Universidad de las Ciencias Informáticas

Título: Plugin de análisis del rendimiento de las consultas en PostgreSQL para la herramienta de administración de bases de datos HABD.

Trabajo de diploma para optar por el título de

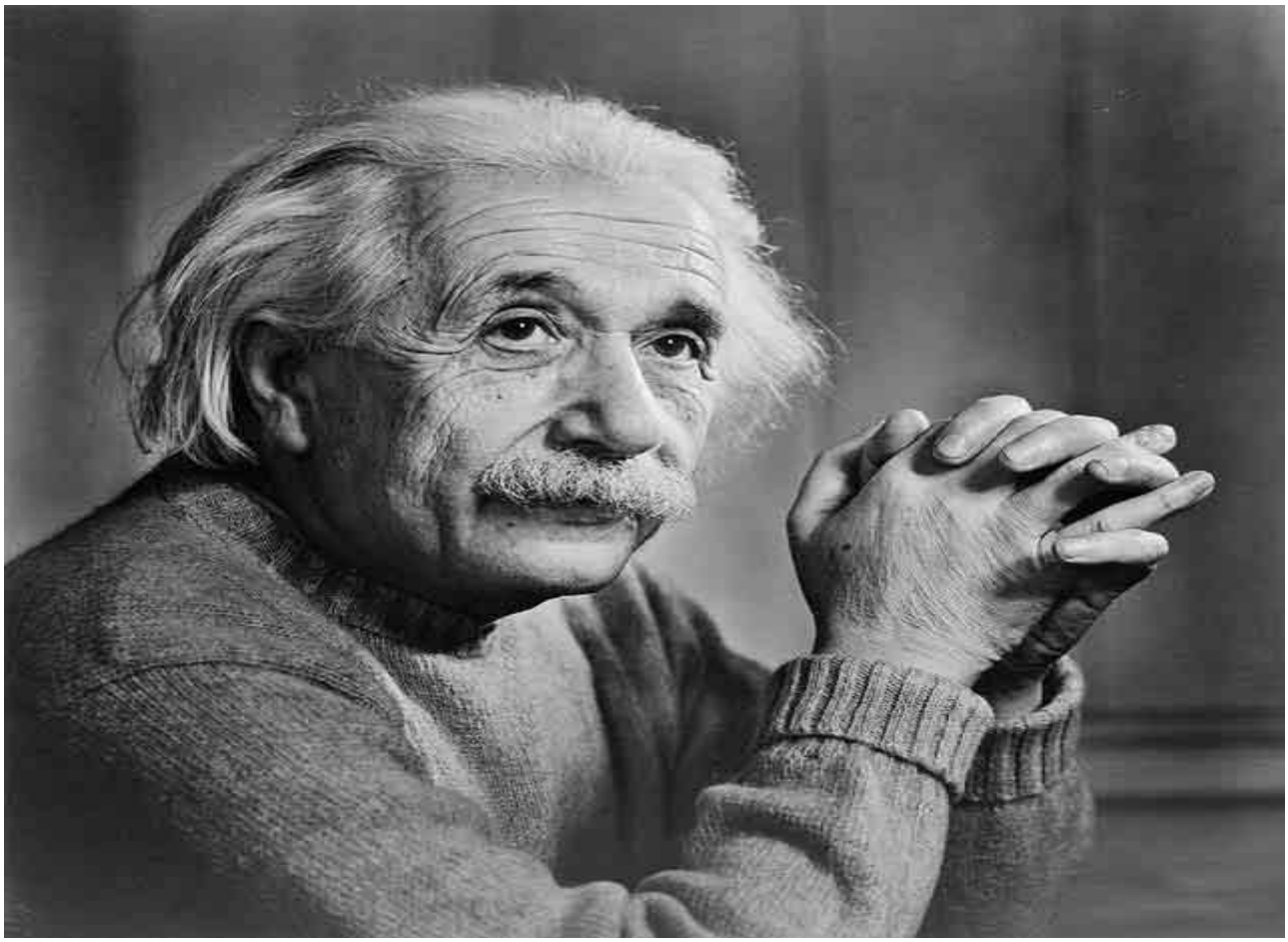
Ingeniero en Ciencias Informáticas.

Autor: Vladimir Ricardo Julián.

Tutor: Ing.Daymel Bonne Solís.

La Habana, Diciembre 2012.

“Año 54 de la Revolución”



"Nunca consideres el estudio como una obligación sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber."

Albert Einstein.

Declaración de autoría:

Declaro que soy el único autor del trabajo “Plugin de análisis del rendimiento de las consultas en PostgreSQL para la Herramienta de Administración de bases de datos HABD.” 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 ____ días del mes de _____ del año _____.

Vladimir Ricardo Julián	Ing. Daymel Bonne Solís
Autor	Tutor

Tutor:

Ing. Daymel Bonne Solís

Universidad de las Ciencias Informáticas, Habana, Cuba.

e-mail: dbonne@uci.cu

Agradecimientos:

Agradezco a toda mi familia que siempre estuvo pendiente de mis resultados en los 5 años en la universidad, en especial a mi madre que es la persona que siempre ha estado ahí para mí y a mi compañera Tahymi que me ha apoyado durante todo este proceso. También a todos mis amigos y amigas con los cuales compartí en la escuela y con los cuales viví experiencias inolvidables, además de todos los profes que siempre me dieron su apoyo incondicional y sincero sin importar la situación o momento en el que se encontraran.

Dedicatoria:

Dedico mi tesis en primer lugar a mi madre por ser lo más grande que tengo en la vida y por ser la persona que más ha sentido todo mi proceso de convertirme en profesional y por darme todo lo que me hacía falta sin importar el costo y también dedicarle mi trabajo a mi hijo Fabio el cual con su presencia me dio más fuerzas para sobreponerme a todos los obstáculos con los cuales tuve que lidiar en esta recta final.

Resumen

Con el desarrollo de la informática, los Sistemas de Administración de Bases de Datos han evolucionado continuamente con el paso de los años. Cuba no ha querido quedarse atrás en este proceso de desarrollo, ya que, con la necesidad de tener aplicaciones y herramientas propias del país, en la Universidad de las Ciencias Informáticas se desarrolló una herramienta de administración de bases de datos llamada HABD. Esta herramienta posee una arquitectura basada en plugin, característica que posibilita a los desarrolladores agregar nuevas funcionalidades de forma sencilla. A pesar de que HABD cuenta con funcionalidades necesarias para el trabajo con el gestor de bases de datos, no cuenta con un analizador del rendimiento de las consultas que se realizan en la misma. Por tal motivo se desarrolló un plugin de análisis de rendimiento de las consultas para bases de datos PostgreSQL. Con el desarrollo del plugin el objetivo fundamental que se persiguió fue permitir a los usuarios realizar el análisis del rendimiento de las consultas, principalmente aquellas que presenten problemas con los tiempos de ejecución. Además de brindar sugerencias para la optimización de las consultas analizadas.

Palabras Clave: análisis del rendimiento, optimización de consultas, plugin.

Tabla de contenido

Índice de figuras

INTRODUCCIÓN

Desde el surgimiento de la informática, el hombre comprendió la necesidad de almacenar grandes cantidades de información. A partir de ese momento las bases de datos han constituido una de las herramientas más utilizadas en la actual sociedad, en la recuperación y almacenamiento de información. Estas han sido empleadas como fuentes de consulta y de producción de conocimientos para investigadores, científicos y académicos en todas las áreas, llegando a ser una de las más importantes herramientas para almacenar grandes volúmenes de información. Posteriormente del surgimiento de las bases de datos no solo bastaba con almacenar la información, sino que se necesitaba de un sistema para gestionar la información contenida en estas. La administración de las bases de datos se realiza con un sistema llamado DBMS (Database Management System: Sistema de Administración de Bases de Datos), el cual es un software que controla la organización, almacenamiento, recuperación, seguridad e integridad de los datos en una base de datos (1). Los sistemas gestores de bases de datos proporcionan a los usuarios la gestión del acceso a la base de datos por múltiples usuarios de manera simultánea y garantizan que no sucedan inconvenientes con la integridad de los mismos. Estos se agrupan en dos categorías: libres y propietarios, destacándose PostgreSQL dentro de los libres, siendo uno de los gestores más utilizados en el mundo; ya que además de ser el gestor de código abierto más avanzado, cuenta con un gran número de desarrolladores que se encargan de dar soporte y documentación la cual es muy amplia y cada día tratan de perfeccionarlo y hacerlo más eficaz y funcional.

Con el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) se ha tratado de satisfacer en Cuba las necesidades de información y conocimiento de todas las personas y ámbitos de la sociedad, destacándose entre las instituciones del país la Universidad de las Ciencias Informáticas (UCI). La UCI está organizada por centros de desarrollo, siendo el Centro de Tecnologías de Gestión de Datos (DATEC) uno de sus miembros. Este centro cuenta con varias líneas entre las que se destaca la llamada PostgreSQL, la cual tiene como objetivo fundamental, contribuir a la soberanía tecnológica aumentando el auge de las tecnologías de bases de datos libres. En el curso 2010-2011 se desarrolló en dicha línea una herramienta de administración de bases de datos llamada HABD, para PostgreSQL. Se obtuvo como resultado de esa implementación un Front-End el cual provee una interfaz amigable y una arquitectura basada en plugin que permite a los desarrolladores incorporar nuevos servicios de manera sencilla. Para los usuarios que trabajan directamente con herramientas de características similares a HABD, se les hace complicado apreciar de forma simple el rendimiento de las consultas cuando estas presentan problemas, principalmente con el tiempo de respuesta de las

mismas. A raíz de esta limitante los usuarios deben dedicar tiempo de más a comprender la causa de la demora y en darle solución, algo que afectaría el trabajo de estos y haría que el sistema dejara de funcionar de forma óptima. Por lo antes expresado surge el siguiente problema de investigación: ¿Cómo realizar el análisis del rendimiento de las consultas que se realizan sobre bases de datos PostgreSQL para mejorarlas, en la herramienta de administración de bases de datos HABD?

Se define por tanto como **objeto de estudio** el análisis del rendimiento de consultas en Sistemas Gestores de Bases de Datos y como **campo de acción** el análisis del rendimiento de consultas realizadas en el Sistema Gestor de Bases de Datos PostgreSQL para la herramienta HABD.

El **objetivo general** de la presente investigación es: Desarrollar un plugin que permita analizar el rendimiento de las consultas realizadas sobre bases de datos PostgreSQL en la herramienta de administración de bases de datos HABD, con el objetivo de brindar opciones de mejoras en el rendimiento de la consulta.

Para dar cumplimiento al objetivo general de la investigación se definen como **objetivos específicos:**

- Análisis de las técnicas para la optimización de consultas y herramientas existentes para la implementación del plugin.
- Implementación del plugin para la herramienta de administración de bases de datos (HABD).
- Validación del plugin obtenido.

Para cumplir los objetivos específicos se planificaron las siguientes **tareas de la investigación:**

- Análisis de las técnicas de optimización de consultas en los sistemas gestores de bases de datos.
- Análisis de las herramientas que realizan análisis de las consultas SQL para bases de datos PostgreSQL.
- Selección de las herramientas, tecnologías y metodología que serán utilizadas para desarrollar el plugin.
- Definición del modelo de dominio.
- Elaboración de las historias de usuarios.
- Implementación de las historias de usuarios.
- Realización de los casos de pruebas.
- Aplicación de los casos de pruebas para validar las funcionalidades implementadas.

Estructura del documento

Capítulo1 “Fundamentación Teórica”.

En este capítulo se analizan las herramientas de administración que trabajan con PostgreSQL, además del análisis de sugerencias para la optimización de las consultas en distintos gestores de bases de datos. Se definen las herramientas, tecnologías y metodología a utilizar en el desarrollo del trabajo.

Capítulo 2 “Descripción e Implementación del Sistema”.

En este capítulo se definen los requisitos necesarios para la correcta implementación del plugin de análisis de rendimiento de las consultas en PostgreSQL, tanto funcionales como no funcionales, descritos mediante las historias de usuarios, además se representan diagramas de clases y dominio, tarjetas CRC, se definen los patrones de arquitectura y diseño y se realizan todas las tareas de la ingeniería, todo esto con un solo objetivo, lograr el desarrollo del plugin para poder integrarlo con la herramienta HABD.

Capítulo3 “Pruebas del sistema”.

En este capítulo se realiza un estudio para definir las pruebas que serán aplicadas al sistema para verificar su correcto funcionamiento .Además se validan las funcionalidades implementadas a través de las estrategias y técnicas definidas. Se muestran los resultados obtenidos luego de haber realizado las pruebas.

Capítulo 1

Fundamentación teórica.

Introducción

En el desarrollo del presente capítulo se analizan los conceptos fundamentales relacionados con las bases de datos, se investigan cómo se observa el rendimiento de las consultas SQL en algunas de las herramientas de administración de bases de datos más importantes y se muestran diferentes características de las tecnologías y herramientas seleccionadas para el posterior desarrollo del plugin de análisis de rendimiento de las consultas en PostgreSQL.

1.1 Conceptos generales

1.1.1 Bases de datos

El término de bases de datos fue escuchado por primera vez en California-USA en un simposio en 1963 y esta se puede definir como un conjunto de información relacionada la cual está estructurada. Desde el ámbito informático se define como bases de datos como un sistema formado por un conjunto de datos almacenados en disco, el cual permite accesos directos a ellos y un grupo de programas que permiten la manipulación de esos datos (2).

La anterior descripción de forma general abarca el concepto de bases de datos, pero particularmente se prefiere escoger la definición dada en el libro: Introducción a los Sistemas de Bases de Datos, del autor Chris J. Date[1]:

"Es posible considerar a la propia **base de datos** como una especie de armario electrónico para archivar; es decir, es un depósito o contenedor de una colección de archivos de datos computarizados" (3).

Los usuarios del sistema pueden realizar una variedad de operaciones sobre dichos archivos.

Por ejemplo (3):

- ✓ Agregar nuevos archivos vacíos a la base de datos.
- ✓ Insertar datos dentro de los archivos existentes.
- ✓ Recuperar datos de los archivos existentes.
- ✓ Modificar datos en archivos existentes.

[1] Chris J. Date: C.J. Date es autor, conferencista, investigador y consultor independiente, especializado en la tecnología de bases de datos.

- ✓ Eliminar datos de los archivos existentes.
- ✓ Eliminar archivos existentes de la base de datos.

1.1.2 PostgreSQL

El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL (y brevemente llamado Postgres95) está derivado del paquete Postgres escrito en Berkeley. PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy en día, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, Perl, tcl y python) (4).

Postgres ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema (4):

- ✓ clases
- ✓ herencia
- ✓ tipos
- ✓ funciones

Otras características aportan potencia y flexibilidad adicional:

- ✓ Restricciones (constraints)
- ✓ Disparadores (triggers)
- ✓ Reglas (rules)
- ✓ Integridad transaccional

Estas características colocan a PostgreSQL en la categoría de las Bases de Datos identificadas como objeto-relacionales (4). Por estas razones el centro DATEC tiene establecido como sistema gestor de bases de datos a dicho gestor.

1.1.3 Plugin

El objetivo principal del trabajo de diploma es desarrollar un plugin para realizar el análisis de las consultas en PostgreSQL, pero es necesario conocer el concepto de un plugin. La palabra plugin viene del inglés “plug in” o “enchufar”. Un plug-in es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande (5).

Existen otras definiciones de plugin las cuales se acercan bastante como son: Pequeño programa que añade alguna función a otro programa, habitualmente de mayor tamaño. Un programa puede tener uno o más conectores. Son muy utilizados en los programas navegadores para ampliar sus

funcionalidades (6); y también se puede detallar como: aplicación informática que interactuando con otra aplicación le aporta una función específica a ésta última, como por ejemplo servir como driver (o controlador) en una aplicación con el fin de hacer funcionar un dispositivo en otro software (7).

Luego de analizar las anteriores definiciones, es válido llamar plugin a lo que se desea implementar, ya que va a ser una nueva función la cual se quiere integrar a una herramienta que en este caso es HABD.

1.1.4 Consultas SQL

A través de consultas SQL se recupera información y se realizan cambios sobre la base de datos de forma sencilla. Las mismas se escriben en el lenguaje SQL, el cual se ha convertido en el lenguaje de consulta relacional más popular. El nombre “SQL” es una abreviatura de Structured Query Language (Lenguaje de consulta estructurado). En 1974 Donald Chamberlain y otros definieron el lenguaje SEQUEL (Structured English Query Language) en IBM Research. Este lenguaje fue implementado inicialmente en un prototipo de IBM llamado SEQUEL-XRM en1974-75. En 1976-77 se definió una revisión de SEQUEL llamada SEQUEL/2 y el nombre se cambió a SQL (4). El mismo es un estándar para interactuar con bases de datos relacionales y es soportado por prácticamente todos los productos de base de datos actuales. El nombre "SQL" significaba originalmente "Lenguaje estructurado de consultas" (3). Una consulta SQL es creada con una instrucción SQL (cadena o instrucción SQL: es la que define un comando SQL, por ejemplo SELECT, UPDATE o DELETE e incluye cláusulas como WHERE o ORDERBY) (8).

El comando más usado en SQL es la instrucción SELECT, a continuación se muestran algunos ejemplos de consultas, principalmente de SELECT (4):

- ✓ SELECT * FROM PART WHERE PRICE > 10;
- ✓ SELECT PNAME, PRICE FROM PART WHERE PRICE > 10;
- ✓ SELECT PNAME, PRICE * 2 AS DOUBLE FROM PART WHERE PRICE * 2 < 50;

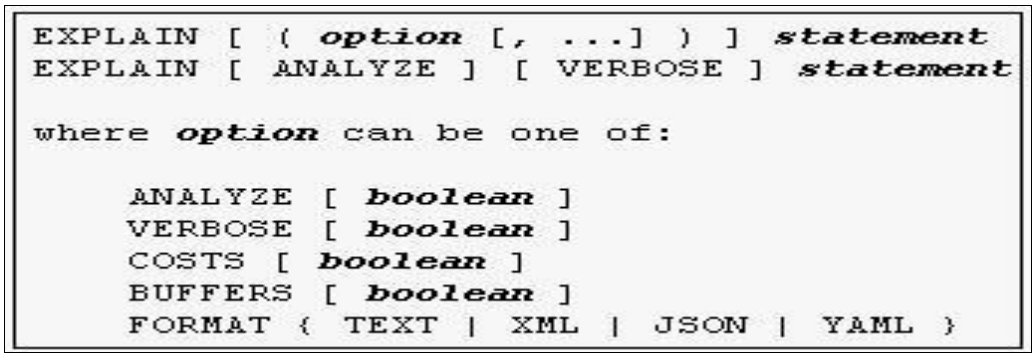
Para un mejor entendimiento se explica la primera sentencia de las expuestas anteriormente, la cual a través de la palabra SELECT: selecciona a todos (*) los atributos de la tabla PART, la búsqueda en la tabla se especifica mediante la palabra FROM y el WHERE es la condición, la cual expone que el precio debe ser mayor que 10.

1.2 EXPLAIN ANALYZE

Es un comando SQL utilizado en consultas lentas para obtener información sobre el origen de la falta de velocidad, permitiendo en muchas ocasiones resolver la causa rápidamente o dando la información necesaria de estas para poder investigar un método de solución (9).

Como resultado de este comando se obtiene un árbol invertido de nodos el cual se lee de abajo hacia arriba, buscando el nodo más bajo con problemas e interpretando de forma integral / global el resultado, ya que muchos de estos nodos pueden ejecutarse de forma paralela y tienden a influenciase mutuamente (9).

Para una correcta interpretación del resultado de este comando es necesario conocer que significa lo que es mostrado en la pantalla al ejecutar el mismo. A continuación se muestra una figura de la sintaxis cuando este se usa:



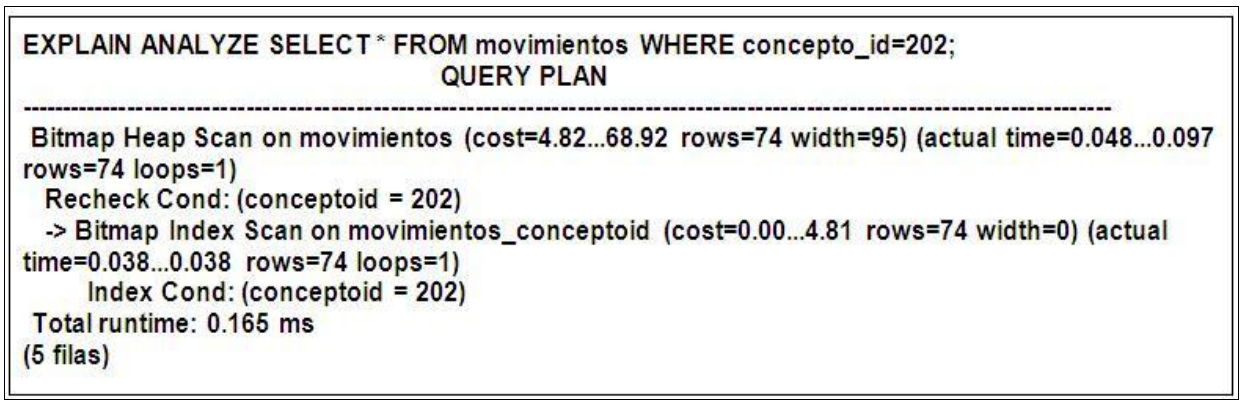
```
EXPLAIN [ ( option [, ...] ) ] statement
EXPLAIN [ ANALYZE ] [ VERBOSE ] statement

where option can be one of:

    ANALYZE [ boolean ]
    VERBOSE [ boolean ]
    COSTS [ boolean ]
    BUFFERS [ boolean ]
    FORMAT { TEXT | XML | JSON | YAML }
```

Figura 1: Sintaxis del EXPLAIN ANALYZE.

A continuación se muestra un ejemplo de cómo se puede utilizar este comando SQL en consultas y el resultado de su ejecución:



```
EXPLAIN ANALYZE SELECT * FROM movimientos WHERE concepto_id=202;
                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on movimientos (cost=4.82...68.92 rows=74 width=95) (actual time=0.048...0.097
rows=74 loops=1)
   Recheck Cond: (conceptoid = 202)
   -> Bitmap Index Scan on movimientos_conceptoid (cost=0.00...4.81 rows=74 width=0) (actual
time=0.038...0.038 rows=74 loops=1)
        Index Cond: (conceptoid = 202)
 Total runtime: 0.165 ms
(5 filas)
```

Figura 2: Resultado del EXPLAIN ANALYZE.

Los números citados son (de izquierda a derecha, acumulando en los nodos superiores sus hijos) (9):

- ✓ Costo de inicio estimado (tiempo gastado antes de que se inicia el barrido de salida).
- ✓ Costo total estimado (si todas las filas fueran de vueltas).
- ✓ Número estimado de filas de salida.
- ✓ Ancho promedio estimado (en bytes) de las filas de salida.

El costo es medido en unidades arbitrarias determinadas por la configuración, tradicionalmente equivale a la cantidad de páginas leídas desde el disco:

(seq_page_cost=1.0).

En la figura se muestra que devuelve los tiempos de ejecución (**actual time** en mili segundos). Si bien las unidades de tiempo y costo son distintas, lo importante es que mantengan las proporciones entre sí.

1.3 Herramientas que utilizan el gestor de bases de datos PostgreSQL

En la actualidad existen herramientas de administración que trabajan con el sistema de gestión de bases de datos PostgreSQL, por las características y ventajas que el sistema brinda. Se analizan a continuación las principales herramientas, para obtener información sobre cómo estas analizan las consultas, en caso que hagan esta función.

1.3.1 pgAdmin lll

Es una aplicación gráfica para gestionar el gestor de bases de datos PostgreSQL, siendo la más completa y popular con licencia Open Source. Está escrita en C++ usando la librería gráfica multiplataforma wxWidgets[2], lo que permite que se pueda usan en Linux, FreeBSD, Solaris, Mac OS X y Windows (10). Es capaz de gestionar versiones a partir de la PostgreSQL 7.3 y superiores ejecutándose en cualquier plataforma, así como versiones comerciales de PostgreSQL como Pervasive Postgres, EnterpriseDB, Mammoth Replicator y SRA PowerGres. El pgAdmin III está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples hasta desarrollar bases de datos complejas. El interfaz gráfico soporta todas las características de PostgreSQL y facilita enormemente la administración (10).

[2] Las wxWidgets son unas bibliotecas multiplataforma y libres, para el desarrollo de interfaces gráficas programadas en lenguaje C++.

Una característica interesante de pgAdmin lll es que, cada vez que se realiza alguna modificación en un objeto, escribe la(s) sentencia(s) SQL correspondiente(s), lo que hace que, además de una herramienta muy útil, sea a la vez didáctica. También incorpora funcionalidades para realizar consultas, examinar su ejecución (como el comando explain y explain analyze) y trabajar con los datos (11).

Esta herramienta incluye (12):

- ✓ Interfaz administrativa gráfica
- ✓ Herramienta de consulta SQL (con un EXPLAIN gráfico)
- ✓ Editor de código procedural
- ✓ Agente de planificación SQL/shell/batch
- ✓ Administración de Slony-I

Como se mencionaba anteriormente el PgAdmin lll utiliza el Explain Analyze para analizar la ejecución de la consulta, el cual muestra los costos y los tiempos de ejecución por nodos, además del tiempo total de la consulta. Por tanto, gracias a toda la información de la ejecución de la consulta que brinda el uso de Explain Analyze, se decide la utilización del mismo en el desarrollo de plugin propuesto para poder realizar el análisis del rendimiento.

1.3.2 PgAccess

PgAccess es una interface gráfica para el gestor de bases de datos PostgreSQL escrito por Constantin Teodorescu en el lenguaje Tcl/Tk[3]. Permite al usuario interactuar con PostgreSQL de una manera similar a muchas aplicaciones de bases de datos para PC, con menús de opciones y diversas herramientas gráficas. Esto significa que el usuario puede evitar la línea de comandos para la mayoría de las tareas. PgAccess no cambia el modo de actuar de PostgreSQL, sólo hace más fácil su uso para aquellos que estén habituados a interfaces gráficas (13). El PgAccess es una aplicación "open source" (código abierto). El código está disponible para el usuario, y puede ser modificado. El usuario puede arreglar fallos, o cambiar la manera de operar de una función (14).

Con las consultas en el PgAccess se realizan un número de funciones entre ellas están (9):

- ✓ Definir, editar y almacenar consultas definidas por el usuario.
- ✓ Guardar formatos de vistas.
- ✓ Almacenar consultas como vistas.

[3] TCL por sus siglas en inglés (Tool Command Language) o en español “Lenguaje de Herramientas de Comando” es un tipo de lenguaje de programación perteneciente a la categoría de lenguajes script (Lenguaje Interpretado).

- ✓ Ejecutar con parámetros opcionales de entrada introducidos por el usuario; por ejemplo: Select * from invoices where year= [parameter "Year of selection"].
- ✓ Visualizar cualquier resultado de una consulta de selección (select).
- ✓ Ejecutar consultas de acción (insert, update, delete).
- ✓ Definir consultas usando un constructor visual de consultas con soporte drag & frop y aliasing de tablas.

Esta herramienta posee funcionalidades muy útiles en el trabajo con las consultas pero a la vez no cuenta con funciones encargadas de realizar análisis de rendimiento de la consulta.

1.3.3 PhpPgAdmin

PhpPgAdmin es una herramienta con una interfaz Web bastante potente que nos permite administrar los servidores de bases de datos PostgreSQL, esta aplicación está escrita en PHP con la intención de manejar la administración de PostgreSQL a través de páginas Web, utilizando Internet y fue basado en otro producto visto anteriormente llamado PhpMyAdmin (15).

PhpPgAdmin es un cliente web que ofrece una manera conveniente de crear objetos como bases de datos, tablas, usuarios, funciones, vistas, disparadores, etc., alterarlos o consultar sus datos usando el estándar SQL. Entre sus características se encuentran (15):

- ✓ PhpPgAdmin puede administrar múltiples servidores PostgreSQL.
- ✓ Soporta las versiones de PostgreSQL 7.0.x, 7.1.x, 7.2.x, 7.3.x, 7.4.x, 8.0.x y 8.1.x.
- ✓ Permite la fácil administración de usuarios y grupos, bases de datos, tablas, índices, triggers o disparadores, reglas, privilegios, vistas, secuencias, funciones, etc.
- ✓ Permite una fácil manipulación de los datos, con un buscador de tablas, vistas y reportes.
- ✓ Permite la ejecución de la mayoría de comandos SQL de manipulación (DML) y definición de datos (DDL).
- ✓ Puede exportar e importar los datos en una variedad de formatos: SQL, XML, XHTML o CSV.
- ✓ Tiene una fácil instalación y configuración.

El PhpPgAdmin realiza un gran número de operaciones, pero no cuenta con el servicio necesario para analizar las consultas, algo que sería muy útil para este tipo de herramientas web.

1.3.4 Herramienta de administración de bases de datos HABD

En el curso 2010-2011 en la línea PostgreSQL del centro DATEC se desarrolló una herramienta de administración de bases de datos para PostgreSQL llamada HABD. El resultado de esta implementación es un Front-End, el cual provee una arquitectura que permite a los desarrolladores incorporar nuevos servicios de manera sencilla por la flexibilidad que posee mediante plugin. La misma contribuye al intercambio entre el usuario y el gestor PostgreSQL facilitando el trabajo entre ambos y tiene como finalidad realizar un grupo de funcionalidades, las cuales serán integradas de forma inmediata a la herramienta al culminar su desarrollo (16).

1.4 Mecanismos para optimizar consultas

Los usuarios de los sistemas de administración de bases de datos, ya sean propietarios o libres, por el sistemático trabajo, han adquirido experiencia y han publicado y comentado en la comunidad muchas formas de mejorar las consultas SQL. Luego de una investigación sobre este tema, se han recopilado una serie de ejemplos y sugerencias de cómo mejorar las consultas, las mismas son expuestas a continuación en dependencia del sistema gestor de bases de datos en que se realizaron.

1.4.1 SQL Server.

SQL Server se ha trazado como objetivo optimizar las consultas SQL lo mejor posible. El mismo no tiene definido un algoritmo o estándar para lograr el mejoramiento, sino que va encaminado a mejorar en las consultas, la utilización correcta de las sentencias, la posición de los operadores y cláusulas, de forma tal que no se realicen operaciones de forma innecesaria y ahorre tiempo en la ejecución del sistema. Algunas recomendaciones para mejorar las consultas son (17):

- ✓ **Uso de DISTINCT**

  Debe usarse solo si se sabe que la consulta puede devolver duplicados, y además esto puede provocar un mal funcionamiento de la aplicación que hace uso de los datos. Esta sentencia genera una gran cantidad de trabajo extra a SQL Server, debido a que consume muchos recursos que son necesarios para otras consultas que sean lanzadas dentro de la base de datos.

- ✓ **Usando TOP**

  Para evitar en la aplicación de forma sencilla el retorno de cientos de registros o incluso miles, es posible usar el operador **TOP** con una sentencia **SELECT**. De este modo se puede ayudar a mejorar el rendimiento.

SELECT TOP 100 fname, lname **FROM** customers **WHERE** state= ‘mo’.

Esta consulta limita los resultados a las 100 primeras filas, incluso si el criterio del **WHERE** devuelve 10000 registros.

- ✓ **Uso de operadores en el WHERE**

  Otro aspecto a destacar es el uso de los operadores dentro de la cláusula **WHERE**, considerando qué operadores tienen mejor rendimiento, se muestra a continuación el orden de mejor a peor rendimiento de los operadores**(=,>,>=,<,<=,LIKE,<>)**.

- ✓ **Uso de NOT IN**

  En caso de hacer uso de este comando, es necesario tener especial cuidado en su uso porque el mismo posee mal rendimiento. En su lugar se utilizarán las siguientes opciones ordenadas de mejor a peor rendimiento:

- ✓ Usar **EXIST** o **NOT EXIST**.
- ✓ Usar **IN**.
- ✓ Realizar un **LEFT OUTER JOIN** y chequear por una condición **null.**

1.4.2 MySQL

Sin duda alguna uno de los sistemas de gestión de bases de datos más populares y utilizados es MySQL. Debido a que la mayoría de los procesos de la sociedad se realizan de forma electrónica a través de aplicaciones web y este es el sistema de gestión de bases de datos preferido en los sitios de lectura intensiva (18).

MySQL desea que las búsquedas en sus bases de datos tengan el mayor rendimiento posible y utilice la menor cantidad de tiempo en sus consultas, un método utilizado es, a través del caché de las consultas, donde se guardan los resultados de las consultas, de esta forma cada vez que se realice la misma consulta los resultados son devueltos de forma inmediata aumentando el rendimiento. Para que este proceso tenga efecto hay que realizar tres pasos muy importantes los cuales se mencionan y explican a continuación (18):

Paso#1.

Habilitar el caché de consultas, estableciendo el tipo”1”.

query_cache_type=1

Paso#2.

Configurar el tamaño máximo de cada consulta que puede almacenar el caché, este es un paso muy importante ya que si el resultado de la consulta es mayor que el límite del caché de la consulta, los resultados no se guardarían. Esta configuración es normalmente establecida a 1Mega:

query_cache_limit=1M

Paso#3.

La cantidad de memoria disponible globalmente para el caché de consultas se establece con el query_cache_size. Esto debería ser bastante grande y debe ser aumentado en tamaño para grandes bases de datos.

query_cache_size=100M

1.4.3 PostgreSQL

Los usuarios del sistema gestor PostgreSQL pretenden hacer eficiente el rendimiento de sus consultas, siendo el trabajo con los índices una de las formas más utilizadas al ejecutar una consulta.

De forma automática, PostgreSQL cuando crea una llave primaria (PK), crea un índice, este proceso de creación de índices se puede realizar de forma forzada según las necesidades del usuario, pero el planificador es el encargado de determinar si es necesario el uso o no de un índice en su consulta. Aparte de los creados por las PK, se puede crear índices en los campos para mejorar el rendimiento de las consultas (al declarar campos como **UNIQUE**). Por mencionar un ejemplo, si tiene una tabla personas (9):

Id sería primary key (se crea un índice por ser PK), nombre varchar, apellidos varchar, dni varchar UNIQUE (se crea un índice por ser **UNIQUE**), f_nacimiento date. En caso que se quiera buscar personas preguntando sobre la fecha de nacimiento se realiza la siguiente consulta:

SELECT id **FROM** personas **WHERE** f_nacimiento<= '01/01/1990'. Este **SELECT** hace un recorrido secuencial, es decir, si tienes un millón de personas este tiene que recorrer un millón de registros (9).

```
pruebas=# create table personas (id serial primary key, nombre varchar, dni integer);
NOTICE:  CREATE TABLE creará una secuencia implícita «personas_id_seq» para la columna serial «personas.id»
NOTICE:  CREATE TABLE / PRIMARY KEY creará el índice implícito «personas_pkey» para la tabla «personas»
CREATE TABLE

pruebas=# insert into personas (nombre, dni) select 'nombre ' || foo, foo from generate_series (1,100000) foo;
INSERT 0 100000
pruebas=# EXPLAIN ANALYZE select * from personas where dni between 2500 AND 5000;

                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Seq Scan on personas (cost=0.00...2133.00 rows=2497 width=20) (actual time=1.425...74.737 rows=2501 loops=1)
   Filter: ((dni >= 2500) AND (dni <= 5000))
 Total runtime: 79.222 ms
(3 rows)
```

Figura 3: Resultado del EXPLAIN ANALYZE.

Luego se crea el índice y se prueba para ver los cambios:

Create index idx_dni **on** personas (dni);

```
pruebas=# create index idx_dni on personas (dni);
CREATE INDEX
pruebas=# EXPLAIN ANALYZE select * from personas where dni between 2500 AND 5000;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Index Scan using idx_dni on personas (cost=0.00...93.20 rows=2497 width=20) (actual time=0.131..6.596 rows=2501 loops=1)
   Index Cond: ((dni >= 2500) AND (dni <= 5000))
 Total runtime: 10.985 ms
(3 rows)
```

Figura 4: Resultado del EXPLAIN ANALYZE.

Al utilizar el índice creado en el **SELECT** anterior, se evidencia un aumento en el rendimiento, observándose en el resultado del EXPLAIN ANALYZE.

1.4.4 Consejos generales para la optimización de las consultas.

De forma general existen otros consejos para optimizar las consultas, aplicables a los gestores que utilizan el lenguaje SQL, los cuales se exponen a continuación:

- **Evite el uso de comodín (%) en el comienzo de un predicado.**

  El predicado **LIKE** 'abc%' hace escaneo completo de tabla. Ejemplo: **SELECT** * **FROM** TABLA1 **WHERE** col1 **LIKE** 'abc%' (19).

- **Evite las columnas innecesarias en la cláusula SELECT.**

Especifique las columnas de la cláusula **SELECT** en lugar de utilizar **SELECT ***. Las columnas innecesarias ponen cargas adicionales sobre la base de datos, lo que ralentiza el SQL no sólo individual, sino todo el sistema (19).

- **La cláusula ORDER BY en SQL es obligatorio si el conjunto de resultados ordenados se espera.**

  El **ORDER BY** palabra clave se utiliza para ordenar el conjunto de resultados por las columnas especificadas. Sin la cláusula **ORDER BY**, el conjunto de resultados se devuelve directamente sin ningún tipo de clasificación. El orden no está garantizado. Tenga en cuenta el impacto en el rendimiento de la adición de la cláusula **ORDER BY**, como la base de datos necesita para ordenar el conjunto de resultados, dando como resultado una de las operaciones más caras de la ejecución de SQL (19).

- **Orden de las tablas**

  Cuando se utilizan varias tablas dentro de la consulta hay que tener cuidado con el orden empleado en la cláusula **FROM**. Si se desea saber cuántos alumnos se matricularon en el año 1996, se escribe **FROM** Alumnos, Matriculas **WHERE** Alumno.IdAlumno = Matriculas.IdAlumno **AND** Matriculas.Año = 1996 el gestor recorrerá todos los alumnos para buscar sus matrículas y devolver las correspondientes. Si se cambia por **FROM** Matriculas, Alumnos **WHERE** Matriculas.Año = 1996 **AND** Matriculas.IdAlumno = Alumnos.IdAlumnos, el gestor filtra las matrículas y después selecciona los alumnos, de esta forma tiene que recorrer menos registros (20).

- **Campos de Filtro**

  Se procurará elegir en la cláusula **WHERE** aquellos campos que formen parte de la clave del fichero por el cual se pregunta. Además se especificarán en el mismo orden en el que estén definidos en la clave. Si se quiere interrogar por campos pertenecientes a índices compuestos es mejor utilizar todos los campos de todos los índices. Ejemplo: tenemos un índice formado por el campo NOMBRE y el campo APELLIDO y otro índice formado por el campo EDAD. La sentencia **WHERE** NOMBRE='Juan' **AND** APELLIDO **LIKE** '%' **AND** EDAD = 20 sería más óptima que **WHERE** NOMBRE = 'Juan' AND EDAD = 20 por que el gestor, en este segundo caso, no puede usar el primer índice y ambas sentencias son equivalentes por que la condición APELLIDO **LIKE** '%' devolvería todos los registros (20).

1.5 Metodología de desarrollo de software

Debido al gran auge y desarrollo que ha alcanzado la producción de software a nivel mundial se ha creado una inmensa cantidad de documentación sobre las políticas, procesos y procedimientos para lograr la calidad del producto final. Entre los principales objetivos de las metodologías de desarrollo de software se encuentran cumplir con los requisitos iniciales del problema y minimizar las pérdidas de tiempo en el proceso de desarrollo así como minimizar riesgos e incrementar las posibilidades de éxito en el desarrollo de los productos.

1.5.1 Extreme Programming (XP)

La metodología XP es adecuada para proyectos pequeños y medianos, su ciclo de vida es iterativo e incremental, cada iteración dura entre una y tres semanas, además de no producir demasiada documentación acerca del diseño o la planificación.

XP fue desarrollada por Kent Beck el cual afirma que:

“Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar” (21).

Extreme Programming surgió como respuesta y posible solución a los problemas derivados del cambio en los requisitos, es una metodología basada en la simplicidad, la comunicación la reutilización del código de desarrollo y la propiedad del código es colectiva. Provee muchas ventajas como tener una programación organizada, menor taza de errores, satisfacer al programador, donde los clientes tienen el control sobre las prioridades, por lo que es la más utilizada en la implementación de nuevas tecnologías donde los requisitos cambian rápidamente.

La línea PostgreSQL del centro DATEC tiene definida como metodología de desarrollo de software a utilizar: Extreme Programming o XP; lo que condiciona su utilización para el desarrollo del plugin.

1.6 Herramientas de desarrollo de software.

1.6.1 Visual Paradigm 8.0

Durante el ciclo de vida de desarrollo de un software las herramientas CASE[4] son de gran beneficio, se definen como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores. Se escoge para la realización del plugin la herramienta CASE Visual Paradigm que soporta el ciclo de vida completo del proceso de desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado ayuda a la construcción de aplicaciones de calidad, a un menor coste. Es una herramienta libre que ayuda a construir aplicaciones rápidamente, soporta a varios usuarios trabajando en el mismo proyecto, con una poderosa robustez, usabilidad y portabilidad. Facilita el modelado colaborativo con Subversion. Proporciona la transformación de diagramas Entidad-Relación en tablas de bases de datos, ingeniería inversa de bases de datos, además posee un potente generador de informes.

1.6.2 Framework Qt 4.8.0

En el proceso de identificar las tecnologías con las que se contarán para realizar una tarea, es preciso identificar el framework con el que se trabajará. Cuando se dice framework se refiere al ambiente de trabajo y ejecución. En general los framework son soluciones completas que contemplan herramientas de apoyo a la construcción (ambiente de trabajo o desarrollo) y motores de ejecución (ambiente de ejecución).

Para la realización de este trabajo, se trabajará con Qt, puesto que es un framework multiplataforma, que se utiliza para el desarrollo de aplicaciones, está escrito en C++, y tiene como principal propósito permitir a los desarrolladores construir aplicaciones multiplataforma a partir de una misma base de código de manera rápida y sencilla. Dicho framework está compuesto por una serie de módulos que proveen funcionalidades específicas a través de una biblioteca de clases multiplataforma y el módulo con el que se trabajará es uno de bases de datos llamado: QtSQL. Es importante destacar que Qt provee poderosas herramientas de desarrollo, entre ellas destaca un completo entorno de desarrollo, llamado Qt Creator, con el cual se trabajará a la hora de realizar la implementación del plugin.

1.6.3 IDE QT Creator 2.4.1

Antes de comenzar a implementar algún proyecto o software, los desarrolladores deben conocer y haber trabajado a fondo con el IDE de programación (Integrated Development Environment) con el cual

[4] Las herramientas CASE (Computer Aided Software Engineering) son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo del sistema de información, completamente o en algunas fases.

se decida implementar. El IDE se define como un programa compuesto por un conjunto de herramientas para un programador. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica.

Para el desarrollo del plugin se trabajará con el Qt Creator, el cual es un IDE multiplataforma para desarrollar aplicaciones en C++ de manera sencilla y rápida. Cuenta con una serie de características como son: editor avanzado en C++, diseñador de formularios (GUI) integrado, herramientas para la administración y construcción de proyectos y completado automático, posee un depurador visual, una ayuda sensible al contexto integrada, resaltado y auto-completado de código.

1.7 Lenguajes a utilizar

1.7.1 Lenguaje C++

Para realizar cualquier tipo de aplicación, software o programa es necesario definir cuál lenguaje de programación se debe utilizar, dependiendo de las características de lo que se desee implementar o por otras causas que implicarán utilizar o no un lenguaje determinado. En el caso del plugin el lenguaje a utilizar es C++, ya que la herramienta HABD está implementada en este lenguaje y para una futura integración lo más adecuado sería desarrollar el plugin en este lenguaje para evitar incompatibilidades.

El mismo fue desarrollado en el año 1972 utilizando el sistema operativo UNIX, con el propósito general de facilitar la programación y realizar tareas anteriormente reservadas al lenguaje ensamblador. C++ es un lenguaje muy potente, flexible y eficaz comparado con el resto de los lenguajes orientados a objetos, características que han hecho que se le considere como lenguaje estándar dentro de la programación orientada a objetos. Este ha sido utilizado para el desarrollo de un gran número de herramientas de trabajo, sistemas operativos, compiladores, procesadores de texto, bases de datos (22).

Ofrece ventajas en cuanto a difusión, al ser uno de los lenguajes más empleados en la actualidad, posee gran número de usuarios y existe una gran cantidad de libros, cursos y sitios web dedicados al lenguaje. Su versatilidad lo caracteriza ya que es un lenguaje de propósito general y puede ser empleado para resolver cualquier tipo de problema, de muy provechosa suele ser su portabilidad porque está estandarizado y un mismo código fuente puede compilar en diversas plataformas, también es eficiente al ser uno de los más rápidos en cuanto a ejecución.

1.7.2 UML

Para modelar el sistema se utilizará el Lenguaje Unificado de Modelado (UML), el cual es un lenguaje usado para especificar, visualizar y documentar los diferentes aspectos relativos a un sistema de software bajo desarrollo, así como para modelado de negocios y almacenamiento de datos (23). Está compuesto por diferentes elementos gráficos que se combinan para formar los diagramas. Entre sus principales características está que contiene corrección de errores viables en todas las etapas, es aplicable para tratar asuntos de escala inherentes a sistemas complejos, tiempo real, cliente-servidor y los modelos permiten la comunicación con el cliente en todas las etapas. El principal beneficio de UML es que unifica distintas notaciones previas.

Hoy en día está consolidado como el lenguaje estándar en el análisis y diseño de sistemas de cómputo y provee ventajas en cuanto a diseño y documentación, descubrimiento de fallas, su código es reutilizable, ahorra tiempo en el desarrollo de software, se hacen las modificaciones de forma fácil y es más sencilla la comunicación con los programadores.

1.8 Control de versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto. Una versión, revisión o edición de un producto, es el estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación. Existen diversas herramientas creadas para realizar este proceso entre las que se destacan: CVS, SVN (Subversion), Git, Mercurial, Bazaar, LibreSource y Monotone. La línea PostgreSQL del centro DATEC tiene instalada y trabaja para gestionar y modificar sus productos, documentos y otros artefactos generados en su proyecto, como herramienta de control de versiones a Subversion.

1.8.1 Subversion

Subversion es un sistema de control de versiones libre y de código fuente abierto. Es decir, Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos (24).

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos de sus respectivas ubicaciones, fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar

todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer porque la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único, si se ha hecho un cambio incorrecto a los datos (24).

1.9 Conclusiones del Capítulo

El cumplimiento de los objetivos trazados para el presente capítulo permite arribar a las conclusiones siguientes:

- ✓ Con el análisis de herramientas de gestión de bases de datos PostgreSQL se concluye que: estas no proporcionan al usuario una opción que muestre de forma clara el rendimiento de las consultas, proponiéndose el desarrollo de un plugin para el análisis de las consultas SQL para la herramienta HABD.
- ✓ El estudio de las técnicas relacionadas con la optimización de consultas SQL permite concluir que para la realización del plugin se escogen las sugerencias brindadas por usuarios de los sistemas gestores de bases de datos MySQL y PostgreSQL, además de los consejos generales expuestos en el capítulo.
- ✓ Se detallan las principales características de herramientas de desarrollo que son previamente condicionadas ya que son con las cuales se desarrolló HABD como: el framework Qt 4.7, el IDE Qt Creator 2.4, el lenguaje de programación C++; así como la metodología XP para lograr una mejor comprensión y utilización en la Fase de Implementación del sistema.
- ✓ Para el modelado se escoge el lenguaje UML y se propone como herramienta CASE el Visual Paradigm 8.0 que soporta UML y es multiplataforma.

Capítulo 2

Descripción de la solución.

Introducción

Describir la solución, a partir de las transformaciones de los requisitos necesarios en una correcta implementación, será el objetivo fundamental del presente capítulo. En consecuencia, se obtendrán un conjunto de artefactos que serán de gran valor para las posteriores etapas del desarrollo como son: las historias de usuarios, diagramas de clases y dominio y tarjetas CRC. Igualmente se realizará, un estudio detallado de los patrones de arquitectura y diseño que pudieran ser aplicados para obtener una solución más flexible, donde se facilite la reutilización del código y la fácil comprensión del mismo.

2.1 Modelo de dominio

Dada la propuesta de la implementación del plugin Análisis del rendimiento de las consultas en PostgreSQL para HABD se propone la definición de un modelo de dominio o modelo conceptual, a pesar que la metodología XP no genera este artefacto, el cual muestra los principales conceptos a utilizar en el desarrollo. Para lograr una mayor comprensión del dominio del problema, se crea el artefacto modelo conceptual o de dominio, que no es más que una representación visual de los conceptos u objetos del mundo real que son significativos para el problema o el área que se analiza; representando las clases conceptuales, no los componentes de software. Puede verse como un modelo que comunica a los interesados, cuáles son los términos importantes y cómo se relacionan entre sí (25). En la siguiente figura se muestra el modelo de dominio del plugin:

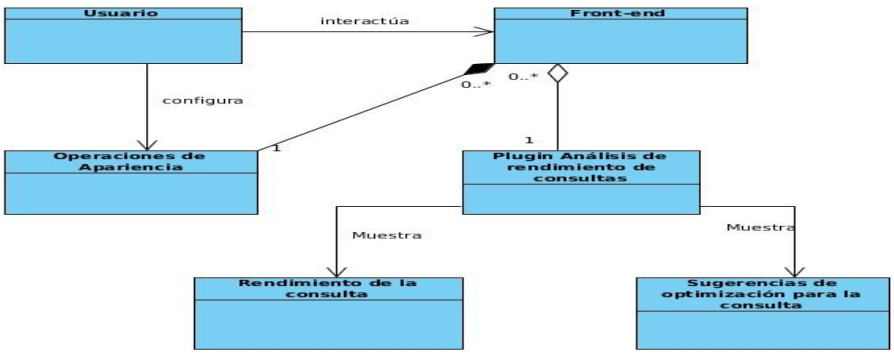


Figura 5: Modelo de dominio.

Se describen a continuación las clases conceptuales de modelo de dominio:

Usuario: Representa a las personas que hacen uso de la herramienta.

Front-End: Representa la interfaz principal de la herramienta, la cual brinda al usuario las opciones tanto de apariencias, como la posibilidad de la utilización de plugin Análisis de rendimiento y demás funcionalidades.

Plugin Análisis de rendimiento de consultas: Se refiere al plugin a desarrollar, el cual es el objetivo de este trabajo.

Opciones de apariencia: Representa las opciones de apariencia que posee la herramienta, las cuales el usuario tiene la posibilidad de utilizar.

Rendimiento de la consulta: Representa una vista que se muestra al usuario, donde se visualiza el rendimiento de la consulta.

Sugerencias de optimización para la consulta: Representa una vista que se muestra al usuario, donde se visualizan las sugerencias de optimización de consultas, con el fin de poder aplicar alguna de las sugerencias mostrar para mejorar la consulta.

2.2 Propuesta del sistema

El plugin de análisis de rendimiento de las consultas realizadas en PostgreSQL para la herramienta de administración de bases de datos HABD, debe permitir que el usuario observe una serie de datos que posibilitarán un mejor entendimiento de la ejecución de las consultas que se deseen analizar. El plugin debe poseer un campo de texto donde se escribirá la consulta a analizar. También debe poseer dos botones: uno para ejecutar las funcionalidades implementadas y otro para reiniciar todos los campos mostrados. Luego de oprimir el botón que ejecutará toda la implementación, la aplicación debe mostrar una interfaz con tres secciones que pueden ser pestañas: una con una tabla que muestre la información de la ejecución de cada nodo de la consulta, otra con un campo de texto donde se visualice la ejecución de la consulta de forma general y por último un campo de texto mostrando posibles sugerencias para mejorar la consulta analizada y así optimizar el rendimiento de la misma. Con todos estos datos que se brinden, el usuario será capaz de tener un mejor entendimiento de cómo está trabajando la consulta.

2.3 Historias de usuarios

Las HU son la forma en que se especifican en XP los requisitos del sistema. Estas se escriben desde la perspectiva del cliente, aunque los desarrolladores pueden brindar también su ayuda en la identificación de las mismas (26). Estas administran los requisitos de los usuarios sin tener que

elaborar gran cantidad de documentos formales sin requerir de mucho tiempo para administrarlos. El contenido de estas debe ser concreto y sencillo.

El tratamiento de las HU es muy dinámico y flexible, en cualquier momento las historias de usuario pueden eliminarse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada HU es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (27).

Los contenidos de las fichas de las HU quedan estructurados de la siguiente forma:

Número: A cada HU se le asigna un número para facilitar su identificación por parte del equipo de desarrollo.

Nombre: Nombre descriptivo de la HU.

Usuario: Nombre de quien desarrolla la HU.

Prioridad en negocio: Grado de prioridad que tiene con respecto al negocio.

Riesgo en desarrollo: Grado de complejidad que le asigna el equipo de desarrollo a la HU luego de analizarla. (Alto, Medio o Bajo).

Puntos estimados: Unidades de tiempo estimadas por el equipo de desarrollo para darle cumplimiento a la HU. Una unidad de tiempo equivale a una semana de trabajo de 40 horas.

Puntos reales: Unidades de tiempo en que el equipo de desarrollo da cumplimiento a la HU.

Iteración asignada: Número de la iteración en la cual será implementada la Historia de Usuario.

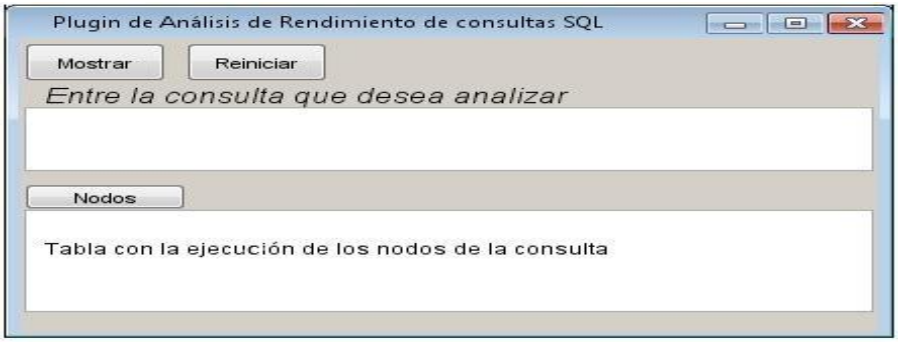
Descripción: Descripción simple sobre lo que debe hacer la funcionalidad en cuestión.

Observaciones: Descripción de alguna observación referente a la HU.

Prototipo de interfaces: Imagen de una futura interfaz de la HU, en caso que tenga.

Se definieron tres Historias de Usuario para darle cumplimiento a la implementación, las cuales se muestran a continuación, además de la ficha de la primera:

- **HU-1:** Mostrar tabla de rendimiento de la consulta.
- **HU-2:** Mostrar plan de ejecución general de la consulta.
- **HU-3:** Mostrar sugerencias de optimización de consultas.

<table>
<tr><td colspan="2">Historia de Usuario</td></tr>
<tr><td>Número: 1</td><td>Nombre de la Historia de Usuario: Mostrar tabla de rendimiento de la consulta.</td></tr>
<tr><td colspan="2">Cantidad de modificaciones a la Historia de Usuario: 3</td></tr>
<tr><td>Usuario: Vladimir Ricardo Julián</td><td>Iteración asignada:1</td></tr>
<tr><td>Prioridad en negocio: Muy Alta</td><td>Puntos estimados: 3</td></tr>
<tr><td>Riesgo en desarrollo: Muy Alto</td><td>Puntos reales: 3</td></tr>
<tr><td colspan="2">Descripción: Se realiza un análisis de los nodos de la consulta en una tabla, identificando mediante colores los nodos que presentan mayores problemas, así como otros datos de interés.</td></tr>
<tr><td colspan="2">Observaciones:</td></tr>
<tr><td colspan="2">Prototipo de interfaces:
 
</td></tr>
</table>

2.4 Lista de reserva del producto

La lista de reserva del producto según la metodología XP representa todos los requisitos funcionales y no funcionales definidos para el desarrollo de una aplicación. Los mismos están ordenados de acuerdo a la prioridad e importancia que tengan en la implementación del sistema, además de la estimación del

tiempo en semanas y el rol del que realiza la estimación. La lista de reserva de producto definida para la implementación de plugin que se propone se encuentra en el anexo #3 de este trabajo de diploma.

2.5 Tareas de ingeniería

Las tareas de ingeniería son las encargadas de darle cumplimiento a las Historias de Usuario para la implementación del sistema. Son elementos más detallados de cada historia de usuarios. A continuación se describen los contenidos de la ficha de las tareas de ingeniería:

Número Tarea: A cada una de las tareas se le asigna un número para facilitar su identificación.

Número de la historia de usuario: Número de la HU a la que está vinculada la tarea.

Nombre Tarea: Nombre descriptivo de la tarea.

Tipo de Tarea: Define el tipo de tarea, ya sea de programación, análisis u otro tipo.

Puntos Estimados: Tiempo en semanas estimadas para el cumplimiento de la tarea.

Fecha Inicio: Fecha del comienzo de la tarea.

Fecha Fin: Fecha del fin de la tarea.

Programador Responsable: Nombre de la persona encargada de darle cumplimiento a la tarea.

Descripción: Breve descripción de la tarea en cuestión.

Se definieron siete tareas de ingeniería para darle cumplimiento a las HU, las que se muestran a continuación, seguida de la ficha de la segunda tarea:

- Capturar la consulta SQL.
- Identificar los nodos críticos.
- Mostrar tabla con los datos de la consulta.
- Mostrar plan de ejecución de la consulta general.
- Cargar las sugerencias de optimización.
- Analizar la consulta.
- Mostrar sugerencias.

Tarea de Ingeniería	
Número Tarea: 2	**Número Historia de Usuario: 1**
Nombre Tarea: Identificar los nodos críticos.	
Tipo de Tarea : Desarrollo	**Puntos Estimados: 1**
Fecha Inicio: 19/3/2012	**Fecha Fin: 23/3/2012**
Programador Responsable: Vladimir Ricardo Julián	
Descripción: Se analizan todos los tiempos de ejecución de cada nodo de la consulta y se calcula el por ciento que representa con respecto al tiempo total de la consulta, luego los nodos toman un color que representa el por ciento (rojo>90%, naranja>50 y amarillo>10) y que da la medida de su rendimiento en la consulta.	

2.6 Plan de iteraciones

El objetivo principal de este plan es definir detalladamente para cada una de las iteraciones a realizarse un conjunto de tareas, actividades y recursos. Para cada iteración existe una serie de objetivos los cuales son usados como: referencia de evaluación para determinar diferentes aspectos, grado de terminación de una determinada función, rendimiento y niveles de calidad. Es importante destacar que para cada plan de iteración es necesario detallar el tiempo estimado.

Release	Descripción de la iteración	Orden de la HU a implementar	Duración total

1	En esta iteración se realiza un análisis del resultado de la ejecución de la consulta, señalándose los nodos que presentan problemas y mostrándose el plan de ejecución de la consulta.	1, 2	4 semanas
2	En esta iteración se brindan posibles sugerencias para mejorar el rendimiento de la consulta.	3	3 semanas

2.7 Tarjetas Clase, Responsabilidades y Colaboración (CRC).

Las tarjetas CRC son una técnica de diseño de software orientado a objetos. Fueron propuestas por Ward Cunningham y Kent Beck. Se utilizan normalmente cuando primero se determinan las clases que se necesitan y cómo van a interactuar y después se implementa la solución.

La técnica de las tarjetas CRC se puede usar para guiar el sistema a través de análisis dirigidos por la responsabilidad. Las clases se examinan, se filtran y se refinan en base a sus responsabilidades con respecto al sistema, y con las que necesitan colaborar para completar sus responsabilidades. Es importante resaltar que esta tarea es permanente durante la vida del proyecto partiendo de un diseño inicial que va siendo corregido y mejorado en el transcurso de este (26).

Una sesión CRC es una técnica donde los miembros del equipo se reúnen en una habitación con una mesa de trabajo de tamaño preferiblemente mediano o grande, las tarjetas CRC en blanco son puestas encima y estos son los encargado mediante la conversación y retroalimentación de ideas, escribir los datos pertenecientes a los nombres de las clases, sus responsabilidades y colaboraciones. Esta técnica permite la total interacción de los miembros del equipo a la hora de resolver un problema y es un tipo de tormenta de ideas. Durante la sesión las tarjetas son movidas de un lado hacia otro y las responsabilidades y colaboraciones pueden cambiar también, lo que se desea lograr es que todos los participantes aporten buenas ideas para el desarrollo del sistema (26).

Características que presentan las tarjetas CRC (26):

- Es una técnica para la representación de sistemas orientados a objetos, para pensar en objetos.
- Son un puente de comunicación entre diferentes participantes.
- Principales desventajas: lentitud y roces.
- Permite ver las clases como algo más que un repositorio de datos.

- Posibilita conocer el comportamiento de cada clase en un alto nivel.

Las partes que componen una tarjeta CRC son las siguientes:

- Clase: Nombre de la clase con que se está modelando.
- Responsabilidades: Descripción de alto nivel del propósito de la clase.
- Colaboración: Indica con qué otras clases se requiere relación para cumplir con las responsabilidades.

Estas tarjetas CRC se hacen con el objetivo de identificar jerarquías de generalización/especificación, o jerarquías de agregación entre las clases, de manera que ayuda al refinamiento de estas. Se definen además con la finalidad de obtener un diseño simple y no incurrir en la implementación de características que no son necesarias.

A continuación se presenta una de las tarjetas CRC pertenecientes al plugin:

Tarjeta CRC	
Clase: Control	
Responsabilidades	**Colaboraciones**
Identifica nodos críticos. Calcula error del planificador. Muestra plan de ejecución de la consulta. Muestra sugerencia.	Nodos

2.8 Diagrama de clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. Para la realización del plugin se decide realizar el diagrama de clases para así tener un mejor entendimiento del sistema, a pesar de que la Metodología XP no genera este tipo de artefacto. La figura a continuación representa el diagrama de clases del plugin:

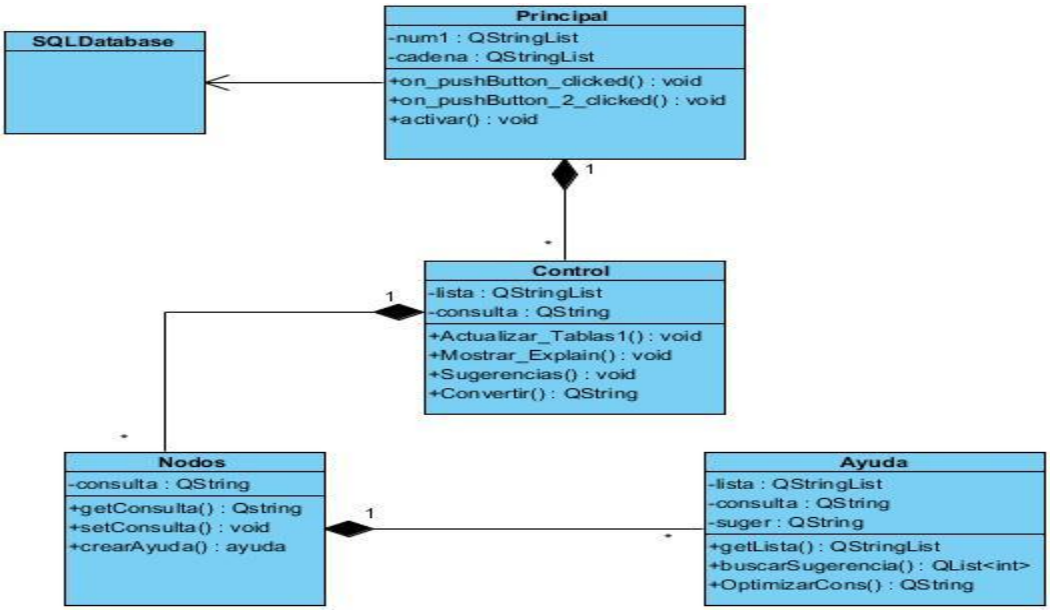


Figura 6: Diagrama de clases.

2.9 Patrones de arquitectura

Los patrones arquitectónicos, o patrones de arquitectura, son patrones de diseño que ofrecen soluciones a problemas de arquitectura en la ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor.

El Patrón Arquitectónico es el nivel en el cual la arquitectura de software:

- Define la estructura básica de un sistema, pudiendo estar relacionado con otros patrones
- Representa una plantilla de construcción que provee un conjunto de subsistemas aportando las normas para su organización.

Ejemplos: Capas, MVC, Tuberías y Filtros, Pizarra

2.9.1 Arquitectura en capas

Hoy en día la arquitectura en capas es uno de los patrones más populares y usados en el mundo, ya que permite distribuir el trabajo de una aplicación por niveles. Estilo arquitectural en capas, se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva

de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan (28).

En este trabajo de diploma se hace uso de la arquitectura en capas, específicamente tres capas, con el objetivo de que el sistema posea una correcta organización, para así tener un orden lógico en la programación de este. Se describen a continuación las tres capas:

Capa de presentación: Esta es la capa con la cual el usuario interactúa directamente, además contiene todas las clases visuales que posee la aplicación. La capa de presentación solamente se comunica con la capa de negocio (29). La clase que compone esta capa, es la clase visual Principal la cual es la encarga de mostrar todos los componentes visuales con los cuales el usuario interactúa y/o trabaja.

Capa de negocio: Esta es la capa que recibe todas las peticiones del usuario y es donde se establecen todas las reglas que deben cumplirse. Esta capa es la intermediaria entre la capa de presentación y la capa de información (29). En el caso de este trabajo la clase Control es la que ocupa el espacio en esta capa ya que, es la responsable de implementar todas las operaciones solicitadas por el usuario a través de la capa presentación.

Capa de acceso a datos/recursos: Esta es la capa encargada de gestionar toda la información, ficheros planos, XML, SGBD, entre otros (29). Es necesario aclarar que la aplicación desarrollada no trabaja con una base de datos, sino con un fichero plano que contiene un tipo de información que forma parte del resultado que se obtiene, siendo la clase Ayuda la encargada de realizar esta operación y a la vez la componente de la capa acceso a datos/recursos.

La figura a continuación representa un diagrama de paquetes que ejemplifica la arquitectura en tres capas de la aplicación:

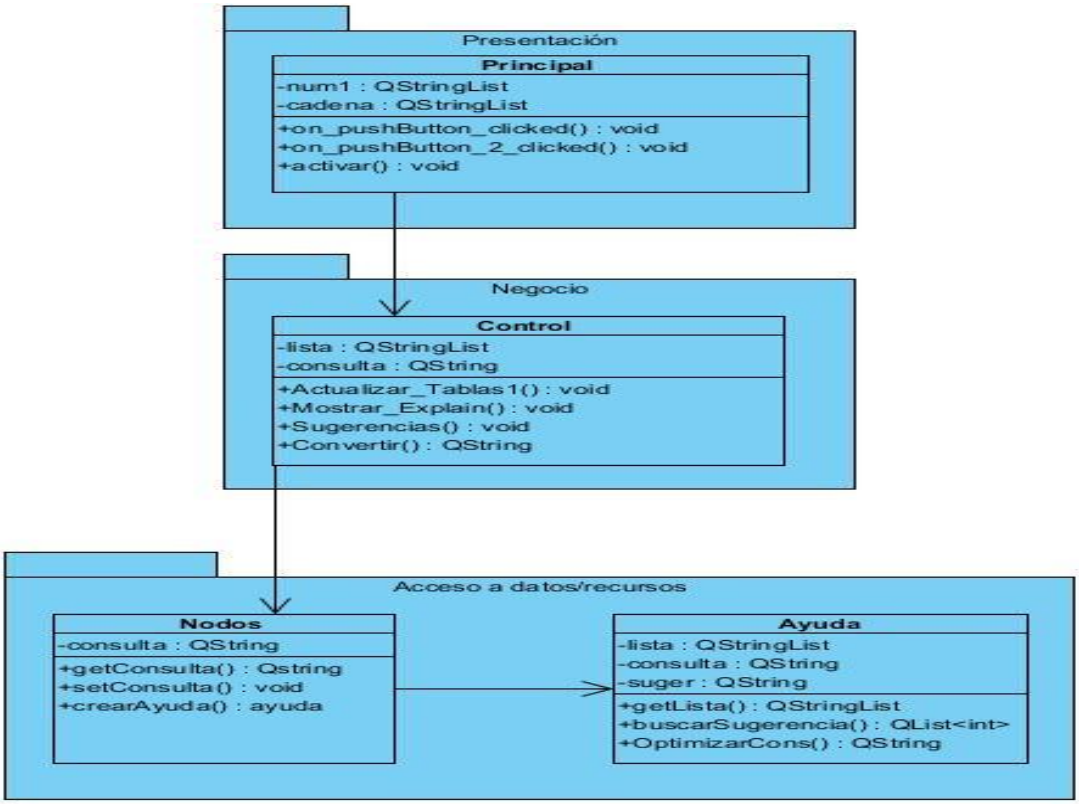


Figura 7: Arquitectura en capas.

2.10 Patrones de diseño

Los patrones de diseño son descripciones de clases cuyas instancias colaboran entre sí. Cada patrón es adecuado para ser adaptado a un cierto tipo de problema. Representa un esquema o micro-arquitectura que supone una solución a problemas (dominios de aplicación) semejantes; una estructura común que tienen aplicaciones semejantes (30).

Los patrones de diseño brindan una solución generalmente ya probada y documentada a problemas que se dan durante el proceso de desarrollo de software. Emplean un conjunto de buenas prácticas que facilitan el trabajo, definen una estructura de clases que da respuesta a uno o varios problemas en particular y presentan la ventaja de que son fáciles de comprender, además de que no dependen del lenguaje, haciéndolos genéricos. Lo complejo es cuando se tiene que decidir cuál usar, pues presentan diferentes soluciones, ya sea a través del empleo de uno u otro, o la combinación de varios. De ahí la importancia de conocer y estudiar los diferentes patrones que existen para poder determinar su uso.

Entre la familia de patrones se puede encontrar los **GRASP"**General Responsibility Assignment Software Patterns**"** y los **GOF"**Abstract Factory**"**. Los **GOF** describen 23 patrones de diseño

comúnmente utilizados y de gran aplicabilidad en problemas de diseño usando modelamiento UML, los que son agrupados en 3 categorías: Creacionales, Estructurales y de Comportamiento.

2.10.1 Patrones GRASP (Del inglés General Responsibility Assignment Software Patterns).

Describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (31).

Experto: Asignar una responsabilidad al más competente en información, la clase cuenta con la información necesaria para cumplir la responsabilidad. Es el principio básico de asignación de responsabilidades que suele utilizarse en el diseño orientado a objetos (31).

Se utiliza de manera implícita en todas las clases del plugin, pues las mismas están diseñadas para que tengan bien definidas sus responsabilidades. Cada uno de los componentes creados para desarrollar la aplicación tiene las bases para el cumplimiento de este patrón, ya sean en los modelos, las vistas o las clases controladoras. Por convención se define que en las Vistas se desarrolla lo que será mostrado al usuario, en el Modelo, por ejemplo es donde se garantiza las validaciones y en los Controladores, donde se definen las funcionalidades.

Controlador: Asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas. Asigna la responsabilidad del manejo de mensajes de los eventos de un sistema a una clase. Garantiza que los procesos sean manejados por la capa Controladora y no por la capa de presentación (31).

La utilización de este patrón se evidencia en la no existencia de llamadas a la base de datos desde las vistas implementadas, sino a través de la clase controladora de cada entidad definida, por ejemplo la clase Control.

Figura 8: Clase Control (donde se evidencia el patrón controlador).

Alta Cohesión: Asignar una responsabilidad de modo que la unión se mantenga a gran escala. Asignar a las clases responsabilidades que trabajen sobre una misma área de aplicación y que no tengan mucha complejidad (31).

Caracteriza a las clases con responsabilidades estrechamente relacionadas que no realizan un trabajo enorme. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande. Este patrón se evidencia en la clase ayuda.

Figura 9: Clase ayuda (donde se evidencia el patrón Alta cohesión).

Bajo Acoplamiento: Asignar una responsabilidad para mantener un engranaje pobre. Es un principio que se debe recordar durante las decisiones de diseño. Soporta el diseño de clases más independientes. Asignar las responsabilidades de forma tal que las clases se comuniquen con el menor número de clases que sea posible (31).

Este patrón se evidencia en la clase ayuda, la misma no depende de otras clases para realizar sus funciones. Ejemplo:

Figura 10: Patrón bajo acoplamiento.

Creador: Es el encargado de guiar la asignación de responsabilidades relacionadas con la creación de objetos o instancias de una clase, el mismo se evidencia ya que, la clase nodos es la responsable de crear un objeto de la clase ayuda con el fin de tener acceso a las funcionalidades de la misma, ejemplo:

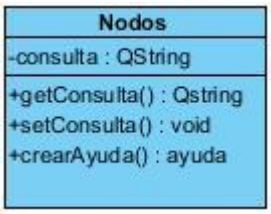


Figura 11: Patrón creador.

2.11 Análisis de la consulta

En el análisis de la consulta se tuvieron en cuenta tres aspectos fundamentales los cuales son necesarios exponer para la comprensión de plugin de análisis de rendimiento. También se debe tener presente ya que es de importancia a la hora de brindar las sugerencias de optimización de la consulta, los mismos se abordan a continuación:

- **El tiempo de ejecución de cada nodo en la consulta**

Al aplicar el comando ***EXPLAIN ANALYZE*** a la consulta se obtiene el plan de ejecución de la consulta, donde son analizados los tiempos de ejecución de cada nodo para conocer si la consulta presenta problemas de rendimiento. La forma de identificar el bajo rendimiento es calculando el porciento que representa el tiempo de cada nodo con respecto al tiempo de ejecución total de la consulta. Si el porciento de un nodo es mayor que 10, quiere decir que la ejecución de nodo se está demorando más de lo normal y el mismo no se ejecute de forma eficiente, ya que lo normal sería que su tiempo fuera menor que el diez porciento. Se definió una escala dependiendo del porciento: si es mayor que 10 y menor que 50, mayor que 50 y menor que 90, y mayor de 90, los cuales son representados con un color distinto: amarillo, naranja y rojo, dando una medida del rendimiento negativo. En el plugin cuando se muestra la información de los nodos de la consulta se muestra claramente lo planteado, en caso de que los nodos se muestren con color blanco significa que la consulta presenta buen rendimiento y en la pestaña de sugerencias del plugin aparecerá en blanco, esto debido a que la consulta no presenta problema.

- **La sintaxis de la consulta**

Esta es la parte principal para poder brindar las sugerencias que permitan mejorar el rendimiento de la consulta, en caso de que la consulta presente bajo rendimiento, lo cual se identifica en los datos que se muestran de los nodos. La estrategia que se aplicó fue trabajar con una funcionalidad que tiene el IDE con el cual se trabajó: Qt Creator. Dicha funcionalidad es el ***contains*** que es aplicado a las

QString, ya que la consulta fue tratada de esta forma para poder acceder a la información que nos brindó la ejecución del comando ***EXPLAIN ANALYZE*** a la consulta, de esta forma se filtran las palabras reservadas del lenguaje SQL que conforman la consulta para luego así ofrecer las sugerencias que permitan optimizar y mejorar el rendimiento de la consulta. Por citar un ejemplo se tiene el uso del ***ORDER BY***, ya que cuando analizamos una consulta que presenta esta instrucción. Al analizar la siguiente consulta: ***SELECT*** *a.attnum,a.attname* ***as*** *campos, pg_catalog.format_type(a.atttypid, a.atttypmod)* ***as*** *tipo, (****SELECT*** *substring(pg_catalog.pg_get_expr(d.adbin, d.adrelid) for 128)* ***FROM*** *pg_catalog.pg_attrdef d* ***WHERE*** *d.adrelid = a.attrelid* ***AND*** *d.adnum = a.attnum* ***AND*** *a.atthasdef)* ***as*** *pordefecto, a.attnotnull* ***as*** *NULO* ***FROM*** *pg_catalog.pg_attribute a* ***WHERE*** *a.attrelid = (****select*** *oid* ***from*** *pg_class* ***where*** *relname='tabla')* ***AND*** *a.attnum > 0* ***AND NOT*** *a.attisdropped* ***ORDER BY*** *a.attnum,* el plugin muestra una sugerencia donde referencia el impacto en el rendimiento que tiene el uso de este.

El error del planificador

Una de las informaciones que nos brinda el plugin es el error del planificador, donde muestra las filas que retornará cada nodo con respecto a las estimadas por el planificador. En este caso también se establece una escala dependiendo del valor que se muestre de este: mayor que 10 y menor que 100, mayor que 100 y menor que 1000 y mayor que 1000, los cuales se pintan de color amarillo, naranja y rojo respectivamente. Además de resaltar el valor con los colores mencionados, siempre aparece delante del valor una flecha: puede ser hacia abajo o arriba. Esta flecha significa si el planificador sobrestima o subestima el número de filas. Sobrestima cuando el número de filas estimadas es mayor que las recorridas realmente y subestima de forma viceversa. El cálculo del error se realiza dividiendo el mayor número entre las filas estimadas y recorridas entre el menor.

2.12 Interfaces de usuario

Luego de culminar la implementación se presentan las principales interfaces de la aplicación, las cuales se explicarán detalladamente. A continuación se presenta la primera vista:

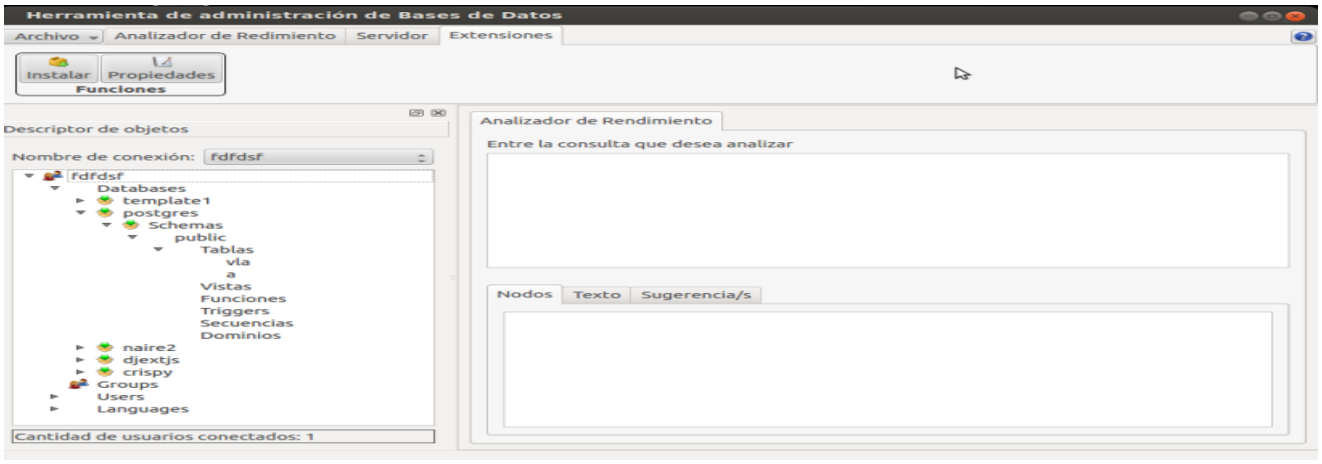


Figura 12: Interfaz principal del plugin.

En la figura se puede apreciar el campo de texto a través del cual se entra la consulta analizar, y en la parte inferior tres pestañas las que se llenarán posteriormente, mostrando el rendimiento de la consulta. Luego de accionar el botón “Mostrar” se llenan las tres pestañas. La primera muestra una tabla con el rendimiento de la consulta, la cual se presenta a continuación:

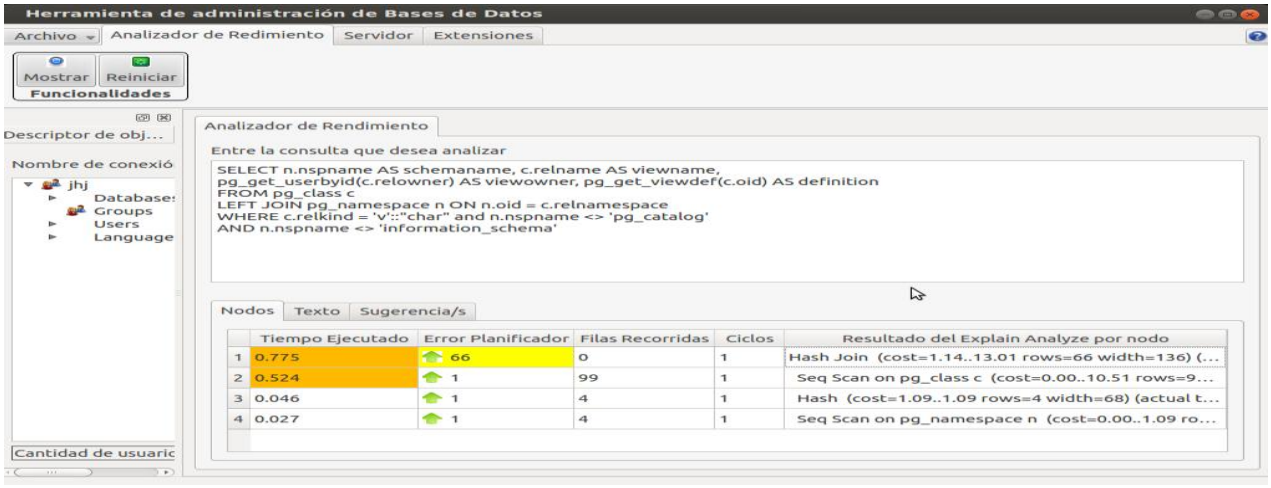


Figura 13: Vista de la pestaña Nodos.

En esta interfaz se puede apreciar que los dos primeros nodos están pintados de color naranja, lo que representa que el tiempo de ejecución de estos se encuentra en el rango del 50 al 90 por ciento con respecto al tiempo total de la consulta. Además el valor del error de panificador de primer nodo está de color amarillo, lo que significa que el valor del mismo es mayor que 10 y menor que 100. En la pestaña texto se muestra el plan de ejecución de la consulta completa, en la figura que se presenta a continuación se verifica lo planteado:

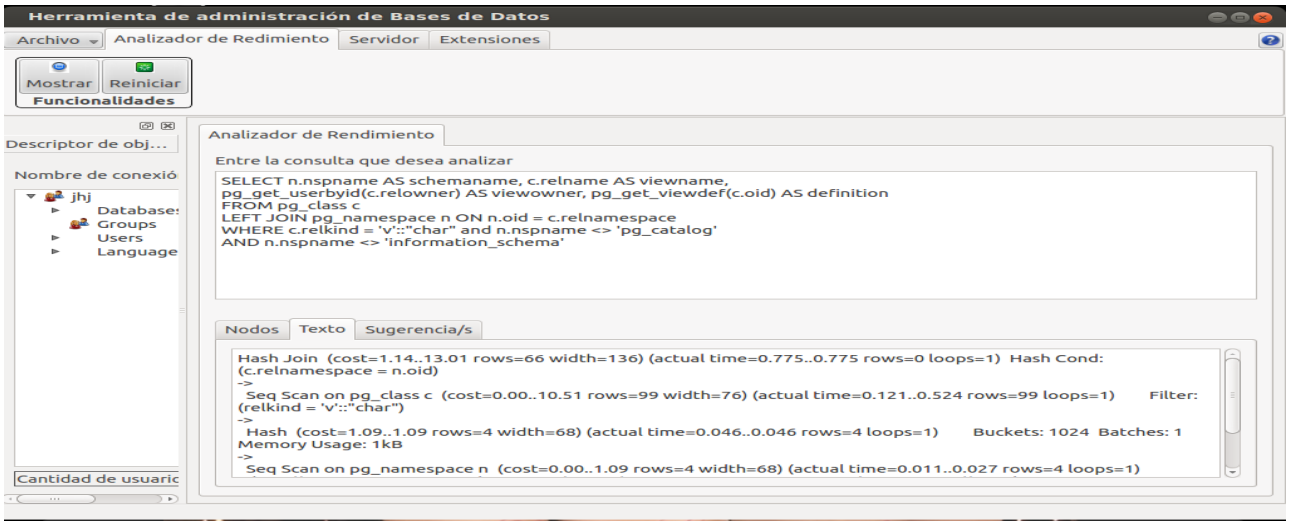


Figura 14: Vista dela pestaña Texto.

Por último se muestra la vista de la pestaña Sugerencias, la cual muestra recomendaciones para optimizar la consulta analizada:

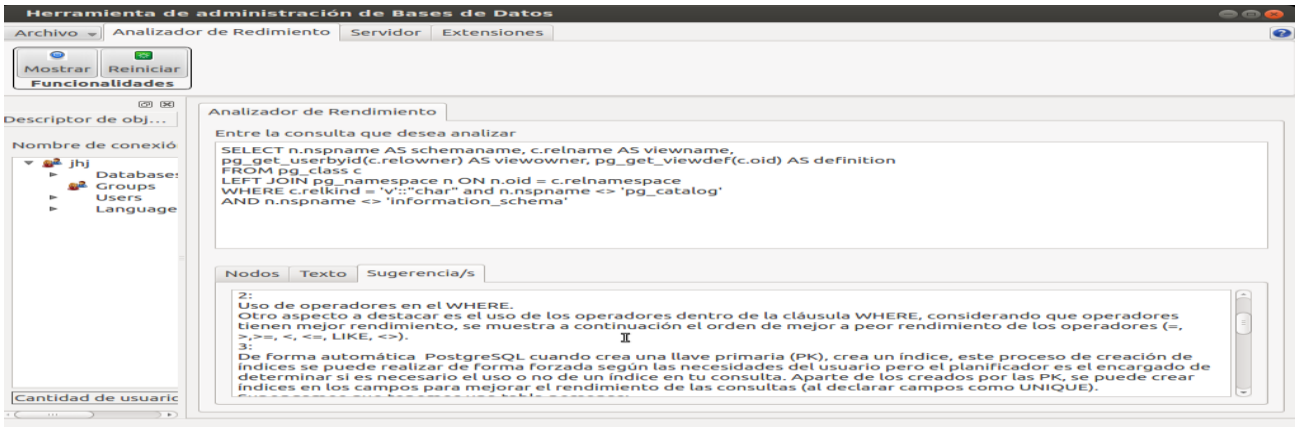


Figura 15: Vista de la pestaña Sugerencias.

2.13 Estándares de codificación

En el proceso de implementación hay que tener presente los estándares de codificación definidos por el desarrollador, los cuales son reglas que se siguen para la escritura de código fuente, de tal manera que a los programadores se les facilite entender el código.

Para el desarrollo del plugin se definen reglas propias del lenguaje C++ a utilizar en la implementación que se muestran a continuación:

Identación

La unidad de identado es de 3 espacios. El uso de la tabulación debe ser evitado no existe un estándar que determine con precisión el ancho que va a producir la tabulación.

Longitud de la Línea

Evitar líneas con más de 80 caracteres. Cuando una sentencia sobrepase una línea simple del editor, esta deber ser separada en otras. Se debe realizar la separación después de un operador, preferentemente luego de una coma. Realizar la ruptura después de un operador disminuye la probabilidad de que al realizar un copiado del código se cometa un error por olvido de la inserción del punto y coma. La siguiente línea debe ser identada con 5 espacios.

Comentarios

Es conveniente dejar información que pueda ser leída tiempo después por personas que necesiten entender qué fue lo que se hizo en el fragmento de código. Los comentarios deben ser escritos correctamente y claros. Generalmente deben usarse comentarios de una sola línea. Reserve los comentarios de bloques para la documentación formal o para comentar porciones de código.

Declaración de variables

Cada variable debe de ser declarada en una línea y comentada. También deben aparecer ordenadas alfabéticamente:

```
//variable con las que trabajo
float celda2;
QString ciclos;
QString estimado;
QString filaReal;
QIcon flecha;
float porciento;
int valor=0;
```

Figura 16: Ejemplo de declaración de variables.

El nombre de las variables debe de comenzar con letras minúsculas y cada palabra relevante por la que esté compuesta debe ser con letra mayúscula. Ejemplo: filaReal.

Declaración de funciones

No debe haber espacio entre el nombre de la función y el paréntesis izquierdo, ni entre este y la lista de parámetros. Debe haber un espacio entre el paréntesis derecho y la llave de comienzo del cuerpo de la función. Las sentencias del cuerpo deben estar en la línea siguiente. La llave que cierra debe

estar alineada con la línea de declaración de la función. Los nombres de las funciones se rigen por las mismas características que el de las variables.

Identificadores

Los identificadores pueden estar formados por cualesquiera de las 26 letras minúsculas o mayúsculas (A .. Z, a... z), los 10 dígitos (0… 9) y el carácter subrayado "_". Debe evitarse el uso de caracteres internacionales (ej.: ñ, ü) porque no siempre pueden ser leídos o entendidos correctamente en todos los lugares. No se debe usar el símbolo dólar "$" o la barra invertida "\" en los identificadores.

Sentencias

Sentencias Simples

Cada línea debe contener como máximo una sentencia. Se debe poner un punto y coma ";" al final de cada sentencia simple. Una sentencia de asignación puede resultar en la asignación de una función o de un objeto como literal y en todos los casos como sentencia de asignación debe estar finalizada con un punto y coma.

Sentencias Compuestas

Las sentencias compuestas son aquellas sentencias que contienen una lista de sentencias encerradas entre llaves:

- Las sentencias encerradas deben ser identadas a 4 espacios.
- La llave que inicia la lista de sentencias debe estar al final de línea de la sentencia compuesta.
- La llave que termina la lista de sentencias debe estar al comienzo de una línea y guardar la misma identeación que la sentencia compuesta en correspondencia con la llave que inicia.
- Las llaves siempre serán usadas para listar todas las sentencias, aunque se trate de una sola, cuando son parte de una estructura de control como if o for. Ello facilita agregar nuevas sentencias sin la introducción accidental de errores.

Etiquetas

Las sentencias etiquetadas son opcionales. Solo estas sentencias deben ser etiquetadas: while, do, for, foreach, switch. Se muestra en la figura siguiente el ejemplo de un while:

```
61      QString texto = "";
62      while (query.next()) {
63        texto += query.value(0).toString();
64      }
65      cadena = texto.split(" ");
66      nodos1= texto.split("->");
67
68      Actualizar_Tablas1();
69      Mostrar_Explain();
70      Sugerencias();
71    }
```

Figura 17: Ejemplo de un while.

Sentencia return

Una sentencia return no debe utilizar paréntesis “()” alrededor del valor que se retorna. La expresión cuyo valor se retorna debe comenzar en la misma línea que la palabra reservada return, terminada con un punto y coma.

Sentencia if

La sentencia if debe ser escrita de esta manera, Ejemplo:

```
262         if (celda2<=10) {
263             ui->tableWidget->item(valor,1)->setBackgroundColor(bla);
264         }
265         if (celda2>10 && celda2<100) {
266             ui->tableWidget->item(valor,1)->setBackgroundColor(ama);
267         }
268         if (celda2>100 && celda2<1000) {
269             ui->tableWidget->item(valor,1)->setBackgroundColor(QColor(255,186,0));
270         }
271         if (celda2>1000) {
272             ui->tableWidget->item(valor,1)->setBackgroundColor(rojo);
273         }
274
275         valor++;
```

Figura 18: Ejemplo de un if.

Estructuras repetitivas

Las estructuras repetitivas deben ser escritas de esta manera:

Ejemplo de un for:

```
290     for(int i=0;i<nodos.length();i++){
291
292         p.append(nodos.at(i)+"\n"+ "->" +"\n" );
293
294     }
295     ui->plainTextEdit_2->setPlainText(p);
```

Figura 19: Ejemplo de un for.

Espacios en blanco

Las líneas en blanco facilitan la lectura determinando secciones de código lógicamente relacionada. Los espacios en blanco pueden o no ser utilizados en las siguientes circunstancias:

- ✓ No se debe utilizar un espacio en blanco entre el identificador de una función y el paréntesis que abre a la lista de parámetros. Ello ayuda a distinguir entre palabras reservadas y llamadas a funciones.
- ✓ Para cualquier operador binario excepto el punto ".", el paréntesis que abre "(" y el corchete que abre "[" todos deben ser separados por un espacio entre operandos y operador.
- ✓ No se debe utilizar el espacio para separar un operador unario de su operando excepto cuando ese operador es una palabra como typeof.
- ✓ Cada punto y coma ";" en una sentencia de control debe ser seguido por un espacio.
- ✓ Debe dejarse un espacio luego de cada coma ",".

Slot

Los slot deben de empezar con slt_ y las señales con sgn_.

Conclusiones del capítulo

Con la culminación de este capítulo permite concluir que:

- ✓ La creación y descripción de los artefactos planteados por la metodología XP como las historias de usuarios, las tareas de ingeniería y tarjetas CRC, permitió el comienzo de la implementación del plugin propuesto.
- ✓ La definición del plan de iteraciones, la identificación y aplicación de los patrones tanto de arquitectura como de diseño, permitió una mejor organización de la implementación y agilizó el trabajo a los desarrolladores.
- ✓ Se implementó el plugin de análisis de rendimiento de las consultas para la herramienta HABD.

Capítulo 3

Validación del sistema.

Introducción

El presente capítulo tiene como objetivo fundamental desarrollar pruebas de aceptación al plugin implementado definidas por la metodología XP, con la misión de comprobar que las funcionalidades del plugin se ejecuten de forma satisfactoria.

2.14 Estrategia de prueba

Uno de los pilares de la metodología XP es el proceso de pruebas. La misma anima a probar constantemente tanto como sea posible. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones.

Las pruebas de aceptación son creadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. Para asegurar el funcionamiento final de una determinada historia de usuario se deben crear “Pruebas de aceptación"; estas pruebas son creadas y usadas por los clientes para comprobar que las distintas historias de usuario cumplen su cometido, especificando uno o diversos escenarios de prueba para comprobar que una historia de usuario ha sido correctamente implementada (32).

Una HU puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de estas pruebas es garantizar que los requisitos han sido cumplidos y que el sistema es aceptable (26).

2.15 Método seleccionado

Las pruebas de aceptación son consideradas como “pruebas de caja negra” (“Black box system tests”). Asimismo, en caso de que fallen varias pruebas, se debe indicar el orden de prioridad de resolución. Una HU no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación (32).

Existen dos enfoques muy utilizados en los diseños de pruebas: enfoques de caja negra y caja blanca.

Caja negra: Comprueba el correcto funcionamiento de los componentes de sistema de información, analizando las entradas y salidas, y verificando que el resultado es el esperado.

Caja blanca: Verifica la estructura interna del componente, con el objetivo de probar las líneas de falla más probables de la aplicación.

Se escoge el método de caja negra ya que estas se centran en los requisitos funcionales del software, comprueban que la funcionalidad del programa o sistema es completamente operativa, que la entrada se acepta de forma adecuada y la salida es correcta, además que verifican que la integridad de la información interna se mantiene. La ejecución del método de caja negra permite encontrar errores típicos como: funciones incorrectas o ausentes, errores de interfaz, errores de estructura de datos o acceso a BD externas, errores de rendimiento y errores de inicialización y de terminación.

2.16 Técnicas de caja negra

Existen varias técnicas que se basan en la filosofía del método de caja negra, algunas de estas son (33):

- ✓ Partición Equivalente
- ✓ Análisis de Valores Límite
- ✓ Grafos de Causa-Efecto
- ✓ Pruebas de Comparación

La técnica escogida para la validación de la aplicación es: partición de equivalencia. Esta se basa en dividir el campo de entrada de un programa en clases de datos, donde una condición de entrada es un valor numérico específico, un rango de valores, un miembro de un conjunto de valores o lógica. Es importante destacar que una clase de equivalencia representa un conjunto de estados válidos y no válidos para una condición de entrada. La técnica de partición equivalente se basa en evaluar las clases de equivalencia para una condición de entrada (33).

2.17 Casos de pruebas basados en Historias de usuarios

Los casos de pruebas se definen como un conjunto de condiciones o variables bajo las cuales el analista determina si los requisitos de una aplicación son parciales o completamente satisfactorios. Con el propósito de comprobar que todos los requisitos de la aplicación sean revisados, se debe

realizar al menos un caso de prueba para cada historia de usuario. Estos comúnmente son utilizados en pruebas de aceptación. A continuación se presenta el caso de prueba para la HU número 1.

Tabla#1.Caso de Prueba 1.Historia de usuario 1

Escenario	**Descripción**	**Variable 1**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Entra la consulta de forma correcta.	El usuario escribe en el editor la consulta de forma correcta.	V(consulta) select rolname from pg_roles where rolcanlogin=false	El sistema muestra en la parte inferior una vista con 3 pestañas. En la pestaña "Nodo" se muestra una tabla con el Tiempo Ejecución, Error del planificador, Filas recorridas, Ciclos y Plan de ejecución del nodo.	El usuario introduce la consulta y oprime el botón "Mostrar".
Escenario	**Descripción**	**Variable 2**	**Respuesta del sistema**	**Flujo central**
EC 1.2 Entra la consulta de forma incorrecta.	El usuario escribe en el editor la consulta de forma incorrecta.	I(consulta) sele rolname from pg_roles where rolcanlogin=false	El sistema muestra un mensaje indicando el error en la consulta.	El usuario entra la consulta y oprime el botón "Mostrar".
Escenario	**Descripción**	**Variable 3**	**Respuesta del sistema**	**Flujo central**
EC 1.3 Dejar el editor en blanco.	El usuario deja el editor en blanco	I	El sistema inhabilita el botón “Mostrar”.	El usuario no escribe en el editor.

Descripción de las variables

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	consulta	campo de texto	No	consulta SQL

1	consulta	campo de texto	No	consulta SQL
1	consulta	campo de texto	Si	

Tabla#2. Caso de prueba 2. Historia de usuario 2.

Escenario	**Descripción**	**Variable 1**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Entra la consulta de forma correcta.	El usuario escribe en el editor la consulta de forma correcta.	V(consulta) select rolname from pg_roles	El sistema muestra en la parte inferior una vista con 3 pestañas. En la pestaña "Texto" se muestra el plan de ejecución general de la consultas.	El usuario introduce la consulta y oprime el botón "Mostrar".
Escenario	**Descripción**	**Variable 2**	**Respuesta del sistema**	**Flujo central**
EC 1.2 Entra la consulta de forma incorrecta.	El usuario escribe en el editor la consulta de forma incorrecta.	I(consulta) sele rolname from pg_roles	El sistema muestra un mensaje indicando el error en la consulta.	El usuario entra la consulta y oprime el botón "Mostrar".
Escenario	**Descripción**	**Variable 3**	**Respuesta del sistema**	**Flujo central**
EC 1.3 Dejar el editor en blanco.	El usuario deja el editor en blanco	I	El sistema inhabilita el botón “Mostrar”.	El usuario no escribe en el editor.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	consulta	campo de texto	No	consulta SQL
1	consulta	campo de texto	No	consulta SQL
1	consulta	campo de texto	Si	

Tabla#3. Caso de prueba 3. Historia de usuario 3.

Escenario	**Descripción**	**Variable 1**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Entra la consulta de forma correcta.	El usuario escribe en el editor la consulta de forma correcta.	V(consulta) select rolname	El sistema muestra en la parte inferior una vista con 3 pestañas. En la pestaña "Sugerencia/s" se muestran sugerencias para optimizar la misma.	El usuario introduce la consulta y oprime el botón "Mostrar".
Escenario	**Descripción**	**Variable 2**	**Respuesta del sistema**	**Flujo central**
EC 1.2 Entra la consulta SQL de forma incorrecta.	El usuario escribe en el editor la consulta SQL de forma incorrecta.	I(consulta) sele rolname	El sistema muestra un mensaje indicando el error en la consulta "Error de sintaxis" y luego te reinicia el editor.	El usuario entra la consulta y oprime el botón "Mostrar".

Escenario	Descripción	Variable 3	Respuesta del sistema	Flujo central
EC 1.3 Dejar el editor en blanco.	El usuario deja el editor en blanco	I	El sistema inhabilita el botón “Mostrar”.	El usuario no escribe en el editor.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	consulta	campo de texto	No	consulta SQL
1	consulta	campo de texto	No	consulta SQL
1	consulta	campo de texto	Si	

2.18 Resultados de las pruebas

Con el propósito de comprobar que todos los requisitos del sistema fueron correctamente implementados, se aplicaron al plugin pruebas de aceptación a través del método de caja negra, aplicando la técnica de partición de equivalencia en dos iteraciones de pruebas, donde se creó un caso de prueba con tres escenarios para cada Historia de Usuario.

Para la identificación de las no conformidades (NC) se definieron los siguientes tipos:

- ✓ Ortográficas (O)
- ✓ De interfaz (DI)
- ✓ Funcionales (F)

Se muestra en la siguiente tabla las no conformidades identificadas por cada iteración de prueba realizada:

Iteración	Cantidad De NC	O	DI	F	NC Resueltas

1	5	Las tildes de las sugerencias faltan. Se detectó un error ortográfico en el nombre del plugin.	La tabla no permite ver el resultado del Explain Analyze completo de los nodos. El plan de ejecución de la consulta no se muestra de forma correcta.	Muestra las mismas sugerencias para distintas consultas.	5
2	2	Se detectó una falta de ortografía en una de las sugerencias	ninguna	Mostraba una sugerencia que no es compatible con PostgreSQL.	2

Conclusiones del capítulo.

Luego de realizar el proceso de pruebas al plugin, se puede concluir que:

- ✓ La realización de las pruebas de aceptación en los distintos escenarios, durante 2 iteraciones, posibilitó al cliente verificar el buen funcionamiento del plugin.
- ✓ La ejecución del método de caja negra permitió identificar las no conformidades y errores presentes en la aplicación, dándole solución a los mismos de forma inmediata.

Conclusiones generales

El desarrollo del presente trabajo propició el cumplimiento de los objetivos y tareas propuestas para su realización, arribándose a las siguientes conclusiones:

- ✓ El análisis de las herramientas de administración existentes, que trabajan con el gestor PostgreSQL, permitió concluir la necesidad de implementar un plugin de análisis del rendimiento de las consultas SQL para la herramienta HABD.
- ✓ La implementación del plugin propuesto con resultados satisfactorios, demostró la viabilidad del mismo, al poder permitir a los usuarios realizar el análisis del rendimiento de las consultas SQL en la herramienta HABD.
- ✓ La aplicación de las pruebas de aceptación verificó el cumplimiento de los requisitos y validó que el plugin funciona correctamente.

Por todo lo anteriormente expresado se puede concluir que se cumplió con el objetivo general propuesto: Desarrollar un plugin que permita analizar el rendimiento de las consultas realizadas sobre bases de datos en PostgreSQL para la herramienta de administración de bases de datos HABD.

Recomendaciones

Luego de lograr el cumplimiento de los objetivos que se trazaron al inicio del trabajo se recomienda que:

- ✓ Se continúe trabajando en versiones futuras del plugin en la optimización de las consultas, aumentado mucho más el número de sugerencias para el mejoramiento de la consulta.
- ✓ El plugin desarrollado pueda analizar el rendimiento de las consultas no solo para PostgreSQL, sino para los demás sistemas gestores de bases de datos existentes.
- ✓ Se automatice el mejoramiento de la consulta sin que el usuario haga modificaciones de forma manual.

Referencias bibliográficas

1. Glosario.net. [En línea] [Citado el: 15 de junio de 2012.] http://tecnologia.glosario.net/terminos-tecnicos-internet/dbms-466.html.

2. Maestros de Web. [En línea] [Citado el: 25 de noviembre de 2011.] http://www.maestrosdelweb.com/principiantes/%C2%BFque-son-las-bases-de-datos/.

3. **J.Date, Chris.** *Introducción a los Sistemas de Bases de Datos.* México : PEARSON EDUCACIÓN, 2001. 968-444-419-2.

4. **Lockhart, Thomas.** *Tutorial de PostgreSQL.*

5. masadelante.com. [En línea] [Citado el: 14 de junio de 2012.] http://www.masadelante.com/faqs/plug-in.

6. PergaminoVirtual. [En línea] [Citado el: 28 de junio de 2012.] http://www.pergaminovirtual.com.ar/definicion/Plug-In.html.

7. SOFTWARELOGÍA.COM. [En línea] [Citado el: 30 de junio de 2012.] http://softwarelogia.com/2008/07/25/%C2%BFque-es-un-plugin/.

8. Manual de Microsoft Acces 2003. [En línea] [Citado el: 25 de noviembre de 2011.] http://www.duiops.net/manuales/access/access4.htm.

9. Manual de usuario de PostgreSQL. [En línea] [Citado el: 17 de junio de 2012.] http://www.ibiblio.org/pub/linux/docs/LuCaS/Postgresql-es/web/navegable/user/app-pgaccess.html.

10. guia-ubuntu. [En línea] [Citado el: 17 de junio de 2012.] http://www.guia-ubuntu.org/index.php?title=PgAdmin_III.

11. [En línea] [Citado el: 18 de junio de 2012.] http://www.pgadmin.org/.

12. Grupo de usuarios PostgreSQL de Argentina. [En línea] [Citado el: 18 de junio de 2012.] http://www.arpug.com.ar/trac/wiki/PgAdmin.

13. Comunidad de Programadores. [En línea] [Citado el: 18 de junio de 2012.] http://www.lawebdelprogramador.com/temas/PostgreSQL/1697-PgAccess_-_a_Tcl_Tk_interface_for_PostgreSQL.html.

14. [En línea] [Citado el: 17 de junio de 2012.] http://mmc.igeofcu.unam.mx/LuCAS/Postgresql-es/web/navegable/pgaccess/tutorial/intro.html.

15. [En línea] [Citado el: 17 de junio de 2012.] http://www.scribd.com/doc/63764883/65/PhpPgAdmin.

16. **Acosta, Ivette Rosa Teodosio.** *Exploración y diseño de la herramienta de administración de bases de datos para .* Habana, Cuba : s.n., 2011.

17. Rendimeinto SQL Server. [En línea] [Citado el: 8 de diciembre de 2011.] http://blogs.msdn.com/b/apinedo/archive/2007/01/24/mejorar-el-rendimiento-de-queries-en-sql-server.aspx.

18. Optimizar rendimiento. [En línea] [Citado el: 4 de diciembre de 2011.] http://www.guatewireless.org/tecnologia/bases-de-datos/mysql-optimizar-el-rendimiento-de-lectura/.

19. Técnicas para mejorar consultas en espacios de trabajo. [En línea] [Citado el: 30 de enero de 2012.] http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.soa.doc/refs/rsdperformanceworkspaces.htm.

20. desarrolloweb. [En línea] [Citado el: 23 de abril de 2012.] http://www.desarrolloweb.com/articulos/2230.php.

21. **David Valdeverde.** David Valdeverde. [En línea] [Citado el: 24 de noviembre de 2011.] http://www.davidvalverde.com/blog/introduccion-a-la-programacion-extrema-xp/.

22. Conceptos del lenguaje C y otros tipos de datos. [En línea] [Citado el: 30 de noviembre de 2011.] http://www.casdreams.com/auladeinformatica/PROG/C1.html.

23. **Tarazona, Ivon.** El uso de UML en los Modelados de Datos. [En línea] [Citado el: 7 de diciembre de 2011.] http://alfa.facyt.uc.edu.ve/computacion/pensum/cs0347/download/exposiciones2005-2006/uml.pdf.

24. **Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato.** *Control de versiones con Subversion.*

25. [En línea] [Citado el: 20 de enero de 2012.] http://ldc.usb.ve/~martinez/cursos/ci3715/clase6_AJ2010.pdf.

26. **BECK, K.** *Planeando en Programación Extrema.* 2000.

27. **Jeffries, R, Anderson, A y Hendrickson, C.** *Extreme Programming Installed.* s.l. : Addison-Wesley, 2002.

28. Ecured. [En línea] [Citado el: 23 de septiembre de 2012.] http://www.ecured.cu/index.php/Estilo_arquitectural_en_capas_%28N-Layer%29.

29. **Buschamann, Frank.** *Pattern-Oriented Software Architecture.* s.l. : Jhon Wiley & Sons, 1996.

30. **Lago, R.** *Patrones de diseño software.* 2007.

31. **Larrman, C.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Mexico : Prentice Hall, 1999.

32. **Joskowicz, J.** *Reglas y Prácticas en Extreme Programming.* 2008.

33. [En línea] [Citado el: 20 de mayo de 2012.] http://ocw.uc3m.es/ingenieria-informatica/ingeniera-del-software-iii/materialclase/ISIII_09_PRUE.pdf.

Bibliografía

1. Glosario.net. [En línea] [Citado el: 15 de junio de 2012.] http://tecnologia.glosario.net/terminos-tecnicos-internet/dbms-466.html.

2. Maestros de Web. [En línea] [Citado el: 25 de noviembre de 2011.] http://www.maestrosdelweb.com/principiantes/%C2%BFque-son-las-bases-de-datos/.

3. **J.Date, Chris.** *Introducción a los Sistemas de Bases de Datos.* México : PEARSON EDUCACIÓN, 2001. 968-444-419-2.

4. **Lockhart, Thomas.** *Tutorial de PostgreSQL.*

5. masadelante.com. [En línea] [Citado el: 14 de junio de 2012.] http://www.masadelante.com/faqs/plug-in.

6. PergaminoVirtual. [En línea] [Citado el: 28 de junio de 2012.] http://www.pergaminovirtual.com.ar/definicion/Plug-In.html.

7. SOFTWARELOGÍA.COM. [En línea] [Citado el: 30 de junio de 2012.] http://softwarelogia.com/2008/07/25/%C2%BFque-es-un-plugin/.

8. Manual de Microsoft Acces 2003. [En línea] [Citado el: 25 de noviembre de 2011.] http://www.duiops.net/manuales/access/access4.htm.

9. Manual de usuario de PostgreSQL. [En línea] [Citado el: 17 de junio de 2012.] http://www.ibiblio.org/pub/linux/docs/LuCaS/Postgresql-es/web/navegable/user/app-pgaccess.html.

10. guia-ubuntu. [En línea] [Citado el: 17 de junio de 2012.] http://www.guia-ubuntu.org/index.php?title=PgAdmin_III.

11. [En línea] [Citado el: 18 de junio de 2012.] http://www.pgadmin.org/.

12. Grupo de usuarios PostgreSQL de Argentina. [En línea] [Citado el: 18 de junio de 2012.] http://www.arpug.com.ar/trac/wiki/PgAdmin.

13. Comunidad de Programadores. [En línea] [Citado el: 18 de junio de 2012.] http://www.lawebdelprogramador.com/temas/PostgreSQL/1697-PgAccess_-_a_Tcl_Tk_interface_for_PostgreSQL.html.

14. [En línea] [Citado el: 17 de junio de 2012.] http://mmc.igeofcu.unam.mx/LuCAS/Postgresql-es/web/navegable/pgaccess/tutorial/intro.html.

15. [En línea] [Citado el: 17 de junio de 2012.] http://www.scribd.com/doc/63764883/65/PhpPgAdmin.

16. **Acosta, Ivette Rosa Teodosio.** *Exploración y diseño de la herramienta de administración de bases de datos para .* Habana, Cuba : s.n., 2011.

17. Rendimeinto SQL Server. [En línea] [Citado el: 8 de diciembre de 2011.] http://blogs.msdn.com/b/apinedo/archive/2007/01/24/mejorar-el-rendimiento-de-queries-en-sql-server.aspx.

18. Optimizar rendimiento. [En línea] [Citado el: 4 de diciembre de 2011.] http://www.guatewireless.org/tecnologia/bases-de-datos/mysql-optimizar-el-rendimiento-de-lectura/.

19. Técnicas para mejorar consultas en espacios de trabajo. [En línea] [Citado el: 30 de enero de 2012.] http://publib.boulder.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.soa.doc/refs/rsdperformanceworkspaces.htm.

20. desarrolloweb. [En línea] [Citado el: 23 de abril de 2012.] http://www.desarrolloweb.com/articulos/2230.php.

21. **David Valdeverde.** David Valdeverde. [En línea] [Citado el: 24 de noviembre de 2011.] http://www.davidvalverde.com/blog/introduccion-a-la-programacion-extrema-xp/.

22. Conceptos del lenguaje C y otros tipos de datos. [En línea] [Citado el: 30 de noviembre de 2011.] http://www.casdreams.com/auladeinformatica/PROG/C1.html.

23. **Tarazona, Ivon.** El uso de UML en los Modelados de Datos. [En línea] [Citado el: 7 de diciembre de 2011.] http://alfa.facyt.uc.edu.ve/computacion/pensum/cs0347/download/exposiciones2005-2006/uml.pdf.

24. **Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato.** *Control de versiones con Subversion.*

25. [En línea] [Citado el: 20 de enero de 2012.] http://ldc.usb.ve/~martinez/cursos/ci3715/clase6_AJ2010.pdf.

26. **BECK, K.** *Planeando en Programación Extrema.* 2000.

27. **Jeffries, R, Anderson, A y Hendrickson, C.** *Extreme Programming Installed.* s.l. : Addison-Wesley, 2002.

28. Ecured. [En línea] [Citado el: 23 de septiembre de 2012.] http://www.ecured.cu/index.php/Estilo_arquitectural_en_capas_%28N-Layer%29.

29. **Buschamann, Frank.** *Pattern-Oriented Software Architecture.* s.l. : Jhon Wiley & Sons, 1996.

30. **Lago, R.** *Patrones de diseño software.* 2007.

31. **Larrman, C.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Mexico : Prentice Hall, 1999.

32. **Joskowicz, J.** *Reglas y Prácticas en Extreme Programming.* 2008.

33. [En línea] [Citado el: 20 de mayo de 2012.] http://ocw.uc3m.es/ingenieria-informatica/ingeniera-del-software-iii/materialclase/ISIII_09_PRUE.pdf.

34. **Pantoja, E.** *El patrón de diseño Modelo-Vista-Controlador (MVC).*

35. **System Sparx.** [En línea] 2007. http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html.

36. **MySQL.** Intartius Corporation. [En línea] 2010. http://instartius.com/site/?page_id=327.

37. **Scribd.** Scribd. [En línea] 2008. es.scribd.com/doc/36570462/postgreSQL-investigacion.

38. **Alvarez, Miguel Angel.** Desarrolloweb.com. [En línea] 2001. http://www.desarrolloweb.com/articulos/392.php.

39. **Arias, Michael.** La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software. [En línea] 2006. http://www.latindex.ucr.ac.cr/intersedes10/10-art_11.pdf.

40. **Curto, Josep.** 2006.

41. **Escribano., Gerardo Fernández.** Introducción a Extreme Programming. [En línea] 2002. http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-XP.pdf.

42. **Flores, Maiqui M.** Nación & Salud. [En línea] 2009. http://www.nacionysalud.com/node/1354.

43. **Gracia, Joaquin.** Diseño de Software Orientado a Objetos. [En línea] 2005. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

44. **Mijares, Marizé, y otros, y otros.** *Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC).* 2006.

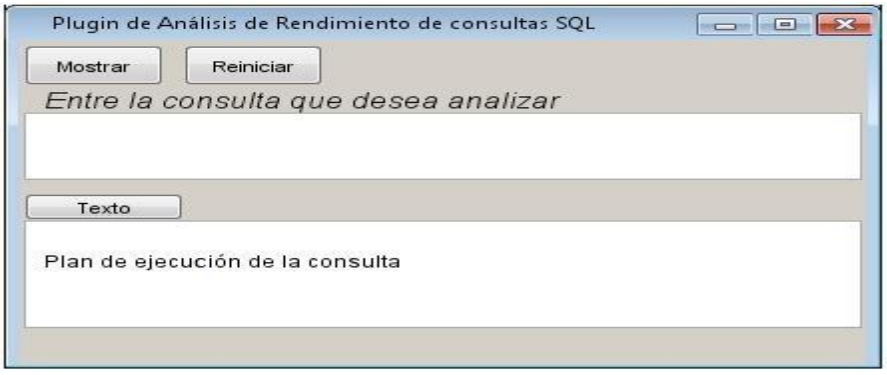
45. **Portillo, José Antonio Pow Sang.** La espcificación de requisitos con casos de uso: Buenas y Malas Prácticas. [En línea] 2008. http://es.scribd.com/doc/14504728/Buenas-Malas-Practicas-CU.

46. **Pressman, Roger S.** *Ingeniería del Software: Un enfoque práctico.* Sexta edición. México : McGraw Hill, 2006.

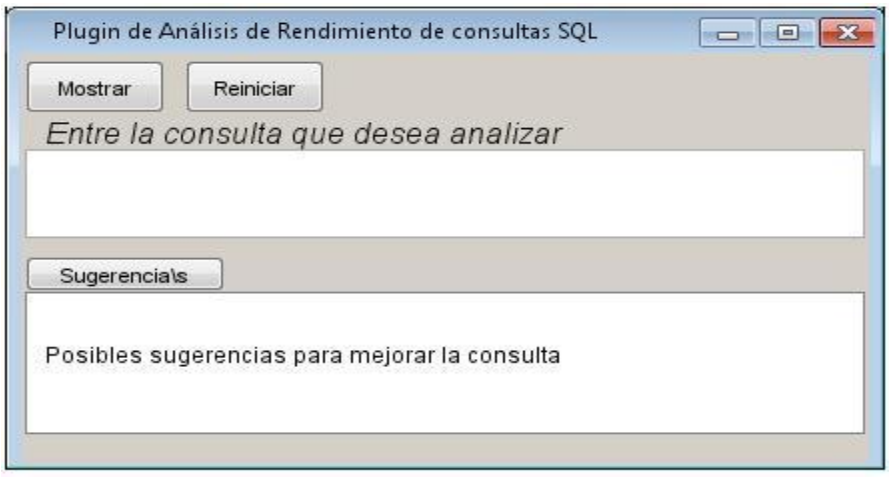
47. **Reynoso, Carlos y Kicillof, Nicolás.** Estilos y Patrones en la Estrategia Arquitectónica de Microsoft. [En línea] 2004. http://carlosreynoso.com.ar/archivos/arquitectura/Estilos.PDF.

Anexos

Anexo #1: Historia de Usuario #2.

	Historia de Usuario
Número: 2	**Nombre de la Historia de Usuario:** Mostrar plan de ejecución general de la consulta.
Cantidad de modificaciones a la Historia de Usuario: 0	
Usuario: Vladimir Ricardo Julián	**Iteración asignada:**1
Prioridad en negocio: Muy Alta	**Puntos estimados: 1**
Riesgo en desarrollo: Muy Alto	**Puntos reales: 1**
Descripción: Se muestra el plan de ejecución de la consulta general, mostrando además el tiempo de ejecución total de la misma.	
Observaciones:	
Prototipo de interfaces: 	

Anexo #2: Historia de Usuario #3.

Historia de Usuario	
Número: 3	**Nombre de la Historia de Usuario:** Mostrar sugerencias de optimización de consulta.
Cantidad de modificaciones a la Historia de Usuario: 0	
Usuario: Vladimir Ricardo Julián	**Iteración asignada: 2**
Prioridad en negocio: Muy Alta	**Puntos estimados: 3**
Riesgo en desarrollo: Muy Alto	**Puntos reales: 3**
Descripción: Se muestran sugerencias de posibles mejoras a la consulta sobre la cual se trabaja.	
Observaciones:	
Prototipo de interfaces: 	

Anexo #3: Lista de Reserva del Producto.

Ítem*	Descripción	Estimación	Estimado por
Prioridad: Muy Alta			
1	Mostrar tabla de rendimiento de la consulta	3 semana	*Programador*
2	Mostrar plan de ejecución general de la consulta	1 semana	*Programador*
3	Mostrar sugerencias de optimización para la consulta	3semanas	*Programador*
Prioridad: Baja			
	Facilidad de uso: Para utilizar el sistema es necesario poseer conocimientos elementales de computación, así como de base de datos. **Soporte**: Se dispondrá de documentación técnica que describa cómo funciona el plugin y detalle cada una de las informaciones o vistas que este muestre. **Software**: Sistema Operativo: Multiplataforma. Librerías QT. Servidor de Bases de datos: SGBD PostgreSQL 8.4.x o versiones superiores. **Hardware:** Se necesita 100 MB de memoria RAM mínimo, 1GB de espacio libre en el disco duro para su instalación y el microprocesador a 300 MHz.		

Anexo #4: Tarjeta CRC "Principal".

Tarjeta CRC	
Clase: Principal	
Responsabilidades	**Colaboraciones**
Muestra componentes visuales del plugin. Muestra datos de la consulta.	Control SQLDatabase

Anexo #5: Tarjeta CRC "Nodos".

Tarjeta CRC	
Clase: Nodos	
Responsabilidades	**Colaboraciones**
Crea objetos de tipo Ayuda	Ayuda

Anexo #6: Tarjeta CRC "Ayuda".

Tarjeta CRC	
Clase: Ayuda	
Responsabilidades	**Colaboraciones**
Carga las sugerencias. Filtra las sugerencias.	

Anexo #7: Tarea de ingeniería #1.

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario: 1**
Nombre Tarea: Capturar la consulta SQL.	
Tipo de Tarea : Desarrollo	**Puntos Estimados: 1**
Fecha Inicio: 12/3/2012	**Fecha Fin: 16/3/2012**
Programador Responsable: Vladimir Ricardo Julián	
Descripción: Se captura la consulta y se le aplica el comando EXPLAIN ANALYZE, obteniéndose el plan de ejecución de la consulta, el cual se guarda en una variable de tipo SQLquery para trabajar con la información contenida en esta.	

Anexo #8: Tarea de ingeniería #3.

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario: 1**
Nombre Tarea: Mostrar tabla con los datos de la consulta.	
Tipo de Tarea : Desarrollo	**Puntos Estimados: 1**
Fecha Inicio: 26/3/2012	**Fecha Fin: 30/3/2012**
Programador Responsable: Vladimir Ricardo Julián	
Descripción: Se muestran los datos de la consulta para que el usuario analice y interprete la ejecución de todos los nodos de la misma.	

Anexo #9: Tarea de ingeniería #4.

Tarea de Ingeniería	
Número Tarea: 4	**Número Historia de Usuario: 2**
Nombre Tarea: Mostrar plan de ejecución de la consulta general.	
Tipo de Tarea : Desarrollo	**Puntos Estimados: 1**
Fecha Inicio: 2/4/2012	**Fecha Fin: 6/4/2012**
Programador Responsable: Vladimir Ricardo Julián	
Descripción: Se muestra el plan de ejecución de la consulta de forma general.	

Anexo #10: Tarea de ingeniería #5.

Tarea de Ingeniería	
Número Tarea: 5	**Número Historia de Usuario: 3**
Nombre Tarea: Cargar las sugerencias de optimización.	
Tipo de Tarea : Desarrollo	**Puntos Estimados: 1**
Fecha Inicio: 9/4/2012	**Fecha Fin: 13/4/2012**
Programador Responsable: Vladimir Ricardo Julián	
Descripción: Se cargan las sugerencias para la optimización de consultas SQL que se encuentran en un fichero estático de la aplicación.	

Anexo #11: Tarea de ingeniería #6.

Tarea de Ingeniería	
Número Tarea: 6	**Número Historia de Usuario: 3**

Nombre Tarea: Analizar la consulta.	
Tipo de Tarea : Desarrollo	**Puntos Estimados: 1**
Fecha Inicio: 16/4/2012	**Fecha Fin: 20/4/2012**
Programador Responsable: Vladimir Ricardo Julián	
Descripción: Se analiza la consulta dependiendo su conformación y se filtran las sugerencias que pueden mejorar el rendimiento de la misma.	

Anexo #12: Tarea de ingeniería #7.

Tarea de Ingeniería	
Número Tarea: 7	**Número Historia de Usuario: 3**
Nombre Tarea: Mostrar sugerencias.	
Tipo de Tarea : Desarrollo	**Puntos Estimados: 1**
Fecha Inicio: 23/4/2012	**Fecha Fin: 27/4/2012**
Programador Responsable: Vladimir Ricardo Julián	
Descripción: Se muestran posibles sugerencias al usuario.	