UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

Particionado de las tablas críticas de la base de datos del Redmine

Trabajo de Diploma para Optar por el Título de Ingeniero en Ciencias Informáticas

Autora:

Nadaisys Teresa Martínez Rivera

Tutores:

Ing. Marianela Gutiérrez Rodríguez

Ing. Anyer Gámez Guedes

La Habana, Junio 2012

"Año 54 de la Revolución"

"La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica"

Aristóteles

DECLARACIÓN DE AUTORÍA

Declaro ser la única autora de este trabajo y autorizo a la Facultad 6 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los _____ del mes de _____del 2012.

_____

Nadaisys Teresa Martínez Rivera

_____ _____

Ing. Marianela Gutiérrez Rodríguez **Ing. Anyer Gámez Guedes**

DATOS DE CONTACTO

- **Autora:** Nadaisys Teresa Martínez Rivera, Universidad de las Ciencias Informáticas, La Habana, Cuba, ntmartinez@estudiantes.uci.cu
- **Turor:** Ing. Marianela Gutiérrez Rodríguez, Universidad de las Ciencias Informáticas, La Habana, Cuba, mgrodriguez@uci.cu
- **Tutor:** Ing. Anyer Gámez Guedes, Universidad de las Ciencias Informáticas, La Habana, Cuba, agguedes@uci.cu

AGRADECIMIENTOS

Quiero agradecer a la Revolución y a nuestro Comandante en jefe Fidel Castro por crear este proyecto futuro y brindarme la oportunidad de formar parte de él.

A mi mami Nora y a mi pa Lázaro por depositar su confianza en mí, por brindarme su amor y cariño incondicional, por susurrarme las palabras de aliento cuando necesitaba seguir adelante, por guiarme siempre por el buen camino, por enseñarme a ser la persona que soy hoy.

A mi tía Eloisa, mi tiota, como cariñosamente le digo; por ser como mi segunda madre, por estar siempre dispuesta a todo por mí y apoyarme en todo momento. Por la confianza depositada, por brindarme su mano como una amiga y también sus consejos.

A mi hermanita, mi nanita linda, por quererme tanto como lo demuestra, por regalarme su sonrisita tierna y sus abrazos. Por ayudarme a levantarme cuando estaba triste pues con solo pensar en ella ya era suficiente para que cambiara mi día. Por ser mi inspiración en todo momento, por ser mi todo.

A mis abues Cándida y Germán por depositar todo su afecto y sus experiencias.

A mi abuelo Cuco y a mi abuela Teresa (a pesar de no haberla conocido) por traer al mundo a ese maravilloso ser que es mi pa y a mi abuelo Cuco por quererme como lo ha hecho.

A mis tíos Dayamí y Gustavo por estar siempre pendientes de mí y entregarme su cariño.

A mis primos y primas, los grandes y los chiquitos, por compartir mis alegrías y hacer de mi vida cada día más feliz.

A toda la familia en general que me apoyó en todo momento.

A mi amiga inolvidable Ivette, la refeita, como cariñosamente le digo, por ser mi pañuelo de lágrimas en mis momentos de tristeza, por compartir su sonrisa en mis momentos de alegría, por estar pendiente de mi familia cuando estuve lejos. Por ayudarme a tomar decisiones importantes y apoyarlas.

A mis amigas la Tay y la Mercy por los consejos brindados, por los abrazos regalados, por todos los momentos compartidos de alegrías y tristezas. Por brindarme sus palabras de aliento en los momentos de nervios (que fueron muchos). Chicas gracias de corazón por todo.

Al grupo de amigos inolvidables, que son un tesoro valioso que tengo guardado para siempre: Yudelkis, Raiko, Yixander, Mikel, Orisbel, Lassie. A todos gracias por las horas de estudio compartidas, por los ratos divertidos que pasamos, por su apoyo incondicional en toda la etapa de la universidad.

A mi amigo Geiber por brindarme su tiempo, por escucharme cuando necesitaba conversar con alguien, por estar siempre ahí para mí.

A mis tutores Marianela y Anyer que sin importar la hora, el día, el lugar o el cansancio me ayudaron y me apoyaron en todo momento. Y también a mi antiguo tutor Maikel que fue el iniciador de esta investigación y por brindarme su ayuda siempre que lo necesité.

A mis amigas Elaimy y Yennis, las amigas de la secundaria por permitirme formar parte de sus vidas y por los consejos dados.

A todos los compañeros de aula y los profesores que tuve a lo largo de mi carrera.

DEDICATORIA

A mis padres por guiarme siempre por el buen camino y convertirme en la persona que soy hoy.

A mi tía Eloisa por su ayuda, su comprensión, por sus palabras, por su paciencia y su dedicación.

A mi hermanita del alma por ser mi fuente de inspiración, por ser mi mundo y mi todo.

A mis abuelos por quererme como lo han hecho siempre.

A toda mi familia en general por la confianza depositada.

RESUMEN

La demanda de los productos de software y los servicios de información tecnológica tienen una de las tasas de crecimiento mundial más elevados en la actualidad. Debido a esto, las empresas y grandes compañías requieren cada vez más de soluciones rápidas y efectivas. El crecimiento y difícil manejo de los volúmenes de información que se generan, explica las razones que progresivamente obligan a las organizaciones a desarrollar las tecnologías de bases de datos. A partir de las necesidades reales de mejorar el rendimiento de la Herramienta de Gestión de Proyectos Redmine en la Universidad de las Ciencias Informáticas (UCI), surge la presente investigación que tiene como objetivo aplicar técnicas de particionado de tablas a la base de datos del Redmine, para reducir los tiempos de respuestas de las consultas que se ejecutan sobre la misma. Para ello se realiza un estudio sobre las técnicas de particionado que existen en PostgreSQL. Se realiza un proceso de selección de las tablas críticas de la base de datos a las cuales se le realizará el particionado. Además se realizan pruebas de rendimiento a la base de datos analizada obteniendo resultados satisfactorios, demostrando que la implementación de la propuesta contribuirá a mejorar los tiempos de respuesta a las consultas que se realizan a la base de datos del Redmine.

Palabras claves: Base de Datos, particionado, Redmine, técnicas.

Índice de tablas

Índice de figuras

ÍNDICE

Introducción

El mundo hoy día está sumergido en un amplio desarrollo que tiene como resultado cuantiosos cambios en todas las esferas de la vida. Sin duda alguna, uno de los sectores que más ha evolucionado en estos últimos tiempos es la Informática y las Comunicaciones para solucionar muchos de los problemas vigentes en la actualidad.

Cuba no está exenta a este constante desarrollo, es por ello que en la última década ha llevado a cabo un proceso de informatización de la Sociedad Cubana. Como parte de esta iniciativa se han creado proyectos futuros que favorezcan el incremento de la producción y exportación de software en el país.

Un ejemplo vigente es la Universidad de las Ciencias Informáticas (UCI), que desde su creación está llamada a convertirse en una potencia en el desarrollo de aplicaciones que beneficien a Cuba y también a otros países hermanos. Con el objetivo de cumplir su trascendental tarea, la UCI organiza su modelo de producción en diferentes centros productivos. El objetivo principal es responder a las necesidades específicas de la universidad y obtener un avance tecnológico que garantice un aumento continuo de exportaciones. Además apoyar desde la gestión de los proyectos y el perfeccionamiento de tecnologías, el trabajo político ideológico y los servicios de formación.

Para lograr un mejor rendimiento en el desarrollo del software, un mejor control y una mayor organización en los centros de producción, los equipos de desarrollo utilizan la herramienta de Gestión de Proyectos Redmine[1]. Esta herramienta permite la gestión de múltiples proyectos a la vez con capacidad suficiente para supervisar el avance del equipo de trabajo. Cuenta con varios módulos que pueden ser modelados de acuerdo a las necesidades de los usuarios y activados para determinados perfiles según los permisos correspondientes; garantizando la seguridad de la información.

Una problemática en términos de rendimiento en cuanto al uso de esta herramienta es el incremento continuo de información que se almacena en su base de datos, provocando el acelerado crecimiento de las tablas que la componen. Una vez que aumenten los volúmenes de datos que se almacena en las tablas, aumenta también la cantidad de filas a recorrer por el sistema cuando se ejecutan consultas sobre ellas. Es por ello que los tiempos de respuesta que se obtienen a estas consultas no son los mejores, provocando en los usuarios que interactúan con la aplicación retraso en la obtención de la información deseada.

[1] Herramienta para la gestión de proyectos y el seguimiento de errores, desarrollada sobre el framework Ruby on Rails. http://www.redmine.org/

Teniendo en cuenta la situación problemática descrita anteriormente se plantea como **problema de la investigación**: ¿Cómo mejorar los tiempos de respuestas de consultas que se realizan a la base de datos del Redmine?

A partir de lo planteado anteriormente se define como **objeto de estudio** las técnicas de optimización en base de datos PostgreSQL, enmarcado en el **campo de acción** particionado de las tablas críticas de la base de datos del Redmine.

Se define como **objetivo general:** aplicar las técnicas de particionado a las tablas críticas del Redmine para reducir los tiempos de respuestas de las consultas que se ejecutan a su base de datos.

A partir del objetivo general, se derivan los siguientes **objetivos específicos**:

1. Caracterizar las técnicas de particionado de tablas en PostgreSQL.
2. Caracterizar el modelo de datos del Redmine para obtener los atributos por los cuales se realizará el particionado.
3. Realizar el particionado a las tablas críticas de la base de datos del Redmine utilizando las técnicas de particionado de tablas estudiadas.
4. Realizar las pruebas de rendimiento a la base de datos del Redmine antes y después del particionado.

Para darle cumplimiento al objetivo planteado se proponen las siguientes **tareas de la investigación**:

1. Identificación de las técnicas de particionado de tablas en PostgreSQL.
2. Identificación de los atributos de las tablas de la base de datos del Redmine.
3. Pruebas de rendimiento utilizando la herramienta pgFouine a la base de datos del Redmine antes de ser particionada.
4. Selección de las tablas críticas del modelo de datos del Redmine a las cuales se realizará el particionado.
5. Aplicación de la técnica de particionado seleccionada.
6. Pruebas de rendimiento utilizando la herramienta pgFouine, a la base de datos del Redmine particionada.
7. Comparación de las pruebas realizadas.

El trabajo está estructurado de la siguiente manera: introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, bibliografías, anexos y glosario de términos. Los capítulos se distribuyen de la siguiente forma:

Capítulo 1: Fundamento teórico

En este capítulo se brinda información referente a la herramienta de gestión de proyectos Redmine, sus características principales y sus propiedades. Se argumenta acerca de los Sistemas de Gestión de Bases de Datos (SGBD) y se especifica cuál se usará en el desarrollo del trabajo. Se aborda el tema de particionado de tablas especificando los tipos de particionado que existen en PostgreSQL y cuál se utilizará para la solución del problema existente. Se referencian las herramientas que se utilizarán para la propuesta de la solución.

Capítulo 2: Diseño e implementación de la solución propuesta

En el desarrollo de este capítulo se obtiene el modelo de datos del Redmine. Se seleccionaron las tablas críticas de la base de datos para realizar el particionado de tablas. Se describe la aplicación de la técnica de particionado seleccionada para arribar a la solución propuesta. Se realizan pruebas de rendimiento a la base de datos antes de ser particionada.

Capítulo 3: Pruebas de rendimiento

En este capítulo se describen las pruebas de rendimiento que se realizaron a la base de datos del Redmine después de realizar el particionado. Además se realizan comparaciones de las pruebas realizadas antes y después de ser particionada la base de datos del Redmine para validar la solución, la cual mejora los tiempos de respuestas a las peticiones que se realizan al Redmine.

Capítulo I: Fundamentos Teóricos

Introducción del capítulo

En el presente capítulo se expone un grupo de conceptos que servirán de base para el desarrollo de la investigación. Un breve enfoque de las herramientas que existen en la actualidad para llevar a cabo la Gestión de Proyectos haciendo énfasis en la herramienta Redmine. Además se explican las técnicas de particionado de datos que existen en PostgreSQL. Se describen brevemente las herramientas a utilizar y el por qué de su selección.

1.1- Bases de datos

Con el desarrollo constante de aplicaciones cada vez más potentes se genera un gran cúmulo de información que necesita ser guardada en un entorno seguro para su posterior uso. Por esta razón el manejo de datos en las empresas se convirtió en un proceso engorroso. A raíz de estos problemas surgieron las bases de datos, que son muy utilizadas actualmente por las propiedades que tienen de almacenar grandes volúmenes de información de una forma segura y persistente. Mantener la información almacenada en una base de datos permite que se pueda acceder a ella desde cualquier lugar y en cualquier instante de tiempo. Hoy día la información que existe sobre las bases de datos es extensa y variada. A continuación se exponen algunos conceptos de bases de datos de diferentes fuentes.

Las bases de datos no son más que un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. Es un conjunto de datos consistente y usualmente persistente, organizados en un modo específico que permite acceder a la información de forma fácil y rápida (1).

Una base de datos es un conjunto de datos almacenados entre los que existen relaciones lógicas y ha sido diseñada para satisfacer los requerimientos de información de una empresa u organización. Se define una sola vez y puede ser utilizada por varios departamentos y usuarios. La base de datos no pertenece a un departamento en específico si no que se comparte por toda la red. En una base de datos, además de los datos, también se almacena su descripción (2).

Se define una base de datos como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular (3).

De los conceptos antes mencionados se concluye para esta investigación que una base de datos es un sistema que almacenará de forma segura y organizada toda la información de una empresa u organización. Los datos que se almacenan están relacionados entre sí y pueden ser consultados por todos los usuarios que pertenezcan al sistema.

1.2- Sistemas de Gestión de Bases de Datos (SGBD).

Mantener la información en una base de datos fue una solución que surgió con el objetivo de almacenar los grandes volúmenes de información que se generaban en las empresas. Para poder gestionar y administrar estas bases de datos se necesitaba de un software que permitiera realizar diferentes acciones sobre los datos guardados. Es por ello que surgen los Sistemas de Gestión de Base de Datos o SGBD como también se conocen.

Un sistema gestor de base de datos se define como el conjunto de programas que administran y gestionan la información contenida en una base de datos. Permite la definición de los datos, el mantenimiento de la integridad de los datos dentro de la base de datos, el control de la seguridad y privacidad de los datos y la manipulación de los datos (4).

Un Sistema de Gestión de Bases de Datos es un conjunto de programas que permite a los usuarios crear y mantener una base de datos. Es un sistema de software de propósito general que facilita el proceso de definir, construir y manipular las bases de datos para diversas aplicaciones (5).

Un SGBD es el software que permite definir, construir y manipular las bases de datos de una forma sencilla y fácil para los usuarios sin necesidad de conocer el funcionamiento interno de una base de datos. Permite acceder a la información almacenada desde varios puntos concurrentemente y en cualquier instante de tiempo.

1.3- Diseño de las Bases de Datos.

Para poder utilizar las bases de datos satisfactoriamente, se debe realizar un proceso de diseño, este diseño contendrá las especificaciones del cliente acerca de lo que desea guardar para su posterior uso. El diseño de la base de datos es una de las actividades más importantes que se debe realizar en el desarrollo de un software. Obtener un buen diseño de base de datos que refleje con claridad y seguridad las necesidades del cliente para el cual se está desarrollando el software garantiza la calidad y la usabilidad del producto final. Para lograr un eficiente diseño de una base de datos deben cumplirse los siguientes pasos:

- ✓ Determinar la finalidad de la base de datos: esto ayudará a entender el propósito por el cual se está creando la base de datos.
- ✓ Buscar y organizar la información necesaria: reunir todos los tipos de información que se desea registrar en la base de datos.
- ✓ Dividir la información en tablas: dividir los elementos de información en entidades o temas principales. Cada tema pasará a ser una tabla de la base de datos.
- ✓ Convertir los elementos de información en columnas: decidir qué información se desea almacenar en cada tabla. Cada elemento se convertirá en un campo y se mostrará como una columna en la tabla.
- ✓ Especificar claves principales: elegir la clave principal de cada tabla. La clave principal es una columna que se utiliza para identificar inequívocamente cada fila.
- ✓ Definir relaciones entre las tablas: examinar cada tabla y decidir cómo se relacionan los datos de una tabla con las demás. Agregar campos a las tablas o crear nuevas tablas para clarificar las relaciones según sea necesario.
- ✓ Ajustar el diseño: analizar el diseño para detectar errores. crear las tablas y agregar algunos registros con datos de ejemplo. Comprobar si se pueden obtener los resultados previstos de las tablas. En caso de fallo realizar los ajustes necesarios en el diseño.
- ✓ Aplicar las reglas de normalización: aplicar reglas de normalización de los datos para comprobar si las tablas están estructuradas correctamente. Realizar los ajustes necesarios en las tablas (6).

Es importante seguir los pasos descritos anteriormente, como resultado de ello se obtendrá un diseño de la base de datos que satisfaga las necesidades del cliente. Uno de los resultados intermedios en el diseño de base de datos es el modelo entidad relación (ER, Entity Relationship). El mismo pertenece a los modelos de datos conceptuales orientados a objetos y que representa los requisitos del mundo real, facilitando la comunicación con el usuario y de este modo verificar si satisface sus necesidades.

1.4- Modelo de datos

En la informática, un modelo de datos es un lenguaje utilizado para la descripción de una base de datos. Por lo general, permite describir las estructuras de datos (el tipo de los datos que incluye y la

forma en que se relacionan), las restricciones de integridad (las condiciones que los datos deben cumplir para reflejar correctamente la realidad deseada) y las operaciones de manipulación de los datos (agregado, borrado, modificación y recuperación de los datos). Un modelo de datos permite describir los elementos que intervienen en una realidad o en un problema dado y la forma en que se relacionan dichos elementos entre sí (7).

Los modelos de datos presentan dos sub-lenguajes:

Lenguaje de Definición de Datos o DDL (Data Definition Language), se basa en describir las estructuras de datos y las restricciones de integridad de una forma abstracta.

Lenguaje de Manipulación de Datos o DML (Data Manipulation Language), se orienta a describir las operaciones de manipulación de los datos. Se le suele conocer como Lenguaje de Consulta o QL (Query Language) (7).

De acuerdo al nivel de abstracción se clasifican en:

Modelos de datos conceptuales: son aquellos que describen las estructuras de datos y restricciones de integridad. Se utilizan durante la etapa de análisis de un problema dado y están orientados a representar los elementos que intervienen y sus relaciones.

Modelos de datos lógicos: se centran en las operaciones y se implementan en algún manejador de base de datos.

Modelos de datos físicos: son estructuras de datos a bajo nivel implementadas dentro del propio manejador (7).

En esta investigación se realizó la ingeniería inversa a la base de datos del Redmine y se obtuvo el modelo de datos de esta herramienta. Este modelo contiene todas las tablas de la base de datos y sus atributos correspondientes.

1.5- Herramienta de Gestión de Proyectos: Redmine

El uso de las herramientas de gestión de proyectos en los procesos de desarrollo de software es de suma importancia, permiten llevar un mejor control de los trabajadores y las diferentes tareas que se le asignan. Agilizan el proceso de toma de decisiones en los proyectos e intervienen en la calidad de los productos finales. Además se puede estimar el tiempo que llevará realizar el software y de esta forma

organizar las tareas necesarias a realizar en el desarrollo del mismo para entregar el producto en tiempo y con la calidad requerida.

Existen diferentes herramientas desarrolladas para este fin, algunas de ellas son: Gantt PV, Gantt Project, Dotproject, Team Work, Planner, Trac y Redmine. Esta última es utilizada en la Universidad de las Ciencias Informáticas (UCI) para llevar a cabo la gestión de los diferentes proyectos que se desarrollan. Para el desarrollo de esta investigación se utilizó el Redmine en su versión 1.3.2.

El Redmine es un sistema multiplataforma con interfaz web, implementado en el lenguaje Ruby sobre el framework Rails, de código abierto y creado bajo licencia GPL. Provee una interfaz sencilla y funcionalidades para la gestión de proyectos. Está orientado a la coordinación de tareas y comunicación de participantes. Se integra a SVN Subversion, sistema que permite llevar un control de todas las versiones que se desarrollan en los proyectos.

Una vez instalada, el administrador puede dar de alta a los proyectos a través de la interfaz web y cada jefe de proyecto puede acceder al mismo y gestionar las tareas de cada uno de los trabajadores que pertenecen a su proyecto. Cada trabajador que responde a un rol, tiene en su página de entrada una lista de las tareas que tiene asignadas según el rol que desempeña dentro del proyecto. A medida que los usuarios trabajan en las actividades orientadas, el Redmine permite establecer un porcentaje de realización de estas tareas y estimar el tiempo que les llevará finalizarlas.

Su funcionamiento interno está basado en el patrón de diseño Modelo Vista Controlador. Posee clases implementadas en el lenguaje Ruby que son las clases que controlan el flujo de datos que se almacena en cada una de las tablas de la base de datos. Contiene páginas html que son las que permiten al usuario gestionar manual y visualmente las tareas, los proyectos y el personal implicado de una manera cómoda y sencilla.

Funcionalidades de la herramienta de Gestión de Proyectos: Redmine.

Gestión de múltiples proyectos: el Redmine permite gestionar múltiples proyectos desde una sola interfaz con una ventana de navegador. La navegación es sencilla, permitiendo cambiar de proyecto en cualquier momento. Cada proyecto puede tener una configuración totalmente diferente y el usuario puede tener un rol distinto en cada uno de los proyectos al cual pertenece. Los proyectos pueden definirse como privados o públicos. En los privados el administrador puede dar acceso a usuarios escogidos dentro del colectivo de trabajadores y los públicos estarán visibles para todo el personal. Dentro de cada proyecto pueden definirse varios sub-proyectos estableciendo una jerarquía de fácil entendimiento para el usuario.

Personalización de proyectos: cada proyecto es totalmente personalizable, dentro de la aplicación pueden encontrarse proyectos distintos entre sí según sus objetivos. Se pueden desactivar o activar diferentes módulos: wiki, foro, noticias, peticiones, control del tiempo, documentos, ficheros o repositorio.

Sistema flexible de seguimiento de tareas: una de las características más útiles del Redmine es la facilidad que brinda para el seguimiento de las peticiones que se asignan a los usuarios y su visualización. Estas peticiones se dividen en tres tipos; errores, tareas y soporte. Cada petición será asignada a un solo miembro del proyecto. Se puede indicar una fecha de inicio y fin para esa petición y llevar un control del tiempo y porcentaje realizado. Permite asignar una prioridad y enlazar con la subida de un fichero. Con los datos que se almacenan, pueden visualizarse las peticiones de manera personalizada estableciendo filtros y contribuir a la generación de informes de tareas o incidencias.

Integración en repositorios de código: el Redmine puede integrarse con un repositorio de código (Subversion, Git, CVS) que esté montado en la misma máquina. La aplicación sirve de interfaz web para el seguimiento del desarrollo de un proyecto. Pueden descargarse los ficheros, ver el historial, los cambios y descargar un archivo a modo de parche para aplicar a código desactualizado. Es un sistema de seguimiento de versiones, aunque no pueden actualizarse los ficheros directamente.

Uso de calendario y diagrama de Gantt: Redmine incluye un calendario para visualizar todas las peticiones a lo largo de un mes elegido, marcando claramente el día de inicio y de fin de cada petición. Igualmente ocurre con la vista en diagrama de Gantt, que va marcando el porcentaje completado conforme avanzan los días. Las peticiones que se visualizan en ambos casos están sujetas a los filtros definidos por el usuario.

Notificaciones: configurando previamente el servidor de correo SMTP, Redmine permite enviar notificaciones por correo electrónico en todos los proyectos, definiendo antes los eventos que activan estos avisos. Además cada usuario en su configuración puede elegir recibir notificaciones de cualquier evento, o solo las relacionadas con su perfil. Puede configurarse además el servidor de correo entrante, permitiendo actualizar peticiones por email e incluso crear nuevas.

Exportación a distintos formatos: los informes de peticiones que pueden generarse añadiendo filtros, y que permiten visualizar las diferentes tareas de un proyecto, pueden exportarse en PDF o formato CSV, facilitando la impresión posteriormente en un formato organizado. Las páginas de la wiki

pueden exportarse en HTML o TXT.

Fallos y/o carencias importantes: Redmine es un gestor de proyectos muy potente y maduro. Además es una herramienta muy flexible y bastante estable. Durante su uso, es difícil o fácil sacarle carencias, siempre dependiendo de lo que esté buscando cada usuario. En esta investigación uno de los fallos que se obtuvo fue el tiempo de respuesta que se obtiene cuando se realizan consultas a su base de datos, retrasando la obtención de información a usuarios que interactúan con la aplicación.

Otras características: Redmine es una herramienta que permite configurar casi todas las funcionalidades que brinda. Otra funcionalidad que se destaca en esta herramienta es la página personal de cada usuario, que ofrece una vista personalizable con información de todos los proyectos donde esté participando. También se pueden subir ficheros, definir campos personalizados para cada módulo, usar la barra de búsqueda global, y ampliar la funcionalidad con otras extensiones. Admite como bases de datos MySQL, SQLite y PostgreSQL.

Diseño de la interfaz: el diseño de la aplicación tiene una interfaz web muy sencilla y fácil de manejar. Para cada proyecto existen una serie de pestañas fijas en la parte superior que organizan los diferentes módulos. Dentro de cada módulo se muestra la información. Destaca la facilidad para configurar los proyectos y los colores elegidos no recargan la herramienta.

Facilidad de uso: la aplicación es muy sencilla de usar y aunque el ámbito principal puede ser el empresarial o el de desarrollo de software, cualquier usuario podría utilizar el Redmine para administrar sus propios proyectos o tareas. La navegación web se hace muy intuitiva a los usuarios que no conozcan mucho acerca de esta herramienta (8).

Portabilidad / Adaptabilidad de la herramienta de gestión de proyectos: Redmine.

Plataformas disponibles: Redmine es una aplicación multiplataforma, para instalarla se necesita una base de datos que puede ser MySQL, PostgreSQL o SQLite y Ruby on Rails en sus versiones apropiadas. A nivel de cliente Redmine puede ser accedido desde cualquier plataforma o sistema operativo a través de un navegador web (8).

Lenguaje de programación y framework de desarrollo de la herramienta de gestión de proyectos: Redmine.

La herramienta de gestión de proyecto Redmine está implementada en Ruby sobre el framework Rails.

Ruby es un lenguaje de scripts, multiplataforma, orientado a objetos y software libre, fue creado por el programador japonés Yukihiro Matz Matsumoto. Hereda varias características de otros lenguajes como: Perl, Smalltalk, Eiffel, Ada y Lisp. Cada día aumentan el número de empresas y usuarios que lo utilizan como lenguaje de programación para el diseño web.

Ralis es un framework para el desarrollo de aplicaciones web, software libre, está basado en el patrón de diseño Modelo Vista Controlador (MVC) y se implementó en una aplicación orientada a la administración de proyectos llamada Basecamp. Cada día se unen más personas al desarrollo de este framework (9).

Según las características antes mencionadas se concluye que el Redmine es una aplicación que permite llevar el control total del equipo de trabajo y las actividades a realizar en el desarrollo de un software. Permite la elaboración de reportes que contribuyen a la toma de decisiones de los líderes de proyecto. Está integrado a un sistema de control de versiones y presenta funcionalidades como el diagrama de Gantt que permite estimar el tiempo que tardará en realizarse un software. Brinda la posibilidad a los administradores de establecer permisos a los usuarios y asignarle un rol a cada uno, con la posibilidad de asignar tareas según el rol que desempeñan dentro del proyecto.

1.6- Buenas prácticas para mejorar los tiempos de respuestas a las peticiones en las bases de datos.

Uno de los problemas más frecuentes que presentan las aplicaciones web en general, son los tiempos de respuestas con que responden a las peticiones que se le realizan. En algunos casos este tiempo no es el más eficiente y se debe a errores cometidos durante el diseño de la base de datos o cuando las tablas que la conforman contienen grandes volúmenes de información almacenada que elevan el tiempo de respuestas a las consultas que se realizan. Otros factores que influyen en el rendimiento de las bases de datos, están dados por las características de hardware que poseen las computadoras que se utilicen. A continuación se explica como mejorar los tiempos de respuesta del intérprete ante determinadas situaciones:

En cuanto a:

Diseño de las tablas

- ✓ Normalizar las tablas, al menos hasta la tercera forma normal para asegurar que no existe duplicidad de datos y se aproveche al máximo el almacenamiento en las tablas.
- ✓ Los primeros campos de cada tabla deben ser aquellos campos requeridos y dentro de estos se definen primeramente los de longitud fija y luego los de longitud variable.

- ✓ Ajustar al máximo el tamaño de los campos y de esta manera se desperdicia menos espacio.
- ✓ En la mayoría de las tablas es habitual que se cree un campo de texto para insertar observaciones o descripciones de esa tabla, si este campo se va a utilizar con poca frecuencia o si se ha definido con gran tamaño, es más factible crear una nueva tabla que contenga la clave primaria y el campo para las observaciones.

Campos a seleccionar

- ✓ En la medida de lo posible, evitar que las sentencias SQL estén dentro del código de la aplicación. Usar vistas o procedimientos almacenados es más factible porque el gestor los guarda compilados.
- ✓ Seleccionar exclusivamente los campos que se necesiten para obtener la información.
- ✓ No utilizar nunca **SELECT *** puesto que el gestor debe leer primero la estructura de la tabla antes de ejecutar la sentencia.
- ✓ Si se utilizan varias tablas en una consulta determinada, especifique siempre a que tabla pertenece cada campo, le ahorrará al gestor el tiempo de localizar a que tabla pertenece este campo.

Campos de Filtro

- ✓ En la cláusula WHERE, se tratarán de elegir aquellos campos que formen parte de la clave del fichero por el cual interrogamos. Se especificarán en el mismo orden en el que estén definidos en la clave.
- ✓ Interrogar siempre por campos que sean clave.

Condiciones de hardware

- ✓ Realizar cambios en los componentes físicos de los equipos dígase memoria RAM (Random Acces Memory, Memoria de Acceso Aleatorio), microprocesador y otros elementos tecnológicos.

Particionado de tablas

- ✓ Aplicar el particionado de tablas en base de datos a aquellas tablas que contienen la mayor cantidad de registros (varios millones o más) dentro de la base de datos.
- ✓ Aplicar el particionado adecuado teniendo en cuentas el SGBD que se utilice.

De las situaciones enunciadas anteriormente el diseño de tablas no se puede utilizar en la base de datos que se analiza puesto que el Redmine cuenta con una BD que cumple con las exigencias necesarias para el diseño. Además cambios en el diseño de la base de datos del Redmine implicaría un mal funcionamiento del sistema. Por otro lado el Redmine tiene implementada a nivel de código la manera en la cual seleccionará los campos de las tablas que se consulten, por tanto los campos a seleccionar no sería factible utilizarla debido a que se violarían los estándares de programación seguidos para cada una de las clases. Similar pasa con la selección de los campos de filtro. En cuanto a las propiedades de hardware, no se puede aumentar las prestaciones físicas del servidor, dado que la entidad no cuenta con recursos financieros que lo permitan.

Como en el modelo de datos del Redmine existen tablas que poseen un alto crecimiento de la información que almacenan y teniendo en cuenta la explicación antes expuesta, para esta investigación se utilizará el particionado de tablas para disminuir los tiempos de respuesta a las consultas que se realizan a la base de datos del Redmine.

1.7- Particionado de tablas

El particionado de tablas es una técnica que se utiliza para reducir la cantidad de lecturas físicas a la base de datos cuando se ejecutan consultas. Se basa en descomponer una tabla con una gran cantidad de información, que se denomina tabla padre; en un conjunto de tablas hijas mejorando el rendimiento a la hora de consultar las tablas y obtener datos de la misma. Se realiza por razones de mantenimiento, rendimiento o gestión. El particionado permite distribuir porciones de una tabla individual en diferentes segmentos rigiéndose por reglas establecidas por el usuario.

Entre las ventajas que presenta se encuentra:

- ✓ Desempeño mejorado para la obtención de registros (querys).
- ✓ Desempeño mejorado para la actualización (update).
- ✓ Índices más pequeños para cada tabla hija contra índices grandes y difíciles de colocar en memoria en una tabla grande.
- ✓ Eliminación rápida de registros.
- ✓ Los datos pueden ser respaldados en medios económicos y pequeños como DVDs o discos duros extraíbles (10).

El particionado de tablas es una forma de organizar los datos por criterios de agrupación, de manera

que cada consulta realizada a la tabla padre se redirija automáticamente hacia grupos de datos con menor cantidad de registros que se encuentran agrupados en las tablas hijas, proporcionando, a la hora de realizar las consultas, un ahorro significativo en el tiempo de respuesta a las mismas.

Según el SGBD que se utilice se realiza el tipo de particionado adecuado, debido a que cada SGBD tiene su modalidad de realizar el particionado. En esta investigación se utilizará el particionado en PostgreSQL que es el SGBD para el cual está diseñada la base de datos del Redmine, pero existen otros SGBD como Oracle y MySQL, que al igual que PostgreSQL, permiten el particionado en tablas de bases de datos.

1.7.2- Particionado de tablas en PostgreSQL

En PostgreSQL el tipo de particionado soportado se denomina **particionado mediante herencia de tablas** y consiste en que cada partición es creada como una tabla hija de una única tabla padre que inicialmente debe ser una tabla vacía y que va a representar a todo el conjunto de datos que luego serán distribuidos por las tablas hijas según los criterios y las reglas que se implementen.

Los tipos de particionado más comunes en PostgreSQL son:

Particionado por rangos: la tabla es particionada mediante rangos definidos en base a la columna de llave primaria o cualquier columna que no se solape entre los rangos de valores asignados a diferentes tablas hijas.

Particionado por lista: la tabla es particionada listando los valores de cada una de las llaves en cada partición (10).

Para poder utilizar dicha técnica en PostgreSQL y se obtengan los mejores resultados es necesario seguir los siguientes pasos:

1. Crear la tabla padre, de la cual todas las tablas hijas heredarán. Esta tabla inicialmente debe estar vacía y no debe tener restricciones ni índices.
2. Crear todas las tablas hijas heredando de la tabla maestra.
3. Agregar a todas las tablas hijas las restricciones correspondientes sobre los datos que albergarán.
4. Para cada partición, crear un índice para la (s) columna (s) llave. Si se necesita una llave única o llave primaria, se necesitará crearla para cada tabla hija.

5. Definir una regla o disparador para redirigir las modificaciones de la tabla padre a la apropiada partición.
6. Verificar que esté habilitado el parámetro constraint_exlusion, en caso de no estarlo, cambiar la configuración de este parámetro a **on** en el archivo de configuración postgresql.conf para que los querys sean optimizados para el particionado (10).

Se puede concluir que el particionado en PostgreSQL es utilizado para reducir la cantidad de datos a recorrer en cada consulta SQL y aumentar el rendimiento. Consiste en segmentar una tabla que se denomina tabla padre, en un conjunto de tablas con menor tamaño, que se llamarán tablas hijas teniendo en cuenta reglas implementadas con anterioridad. Se puede aplicar de dos formas: por lista y por rango. Se debe aplicar cuando existen tablas con grandes volúmenes de datos que pueden influir en el rendimiento, específicamente en el tiempo de respuesta a las consultas que se realicen sobre estas. Las ventajas principales que posee, es que una vez particionada la tabla, el gestor no debe recorrerla completamente si no que buscará en segmentos de datos más pequeños organizados en las tablas hijas según reglas definidas previamente. Este proceso facilita la obtención de la información deseada con mayor rapidez.

En esta investigación el tipo de particionado que se aplicará se seleccionará teniendo en cuenta las características de cada tabla a particionar. Las tablas de una base de datos contienen sus atributos propios, teniendo en cuenta las características de cada uno de los atributos se seleccionará el adecuado para la realización del particionado.

1.8- Herramientas informáticas utilizadas

En el desarrollo del presente trabajo de diploma se hará uso de diferentes herramientas informáticas que permitirán el avance de la investigación.

1.8.1- Herramienta CASE: Visual Paradigm for UML

Para la obtención del modelo de datos de la herramienta de gestión de proyectos: Redmine y el rediseño del mismo se utilizará el Visual Paradigm for UML 8.0 como herramienta CASE.

Esta herramienta soporta el ciclo de vida completo del desarrollo de software: análisis y diseño, construcción, pruebas y despliegue. El software de modelado UML (Unified Modeling Language, Lenguaje Unificado de Modelado) ayuda a una rápida construcción de aplicaciones de calidad a un menor coste. Permite diseñar todos los tipos de diagramas de clases, con sus atributos y las relaciones que existen entre ellas. Otra de las funcionalidades que tiene esta herramienta es que permite realizar

la ingeniería inversa de los datos característica fundamental a tener en cuenta para esta investigación debido a que se obtendrá el modelo de datos del Redmine y a partir de este se desarrollarán las demás tareas propuestas.

Ofrece:

- ✓ Entorno de creación de diagramas para UML 2.1.
- ✓ Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- ✓ Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- ✓ Capacidades de ingeniería directa e inversa.
- ✓ Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- ✓ Disponibilidad de múltiples versiones, para cada necesidad.
- ✓ Disponibilidad de integrarse en los principales IDEs.
- ✓ Disponibilidad en múltiples plataformas (11).

1.8.2- Servidor de aplicaciones web: Apache

El servidor que se utilizará será Apache2. El mismo es un servidor de páginas web de código abierto multiplataforma y modular. Fue creado en el año 1996 y se desarrolla con todo éxito dentro del proyecto HTTP Server (httpd) de la Apache Software Foundation.

Presenta entre otras características: mensajes de error altamente configurables, bases de datos de autentificación y negociado de contenido.

Su flexible sistema modular, permite cargar y descargar módulos sin necesidad de tocar el kernel. Dispone de una herramienta (APXS) que facilita la compilación e instalación de estos módulos, ya sean del mismo Apache o de terceras partes. Los módulos se cargan en memoria cuando los necesita y se descargan automáticamente cuando dejan de utilizarse.

Dentro de sus características principales se encuentran:

- ✓ Trabaja sobre múltiples plataformas (Unix, Linux, MacOSX, Vms, Win32, OS2, etc.)
- ✓ Incluye módulos que se cargan de forma dinámica.
- ✓ Soporta CGI, Perl, PHP.
- ✓ Soporte para Bases de datos.

- ✓ Soporte SSL para transacciones seguras.
- ✓ Incluye soporte para host virtuales.
- ✓ Soporta HTTP 1.1.
- ✓ Código abierto.
- ✓ Rápido y eficiente (12).

1.8.3- Herramienta de administración de bases de datos: PgAdmin III

Para la gestión y administración de la base de datos se utilizará el PgAdmin III. Aplicación de alto desempeño para la administración y desarrollo de los servidores de base de datos para PostgreSQL. Permite trabajar eficientemente con las tablas de la base de datos ya sea para realizar el particionado de tablas y también para ejecutar las consultas necesarias a la misma.

Soporta grandes volúmenes de datos que permitirá realizar las pruebas necesarias una vez propuesta la solución.

1.8.4- Herramienta para el monitoreo del servidor de PostgreSQL: pgFouine

Las herramientas de monitoreo de base de datos, como su nombre lo indica, se utilizan para monitorear todas las operaciones que se realizan en las bases de datos, dígase consultas, actualizaciones o búsquedas. Según su funcionamiento, generan reportes de los tiempos que demoran en realizarse estas operaciones y otros datos que contribuyen a la toma de decisiones. Existen muchas herramientas creadas para este fin, algunas de ellas son: DBF reader and writer, PHP Automatic form generator, phpexcel SQL to Excel, Odp .Net y pgFouine.

Para el desarrollo de esta investigación se utilizará como herramienta de monitoreo el pgFouine. Algunas características de esta herramienta: está escrita en php, software libre, orientado a objetos y basa su funcionamiento en analizar los registros (logs) de PostgreSQL. Para ello se especifican en el archivo de configuración de PostgreSQL diferentes parámetros que permitirán generar el informe que contendrá todas las funciones que se realizan en las bases de datos PostgreSQL y contribuir a la toma de decisiones para mejorar los tiempos de respuesta a las peticiones en la base de datos.

Las bases de datos se monitorean con el objetivo de:

- ✓ Verificar el consumo de recursos.
- ✓ Tiempo de respuesta del servidor.

- ✓ Recabar la mayor cantidad de información a fin de poder tener los suficientes datos para ubicar donde está el problema.
- ✓ Detección de problemas de hardware o red.
- ✓ Obtener información de una determinada tarea o consulta (13).

1.8.5- Herramienta de generación de datos de prueba: EMS Data Generator para PostgreSQL

En esta investigación se utilizará el EMS Data Generator 2005 para PostgreSQL como herramienta de generación de datos. Esta herramienta simula el entorno de producción de base de datos y permite rellenar varias tablas por separado y a la misma vez, permitiendo configurar o establecer parámetros de generación para cada tipo de campos de las diferentes tablas.

Presenta disímiles características: muestra una interfaz de usuario fácil de usar con la posibilidad de guardar y editar los datos generados en la secuencia de comandos SQL sin ejecutar consultas en el servidor. Puede generar datos por la red, es decir, no es necesario estar en la misma máquina donde se encuentra montado el servidor web para generar datos en una base de datos determinada perteneciente a ese servidor. Brinda la posibilidad de pre visualizar los datos generados y también muestra mensajes de error cuando los datos generados no cumplen con las especificaciones de la tabla, por ejemplo cuando existe una llave primaria duplicada.

1.8.6- Sistema Gestor de Base de Datos: PostgreSQL

En esta investigación se hará uso del Sistema Gestor de Base de datos PostgreSQL en su versión 8.4. A pesar de existir versiones superiores como la versión 9.1, se utilizará la versión 8.4 debido a que el script de la base de datos que se analiza está diseñado para esta versión. Después de una década de desarrollo, esta herramienta se ha convertido en el gestor de bases de datos de código abierto más avanzado de la actualidad, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones y funciones definidas por el usuario). Cuenta también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, Perl, PHP y Python).

PostgreSQL ofrece muchas ventajas respecto a otros gestores de bases de datos, entre las cuales se pueden encontrar:

- ✓ **Instalación ilimitada**: PostgreSQL no tiene costos asociados a la licencia del software, lo cual posibilita que se pueda instalar en varios servidores sin violar ningún acuerdo de licencia.

- ✓ **Soporte técnico:** PostgreSQL cuenta con numerosas ofertas de soporte por parte de empresas de software libre. Además existen comunidades de profesionales y entusiastas de las cuales se pueden obtener grandes beneficios para la misma.
- ✓ **Ahorros considerables en costos de operación:** el software es diseñado y creado para tener presente un mantenimiento y ajuste mucho menor que los productos de los proveedores comerciales, conservando todas las características, estabilidad y rendimiento.
- ✓ **Estabilidad y confiabilidad:** en contraste a muchos sistemas de bases de datos comerciales es extremadamente común que las compañías reporten que PostgreSQL no presenta caídas en varios años de operación de alta actividad.
- ✓ **Extensible:** el código fuente está disponible sin costo para todo aquel que necesite extender o personalizar PostgreSQL de alguna manera.
- ✓ **Multiplataforma y variadas herramientas gráficas de diseño y administración:** existen instaladores de los servicios y clientes PostgreSQL para los sistemas operativos Windows y las distribuciones de Linux. También se dispone de diversas herramientas gráficas de alta calidad para administrar las bases de datos y para hacer diseño: Toad, Data Architect, entre otras (14).

Conclusiones del capítulo

En este capítulo se explicaron las características fundamentales de la herramienta de gestión de proyectos: Redmine. Se seleccionó el particionado de tablas en base de datos como la técnica factible para mejorar los tiempos de respuesta a las consultas que se realizan y se explicó el por qué de su selección. Se abordaron las técnicas de particionado que existen en PostgreSQL. Por último se expuso una breve descripción de las herramientas definidas: para el modelado de software la herramienta CASE Visual Paradigm. Como servidor de aplicaciones web: Apache2. Para la administración de la base de datos la herramienta PgAdmin III. Como herramienta de monitoreo el pgFouine y como Sistema de Gestión de Base de Datos el PostgreSQL en su versión 8.4.

Capítulo II: Diseño e implementación de la solución

Introducción del capítulo

En este capítulo se describen procedimientos necesarios a seguir para realizar el particionado con el objetivo de obtener mejores tiempos de respuesta a las peticiones que se realizan a la base de datos del Redmine. Esta propuesta está basada en los estudios realizados en el capítulo anterior y se sustenta teniendo en cuenta las dificultades encontradas al realizar diferentes consultas a la base de datos del Redmine.

2.1- Obtención del modelo de datos del Redmine.

Con el objetivo de lograr una mejor comprensión del funcionamiento interno del Redmine y las tablas que componen su base de datos se obtuvo el modelo de datos del Redmine (Ver Figura 1 y 2). El modelo está compuesto por cuarenta y siete tablas, cada una de ellas con sus atributos propios y sus funcionalidades específicas dentro de la base de datos. Se muestran dos porciones del modelo de datos con el objetivo de facilitar la visualización. El modelo general con todas las tablas que lo componen se muestra en el Anexo 1 (Ver en la sección Anexos).

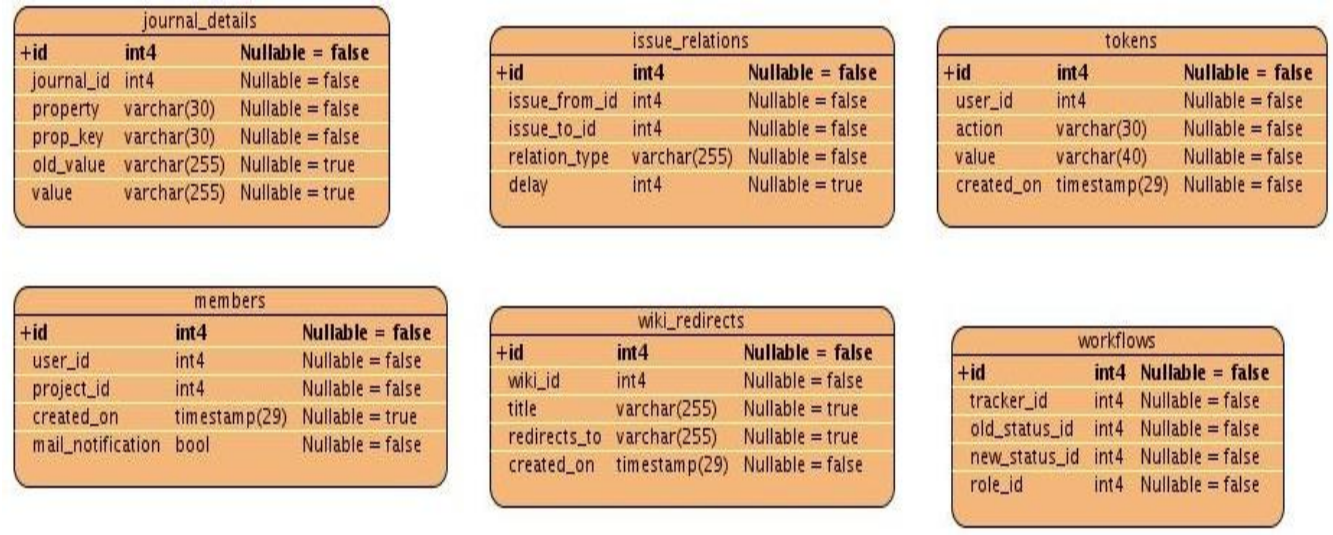


Figura 1. Porción del Modelo de datos del Redmine

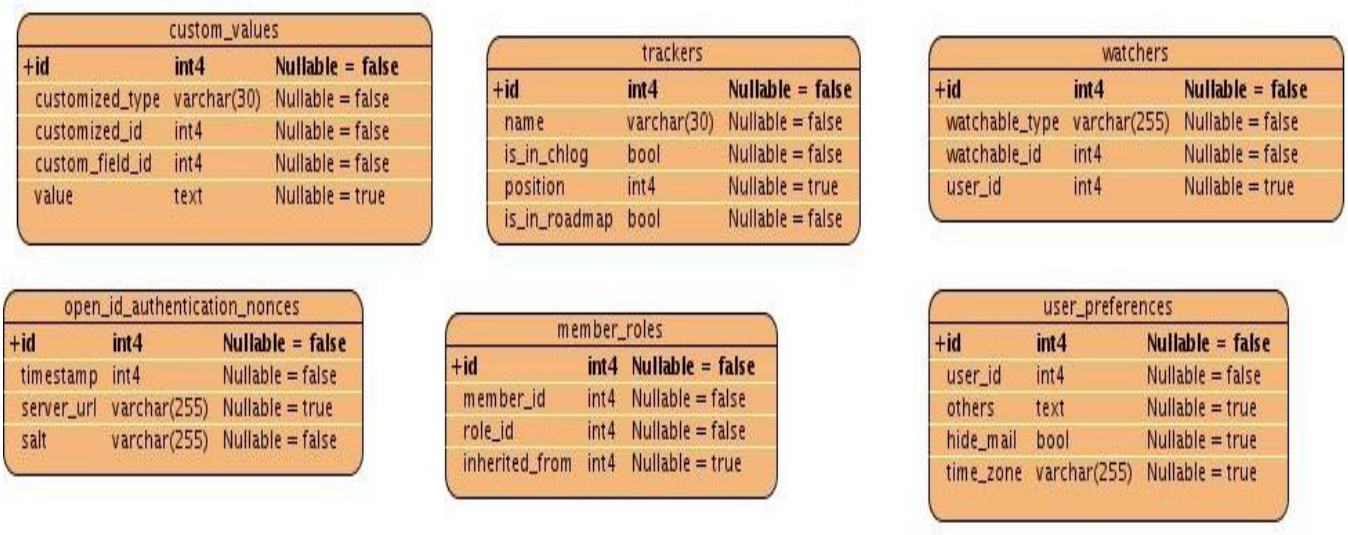


Figura 2. Porción del Modelo de datos del Redmine

2.2- Selección de las tablas críticas

Para realizar el particionado en una base de datos primeramente se deben identificar las tablas a la cual se le aplicará esta técnica, puesto que solo se realizará esta acción a aquellas tablas que afecten el rendimiento en la base de datos y se denominarán tablas críticas.

El proceso de selección de las tablas críticas dentro de una base de datos es una actividad difícil de realizar, debido a que todas las tablas no tienen las mismas características y funcionalidades. Además se debe analizar la base de datos de una organización en particular acorde al funcionamiento del entorno, puesto que esa misma base de datos en otro entorno de trabajo pudiera generar la misma cantidad, menos e incluso más tablas candidatas para realizar el particionado. Por tal razón se enfocarán algunos aspectos generales de cuando es conveniente realizar el particionado en tablas de base de datos.

Considerando los distintos tipos de tablas que existen en una aplicación se expone una clasificación de las tablas a efectos de particionado.

- ✓ Tablas maestras: son tablas que por lo general abundan mucho en las bases de datos, son de tamaño medio y no se modifican con regularidad. Por lo general cada registro tiene una vida larga que casi siempre es la vida de la aplicación.
- ✓ Tablas de movimientos: son pocas las que existen de este tipo en las bases de datos. Suelen ser las tablas más grandes de una aplicación y por lo general solo se le realizan consultas de

INSERT y DELETE. La permanencia de cada registro en la tabla suele ser de un tiempo determinado y prefijado.

- ✓ Tablas decodificadoras: de este tipo de tablas existen muchas en las bases de datos. Son tablas pequeñas que por lo general no se modifican nunca. Cada registro tiene una vida larga, la mayoría de las veces la vida de la aplicación.
- ✓ Tablas históricas: son aquellas en las que se guarda información para toda la vida. Los registros nunca se borran ni se modifican y la consulta que se realiza solo es INSERT (15).

Teniendo en cuenta la clasificación antes expuesta si la tabla es:

Tablas decodificadoras: no es necesario realizar el particionado, como se explicaba anteriormente, son tablas pequeñas que no afectan el rendimiento en la base de datos.

Tablas maestras: la mayoría de las veces estas tablas no contienen campos con criterio de particionado suficientemente claros, además de que su tamaño no suele ser muy grande. Es por ello que no es recomendable realizar el particionado cuando existan tablas de este tipo.

Tablas de movimiento: estas tablas como se explicaba anteriormente son las más grandes de la aplicación y por tanto las mayores candidatas a realizarle el particionado. Por lo general tienen un campo de tipo *timestamp* (valor estrictamente creciente, por ejemplo la fecha). Si se decide por rango debe establecerse de forma tal que en el borrado de datos se borren particiones enteras y que el número de particiones no supere las 100.

Tablas históricas: es conveniente realizar el particionado por los mismos criterios que las tablas de movimiento pero sin preocuparse por el borrado puesto que se explicaba anteriormente que en estas tablas no se realiza DELETE (15).

Para el desarrollo de esta investigación se tomarán los criterios descritos y se añadirán a ellos que:

- ✓ La tabla sea consultada con mayor frecuencia dentro de la base de datos
- ✓ El volumen de información que se almacena crezca constantemente.

Para la identificación de las tablas críticas se tuvo en cuenta la clasificación dada anteriormente. Otra de las actividades que permitieron la selección fue la realización de consultas al Catálogo de PostgreSQL. En esta ocasión se utilizó **pg_size_pretty**. Esta consulta permite saber el peso de una base de datos y también de las tablas que la componen.

Para poder utilizar esta consulta se creó una base de datos llamada redmine_datec. Se restauró un backup del centro DATEC (Centro de Tecnologías de Gestión de Datos) que contiene datos reales que se manejan en los proyectos de este centro.

Inicialmente se ejecutó la consulta: *select pg_size_pretty(pg_database_size('redmine_datec'));* con el objetivo de obtener el peso de la base de datos. En este caso la consulta arrojó el resultado 66 mb como peso de la base de datos.

Luego se identificaron las tablas que más peso tienen dentro de la base de datos. Para ello se ejecutó la consulta:

```
SELECT nspname, relname, pg_size_pretty(tablesize+indexsize+toastsize+toastindexsize) AS
totalsize
FROM (SELECT ns.nspname, cl.relname, pg_relation_size(cl.oid) AS tablesize,
COALESCE((SELECT SUM(pg_relation_size(indexrelid))::bigint
FROM pg_index WHERE cl.oid=indrelid), 0) AS indexsize,
CASE WHEN reltoastrelid=0 THEN 0
ELSE pg_relation_size(reltoastrelid)
END AS toastsize,
CASE WHEN reltoastrelid=0 THEN 0
ELSE pg_relation_size((SELECT reltoastidxid FROM pg_class ct
WHERE ct.oid = cl.reltoastrelid))
END AS toastindexsize
FROM pg_class cl, pg_namespace ns
WHERE cl.relnamespace = ns.oid
AND ns.nspname NOT IN ('pg_catalog', 'information_schema')
AND cl.relname IN
(SELECT table_name FROM information_schema.tables
WHERE table_type = 'BASE TABLE')) ss
ORDER BY tablesize+indexsize+toastsize+toastindexsize DESC;
```

Esta consulta devuelve las tablas de la base de datos especificando para cada una de ellas el nombre y el peso dentro de la base de datos ordenados de forma descendente. Luego de ejecutar la consulta se obtuvieron los resultados que se muestran en la Figura 3. Se muestran solo algunas tablas para facilitar la visualización. El resultado obtenido en todas las tablas se muestra en el Anexo 2. Además

en el Anexo 3 se muestra una gráficamente que representa el peso de las tablas de la base de datos del Redmine con respecto al peso total de la base de datos (Ver en la sección Anexos).

nspname name	relname name	totalsize text
public	changes	42 MB
public	custom_values	5544 kB
public	journals	2488 kB
public	issues	2184 kB
public	journal_details	1736 kB
public	attachments	384 kB
public	watchers	312 kB
public	time_entries	288 kB
public	workflows	256 kB
public	members	224 kB
public	wiki_content_versions	216 kB
public	member_roles	216 kB
public	users	192 kB
public	enabled_modules	128 kB

Figura 3. Peso de las tablas de la base de datos del Redmine

Según los resultados obtenidos se observa que las tablas más grandes de la base de datos en cuanto a su peso son: changes, custom_values, journals, issues y journal_details. Como son las tablas más grandes de la aplicación se clasifican en tablas de movimiento y por ende las candidatas para realizar el particionado teniendo en cuenta los criterios emitidos anteriormente.

Se definen como tablas críticas de la base de datos del Redmine las tablas changes, custom_values, journals, issues, journal_details y users (Ver Figura 4). La tabla users no se encuentra dentro de las que más peso tienen en la base de datos pero se decide escoger como tabla crítica para esta investigación debido a que los administradores del Redmine del centro DATEC, expresan *"que se trabaja constantemente con los usuarios del sistema, ya sea para adicionar o eliminar tareas o modificar las propiedades de los mismos. También cada usuarios que se conecta al Redmine genera información acerca de las actividades que realiza y parte de estas informaciones se guardan en la tabla users"*. Como es una de las tablas que se consulta con más frecuencia se decide adicionar como tabla crítica de la base de datos a la tabla users.

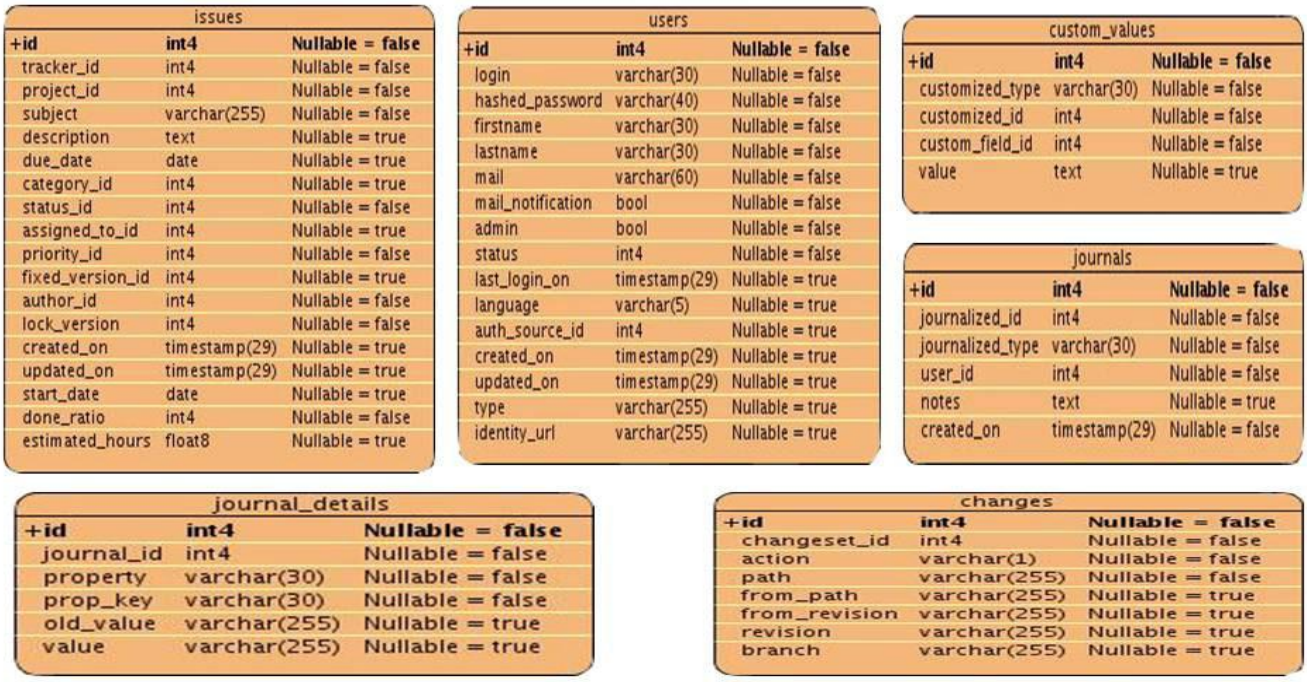


Figura 4. Tablas críticas identificadas en la base de datos del Redmine

2.3- Funcionalidades de las tablas críticas

Con el objetivo de comprender el comportamiento de las tablas críticas seleccionadas dentro de la base de datos del Redmine, se muestran en las Tablas 1, 2, 3, 4, 5 y 6 el funcionamiento de las mismas y sus atributos. Estas funcionalidades permitirán seleccionar la técnica de particionado a emplear en cada una de las tablas y también el atributo por el cual se realizará el particionado.

Tabla 1. Funcionalidad de la tabla users

Tabla users: almacena la información de todos los usuarios y grupos del Redmine	
Atributos	**Función**
id	Llave primaria que representa el identificador del usuario en la base de datos.
login	El nombre con el cual se loguerá el usuario en el sitio.
hashed_password	Contraseña con la que se logueará el usuario, se almacena encriptada
firstname	Primer nombre del usuario

lastname	Apellido
mail	Dirección de correo electrónico
mail_notification	Booleano que representa si se enviará notificación por correo al usuario creado.
admin	Booleano que representa si el usuario es administrador del sitio.
status	Entero que define si el usuario está activo o no. Si toma valor 1 está activo, si toma valor 0 no lo está
last_login_on	La última fecha y hora en que se logueó el usuario.
language	Lenguaje que tendrá la sesión de este usuario.
create_on	Fecha y hora en la que fue creado el usuario.
updated_on	Fecha y hora en la que se actualizó algún dato del usuario.
type	Que tipo es dentro del redmine en este caso es usuario.

Tabla 2. Funcionalidad de la tabla issues

Tabla issues: almacena la información referente a las peticiones que se crean en el Redmine	
Atributos	**Función**
id	Identificador de la tabla
tracker_id	Identificador de la petición.
project_id	Identificador del proyecto al cual está asignada esa petición.
subject	Título de la petición
description	Campo que permite dar una breve descripción de la petición.
due_date	Día final de la petición.
asigned_to_id	Identificador del usuario al cual se le asignó la petición.

priority_id	Identificador de la prioridad, en este caso la prioridad puede tomar valor como normal, high, medium. Cada una de estas prioridades tiene un identificador el cual se guarda en este campo de la base de datos.
author_id	Identificador del usuario que adicionó la petición
created_on	Fecha y hora en la que se creó la petición.
updtae_on	Fecha y hora que se actualizó la petición
start_date	Día que se inició la petición
estimates_hours	Horas estimadas que se necesitan para realizar la tarea.
status_id	Estado de la petición

Tabla 3. Funcionalidad de la tabla custom_values

Tabla custom_values: almacena los valores que van tomando los campos personalizados	
Atributos	**Función**
Id	Identificador de la tabla
customized_type	Tipo de campo personalizado
customized_id	Especifica el identificador para el valor que toma este campo personalizado
custom_field_id	Identificador del campo personalizado
value	Valor que toma el campo personalizado

Tabla 4. Funcionalidad de la tabla journals

Tabla journals: almacena la información de las actividades diarias de los usuarios en sus peticiones o tareas	
Atributos	**Función**
id	Identificador de la tabla
jounalized_id	Identificador de la actualización de la tarea
journalized_type	Este campo almacena que tipo de actividad realizó, en esta base de datos todos los tipos que se almacenan es issue, es decir actividades sobre las peticiones
user_id	Identificador del usuario el cual realizó algún cambio
notes	Campo que permite insertar una descripción
created_on	Fecha y hora en la que se creó

Tabla 5. Funcionalidad de la tabla journal_details

Tabla journal_details: almacena la información de las peticiones por cada usuario, es decir las actualizaciones que realizan los usuarios en cada una de sus tareas	
Atributos	**Función**
id	Identificador de la tabla
journal_id	Identificador de la tabla journals
property	Valor que toma la tarea, por ejemplo si es una actualización de porciento se refleja en este campo el valor: **attr**, si se sube un archivo o documento, el campo toma valor **attachment**
old_value	Valor anterior que tenia la tarea en el porciento de realizada
value_character	Nuevo valor que toma el porciento de la tarea

Tabla 6. Funcionalidad de la tabla changes

Tabla changes: almacena información referente a todos los cambios que se realicen en el Redmine	
Atributos	**Función**
id	Identificador de la tabla
changeset_id	Identificador de la tabla changeset
action	Guarda un valor según el tipo de acción que realiza el usuario en el Redmine, Si es actualización de porciento toma valor A, si es subida de un documento almacena D, si es creación de usuarios y grupos almacena M y por último si es cambios en las configuraciones de los usuarios o proyectos por algún administrador del sitio se almacena la letra R

2.4- Pruebas de rendimiento a la base de datos del Redmine antes de realizar el particionado

Se realizaron pruebas de rendimiento antes de proceder al particionado de las tablas con el fin de verificar los tiempos de respuesta que se obtienen y poder realizar una comparación una vez realizado el particionado y verificar que la técnica empleada disminuye los tiempos de respuesta a las consultas que se realizan a la base de datos.

El entorno de prueba utilizado:

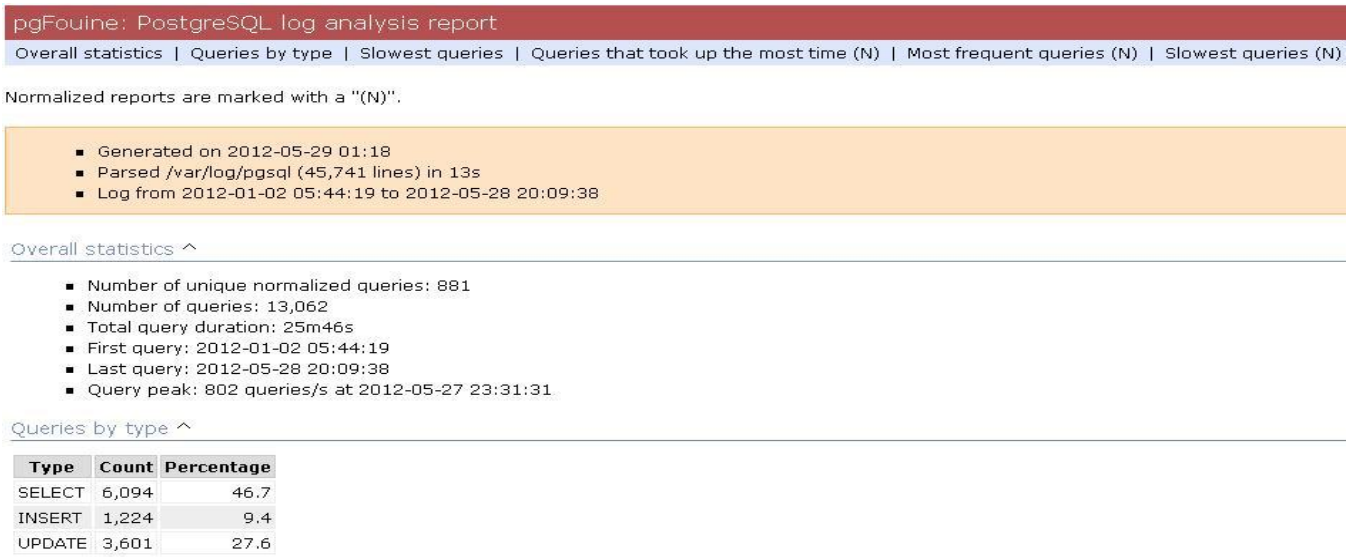
- ✓ PC de escritorio: Marca Haier
- ✓ Memoria RAM: 1 GB
- ✓ Microprocesador: Intel Pentium D
- ✓ Velocidad del microprocesador: 3.0 Mhz
- ✓ Tarjeta madre o motherboard: Modelo P5LD2
- ✓ Capacidad disco duro: 250 GB

Utilizando la herramienta EMS Data Generator 2005 para PostgreSQL se generaron datos a cada una de las tablas críticas seleccionadas (Ver Tabla 7).

Tabla 7. Cantidad de datos generados en las tablas críticas

Tablas críticas	Cantidad de datos generados
issues	1 000 000
custom_values	500 000
changes	2 615 799
journal_details	1 282 798
user	658 800
journal	500 000

Se realizaron diferentes consultas a todas las tablas que componen la base de datos. Se realizó una monitorización con la herramienta pgFouine (Ver Figura 5, 6, 7 y 8) dando como resultado que los tiempos más elevados obtenidos fueron a las consultas realizadas a las tablas críticas seleccionadas (Ver Tabla 8). En el Anexo 4 (Ver sección Anexos) se muestran tiempos de respuesta obtenidos a otras consultas que se realizaron a la base de datos del Redmine monitorizado con la herramienta pgFouine. Se evidencia que los tiempos a estas tablas no son relevantes por tanto no se escogieron como tablas críticas.


pgFouine: PostgreSQL log analysis report

Overall statistics | Queries by type | Slowest queries | Queries that took up the most time (N) | Most frequent queries (N) | Slowest queries (N)

Normalized reports are marked with a "(N)".

- Generated on 2012-05-29 01:18
- Parsed /var/log/pgsql (45,741 lines) in 13s
- Log from 2012-01-02 05:44:19 to 2012-05-28 20:09:38

Overall statistics

- Number of unique normalized queries: 881
- Number of queries: 13,062
- Total query duration: 25m46s
- First query: 2012-01-02 05:44:19
- Last query: 2012-05-28 20:09:38
- Query peak: 802 queries/s at 2012-05-27 23:31:31

Queries by type

Type	Count	Percentage
SELECT	6,094	46.7
INSERT	1,224	9.4
UPDATE	3,601	27.6

Figura 5. Cantidad de consultas realizadas a la base de datos

Figura 6. Monitorización a las tablas issues y changes

Figura 7. Monitorización a la tabla users

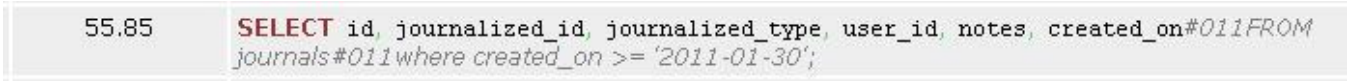


Figura 8. Monitorización a la tabla journals

Tabla 8. Tiempo de respuesta a las consultas que se realizan a las tablas críticas de la base de datos del Redmine antes de realizar el particionado.

Consulta	issues	users	journal	Journal_details	custom_values	changes
INSERT	0,18 s	0,13 s	0,06 s	0,24 s	0,08 s	0,02 s
DELETE	0,02 s	0,03 s	0,01 s	0,05 s	0,03 s	0,03 s
SELECT	50,15 s	20,01 s	56,67 s	5,29 s	3,95 s	10,23 s
UPDATE	0,31s	0,05 s	0,01 s	0,04 s	0,03 s	0,02 s

Como se evidencia en la tabla anterior, las consultas que más tiempo demoran en ejecutarse en todas las tablas seleccionadas son los SELECT. La Figura 9 representa las consultas SELECT contra el tiempo de respuesta que generaron.

Figura 9. Tiempo de respuesta a las consultas SELECT antes de realizar el particionado

2.5- Aplicación de la técnica de particionado

Luego de haber obtenido las tablas críticas se procede a realizar el particionado adecuado para cada una de las tablas. Como se explicaba en el capítulo anterior, el particionado soportado en PostgreSQL se aplica de dos maneras: por lista y por rango. Entre estas técnicas no existe una más eficiente que otra, sino que son implementadas en las tablas teniendo en cuenta las características individuales de cada una. En la Tabla 9 se muestra el tipo de particionado seleccionado y el atributo por el cual se realizará el particionado para cada una de las tablas críticas de la base de datos del Redmine.

Tabla 9. Tipo y atributo de particionado seleccionado para cada una de las tablas críticas

Tabla crítica	Tipo de particionado	Atributo seleccionado
issues	rango	created_on
custom_values	lista	customized_type
changes	lista	action
journal_details	lista	property

user	lista	id
journal	rango	created_on

Para las tablas custom_values, changes, journal_details y user se decide aplicar el particionado por lista debido a que los atributos seleccionados cumplen con los requerimientos para esta técnica. Cada uno de ellos toma valores específicos para cada tupla, permitiendo dividir la tabla padre en una cantidad de tablas hijas menor que 100, características que debe cumplirse para poder realizar el particionado por el atributo especificado.

Ejemplo del particionado por lista en la tabla **custom_values**

Paso 1: Verificar que en el archivo de configuración de PostgreSQL: postgresql.conf el parámetro constrain_exclusion esté activado, es decir constrain_exclusion = on.

Paso 2: Para realizar el particionado la tabla padre debe estar vacía. Si la tabla tuviera datos, como es el caso de este ejemplo, se procede a crear una tabla temporal que almacenará los datos de la tabla padre, evitando la pérdida de los mismos y luego se realiza un **truncate** a la tabla para dejarla completamente vacía.

create table temporal1 (like custom_values);
*insert into temporal1 select * from custom_values;*
truncate table custom_values;

Paso 3: Luego se crean las tablas que heredarán de la tabla padre (tablas hijas) con las restricciones (check) en el atributo por el cual se va a realizar el particionado. Se debe tener en cuenta que al utilizar el particionado por lista, como en este caso, se crearán tantas tablas hijas como valores que pueda tomar el atributo seleccionado.

create table cv_project (check (customized_type=Project)) inherits (custom_values);
create table cv_principal (check (customized_type=Principal)) inherits (custom_values);
create table cv_issue (check (customized_type=Issue)) inherits (custom_values);

Paso 4: Se crean los índices, necesarios para la optimización de las consulta en cada una de las particiones creadas.

create index cv_project_idx on cv_project (customized_type);
create index cv_principal_idx on cv_principal (customized_type);
create index cv_issue_idx on cv_issue (customized_type);

Paso 5: Se procede a crear la función que se encargará de distribuir los valores de la tabla padre hacia las hijas según las condiciones especificadas.

```
create or replace function particionar1 ()
returns trigger as $$
begin
if (new.customized_type='Project') then insert into cv_project values (new.*);
elseif (new.customized_type='Principal') then insert into cv_principal values (new.*);
elseif (new.customized_type='Issue') then insert into cv_issue values (new.*);
end if;
return null;
end
$$
```

Paso 6: Se especifica el lenguaje de ejecución de la función.

language plpgsql;;;

Paso 7: Se crea el disparador que ejecutará la función anteriormente implementada cada vez que se realice una acción sobre esta tabla.

create trigger part_custom_values before insert on custom_values
for each row execute procedure particionar1 ();

Paso 8: Se restauran los valores que inicialmente tenía la tabla padre guardados en la tabla *temporal1*.

*insert into custom_values select * from temporal1;*

Paso 9: Se borra la tabla *temporal1*

drop table temporal1;

Para las tablas issues y journals se decide aplicar el particionado por rango debido a que no contienen atributos factibles para el particionado por lista. Sin embargo ambas tienen un atributo el cual almacena valores de tipo *timestamp* o fecha, candidato para realizar el particionado basándose en él.

A continuación se muestra un ejemplo de cómo se realizó el particionado por rango en la tabla issues. Para la implementación de este tipo de particionado se deben ejecutar los pasos descritos anteriormente, solo modificando en el Paso 2 las restricciones (cheks) del atributo por el cual se va a realizar el particionado y en el Paso 3 las condiciones de inserción en las tablas hijas.

Paso 1: Verificar que en el archivo de configuración de POstgreSQL: postgresql.conf el parámetro constrain_exclusion esté activado, es decir constrain_exclusion = on.

Paso 2: Para realizar el particionado la tabla padre debe estar vacía. Si la tabla tuviera datos, como es el caso de este ejemplo, se procede a crear una tabla temporal que almacenará los datos de la tabla padre, evitando la pérdida de los mismos y luego se realiza un **truncate** a la tabla para dejarla completamente vacía.

create table temporal (like issues);
*insert into temporal select * from issues;*
truncate table issues;

Paso 3: Luego se crean las tablas que heredarán de la tabla padre (tablas hijas) con las restricciones (check) en el atributo por el cual se va a realizar el particionado. Se debe tener en cuenta que al utilizar el particionado por rango, como en este caso, se crearán tantas tablas hijas como la cantidad de rango de valores que pueda tomar el atributo seleccionado. En este caso se crearon particiones para almacenar las peticiones teniendo en cuenta la fecha de creación de la tarea, de esta manera quedarán almacenadas por meses todas las peticiones que se inserten en el Redmine.

create table issues_sep (check (created_on BETWEEN '2011-09-01' AND '2011-09-30')) inherits (issues);
create table issues_oct (check (created_on BETWEEN '2011-10-01' AND '2011-09-31')) inherits (issues);
create table issues_nov (check (created_on BETWEEN '2011-11-01' AND '2011-11-30')) inherits (issues);
create table issues_dic (check (created_on BETWEEN '2011-12-01' AND '2011-12-31')) inherits (issues);
create table issues_ene (check (created_on BETWEEN '2012-01-01' AND '2012-01-31')) inherits (issues);

create table issues_feb (check (created_on BETWEEN '2012-02-01' AND '2012-02-28')) inherits (issues);
create table issues_mar (check (created_on BETWEEN '2012-03-01' AND '2012-03-31')) inherits (issues);
create table issues_abr (check (created_on BETWEEN '2012-04-01' AND '2012-04-30')) inherits (issues);
create table issues_may (check (created_on BETWEEN '2012-05-01' AND '2012-05-31')) inherits (issues);
create table issues_jun (check (created_on BETWEEN '2012-06-01' AND '2012-06-30')) inherits (issues);
create table issues_jul (check (created_on BETWEEN '2012-07-01' AND '2012-07-31')) inherits (issues);
create table issues_ago (check (created_on BETWEEN '2012-08-01' AND '2012-08-31')) inherits (issues);

Paso 4: Se crean los índices, necesarios para la optimización de las consulta en cada una de las particiones creadas.

create index issues_sep_idx on issues_sep (created_on);
create index issues_oct_idx on issues_oct (created_on);
create index issues_nov_idx on issues_nov (created_on);
create index issues_dic_idx on issues_dic (created_on);
create index issues_ene_idx on issues_ene (created_on);
create index issues_feb_idx on issues_feb (created_on);
create index issues_mar_idx on issues_mar (created_on);
create index issues_abr_idx on issues_abr (created_on);
create index issues_may_idx on issues_may (created_on);
create index issues_jun_idx on issues_jun (created_on);
create index issues_jul_idx on issues_jul (created_on);
create index issues_ago_idx on issues_ago (created_on);

Paso 5: Se procede a crear la función que se encargará de distribuir los valores de la tabla padre hacia las hijas según las condiciones especificadas.

```
create or replace function particionar ()
returns trigger as $$
begin
if (new.created_on  BETWEEN '2011-09-01' AND '2011-09-30') then insert into issues_sep values
(new.*);
elseif (new.created_on  BETWEEN '2011-10-01' AND '2011-09-31') then insert into issues_oct values
(new.*);
elseif (new.created_on  BETWEEN '2011-11-01' AND '2011-11-30') then insert into issues_nov values
(new.*);
elseif (new.created_on  BETWEEN '2011-12-01' AND '2011-12-31') then insert into issues_dic values
(new.*);
elseif (new.created_on  BETWEEN '2012-01-01' AND '2012-01-31') then insert into issues_ene values
(new.*);
elseif (new.created_on  BETWEEN '2012-02-01' AND '2012-02-28') then insert into issues_feb values
(new.*);
elseif (new.created_on  BETWEEN '2012-03-01' AND '2012-03-31') then insert into issues_mar values
(new.*);
elseif (new.created_on  BETWEEN '2012-04-01' AND '2012-04-30') then insert into issues_abr values
(new.*);
elseif (new.created_on  BETWEEN '2012-05-01' AND '2012-05-31') then insert into issues_may values
(new.*);
elseif (new.created_on  BETWEEN '2012-06-01' AND '2012-06-30') then insert into issues_jun values
(new.*);
elseif (new.created_on  BETWEEN '2012-07-01' AND '2012-07-31') then insert into issues_jul values
(new.*);
elseif (new.created_on  BETWEEN '2012-08-01' AND '2012-08-31') then insert into issues_ago values
(new.*);
end if;
return null;
end
$$
```

Paso 6: Se especifica el lenguaje de ejecución de la función.

```
language plpgsql;;;
```

Paso 7: Se crea el disparador que ejecutará la función anteriormente implementada cada vez que se realice una acción sobre esta tabla.

create trigger particionar_issues before insert on issues
for each row execute procedure particionar ();

Paso 8: Se restauran los valores que inicialmente tenía la tabla padre guardados en la tabla *temporal*.
*insert into issues select * from temporal;*

Paso 9: Se borra la tabla *temporal*
drop table temporal;

Conclusiones del capítulo

En este capítulo se obtuvo el modelo de datos de la herramienta de gestión de proyectos Redmine. Se seleccionaron las tablas críticas de la base de datos: changes, custom_values, journals, issues, journal_details y users. Se realizaron diferentes consultas a todas las tablas de la base de datos con el objetivo de obtener los tiempos de respuesta a las mismas. Se especificaron las funcionalidades que tienen las tablas críticas seleccionadas y cada uno de los atributos que la componen dentro de la base de datos. Se seleccionó la técnica de particionado para cada una de las tablas críticas según sus características y el atributo de partición. Se implementaron las técnicas de particionado por lista y por rango para cada una de las tablas críticas.

Capítulo III: Pruebas de rendimiento

Introducción del capítulo

El presente capítulo presenta las pruebas realizadas una vez realizado el particionado de las tablas críticas seleccionadas de la base de datos del Redmine. Se confeccionaron gráficas y tablas que demuestran los tiempos de respuestas obtenidos de las consultas realizadas. Además se expone una comparación entre tiempos de respuesta antes y después de ser particionadas las tablas para validar que las técnicas implementadas en el transcurso de la investigación, es la vía para mejorar los tiempos de respuestas a las peticiones que se realizan a la base de datos del Redmine.

3.1- Pruebas de rendimiento después de realizar el particionado

Luego de haber realizado el particionado de las tablas seleccionadas se ejecutaron las mismas consultas realizadas en el capítulo anterior a las tablas críticas, con un total de datos similar. La Tabla 10 muestra los resultados obtenidos con el monitoreo.

Tabla 10. Tiempo de respuesta a las consultas que se realizan a las tablas críticas de la base de datos del Redmine después de realizar el particionado.

	issues	users	journals	journal details	custom values	changes
INSERT	0,09 s	0,04 s	0,01 s	0,24 s	0,08 s	0,02 s
DELETE	0,02 s	0,02 s	0,01 s	0,05 s	0,03 s	0,03 s
SELECT	48,32 s	18,08 s	55,08 s	2,65 s	3,81 s	9,83 s
UPDATE	0,31 s	0,02 s	0,01 s	0,04 s	0,03 s	0,02 s

Como se evidencia en la tabla anterior surgieron cambios en los tiempos de respuesta a las consultas que se realizaron sobre las tablas. La Figura 10 representa las consultas SELECT contra los tiempos de respuesta generados luego de haber particionado las tablas críticas.

Figura 10. Tiempo de respuesta a las consultas SELECT luego de realizar el particionado

3.2- Comparación de las pruebas realizadas

A raíz de las pruebas realizadas en los epígrafes anteriores, cuando se realiza el particionado de las tablas críticas se observa una mejora en el tiempo de respuesta a las operaciones que se realizan sobre ellas, principalmente en las consultas SELECT.

La Figura 11 representa los tiempos de respuestas de las consultas SELECT ejecutadas antes y después de realizar el particionado separadas por cada tabla. Se evidencia que las tablas particionadas disminuyen el tiempo de respuesta a las peticiones que se realizan sobre ellas.

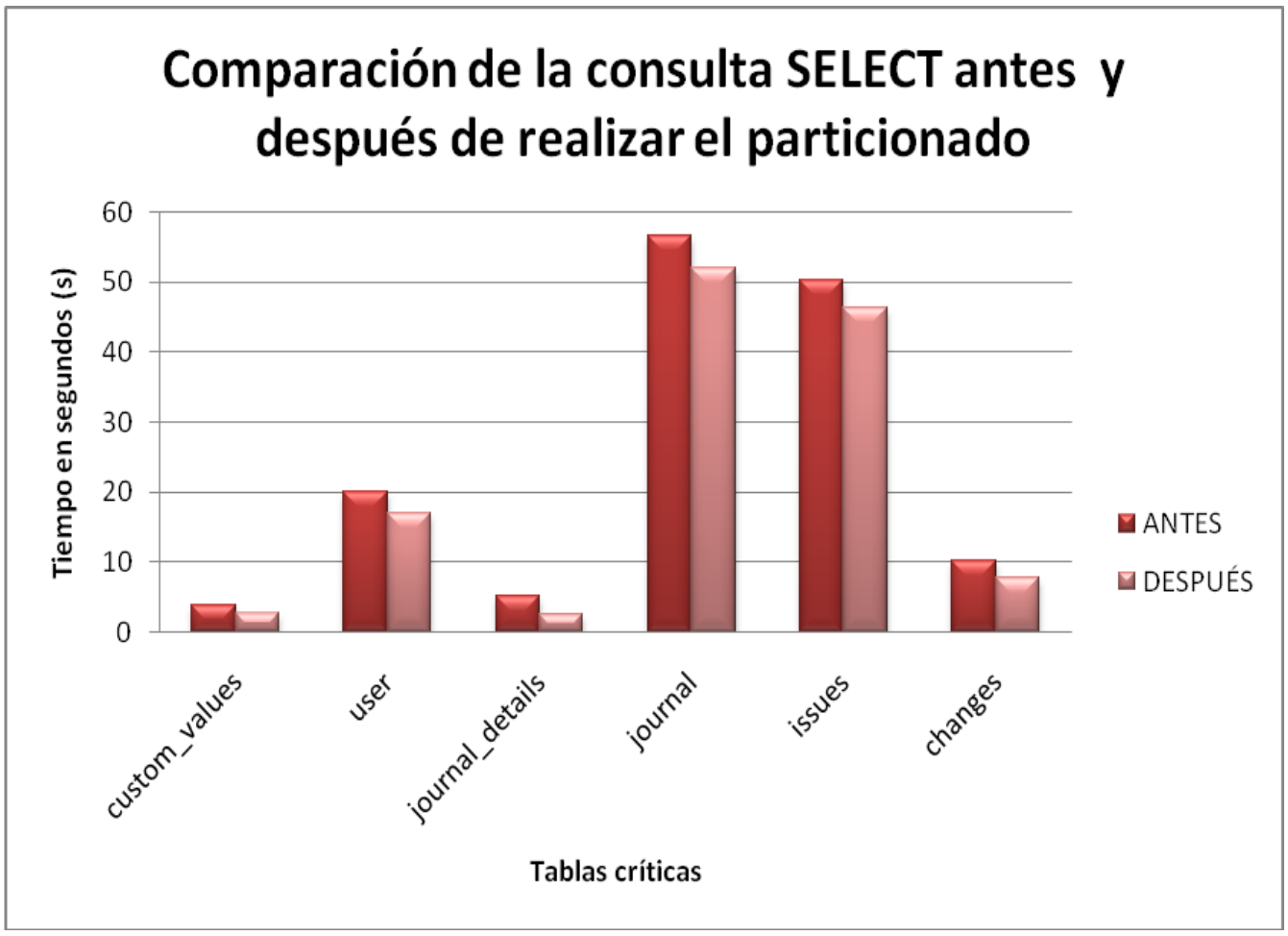


Figura 11. Comparación de las pruebas realizadas antes y después de realizar el particionado de las tablas críticas

Para demostrar de manera general el rendimiento de la base de datos del Redmine teniendo en cuenta los datos obtenidos en las pruebas realizadas, se muestran en las Figura 12, 13, 14 y 15, el rendimiento antes y después de ser particionadas las tablas críticas divididos por tipos de consultas.

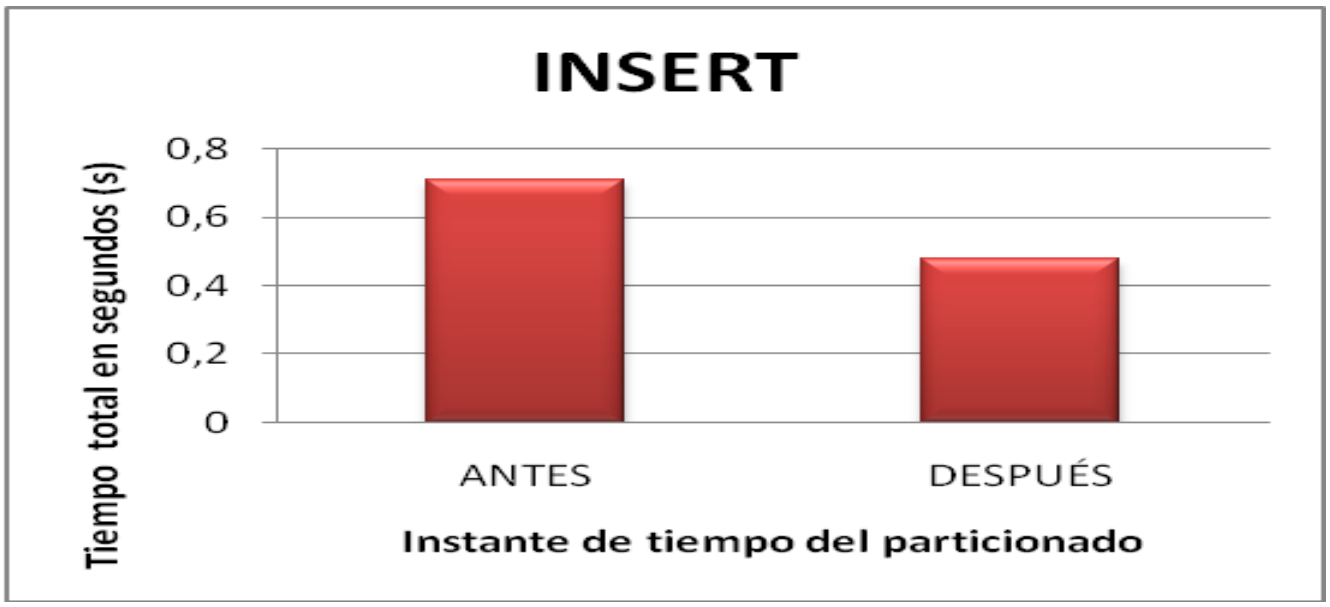


Figura 12. Comportamiento de los tiempos obtenidos a las consultas INSERT realizadas antes y después del particionado de las tablas críticas

Figura 13. Comportamiento de los tiempos obtenidos a las consultas DELETE realizadas antes y después del particionado de las tablas críticas

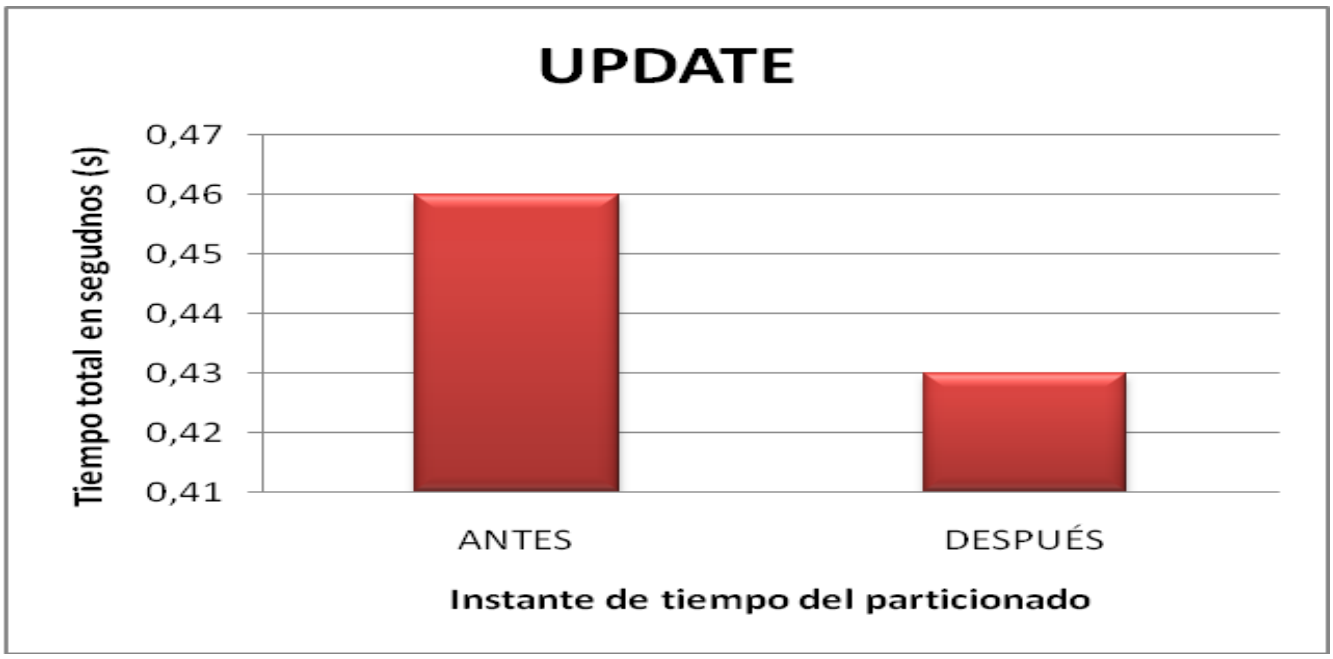


Figura 14. Comportamiento de los tiempos obtenidos a las consultas DELETE realizadas antes y después del particionado de las tablas críticas

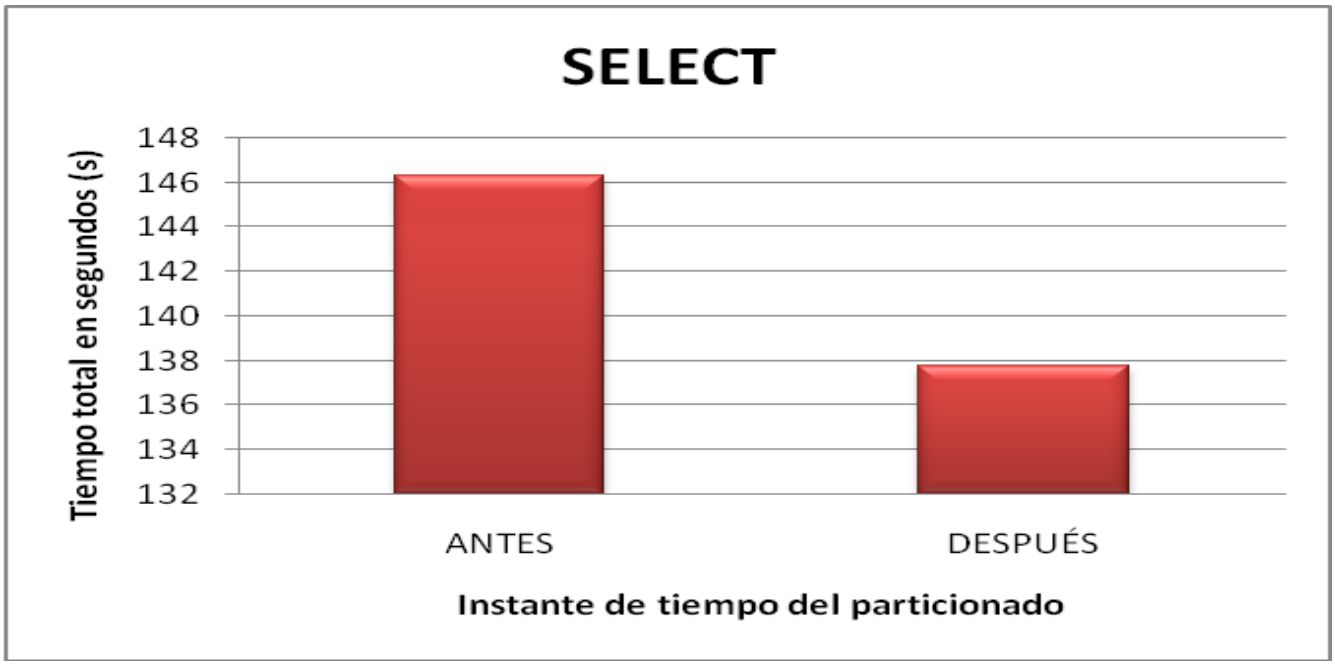


Figura 15. Comportamiento de los tiempos obtenidos a las consultas SELECT realizadas antes y después del particionado de las tablas críticas

Se evidencia en las gráficas anteriores, que en la base de datos del Redmine se obtuvo un mejor rendimiento cuando se implementaron técnicas de particionado a las tablas críticas que se identificaron.

Conclusiones del capítulo

Se realizaron las consultas a las tablas críticas seleccionadas después de realizar el particionado. Se obtuvieron los reportes de monitorización con la herramienta pgFouine. Se realizó una comparación entre los tiempos obtenidos antes de realizar el particionado con los tiempos obtenidos después de aplicar el particionado en las tablas críticas seleccionadas.

Conclusiones Generales

Luego de haber concluido el desarrollo de la investigación se obtuvo como resultado

- ✓ El Modelo de datos del Redmine y catálogo que refleja las características y funciones de las tablas y los atributos que componen la base de datos del Redmine.
- ✓ Se identificaron seis tablas críticas de la base de datos del Redmine: changes, custom_values, journals, issues, journal_details y users, a las cuales se les aplicó las técnicas de particionado del Sistema Gestor de Base de Datos PostgreSQL por lista y por rango.
- ✓ Se realizaron las pruebas de rendimiento a la base de datos del Redmine antes y después de ser particionada, evidenciando la disminución de tiempo en las consultas SELECT, INSERT, UPDATE y DELETE, contribuyendo al mejoramiento de los tiempos de respuestas de estas consultas que permitirán una mayor satisfacción del usuario al ejecutarlas.

Recomendaciones

Se recomienda

- ✓ Aplicar pruebas de rendimiento al servidor real del Redmine en la UCI para obtener los tiempos de respuesta que se obtienen cuando se realizan consultas a la base de datos.
- ✓ Aplicar técnicas de particionado de tablas a la base de datos del Redmine real de la UCI para minimizar los tiempos de respuesta a las consultas que se realizan a la base de datos.

Referencias Bibliográficas

1. Buenas Tareas. [En línea]. [Citado el: 10 de 4 de 2012.] http://www.buenastareas.com/ensayos/Sistemas-De-Bd/1598603.html.

2. **García, Maríaa J Somodevilla.** [En línea] 2011. [Citado el: 4 de 5 de 2012.] https://docs.google.com/viewer?a=v&q=cache:Ty-VLCnSKEoJ:www.cs.buap.mx/~mariasg/bases/Audiovisuales/Introduccion.ppt+Una+base+de+datos+es+un+conjunto+de+datos+almacenados+entre+los+que+existen+relaciones+l%C3%B3gicas+y+ha+sido+dise%C3%B1ada+para+satisface.

3. **Valdés, Damián Pérez.** Maestros del web. [En línea] [Citado el: 12 de 9 de 2011.] http://www.maestrodelweb.com/principiantes/¿que-son-las-bases-de-datos/. 2.

4. Desarrolloweb.com. [En línea] [Citado el: 24 de 05 de 2012.] http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

5. **Elmasri, Ramez y Navathe, Shamkant B.***Sistemas de Bases de Datos Conceptos fundamentales.* Mexico DF : s.n.

6. **Officce, Microsoft.** Microsoft Office. [En línea] http://office.microsoft.com/es-hn/access-help/conceptos-basicos-del-diseno-de-una-base-de-datos-HA001224247.aspx.

7. Miedo al miedo. [En línea]. [Citado el: 28 de 04 de 2012.] http://miedoalmiedo.com3.tv/definicion-de-modelo-de-datos-qu-es-significado-y-concepto/.

8. Ceslcam Centro de Excelencia de Software libre. [En línea] 7 de 12 de 2010. [Citado el: 25 de 10 de 2011.] http://www.ceslcam.com/analisis-de-aplicaciones/analisis-de-aplicacion-redmine.html.

9. **Santos, Rubén Dávila.** maestros del web. [En línea] [Citado el: 15 de 9 de 2011.] http://www.maestrosdelweb.com/editorial/rubyonrails/.

10. **Martínez, Gilberto Castillo.** PostgreSQL. [En línea] 19 de 1 de 2010. [Citado el: 5 de 11 de 2010.] http://archives.postgresql.org/pgsql-es-ayuda/2010-01/msg00234.php.

11. **Rosayda Valiente Mesa, Idelsis Castillo Pérez.***LIMS de calidad del Centro de Ingeniería Genética y Biotecnología: Desarrollo de la Base de Datos del módulo Sección de Mejoramiento de la Calidad. Ciudad de La Habana : s.n., 2008.* Ciudad de La Habana : s.n., 2008.

12. **Fundation, The Apache Software.** ABCdatos. [En línea] 20 de 10 de 2010. [Citado el: 13 de 12 de 2010.] http://www.abcdatos.com/webmasters/programa/z2818.html..

13. **SIU, Consorcio.***Monitoreando y midiendo el rendimiento de un servidor Postgresql en Sistemas Windows, Linux y Solaris.*

14. *HERRAMIENTA PARA REALIZAR EL PARTICIONADO DE TABLAS EN BASE.* **Rosabal Alarcon, Maikel Fernando y Rosales Garcia, Adonis Ricardo.** La Habana : s.n., 2011.

15. www.ora-600.es. [En línea] [Citado el: 28 de 5 de 2012.] http://www.ora-600.es.

Bibliografía

1. Buenas Tareas. [En línea]. [Citado el: 10 de 4 de 2012.] http://www.buenastareas.com/ensayos/Sistemas-De-Bd/1598603.html.

2. **García, Maríaa J Somodevilla.** [En línea] 2011. [Citado el: 4 de 5 de 2012.] https://docs.google.com/viewer?a=v&q=cache:Ty-VLCnSKEoJ:www.cs.buap.mx/~mariasg/bases/Audiovisuales/Introduccion.ppt+Una+base+de+datos+es+un+conjunto+de+datos+almacenados+entre+los+que+existen+relaciones+l%C3%B3gicas+y+ha+sido+dise%C3%B1ada+para+satisface.

3. **Valdés, Damián Pérez.** Maestros del web. [En línea] [Citado el: 12 de 9 de 2011.] http://www.maestrodelweb.com/principiantes/¿que-son-las-bases-de-datos/. 2.

4. Desarrolloweb.com. [En línea] [Citado el: 24 de 05 de 2012.] http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

5. **Elmasri, Ramez y Navathe, Shamkant B.***Sistemas de Bases de Datos Conceptos fundamentales.* Mexico DF : s.n.

6. **Officce, Microsoft.** Microsoft Office. [En línea] http://office.microsoft.com/es-hn/access-help/conceptos-basicos-del-diseno-de-una-base-de-datos-HA001224247.aspx.

7. Miedo al miedo. [En línea]. [Citado el: 28 de 04 de 2012.] http://miedoalmiedo.com3.tv/definicion-de-modelo-de-datos-qu-es-significado-y-concepto/.

8. Ceslcam Centro de Excelencia de Software libre. [En línea] 7 de 12 de 2010. [Citado el: 25 de 10 de 2011.] http://www.ceslcam.com/analisis-de-aplicaciones/analisis-de-aplicacion-redmine.html.

9. **Santos, Rubén Dávila.** maestros del web. [En línea] [Citado el: 15 de 2 de 2012.] http://www.maestrosdelweb.com/editorial/rubyonrails/.

10. **Martínez, Gilberto Castillo.** PostgreSQL. [En línea] 19 de 1 de 2010. [Citado el: 5 de 11 de 2011.] http://archives.postgresql.org/pgsql-es-ayuda/2010-01/msg00234.php.

11. **Rosayda Valiente Mesa, Idelsis Castillo Pérez.***LIMS de calidad del Centro de Ingeniería Genética y Biotecnología: Desarrollo de la Base de Datos del módulo Sección de Mejoramiento de la Calidad. Ciudad de La Habana : s.n., 2008.* Ciudad de La Habana : s.n., 2008.

12. **Fundation, The Apache Software.** ABCdatos. [En línea] 20 de 10 de 2010. [Citado el: 13 de 12 de 2011.] http://www.abcdatos.com/webmasters/programa/z2818.html..

13. **SIU, Consorcio.***Monitoreando y midiendo el rendimiento de un servidor Postgresql en Sistemas Windows, Linux y Solaris.*

14. *HERRAMIENTA PARA REALIZAR EL PARTICIONADO DE TABLAS EN BASE.* **Rosabal Alarcon, Maikel Fernando y Rosales Garcia, Adonis Ricardo.** La Habana : s.n., 2011.

15. www.ora-600.es. [En línea] [Citado el: 28 de 5 de 2012.] http://www.ora-600.es.

16. **Sicilia, Miguel Angel.** connexions. [En línea] 25 de 11 de 2008. [Citado el: 2 de 3 de 2001.] http://cnx.org/content/m17436/latest/.

17. **Chachin, Carlos.** DosIdeas. [En línea] 13 de 5 de 2009. [Citado el: 21 de 10 de 2010.] http://www.dosideas.com/noticias/base-de-datos/576-incrementar-rendimiento-de-bbdd-con-qparticionado-de-tablasq.html.

18. Ingeniería de Software 1. [En línea] 14 de 4 de 2010. [Citado el: 5 de 10 de 2010.] http://www.sribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE.

19. [En línea] [Citado el: 25 de 2 de 2011.] http://softbb.org/es/722314.aspx.

20. CANSI. [En línea] [Citado el: 5 de 1 de 2011.] http://www.cavsi.com/preguntasrespuestas/que-es-un-sistema-gestor-de-bases-de-datos-o-sgbd/.

21. ICT. [En línea] 2008. [Citado el: 15 de 1 de 2011.] http://ict.udlap.mx/people/carlos/is341/bases02.html.

22. Molinux. [En línea] [Citado el: 15 de 10 de 2010.] http://comunidad.molinux.info/index.php/Redmine.

23. Office Microsoft. [En línea] 2010. [Citado el: 1 de 1 de 2010.] http://office.microsoft.com/es-es/access-help/conceptos-basicos-del-diseno-de-una-base-de-datos-HA001224247.aspx..

24. The::Beastieux. [En línea] 27 de 11 de 2009. [Citado el: 20 de 11 de 2010.] http://saforas.wordpress.com/2009/11/27/postgresql-particionado-de-tablas/.

25. **Jhen.** ManualesdeAyuda.com. [En línea] 23 de 9 de 2006. [Citado el: 20 de 2 de 2011.]

26. **Nero, Margarita.** Metodología del diseño de base de datos Capítulo 15. [En línea] [Citado el: 18 de 11 de 2010.] http://www.slideshare.net/emnero2/diseo-de-base-de-datos-360943.

27. **Nojera Espino, Edwin Omar.** Practica Extra Ruby on Rails. [En línea] 15 de 5 de 2009. [Citado el: 21 de 10 de 2010.] http://www.scribd.com/doc/15485640/Ruby-on-Rails#autor_page_3..

28. **respinosa.** DATA PRIX. [En línea] 25 de 2 de 2010. [Citado el: 8 de 2 de 2011.] http://www.dataprix.com/blogs/respeinosamilla/tabla-hechos-venta-particionado-mysql.

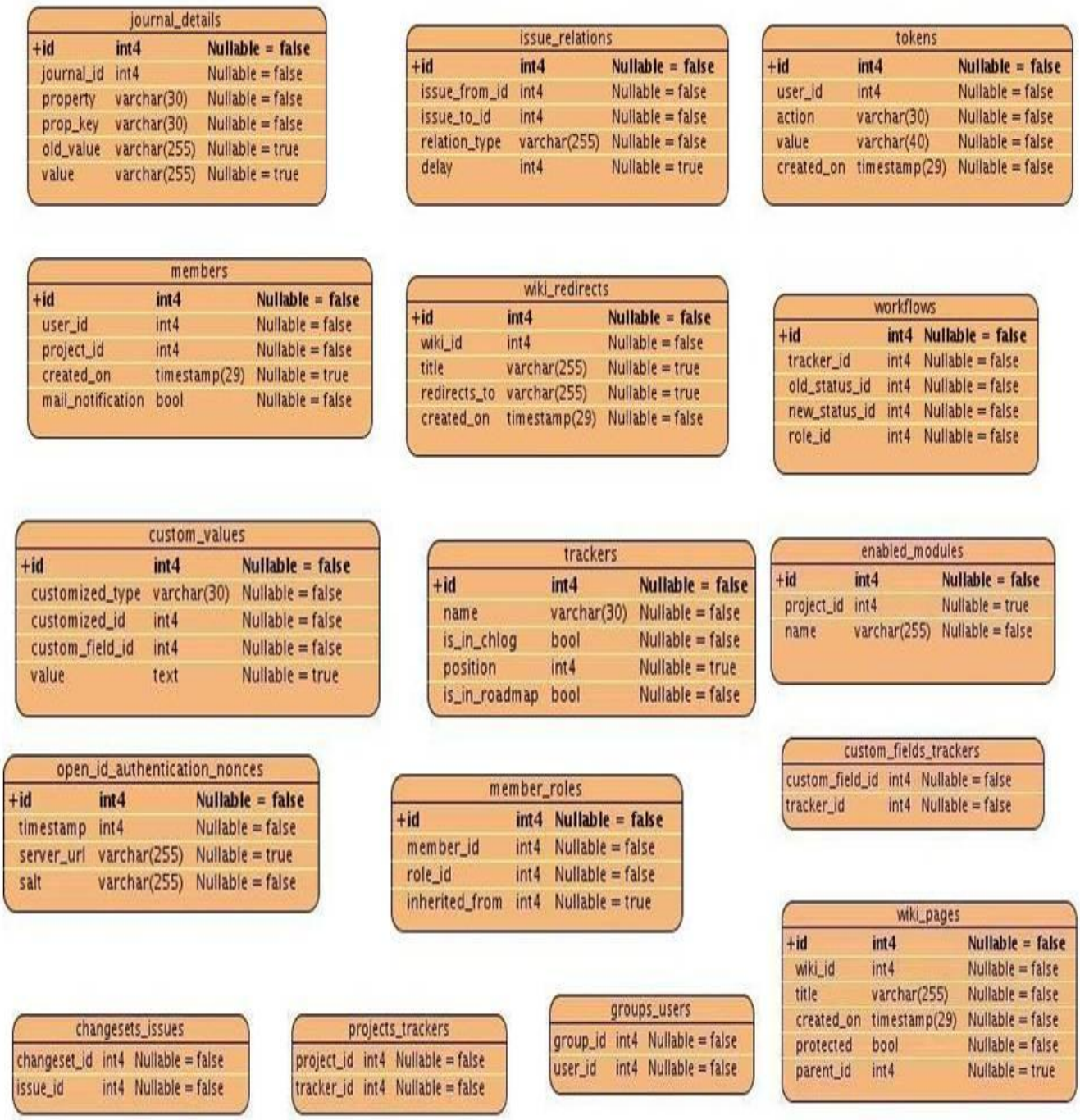
29. **respinosamilla.** DATA PRIX. [En línea] 24 de 4 de 2010. [Citado el: 10 de 2 de 2011.] http://www.dataprix.com/blogs/respinosamilla/particionado_tablas_oracle.

30. **Software, Sitio de descargas de.** Free Download Manager. [En línea] 5 de 3 de 2007. [Citado el: 26 de 1 de 2011.] http://www.freedownloadmanager.org/es/downloads/Paradim_Visual_para_UML.

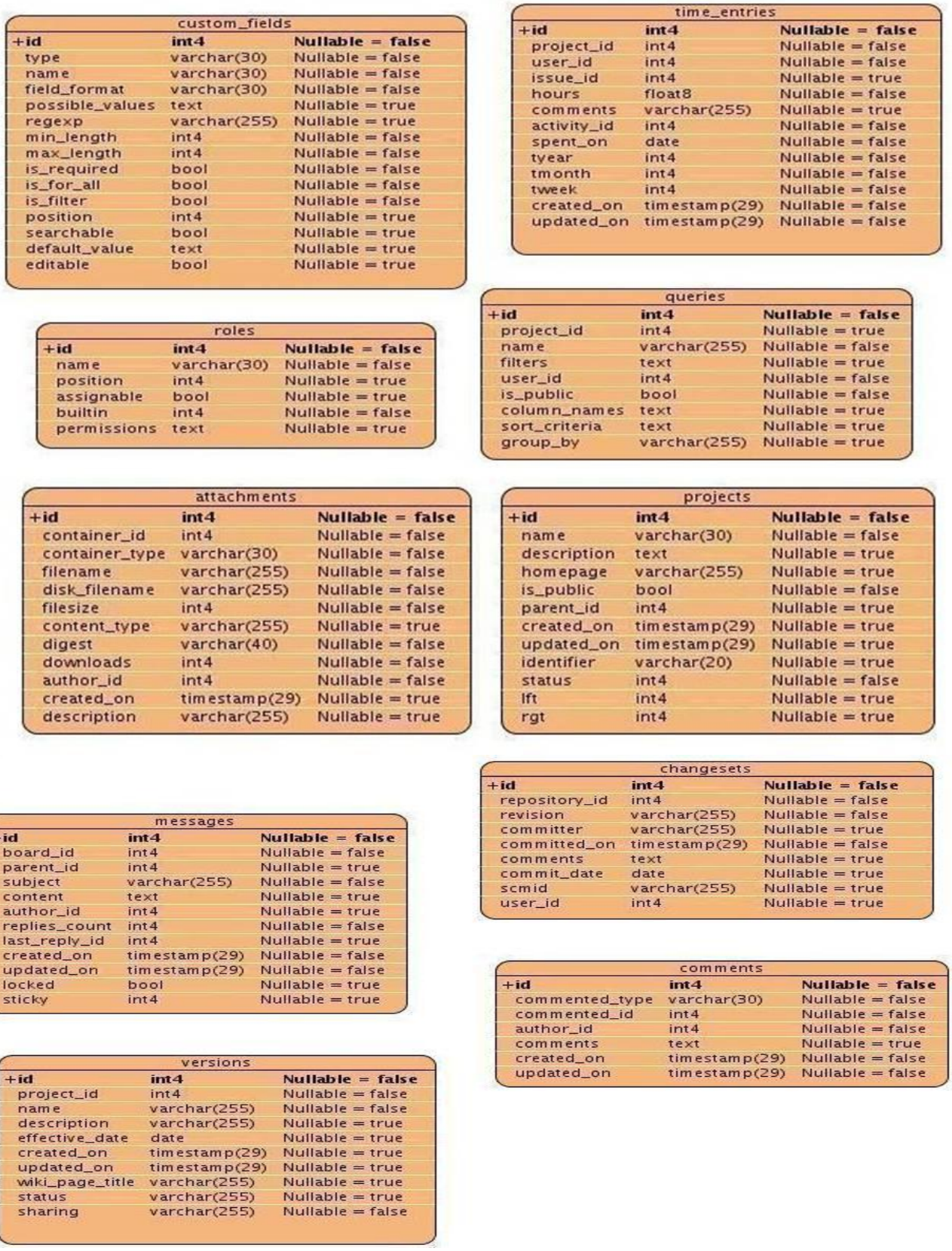
31. SQLManager.net. [En línea] [Citado el: 8 de abril de 2011.] http://www.sqlmanager.net/products/postgresql/datagenerator.

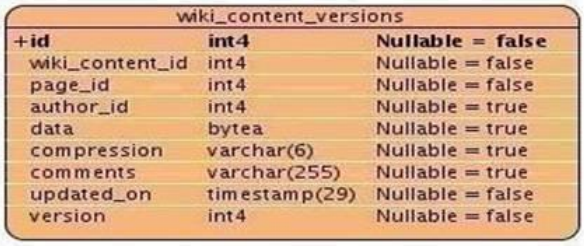
Anexos

Anexo 1. Modelo de datos del Redmine

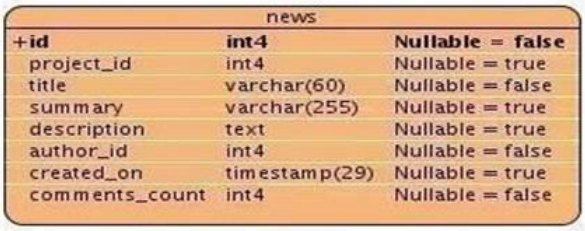



watchers
+id int4 Nullable = false
watchable_type varchar(255) Nullable = false
watchable_id int4 Nullable = false
user_id int4 Nullable = true
wikis
+id int4 Nullable = false
project_id int4 Nullable = false
start_page varchar(255) Nullable = false
status int4 Nullable = false
enumerations
+id int4 Nullable = false
name varchar(30) Nullable = false
position int4 Nullable = true
is_default bool Nullable = false
type varchar(255) Nullable = true
active bool Nullable = false
project_id int4 Nullable = true
parent_id int4 Nullable = true
user_preferences
+id int4 Nullable = false
user_id int4 Nullable = false
others text Nullable = true
hide_mail bool Nullable = true
time_zone varchar(255) Nullable = true
custom_fields_projects
custom_field_id int4 Nullable = false
project_id int4 Nullable = false
open_id_authentication_associations
+id int4 Nullable = false
issued int4 Nullable = true
lifetime int4 Nullable = true
handle varchar(255) Nullable = true
assoc_type varchar(255) Nullable = true
server_url bytea Nullable = true
secret bytea Nullable = true
repositories
+id int4 Nullable = false
project_id int4 Nullable = false
url varchar(255) Nullable = false
login varchar(60) Nullable = true
password varchar(60) Nullable = true
root_url varchar(255) Nullable = true
type varchar(255) Nullable = true
schema_migrations
version varchar(255) Nullable = false
issue_categories
+id int4 Nullable = false
project_id int4 Nullable = false
name varchar(30) Nullable = false
assigned_to_id int4 Nullable = true
issues
+id int4 Nullable = false
tracker_id int4 Nullable = false
project_id int4 Nullable = false
subject varchar(255) Nullable = false
description text Nullable = true
due_date date Nullable = true
category_id int4 Nullable = true
status_id int4 Nullable = false
assigned_to_id int4 Nullable = true
priority_id int4 Nullable = false
fixed_version_id int4 Nullable = true
author_id int4 Nullable = false
lock_version int4 Nullable = false
created_on timestamp(29) Nullable = true
updated_on timestamp(29) Nullable = true
start_date date Nullable = true
done_ratio int4 Nullable = false
estimated_hours float8 Nullable = true
settings
+id int4 Nullable = false
name varchar(255) Nullable = false
value text Nullable = true
updated_on timestamp(29) Nullable = true
users
+id int4 Nullable = false
login varchar(30) Nullable = false
hashed_password varchar(40) Nullable = false
firstname varchar(30) Nullable = false
lastname varchar(30) Nullable = false
mail varchar(60) Nullable = false
mail_notification bool Nullable = false
admin bool Nullable = false
status int4 Nullable = false
last_login_on timestamp(29) Nullable = true
language varchar(5) Nullable = true
auth_source_id int4 Nullable = true
created_on timestamp(29) Nullable = true
updated_on timestamp(29) Nullable = true
type varchar(255) Nullable = true
identity_url varchar(255) Nullable = true
auth_sources
+id int4 Nullable = false
type varchar(30) Nullable = false
name varchar(60) Nullable = false
host varchar(60) Nullable = true
port int4 Nullable = true
account varchar(255) Nullable = true
account_password varchar(60) Nullable = true
base_dn varchar(255) Nullable = true
attr_login varchar(30) Nullable = true
attr_firstname varchar(30) Nullable = true
attr_lastname varchar(30) Nullable = true
attr_mail varchar(30) Nullable = true
onthefly_register bool Nullable = false
tls bool Nullable = false
wiki_contents
+id int4 Nullable = false
page_id int4 Nullable = false
author_id int4 Nullable = true
text text Nullable = true
comments varchar(255) Nullable = true
updated_on timestamp(29) Nullable = false
version int4 Nullable = false
journals
+id int4 Nullable = false
journalized_id int4 Nullable = false
journalized_type varchar(30) Nullable = false
user_id int4 Nullable = false
notes text Nullable = true
created_on timestamp(29) Nullable = false
documents
+id int4 Nullable = false
project_id int4 Nullable = false
category_id int4 Nullable = false
title varchar(60) Nullable = false
description text Nullable = true
created_on timestamp(29) Nullable = true
issue_statuses
+id int4 Nullable = false
name varchar(30) Nullable = false
is_closed bool Nullable = false
is_default bool Nullable = false
position int4 Nullable = true
default_done_ratio int4 Nullable = true


custom_fields
+id int4 Nullable = false
type varchar(30) Nullable = false
name varchar(30) Nullable = false
field_format varchar(30) Nullable = false
possible_values text Nullable = true
regexp varchar(255) Nullable = true
min_length int4 Nullable = false
max_length int4 Nullable = false
is_required bool Nullable = false
is_for_all bool Nullable = false
is_filter bool Nullable = false
position int4 Nullable = true
searchable bool Nullable = true
default_value text Nullable = true
editable bool Nullable = true
time_entries
+id int4 Nullable = false
project_id int4 Nullable = false
user_id int4 Nullable = false
issue_id int4 Nullable = true
hours float8 Nullable = false
comments varchar(255) Nullable = true
activity_id int4 Nullable = false
spent_on date Nullable = false
tyear int4 Nullable = false
tmonth int4 Nullable = false
tweek int4 Nullable = false
created_on timestamp(29) Nullable = false
updated_on timestamp(29) Nullable = false
queries
+id int4 Nullable = false
project_id int4 Nullable = true
name varchar(255) Nullable = false
filters text Nullable = true
user_id int4 Nullable = false
is_public bool Nullable = false
column_names text Nullable = true
sort_criteria text Nullable = true
group_by varchar(255) Nullable = true
roles
+id int4 Nullable = false
name varchar(30) Nullable = false
position int4 Nullable = true
assignable bool Nullable = true
builtin int4 Nullable = false
permissions text Nullable = true
attachments
+id int4 Nullable = false
container_id int4 Nullable = false
container_type varchar(30) Nullable = false
filename varchar(255) Nullable = false
disk_filename varchar(255) Nullable = false
filesize int4 Nullable = false
content_type varchar(255) Nullable = true
digest varchar(40) Nullable = false
downloads int4 Nullable = false
author_id int4 Nullable = false
created_on timestamp(29) Nullable = true
description varchar(255) Nullable = true
projects
+id int4 Nullable = false
name varchar(30) Nullable = false
description text Nullable = true
homepage varchar(255) Nullable = true
is_public bool Nullable = false
parent_id int4 Nullable = true
created_on timestamp(29) Nullable = true
updated_on timestamp(29) Nullable = true
identifier varchar(20) Nullable = true
status int4 Nullable = false
lft int4 Nullable = true
rgt int4 Nullable = true
changesets
+id int4 Nullable = false
repository_id int4 Nullable = false
revision varchar(255) Nullable = false
committer varchar(255) Nullable = true
committed_on timestamp(29) Nullable = false
comments text Nullable = true
commit_date date Nullable = true
scmid varchar(255) Nullable = true
user_id int4 Nullable = true
messages
+id int4 Nullable = false
board_id int4 Nullable = false
parent_id int4 Nullable = true
subject varchar(255) Nullable = false
content text Nullable = true
author_id int4 Nullable = true
replies_count int4 Nullable = false
last_reply_id int4 Nullable = true
created_on timestamp(29) Nullable = false
updated_on timestamp(29) Nullable = false
locked bool Nullable = true
sticky int4 Nullable = true
comments
+id int4 Nullable = false
commented_type varchar(30) Nullable = false
commented_id int4 Nullable = false
author_id int4 Nullable = false
comments text Nullable = true
created_on timestamp(29) Nullable = false
updated_on timestamp(29) Nullable = false
versions
+id int4 Nullable = false
project_id int4 Nullable = false
name varchar(255) Nullable = false
description varchar(255) Nullable = true
effective_date date Nullable = true
created_on timestamp(29) Nullable = true
updated_on timestamp(29) Nullable = true
wiki_page_title varchar(255) Nullable = true
status varchar(255) Nullable = true
sharing varchar(255) Nullable = false

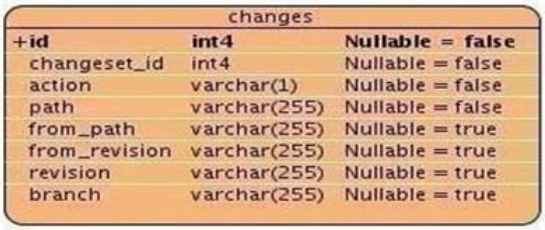


wiki_content_versions		
+id	int4	Nullable = false
wiki_content_id	int4	Nullable = false
page_id	int4	Nullable = false
author_id	int4	Nullable = true
data	bytea	Nullable = true
compression	varchar(6)	Nullable = true
comments	varchar(255)	Nullable = true
updated_on	timestamp(29)	Nullable = false
version	int4	Nullable = false

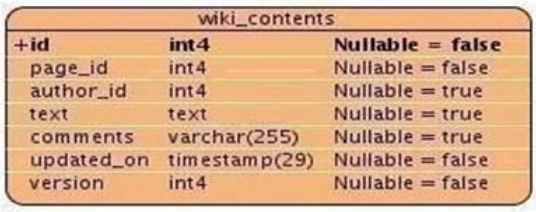


news		
+id	int4	Nullable = false
project_id	int4	Nullable = true
title	varchar(60)	Nullable = false
summary	varchar(255)	Nullable = true
description	text	Nullable = true
author_id	int4	Nullable = false
created_on	timestamp(29)	Nullable = true
comments_count	int4	Nullable = false

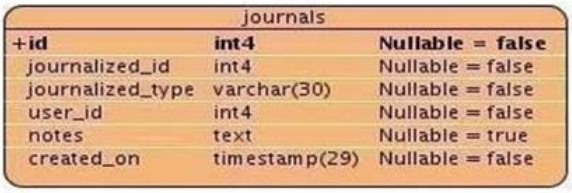
boards		
+id	int4	Nullable = false
project_id	int4	Nullable = false
name	varchar(255)	Nullable = false
description	varchar(255)	Nullable = true
position	int4	Nullable = true
topics_count	int4	Nullable = false
messages_count	int4	Nullable = false
last_message_id	int4	Nullable = true



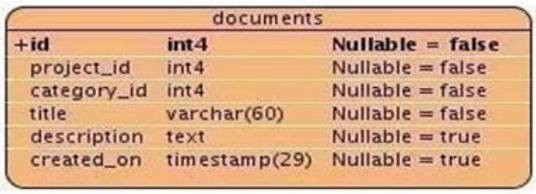
changes		
+id	int4	Nullable = false
changeset_id	int4	Nullable = false
action	varchar(1)	Nullable = false
path	varchar(255)	Nullable = false
from_path	varchar(255)	Nullable = true
from_revision	varchar(255)	Nullable = true
revision	varchar(255)	Nullable = true
branch	varchar(255)	Nullable = true



wiki_contents		
+id	int4	Nullable = false
page_id	int4	Nullable = false
author_id	int4	Nullable = true
text	text	Nullable = true
comments	varchar(255)	Nullable = true
updated_on	timestamp(29)	Nullable = false
version	int4	Nullable = false



journals		
+id	int4	Nullable = false
journalized_id	int4	Nullable = false
journalized_type	varchar(30)	Nullable = false
user_id	int4	Nullable = false
notes	text	Nullable = true
created_on	timestamp(29)	Nullable = false



documents		
+id	int4	Nullable = false
project_id	int4	Nullable = false
category_id	int4	Nullable = false
title	varchar(60)	Nullable = false
description	text	Nullable = true
created_on	timestamp(29)	Nullable = true

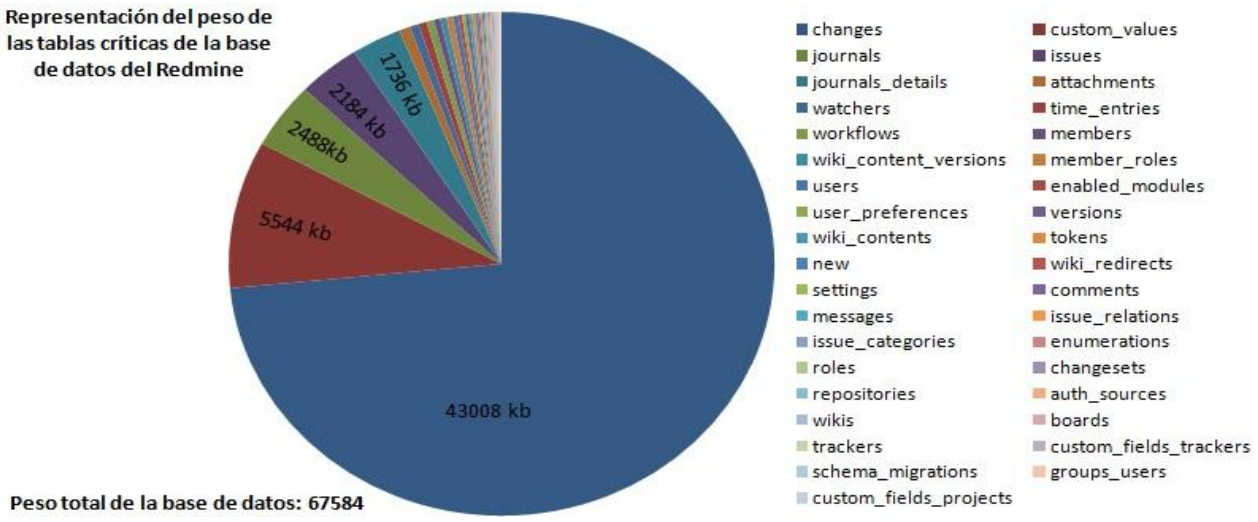
issue_statuses		
+id	int4	Nullable = false
name	varchar(30)	Nullable = false
is_closed	bool	Nullable = false
is_default	bool	Nullable = false
position	int4	Nullable = true
default_done_ratio	int4	Nullable = true

Anexo 2. Peso de las tablas de la base de datos del Redmine

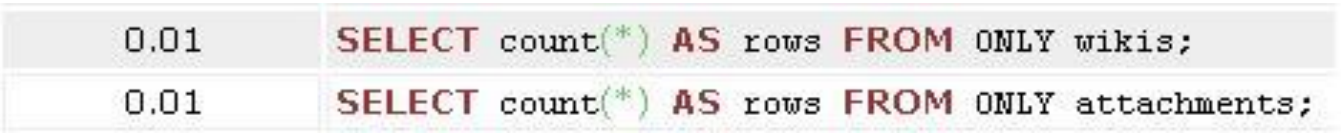
nspname name	relname name	totalsize text
public	changes	42 MB
public	custom_values	5544 kB
public	journals	2488 kB
public	issues	2184 kB
public	journal_details	1736 kB
public	attachments	384 kB
public	time_entries	288 kB
public	workflows	256 kB
public	members	224 kB
public	member_roles	216 kB
public	wiki_content_versions	216 kB
public	users	192 kB
public	enabled_modules	128 kB
public	user_preferences	120 kB
public	versions	120 kB
public	tokens	112 kB
public	wiki_contents	112 kB
public	news	104 kB
public	projects_trackers	88 kB
public	documents	80 kB
public	projects	80 kB
public	queries	72 kB
public	issue_statuses	72 kB
public	wiki_pages	72 kB
public	custom_fields	64 kB
public	wiki_redirects	64 kB
public	settings	64 kB

public	comments	64 kB
public	messages	56 kB
public	issue_relations	56 kB
public	issue_categories	56 kB
public	enumerations	56 kB
public	roles	48 kB
public	changesets	48 kB
public	repositories	48 kB
public	auth_sources	48 kB
public	wikis	40 kB
public	boards	32 kB
public	trackers	24 kB
public	custom_fields_trackers	24 kB
public	schema_migrations	24 kB
public	groups_users	24 kB
public	custom_fields_projects	24 kB
public	open_id_authentication_associations	16 kB
public	open_id_authentication_nonces	16 kB
public	sesions	16 kB
public	changesets_issues	8192 bytes

Anexo 3. Peso de las tablas con respecto al peso total de la base de datos



Anexo 4. Monitorización a la base de datos del Redmine. Consultas realizadas a las tablas attachments, projects y wikis



Glosario de Términos

UCI: siglas que responden al nombre Universidad de las Ciencias Informáticas, centro de altos estudios ubicado en la Capital de Cuba, La Habana.

SGBD: siglas que responden a Sistema de Gestión de Bases de Datos.

MVC: siglas que responden a Modelo Vista Controlador, patrón de arquitectura que se emplea en la realización de softwares informáticos.

UPDATE: palabra reservada del lenguaje SQL que se utiliza para realizar consultas de actualización de datos en las tablas.

QUERYS: se refiere a las consultas que se realizan en las bases de datos.

UML: siglas que responden al lenguaje de modelado de sistemas de software, por sus siglas en inglés Unified Modeling Language, en español Lenguaje Unificado de Modelado.

HTML: siglas que responden al lenguaje de marcado en la elaboración de páginas web, por sus siglas en inglés HyperText Markup Language, en español Lenguaje de Marcado de Hipertexto.

LOGS: registros donde se guardan todas las operaciones realizadas sobre los servidores PostgreSQL

SQL: siglas que responden en inglés a Structured Query Language, en español Lenguaje de Consulta Estructurado, es un lenguaje de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre ellas.