcionar correctamente desde la BD PostgreSQL.

La implementación de SIGEPOL cuenta con 10 paquetes en los cuales están todas las funcionalidades del sistema agrupadas por módulos, de estos paquetes se deben tener en cuenta a la hora de comenzar a realizar el proceso de migración de la BD los siguientes: administración, común, configuración, core, denuncia, dependencia, operativo, reseña y servicio. Dicha estructura física se puede apreciar en la Figura 1.

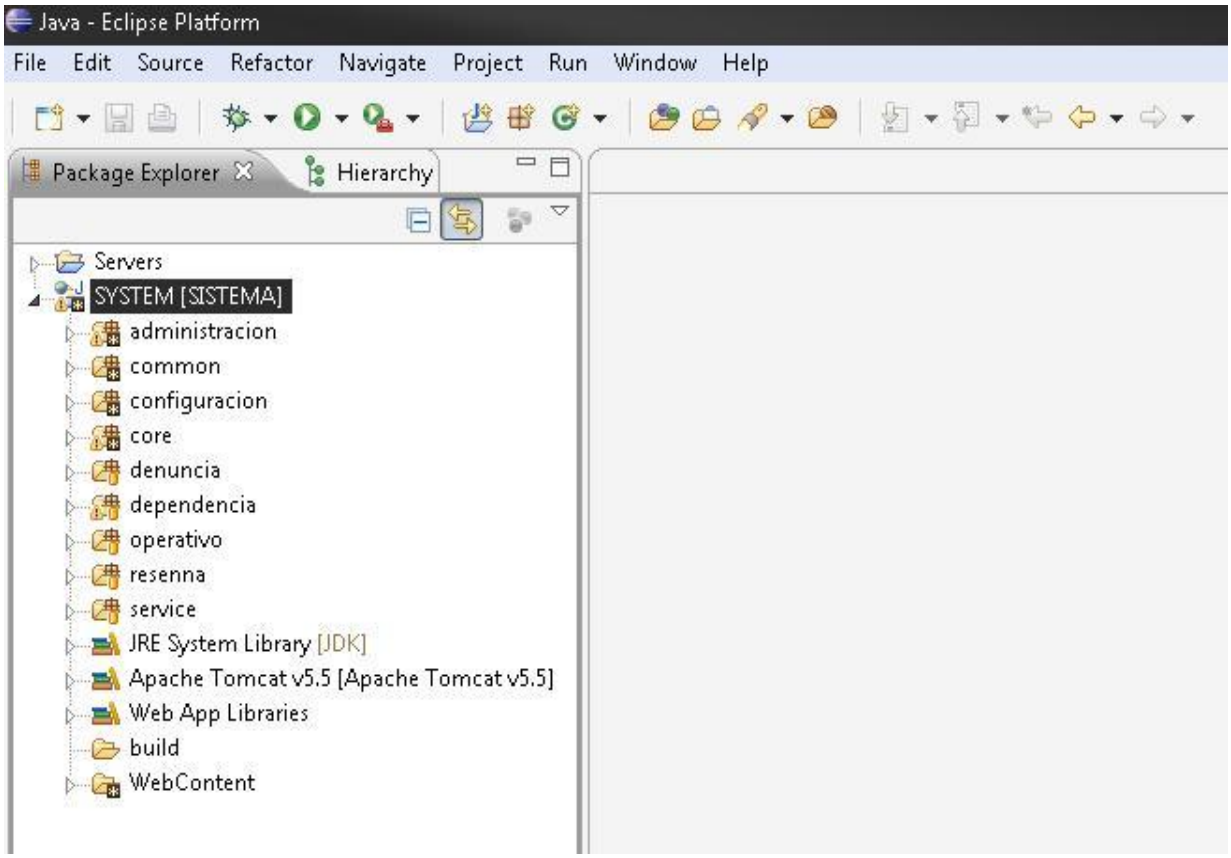

Figura 1: Estructura física de SIGEPOL

La migración de la BD a PostgreSQL sólo afecta una pequeña parte de la implementación de la aplicación SIGEPOL, la concerniente al acceso a datos, donde debe ser realizados algunos cambios que permitan que la aplicación funcione con este SGBD tal como lo hacia anteriormente con Oracle, los principales cambios son comentados a continuación:

- Se debe realizar la configuración del driver de PostgreSQL, la siguiente figura muestra cómo quedó dicha configuración, la misma debe hacerse en el paquete configuración sobre el archivo dbcp.properties:

```
dbcp.connection.url= jdbc:postgresql://10.31.19.19:5432/dbsigepol
dbcp.connection.driver_class= org.postgresql.Driver
dbcp.connection.username= sigepol
dbcp.connection.password= sigepol
```

Figura 2: Configuración del driver de PostgreSQL.

- Otro de los archivos a configurar dentro de este mismo paquete es el llamado sigepol-ibatis-config.xml, encargado de realizar la conexión con la BD. En general, esta configuración es recomendada ya que brinda más flexibilidad y potencia a las aplicaciones, posibilitando utilizar transacciones complejas, definiendo el momento de iniciar o finalizar así como controlar el estado de una transacción. El cambio a realizar en la configuración de este archivo se puede apreciar en la siguiente figura:

```
<property name="defaultAutoCommit" value="false"/>
```

Figura 3: Configuración del datasource.

- Debido a esta configuración del datasource se hizo necesario la creación de una funcionalidad que se encargara de manejar una transacción, la misma debe ser agregada a la clase abstracta AbstractIbatisDAO, ubicada en el paquete dao, el cual está contenido dentro del paquete general core. A continuación se muestra la nueva funcionalidad que se debe agregar a dicha clase:

```
public void executeQueryIssue(String query, Object object) throws Exception{

        SqlMapClient sqlMap = sqlMapClientTemplate.getSqlMapClient();
        BasicDataSource bdt = (BasicDataSource)sqlMapClientTemplate.getDataSource();
        Connection conn = null;
        SqlMapSession session = null;
        try {

            conn = bdt.getConnection();
            session = sqlMap.openSession(conn);
            session.queryForObject(query, object);
            conn.commit();

        } catch (Exception e) {
            e.getCause();
        } finally {
            if (session != null)
                session.close();
            if (conn != null)
                conn.close();
        }
    }
```

Figura 4: Funcionalidad encargada de manejar una transacción de tipo local en Ibatis.

- El próximo y último paso a realizar está en el mapeo de cada uno de los paquetes antes mencionados, donde hay que realizar solo un cambio. Los mapeos pueden ser encontrados dentro de un paquete llamado **dao,** el cual está contenido dentro de cada uno de los paquetes generales a los que se hizo referencia anteriormente. A continuación se muestra un ejemplo, donde se pueden encontrar estos mapeos dentro del paquete administración**.**

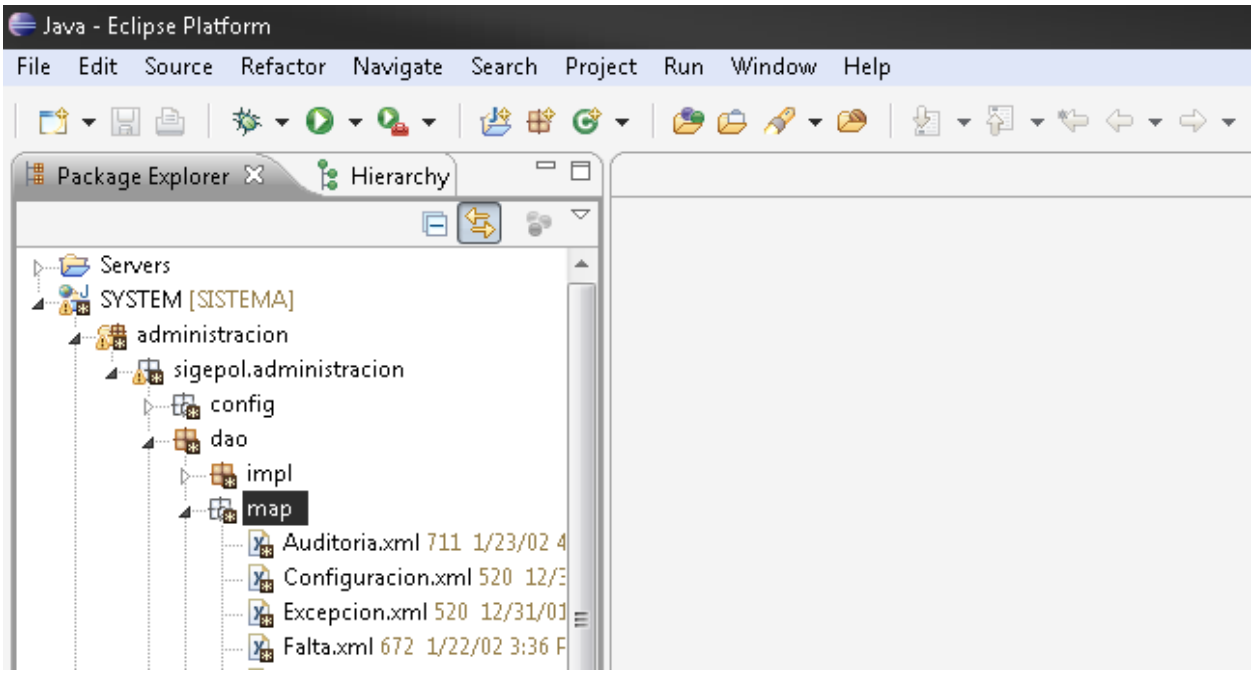

Figura 5: Ubicación de los XML en el paquete **administración.**

Una vez que se conoce la ubicación de todos los XML, se puede proceder a realizar el cambio para cada uno de los paquetes del sistema. El cambio se realizará para el caso de la llamada a los procedimientos almacenados en los cuales se retorna un cursor como parámetro de salida, la declaración para este tipo de parámetro se encuentra dentro de los parameterMap de los XML. El cambio a realizar sería de la siguiente manera.

Para Oracle:

```
<parameter property="Faltas" mode="OUT" javaType="java.sql.ResultSet" jdbcType="ORACLECURSOR "
resultMap="Falta.resultMostrarFalta"/>
```

El cambio sólo implica la sustitución de la palabra ORACLECURSOR por OTHER en la propiedad jdbcType**.**

```
<parameter property="Faltas" mode="OUT" javaType="java.sql.ResultSet" jdbcType="OTHER"
resultMap="Falta.resultMostrarFalta"/>
```

Una vez realizado este cambio en cada uno de los XML de SIGEPOL se podrá desplegar la aplicación con PostgreSQL como SGBD, soportando las mismas funcionalidades que se brindaban desde Oracle.

CTAISC: Centro de Tratamiento y Análisis de Información de Seguridad Ciudadana.

Licencias copyleft o **Licencias robustas:** tipo de licencia que impone condiciones en caso de que se quiera redistribuir el software, condiciones que van en la línea de forzar a que se sigan cumpliendo las condiciones de la licencia después de la primera redistribución.

ACID es un acrónimo de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad.

Atomicidad: es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.

Consistencia: es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto, se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos.

Aislamiento: es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sea independiente y no generen ningún tipo de error.

Durabilidad: es la propiedad que asegura que una vez realizada la operación, esta persistirá y no se podrá deshacer aunque falle el sistema.

Timeout: Mensaje de error cuando el tiempo de espera se agotó (por ejemplo, para conectarse a un servidor).

Deadlock o bloqueo mutuo: es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos. A diferencia de otros problemas de concurrencia de procesos, no existe una solución general para los interbloqueos.

Multithread: multi hilos.

mSQL: mSQL o Mini SQL es un cliente/servidor de bases de datos ligero de Hughes Technologies.

Glosario de Términos

Arquitecturas Shared Nothing: Consiste en una arquitectura distribuida en el que cada nodo es independiente, autosuficiente y tiene un único punto de contención en todo el sistema. Típicamente se contrasta con los sistemas que mantienen una gran cantidad de datos almacenados en forma centralizada, ya sea una base de datos, un servidor de aplicaciones, o cualquier otro donde esté centralizada la funcionalidad.

Thread: hilo de ejecución.

SQL: Structured Query Language o Lenguaje Estructurado de Consultas.

Campo: Unidad menor de información sobre un objeto (almacenada en la BD) y representa una propiedad de un objeto.

Tupla: Ocurrencia de una colección identificable de campos asociados que representan un objeto con sus propiedades.

NDB API: Interfaz de aplicación de MySQL Cluster que implementa transacciones.