Aplicación Gráfica de Análisis de Costos para PostgreSQL¨PGSql Analyze¨

Denys Marcelo Rodríguez Domínguez

13 de abril de 2005

Índice general

Índice de figuras

Índice de cuadros

Introducción

El presente trabajo forma parte de las actividades necesarias para optar al Título de Ingeniero de Ejecución en Computación e Informática.

Este consiste en la desarrollo de una Aplicación basado en tecnologías .NET[1], que ayude a mejorar el desempeño de consultas SQL.

En la primera parte del informe se dará a conocer el porque surge la necesidad de desarrollar la aplicación, se establecerá los objetivos generales y específicos a los cuales se quiere llegar con este proyecto, se dará a conocer los alcances y aportes de la aplicación a desarrollar.

Después de establecer lo anterior se hará una descripción de todas las tecnologías utilizadas en el desarrollo de la herramienta, tales como el sistema operativo Linux, PostgreSQL, Mono, GTK#, entre otros.

Posteriormente se describirá la metodología utilizada en el desarrollo de la aplicación.

Luego de esto se procede a realizar el estudio de factibilidad para saber si la construcción de la herramienta es económica, operacional y funcionalmente factible.

Al terminar con este análisis se comienza con la etapa de modelamiento, estableciendo los requerimiento a través del diseño de Diagramas de Casos de Uso, Colaboración, Análisis de Casos de Uso y por ultimo el diseño de las Interfaces para su posterior construcción.

Al concluir se realiza una serie de pruebas a la aplicación para confirmar que

[1]Es el conjunto de nuevas tecnologías de Microsoft.

todo lo establecido en el análisis previo concuerda con el producto final.

Capítulo 1

Descripción del proyecto

1.1. Descripción del proyecto

Aplicación gráfica de Análisis de Costos para PostgreSQL ¨PGSql Analyze¨

1.2. Origen del proyecto

En la actualidad, existen muchas aplicaciones o sistemas que tienen interacción con base de datos, los usuarios hoy en día solicitan grandes cantidades de información y de la forma más rápida posible, esto da muchos problemas a los administradores de base de datos porque son ellos los que tienen que proveer alguna solución a estos requerimientos.

Muchos de los problemas producidos son producto de cuellos de botella en el procesamiento de las consultas, esto se debe a una mala configuración del servidor de base de datos o por mala confección de la consulta SQL que obtiene los datos, este último punto es el que se pretende abordar, ya que se necesita una aplicación que ayude a la optimización de consultas del motor de Base de Datos, conocer el comportamiento del motor para realizar las tareas / consultas solicitadas, con el fin de que los procesos se agilicen, disminuyendo así el tiempo de respuesta, y por consiguiente mejorar el rendimiento del sistema o aplicación que las utiliza.

En la búsqueda de un motor de Base de Datos, que cumpla fielmente con todas las características de un RDBMS[1], nos encontramos con PostgreSQL, base de datos desarrollada originalmente en el Departamento de Ciencias de la Computación de la Universidad de California en Berkeley (1986). Además de ser un RDBMS, la licencia es open-source, no teniendo restricción en como el código es usado, lo cual le da un valor agregado al motor.

PostgreSQL es también distinto a otros gestores de datos en que el servidor puede incorporar código escrito por el usuario a través de bibliotecas de carga dinámicas. O sea, el usuario puede especificar un archivo de código objeto (p. eje., un archivo compilado .o o bibliotecas de intercambio) con lo que se implementa un nuevo tipo o funciones así PostgreSQL cargará sólo lo que requiera. Siendo esto una gran diferencia con otras sistemas relacionales comerciales, debido que sólo se pueden extender cambiando los procedimientos codificados del RDBMS o cargando módulos especialmente escritos por el vendedor de RDBMS.[2]

Uno de los inconvenientes que presenta PostgreSQL hoy en día, es que su manipulación y administración se realiza por consola o terminal (linea de comandos Linux - UNIX), lo cual es una gran desventaja comparando con las soluciones comerciales actuales como Microsoft SQL-Server u ORACLE.

Tomando en cuenta los antecedentes anteriores, se pretende aportar con la construcción de una aplicación gráfica orientada a desarrolladores y administradores de Bases de Datos, que les permita realizar análisis de rendimiento de sus consultas y procesos asociados.

1.3. Objetivos del proyecto

1.3.1. Objetivos generales

Construir una aplicación que provea una representación gráfica del árbol de una consulta SQL, que permita ver el costo de procesamiento de cada parte de

[1] Relational Database Management System
[2] PostgreSQL Programmer's Manual.

la consulta, facilitando la detección de cuellos de botella y sea capaz de sugerir optimizaciones que aminoren el trabajo a los desarrolladores.

1.3.2. Objetivos específicos

- Investigar:
  - El funcionamiento interno de un RDBMS, en este caso particular, PostgreSQL.
  - Mecanismos de optimizaciones que se puedan aplicar a PostgreSQL, para mejorar el uso de recursos de HW, tales como el uso de CPU, Disco y Memoria
  - Mecanismos de optimizaciones que provea PostgreSQL, para mejorar el diseño de consultas, procedimientos almacenados, entre otros. Los mecanismos son :
    - VACUUM
    - EXPLAIN
    - ANALYZE
  - Aplicación de optimizaciones mediante el control fino de índices en la construcción de tablas.
  - Investigar sobre la plataforma .NET, sus características y conceptos básicos.
  - Descripción de MONO (implementacion libre de la plataforma .NET) .
  - Descripción de bibliotecas utilizadas en el desarrollo de la aplicación como son GTK+.
- Construir una aplicación que permita:

- Una fácil lectura de la información con el objetivo de determinar y mostrar los cuellos de botella de una consulta SQL.
- Desplegar el método de búsqueda y procesamiento de una consulta SQL, procedimientos almacenados, entre otros.
- Mostrar gráficamente los algoritmos empleados por el motor de la Base de Datos para llevar a cabo las consultas.
- Conocer el tiempo real y estimado de cada una de las partes de una consulta SQL.
- Ofrecer recomendaciones que permitan al desarrollador optimizar las consultas y/o procedimientos almacenados.

- Crear una aplicación con licencia GPL.

1.3.3. Alcances

- La aplicación será liberada bajo licencia GPL.
- El software se desarrollará bajo plataforma Linux.

1.3.4. Aportes

- Incorporar tecnologías de open-source[3], como MONO, GTK#, XML, PostgreSQL, entre otros, la cual no es muy utilizada con frecuencia en Proyectos de Titulo.
- Incorporar una herramienta open-source, a la comunidad Linux.
- Agregar una nueva herramienta gráfica a PostgreSQL, que provea información detallada sobre los recursos consumidos por una consulta SQL, siendo esta de gran ayuda para mejorar el rendimiento de las consultas. La herramienta permitirá hacer lo siguiente :

[3] Código abierto.

- Editor para la entrada de consultas SQL.
- Coloreo de sintaxis, esto ayuda a los usuarios a distinguir palabras claves del SQL.
- La salida puede ser configurable, ya sea en modo texto o de forma gráfica.
- Diagrama gráfico de la consulta, este diagrama muestra los pasos lógicos, de un plan de ejecución de una consulta SQL. Esta característica permite visualizar de mejor forma la consulta ejecutada y los costos asociados a ella.
- Sugerencias de creación de índices, la aplicación sugerirá al usuario donde crear un índice.

Capítulo 2

Aspectos tecnológicos

2.1. Introducción

A continuación se describirán las tecnologías utilizadas para llevar a cabo este proyecto, que tiene como finalidad la construcción de una aplicación que permita ver los costos asociados a una consulta SQL. Para que la aplicación trabaje de forma óptima, existen ciertos aspectos que deben considerarse en la base de datos (PostgreSQL), como requerimientos mínimos de funcionamiento tanto de HW como SW y así esta pueda entregar valores verdaderos.

2.2. Linux

Linux estrictamente se refiere al núcleo Linux, pero es más comúnmente utilizado para describir un sistema operativo tipo Unix, que utiliza primordialmente filosofía y metodologías libres, también conocido como GNU/Linux. Está formado mediante la combinación del núcleo Linux con las bibliotecas y herramientas del proyecto GNU y de muchos otros proyectos/grupos de software (libre o propietario). La primera versión del núcleo Linux fue escrita por el programador finlandés Linus Torvalds y liberada en 1991, combinado con componentes de GNU. El núcleo no es parte oficial del proyecto GNU (el cual posee su propio núcleo,

llamado Hurd)[1], pero es distribuido bajo los términos de la licencia GPL (GNU General Public License) (ver anexo).

Algunas ventajas de utilizar Linux:

- Es gratuito. Es de libre distribución, cualquier persona puede regalarlo, venderlo o prestarlo.
- Es más seguro.
- Menores tiempos de desarrollo debido a la amplia disponibilidad de herramientas y librerías.
- Se puede acceder a su código y aprender de él.
- Se puede modificar, adaptándolo para realizar tareas específicas, según requerimientos particulares.

2.3. PostgreSQL

2.3.1. Historia

PostgreSQL, desarrollada originalmente en el Departamento de Ciencias de la Computación de la Universidad de California en Berkeley, fue pionera en muchos de los conceptos de bases de datos relacionales orientadas a objetos que ahora empiezan a estar disponibles en algunas bases de datos comerciales. Ofrece soporte al lenguaje SQL92/SQL3, integridad de transacciones, y extensibilidad de tipos de datos. PostgreSQL es un descendiente de dominio público y código abierto del código original de Berkeley.

[1] Wikipedia (Enciclopedia Libre).

2.3.2. Visión general del motor

Descripción general de como el motor procesa una consulta hasta, que se obtienen los datos y son mostrados al usuario ver (figura 2.1).

Como PostgreSQL procesa una consulta.

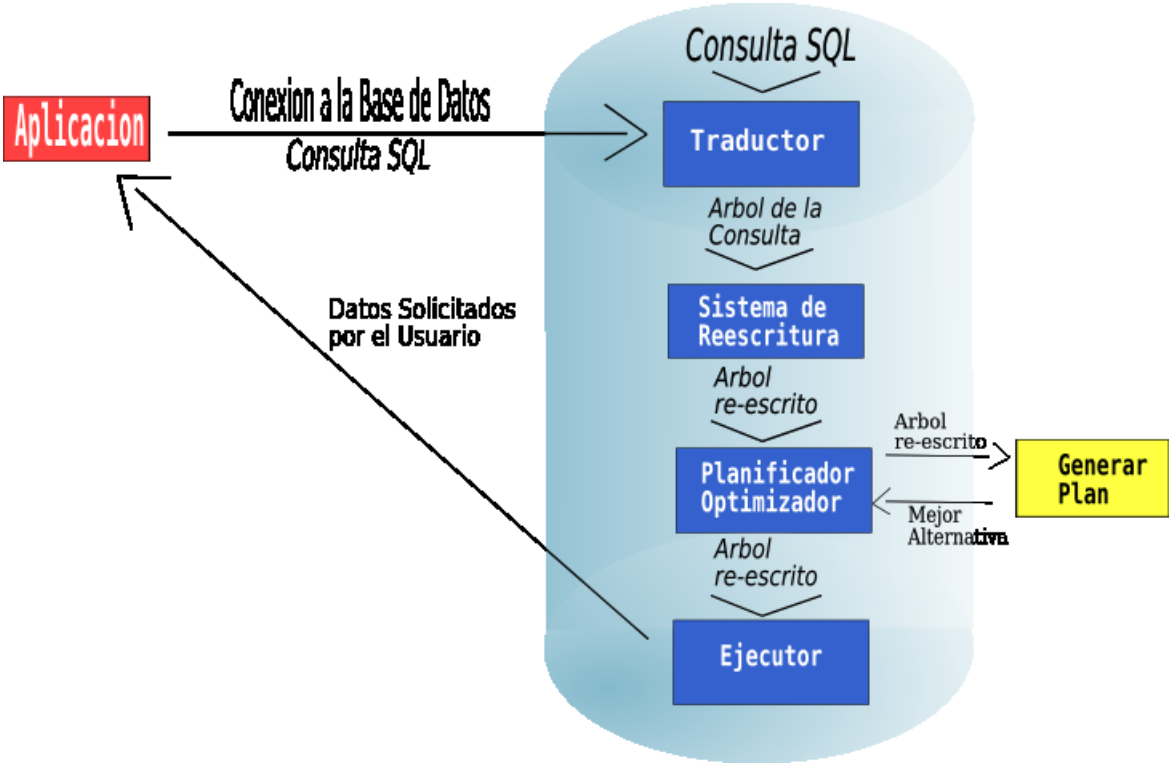

Figura 2.1: Fuente: Realizada por el autor.

- Una aplicación establece una conexión con el servidor de PostgreSQL. La aplicación transmite una consulta al servidor y aguarda por los datos solicitados. Por ejemplo :

```
(estableciendo la conexión con el motor)
```

```
psql [2] -U user -d db -h localhost -W
```

(solicitando datos al motor)

```
SELECT nombre FROM login;
```

- El motor verifica la validez de sintaxis (*traductor*) de la consulta recibida, si la sintaxis es correcta, se construye un *árbol de la consulta* de lo contrario se devuelve mensaje de error. Por ejemplo:

```
SELECT nombre FROM login;
```

- El *sistema de reescritura* toma el árbol de la consulta creado en el paso del *traductor* y busca reglas (las que se encuentran almacenadas en los catálogos del sistema), para poder aplicarselas al árbol de la consulta y realizar las transformaciones pertinentes. A continuación se verá una aplicación del sistema de reescritura en la realización de las vistas. Por ejemplo:

```
CREATE RULE view_rule AS ON SELECT TO test_view
DO INSTEAD
    SELECT s.sname, p.pname

       FROM supplier s, sells se, part p
    WHERE s.sno = se.sno and p.pno = se.pno;
```

- Esta regla se disparará cada vez que se detecte un select contra la relación *test_view*. En lugar de seleccionar las tuplas de *test_view*, se ejecutará la instrucción SELECT dada en la parte de la acción de la regla. Luego se aplica la consulta contra *test_view*.

[2] Interfaz de usuario para realizar consultas SQL interactivas,
donde el -U es el usuario a conectarse, -d base de datos a conectarse,
-h es el host a conectarse y por último -W para solicitar la contraseña.

```
SELECT sname FROM test_view
    WHERE sname <> 'Denys';
```

- Reescritura de *test_view*

  1. Toma la consulta dada por la parte de la acción de la regla.

  2. Adapta la lista-objetivo para recoger el número y orden de los atributos dados en la consulta del usuario.

  3. Añade la cualificación dada en la cláusula WHERE de la consulta del usuario a la cualificación de la consulta dada en la parte de la acción de la regla. Dada la definición de la regla anterior, será reescrita de la siguiente forma:

```
SELECT s.sname FROM supplier s, sells se, part p
    WHERE
        s.sno = se.sno
            AND p.pno = se.pno
            AND s.sname <> 'Denys';
```

- El *planificador/optimizador, planner* de ahora en adelante, toma el árbol reescrito y creará varios planes para la consulta, siendo estos enviados al ejecutor. Por ejemplo si existe un índice sobre una relación que va ser escaneada[3], el *planner* creará 2 posibles planes, uno será un barrido secuencial y el otro usará el índice para la búsqueda. El próximo paso es calcular un costo estimado de ejecución para los 2 planes, siendo seleccionado el menos costoso.

  El *planner* decide qué planes deberían generarse basándose en los tipos de índices definidos sobre las relaciones que aparecen en una consulta. Siempre existe la posibilidad de realizar un barrido secuencial de una relación, de modo que siempre se crea un plan que sólo utiliza barridos secuenciales.

[3] barrido

Si se asume que hay definido un índice en una relación (por ejemplo un índice B-Tree) y una consulta contiene la restricción relación.atributo OPR[4] constante. Si relación.atributo acierta a coincidir con la clave del índice B-tree y OPR esta listado en la clases de operadores de índices se crea un plan utilizando el índice B-tree, permitiendo así una búsqueda más rápida en la relación. Si hay otros índices presentes y las restricciones de la consulta aciertan con una clave de un índice, se considerarán por lo tanto otros planes.

Tras encontrar todos los planes utilizables para revisar relaciones únicas, se crean los planes para cruzar (join) relaciones. El *planner* considera sólo cruces entre cada dos relaciones para los cuales existe una cláusula de cruce correspondiente (es decir, para las cuales existe una restricción como WHERE rel1.atr1=rel2.atr2). Se generan todos los posibles planes para cada cruce considerado por el *planner*. Las tres posibles estrategias de cruce son:

- Cruce de iteración anidada (*Nested Loop*) ver (figura:2.2): La relación derecha se recorre para cada tupla encontrada en la relación izquierda. Esta estrategia es fácil de implementar pero puede consumir mucho tiempo.

[4] Operador(<,>,=,=>).

Nested Loop

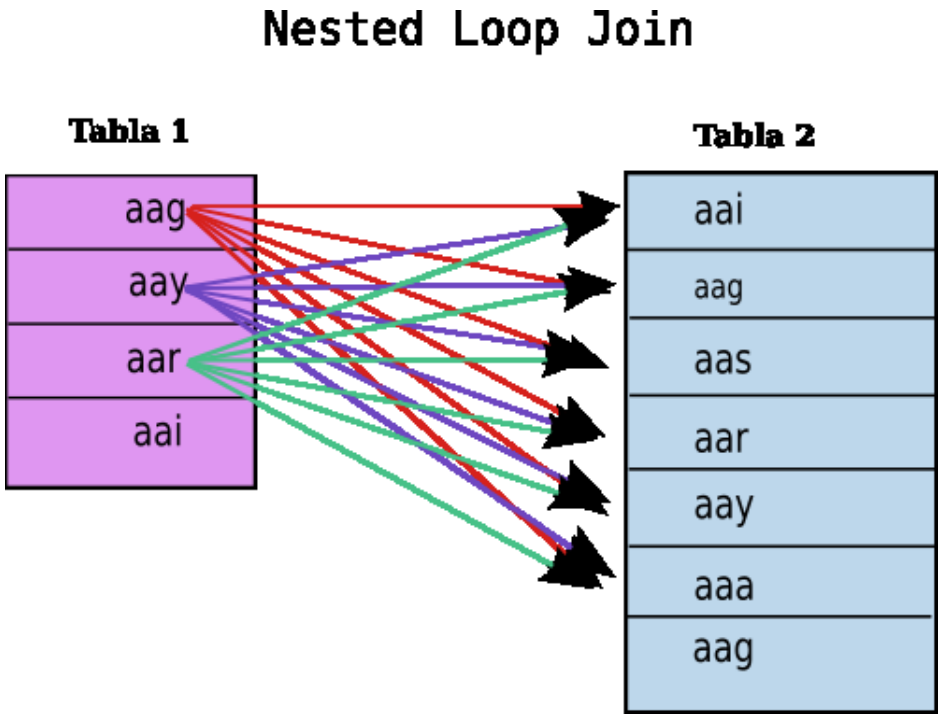

Figura 2.2: Fuente: Bruce Momjian, PostgreSQL Internals Trough Pictures.

- Cruce de ordenación mezclada (*Merge Join*) ver (figura:2.3): Cada relación es ordenada por los atributos del cruce antes de iniciar el cruce mismo. Después se mezclan las dos relaciones teniendo en cuenta que ambas relaciones están ordenadas por los atributos del cruce. Este modelo de cruce es más atractivo porque cada relación debe ser barrida sólo una vez.

Merge-Join

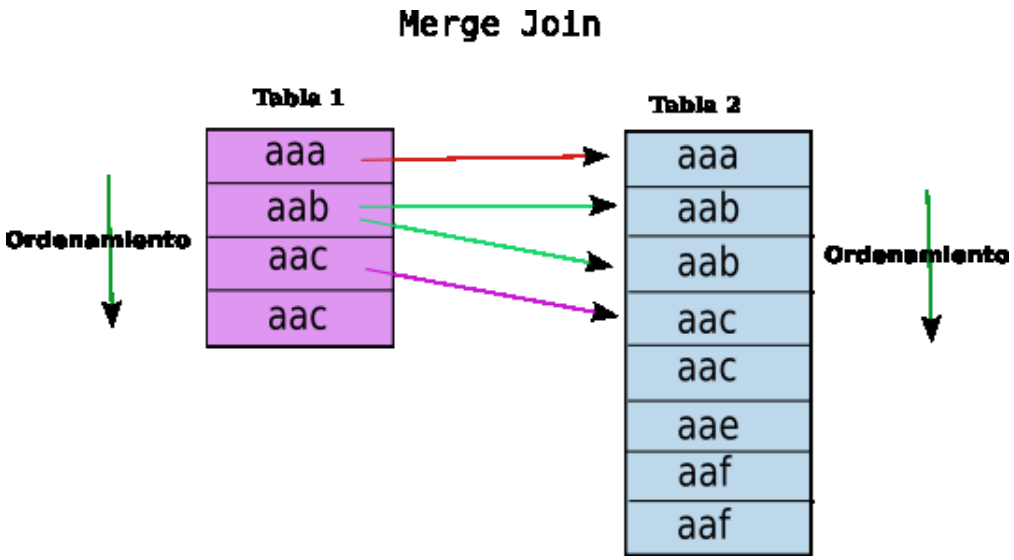

Figura 2.3: Fuente: Bruce Momjian, PostgreSQL Internals Trough Pictures.

- Cruce indexado (*Hash Join*) ver (figura 2.4): La relación de la derecha se indexa primero sobre sus atributos para el cruce y se carga en una tabla *hash*. A continuación, se barre la relación izquierda, y los valores apropiados de cada tupla encontrada se utilizan como clave indexada para localizar las tuplas de la relación derecha.

Hash-Join

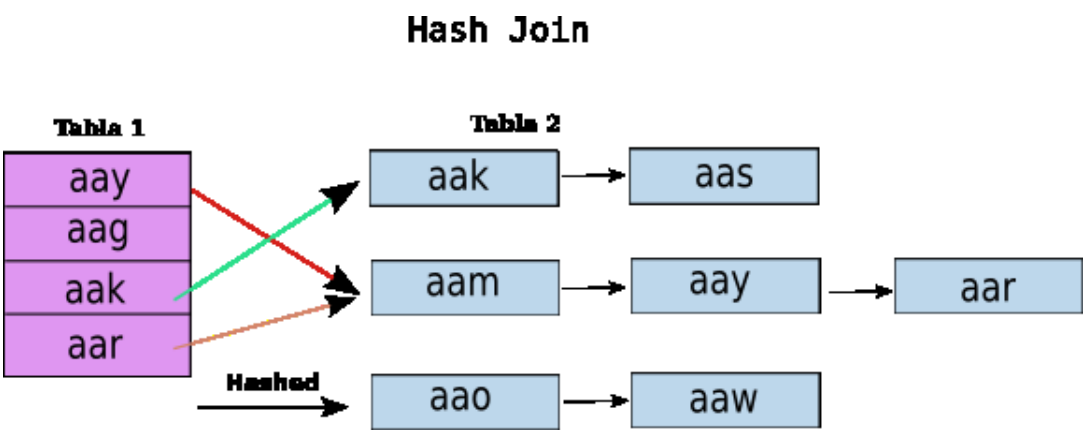

Figura 2.4: Fuente: Bruce Momjian, PostgreSQL Internals Trough Pictures.

- El *ejecuto*r de modo recursivo recupera las tuplas del árbol del plan. El ejecutor hace uso del sistema de almacenamiento mientras está revisando las relaciones, realiza ordenaciones (sorts) y joins, evalúa cualificaciones y finalmente devuelve las tuplas derivadas.

2.3.3. Preparación del entorno

Uno de los principales objetivos en afinar un sistema (hardware, software) es eliminar los cuellos de botella, estos producen una gran limitante al rendimiento de la Base de Datos.

Existen 3 aspectos importantes para mejorar el desempeño de las base de datos. Uno es mejorar el uso de CPU, Memoria y Disco (Hardware), el segundo es optimizar la configuración del mismo motor (Software) y el tercero es optimizar la codificación de las consultas (SQL).[5]

Para que PostgreSQL funcione de manera correcta, se necesitan afinar ciertos parámetros de configuración. Con esto se ayuda al usuario a optimizar el desem-

[5]Microsoft SQL Server 7.0 Performance Tunning.

peño de PostgreSQL en el uso de recursos de hardware. Debido a que muchos de los problemas de tiempo respuesta en consultas son por mal uso de hardware precisamente, luego esta la poca eficiencia en el uso de SQL.

Si una vez que se ha configurado bien el motor de PostgreSQL, y todavía existe lentitud en la respuesta de las consultas, entonces es altamente probable que la consulta, no esta correctamente construida. Es aquí donde va a puntar la aplicación, ya que como anteriormente se menciono se carece de una herramienta gráfica que facilite esta labor.

Se requiere:

- Una fácil lectura de la información con el objetivo de determinar y mostrar los cuellos de botella de una consulta SQL.
- Desplegar el método de búsqueda y procesamiento de una consulta SQL, procedimientos almacenados, entre otros.
- Mostrar gráficamente los algoritmos empleados por el motor de la Base de Datos, para llevar a cabo las consultas.
- Conocer el tiempo real y estimado de cada una de las partes de una consulta SQL.
- Ofrecer recomendaciones, que permitan al desarrollador optimizar las consultas y/o procedimientos almacenados.

En la actualidad solo desarrolladores avanzados tienen la capacidad de procesar la información entregada por el EXPLAIN o EXPLAIN ANALYZE, debido que la salida es texto plano, complicando en gran medida la lectura a los desarrolladores novatos, siendo estos últimos los más propensos a desarrollar consultas costosas en tiempo de ejecución.

2.3.3.1. Mejoras en rendimiento de hardware y software

A continuación se habla de algunos aspectos importantes a considerar al momento de afinar la Base de Datos.

Adaptación del articulo de Bruce Momjian (PostgreSQL Hardware Performance Tunning).

2.3.3.2. RAM y paginación

Es preciso recordar, aunque sea de forma somera, el papel de la memoria RAM en una computadora, así como los efectos indeseados de la paginación.

La memoria RAM por sus caracteristicas es un recurso limitado dividido en segmentos. Los segmentos, de un tamaño fijo, se agrupan formando páginas de memoria. En la memoria se guarda todo lo que la CPU necesita para hacer su trabajo, esto incluye programas, datos requeridos por los programas, el kernel[6], entre otros y por supuesto, las zonas de trabajo de PostgreSQL.

Para entender este proceso se necesita conocer como el Sistema Operativo (Linux) ver (figura: 2.5), administra la memoria.

Cuando no existe suficiente memoria RAM, el kernel lo que hace es escribir páginas de memoria en el Disco Duro, esto se conoce como *swap*[7]. El Sistema Operativo se encarga de mover las páginas que no están siendo utilizadas frecuentemente a la *swap*, conociéndose como *swap pageout*, lo cual no es problema ya que esto sucede durante tiempos de inactividad de la CPU. El problema surge cuando se copia de swap a RAM (*swap pagein*), es decir, del disco duro a RAM y por lo tanto a CPU, para la copia swap pagein solo se logra deteniendo el programa que lo solicita.

El resto de la afinación va a consistir en optimizar el uso de memoria para PostgreSQL, evitando en lo posible el número de intercambios con la swap (*swap pagein*). El mejor ajuste de los parámetros de configuración es aquél que dispone la máxima disponibilidad en memoria para la Base de Datos, sin perjudicar al resto de los elementos que también deben estar en la memoria.

[6] Núcleo de LINUX

[7] Sustituto de memoria RAM que se asigna al disco duro (Linux).

Representación swap (page-in, page-out).

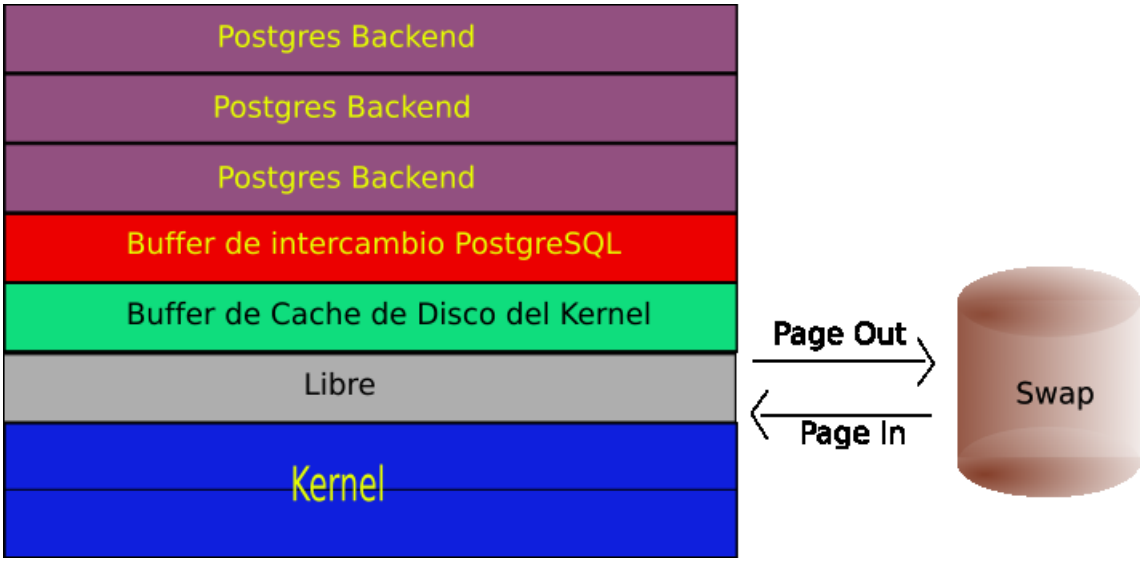

Figura 2.5: Fuente: Adaptación de Diagrama Bruce Momjian.

2.3.3.3. Buffer de intercambio de PostgreSQL

PostgreSQL para arrancar necesita de cierta cantidad de memoria (*shared_buffers*) para procesar las consultas, pero el tamaño que se le asigne no debe afectar a los demás procesos como el sistema operativo, siendo este de principal importancia, ya que todas las aplicaciones se sustentan en el, incluso el mismo motor de Base de Datos.[8]

Para conocer mejor de que forma PostgreSQL accede a la memoria ver (figura 2.6), se puede observar que el motor de PostgreSQL busca si los bloques (para acceder a las tablas), se encuentran en el *Buffer de intercambio de PostgreSQL* (*shared_buffers*), de ser así continua la lectura/escritura. De lo contrario al no encontrarse estos bloques, se cargan desde el *Buffer de Disco del Kernel* (Kernel Disk Buffer Cache), si aún no se encuentran en este buffer, se necesitarán cargar

[8]Documentación de PostgreSQL.

del disco. Esta operación de por sí, es muy costosa para el Sistema Operativo.

La configuración predeterminada de PostgreSQL, permite 64 buffer de intercambio, cada uno de un tamaño de 8 kilobytes. Incrementar la cantidad de buffers (a través del comando *postmaster*[9]) permite almacenar mayor cantidad de bloques en el *buffer de intercambio de PostgreSQL*, de este modo se evita una operación costosa para el Sistema Operativo, mejorando de forma considerable el rendimiento de la base de datos.

Lo anterior induce a que muchas personas piensen que lo mas conveniente, es asignarle todo el recurso de la RAM a PostgreSQL, pero como se explicó anteriormente (en 2.3.3.2 en la página 22), no es recomendable hacer esto. La asignación recomendada para *buffer de intercambio de PostgreSQL* es darle mayor tamaño no utilizado por otros procesos (el espacio vacío).

Si el espacio ocupado *shared_buffers* excede el tamaño máximo de segmentos (en Linux de forma predeterminada es de 32 megas) de memoria PostgreSQL se negará a arrancar.

[9]Comando para arrancar la base de datos.

Representación interna de PostgreSQL.

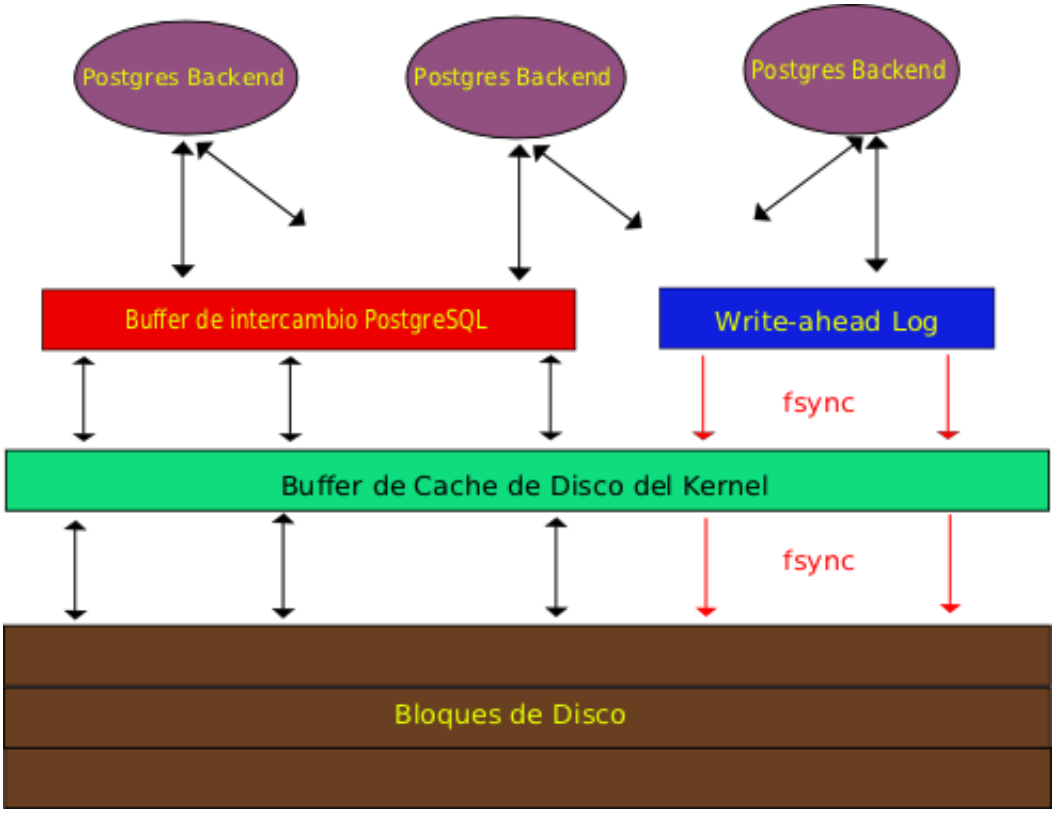

Figura 2.6: Fuente: Adaptación Diagrama Bruce Momjian.

2.3.3.4. Tamaño de la Memoria de Ordenamiento

Cuando PostgreSQL ordena grandes tablas o resultados, lo que hace es colocar en archivos temporales el resultado de la clasificación. Por lo tanto es recomendable que el motor cree pocos archivos temporales mejorando con esto la rapidez del ordenamiento. Si los ordenamientos demoran mucho se produce *swap*, ya que mucho bloques permanecen en inactividad durante el ordenamiento.

Al motor se le puede indicar como tratar los archivos temporales cambiando

el parámetro *Batch Size*[10].

2.3.3.5. Acceso a Disco

La naturaleza física de los discos duros hace que su desempeño se vea disminuido con respecto a otros medios de almacenamiento ya que sus componentes mecánicos y sus constantes movimientos de platos y cabezales no permiten una lectura / escrituta rápida como la memoria principal.

En el caso de PostgreSQL se prefiere que la exploración sea secuencial, ya que con esto se consigue que el cabezal ejecute menos movimientos.

Algo que se debe evitar son grandes cantidades de solicitudes de lectura / escritura (conocido como saturación), este caso se da por ejemplo si se solicita información a la Base de Datos, el motor lo que hará es sacar dicha información y al mismo tiempo estará escribiendo en los *logs*[11], significando esto que cabezal del disco duro este en constante movimiento lo cual hace que el rendimiento de la Base de Datos baje considerablemente.

La solución para evitar la saturación es mover algunos archivos (logs, índices, joins, entre otros) de PostgreSQL a otros discos (claro que con la salvedad que todos los discos tengan igual sistema de archivos ext2, ext3, xfs).

2.3.3.6. Sistema de Archivos

Algunos sistemas operativos pueden soportan discos con múltiples sistemas de archivos. En tal caso, esto hace difícil la elección de cual sistema de archivos cumple mejor desempeño. PostgreSQL usualmente tiene mejor rendimiento sobre sistemas de archivos UNIX tales como BSD USF/SFF, porque el tamaño de una página de PostgreSQL es igual a un bloque de UFS (8 kilobytes).

NFS y otros sistemas de archivos remotos no son recomendados para usar con PostgreSQL. NFS no tiene la misma semántica de un sistema de archivos local, y

[10] Parámetro de configuración de PostgreSQL.
[11] Archivo que almacena todo lo que pasa en la base de datos.

estas inconsistencias puede provocar información poco confiable y problemas de recuperación de la información.

Afortunadamente, PostgreSQL no requiere de mucho afinamiento, muchos de los parámetros son ajustados automáticamente para mantener un óptimo desempeño.

2.3.4. Índices

Los índices son un sistema especial que utilizan las bases de datos para mejorar su rendimiento global. Al definir índices en las columnas de una tabla, se le indica a PostgreSQL que preste atención especial a dichas columnas.

Los índices permiten que las consultas se ejecuten más rápido, pero no se puede pensar que lo mejor sería indexar todas las columnas. Es necesario tener claro que los índices tienen un precio. Cada vez que se realiza un INSERT sobre una tabla, PostgreSQL tiene que actualizar cada índice en la tabla para reflejar los cambios en los datos.

Los índices también benefician a comandos como UPDATE y DELETE con condiciones de búsqueda. Índices pueden además ser usados en consultas con JOINS. Así de este modo, un índice definido sobre la condición de un JOIN incrementa de forma considerable la velocidad de búsqueda.

Por ejemplo para la siguiente consulta:

```
SELECT * FROM a,b,c
    WHEREa.id = b.id AND b.ref = c.id;
```

El *planner* es libre de elegir que tablas va a unir para obtener los datos. Así el *planner* , podría generar un plan que una A con B usando la condición del WHERE a.id = b.id, y del resultado unirlo con C usando la otra condición del WHERE. O podría generar otro plan que una B con C y el resultado con A. Pero todos lo planes que va a generar son ineficientes, ya que para poder hacer la unión de las tablas ya sea A con B o B con C necesita hacer un producto cartesiano (A * B ó B * C).

La alternativa a esta consulta es utilizar la cláusula JOIN (en PostgreSQL permite la unión entre 2 tablas), si bien es cierto el resultado será el mismo pero con la diferencia que la búsqueda de los datos será mucho más eficiente.

Ahora la consulta queda de la siguiente manera:

```
SELECT * FROM a
    LEFT JOIN
    (b JOIN c ON (b.ref = c.id)) ON (a.id = b.id);
```

Las condiciones de esta consulta son similares a la anterior, pero semánticamente son distintas porque una fila es retornada, por cada fila de A que no se encuentre en la unión de B y C. Por lo tanto el *planner* no tiene elección en el orden de unión de los datos (como en el ejemplo anterior), no obstante el *planner* primero tiene que hacer una unión entre B con C y el resultado unirlo con A (asocitividad). Así de este modo esta consulta toma menos tiempo en ejecutarse que la anterior.

De la explicación anterior se deduce que la razón para tener un índice en una columna es para permitirle al motor de PostgreSQL que ejecute las búsquedas tan rápido como sea posible (y evitar los barridos completos de tablas). Un índice contiene una entrada por cada valor único en la columna. En el índice, PostgreSQL debe considerar cualquier valor duplicado. Estos valores duplicados decrementan la eficiencia y la utilidad del índice.

Así que antes de indexar una columna, se debe considerar que porcentaje de entradas en la tabla son duplicadas. Si el porcentaje es demasiado alto, seguramente no veremos alguna mejora con el uso de un índice.

Otra cosa a considerar es la frecuencia de uso de los índices o en la cantidad de datos solicitados. PostgreSQL puede usar un índice para una columna en particular, siempre que dicha columna aparezca en la cláusula WHERE de la consulta. Si muy rara vez se usa la columna en una cláusula WHERE, seguramente no tiene mucha sentido indexar dicha columna. Así de esta manera, probablemente sea más eficiente efectuar la búsqueda completa de la tabla, debido al poco uso de la columna en la consulta o porque la cantidad de datos solicitados es ínfima .

2.3.4.1. Tipos de índices

Como se explicó previamente los índices mejoran de gran manera la búsqueda de información, pero cabe señalar que no siempre el índice será utilizado por el *planner*, esto va a depender de los cálculos echos por el motor, ya que es el quien decide que plan ha sido el menos costoso.

Por esto que es de vital importancia saber para los desarrolladores si el *planner*, esta utilizando el índice, si este realmente es un aporte al mejoramiento del proceso de la consulta, lo empeora o es indiferente. En muchas ocasiones los desarrolladores crean índices y asumen que serán empleados, pero no siempre es así (los motivos estan dados anteriormente) debido a estas razones que sería de gran ayuda contar con alguna aplicación que permita ver de forma gráfica la planificación de la consulta echa por el *planner*.

PostgreSQL provee diversos tipos de índices: B-Tree, R-Tree y Hash ver (cuadro 2.1). Cada tipo usa un algoritmo diferente y se utilizan para diferentes tipos de consultas.

Algoritmos de Indices

Índices	Algoritmos
B-Tree	Lehman-Yao
R-Tree	Guttman
Hash	Litwins linear Hashing

Cuadro 2.1: Fuente: Documentación PostgreSQL.

2.3.4.1.1. B-Tree Es un árbol con raíz donde :

- Cada nodo x tiene:
  - N[x] numero de claves
  - Las clave estan ordenadas de mayor a menor $k_1[x]$ <...$k_n[x]$

- Una variable Boolean hoja[x], que es verdadera cuando el nodo x es una hoja.

- Si el nodo es interno, tiene n[x] +1 punteros hijos.
- k_i[x] separa los rangos que se almacenan.
- Las hojas estan a la misma altura.
- dado un t >= 2, nodos excepto la raiz tiene t-1 claves como minimo y 2t-1 claves como máximo.[12]

B-Tree del alfabeto con t=3

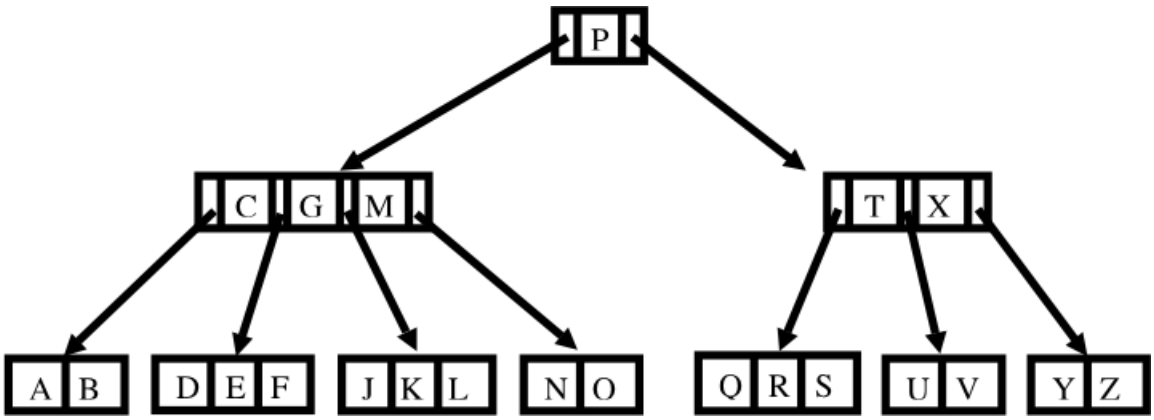

Figura 2.7: Fuente: www.inf.udec.cl/~andrea/cursos/retrieval/indexes.pdf

Dado que n>=1, entonces cualquier n-key B-Tree de altura h y minimo grado t>=2 satisface que

h <= $\log_t(n+1/2)$

En PostgreSQL al crear un índice con el comando CREATE INDEX por omisión crea un B-TREE, usado principalmente para consultas que utilizan los siguientes operadores de comparación tales como <, <=, =, >, >=.

[12]Definición de www.inf.udec.cl/~andrea/cursos/retrieval/indexes.pdf

2.3.4.1.2. R-Tree Es una estructura de datos para búsqueda en espacios métricos. R-Tree puede manejar datos multi-dimensionales, por ejemplo si un índice se crea sobre un tipo de dato *point,* el *planner* puede responder de manera mas eficiente consultas como: 'Seleccionar todos los puntos dentro de un rectangulo'.[13]

El motor de PostgreSQL utilizará los índices R-TREE cuando necesite hacer comparaciones utilizando estos operadores tales como < <, &<, &>, > >, @, ~=, &&(Funciones geométricas).

2.3.4.1.3. Hash Es un método de búsqueda que aumenta la velocidad de búsqueda, pero que no requiere que los elementos estén ordenados. Consiste en asignar a cada elemento un índice mediante una transformación del elemento. Esta correspondencia se realiza mediante una función de conversión, llamada función hash.

En PostgreSQL el índice hash sólo puede manejar simples comparaciones de igualdad, el operador utilizado es =.

2.3.5. Mecanismos provistos por PostgreSQL para mejorar el rendimiento

Existen diversos mecanismos asociados al mantenimiento de la Base de Datos que permiten que PostgreSQL trabaje de una manera óptima, estos se pueden acceder a través de sentencias tales como : VACUUM, EXPLAIN, ANALYZE.

2.3.5.1. VACUUM

VACUUM es un comando SQL para recuperar almacenamiento por tuplas borradas. Cuando las tuplas son borradas o están obsoletas por alguna actualización del motor, estas no son removidas físicamente de sus tablas, ellas permanecen solo hasta que se aplica el comando *vacuum.*

Razones por lo cual ejecutar periódicamente un *vacuum* a la Base de Datos

[13] http://www.postgresql.org/docs/faq-english.html.

1. Recupera espacio en disco ocupado por tuplas borradas o actualizadas.
2. Actualizar la estadística utilizada por el planificador de consultas de PostgreSQL.
3. Protección contra los daños de perdida de información.[14]

2.3.5.2. EXPLAIN

PostgreSQL realiza un plan de cada consulta que recibe. Al usar el comando EXPLAIN se puede observar que tipo plan a creado el *planner* para la consulta.

Los número desplegados por EXPLAIN son:

- Costo estimado en arrancar la consulta (Tiempo estimado antes de mostrar las filas).
- Número estimado de filas mostradas.
- Carga estimada (en bytes) de filas mostradas.

Los costos son medidos en unidades de páginas de disco obtenidas.

El cálculo del costo para la consulta

```
EXPLAIN SELECT * FROM usuarios;
  QUERY PLAN
-------------------------------------------------------------
Seq Scan on usuarios  (cost=0.00..333.00 rows=10000 width=148)
```

Para la tabla usuarios el total de filas es 10000 con 233 (tabla *pg_class* la columna *realpages*) páginas de disco, entonces el costo estimado es el total de filas de la tabla 10000 * *cpu_tuple_cost* (el cual es 0.01) más el resultado de las páginas de disco 233. Esto es solo un ejemplo de como calcula el costo el planner, ya que va a depender de que la información estadística del motor este actualizada y además del tipo de optimización que el mimo motor hace internamente.[15]

[14] Documentación de PostgreSQL.
[15] Documentación de PostgreSQL.

2.3.5.3. ANALYZE

Comando utilizado para actualizar las estadísticas de las tablas. Estas estadísticas son utilizadas por el *planner* para la elección del plan mas óptimo para las consultas.

Para actualizar las estadísticas de las tablas, el comando lo que hace es buscar si existen cambios en la tabla a analizar. Por ejemplo, existe una tabla productos con 0 elementos, al ver la información en la tabla *pg_class*[16], especificamente en la columna *reltuples*[17] indica que tiene 0 filas la tabla productos, ahora si se hace un *INSERT* en la tabla y nuevamente se observa la tabla *pg_class* en la columna *reltuple*s seguirá indicando las 0 filas, ya que no se ha actualizado. Esto hará que el *planner* no genere un plan óptimo para la tabla productos. Por esta razón que se recomienda aplicar el comando ANALYZE a la tabla productos, ya que el motor buscará los cambios realizados (en este caso la inserción) y actualizará la estadística almacenada en la tabla de sistema *pg_class*. Esto es solo un ejemplo genérico, de como realiza el trabajo ANALYZE.

2.4. Tecnología .NET

Es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con el objetivo de obtener una plataforma sencilla y potente para distribuir el software en forma de servicios que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados. Ésta es la llamada plataforma .NET, y a los servicios antes comentados se les denomina servicios Web.NET.

[16]Tabla almacena todas las tablas del motor con sus estadísticas.
[17]Número de filas en la tabla.

2.4.1. Common language runtime (CLR)

El Common Language Runtime (CLR) es el núcleo de la plataforma .NET. Es el motor encargado de gestionar la ejecución de las aplicaciones desarrolladas para la plataforma y a las que ofrece numerosos servicios que simplifican su desarrollo y favorecen su fiabilidad y seguridad. Las principales características y servicios que ofrece el CLR son:

- *Modelo de programación consistente*: A todos los servicios y facilidades ofrecidos por el CLR se accede de la misma forma: a través de un modelo de programación orientado a objetos. Esto es una diferencia importante respecto al modo de acceso a los servicios ofrecidos por los algunos sistemas operativos actuales (por ejemplo, los de la familia Windows), en los que a algunos servicios se les accede a través de llamadas a funciones globales definidas en DLLs y a otros a través de objetos (objetos COM en el caso de la familia Windows)

- *Modelo de programación sencillo*: Con el CLR desaparecen muchos elementos complejos incluidos en los sistemas operativos actuales (registro de Windows, GUIDs, HRESULTS, IUnknown, etc.) El CLR no es que abstraiga al programador de estos conceptos, sino que son conceptos que no existen en la plataforma .NET

- *Eliminación del infierno de las DLLs*: En la plataforma .NET desaparece el problema conocido como *infierno de las DLLs* que se da en los sistemas operativos actuales de la familia Windows, problema que consiste en que al sustituirse versiones viejas de DLLs compartidas por versiones nuevas puede que aplicaciones que fueron diseñadas para ser ejecutadas usando las viejas dejen de funcionar si las nuevas no son 100 % compatibles con las anteriores. En la plataforma .NET las versiones nuevas de las DLLs pueden coexistir con las viejas, de modo que las aplicaciones diseñadas para ejecutarse usando las viejas podrán seguir usándolas tras instalación de las

nuevas. Esto, obviamente, simplifica mucho la instalación y desinstalación de software.

- *Ejecución multiplataforma:* El CLR actúa como una máquina virtual, encargándose de ejecutar las aplicaciones diseñadas para la plataforma .NET. Es decir, cualquier plataforma para la que exista una versión del CLR podrá ejecutar cualquier aplicación .NET. Microsoft ha desarrollado versiones del CLR para la mayoría de las versiones de Windows: Windows 95, Windows 98, Windows ME, Windows NT 4.0, Windows 2000, Windows XP y Windows CE (que puede ser usado en CPUs que no sean de la familia x86) Por otro lado Microsoft ha firmado un acuerdo con Corel para portar el CLR a Linux y también hay terceros que están desarrollando de manera independiente versiones de libre distribución del CLR para Linux. Asimismo, dado que la arquitectura del CLR está totalmente abierta, es posible que en el futuro se diseñen versiones del mismo para otros sistemas operativos.

- *Integración de lenguajes:* Desde cualquier lenguaje para el que exista un compilador que genere código para la plataforma .NET es posible utilizar código generado para la misma usando cualquier otro lenguaje tal y como si de código escrito usando el primero se tratase. Microsoft ha desarrollado un compilador de C# que genera código de este tipo, así como versiones de sus compiladores de Visual Basic (Visual Basic.NET) y C++ (C++ con extensiones gestionadas) que también lo generan y una versión del intérprete de JScript (JScript.NET) que puede interpretarlo. La integración de lenguajes esta que es posible escribir una clase en C# que herede de otra escrita en Visual Basic.NET que, a su vez, herede de otra escrita en C++ con extensiones gestionadas.

- *Gestión de memoria:* El CLR incluye un recolector de basura que evita que el programador tenga que tener en cuenta cuándo ha de destruir los objetos que dejen de serle útiles. Este recolector es una aplicación que se activa cuando se quiere crear algún objeto nuevo y se detecta que no queda memo-

ria libre para hacerlo, caso en que el recolector recorre la memoria dinámica asociada a la aplicación, detecta qué objetos hay en ella que no puedan ser accedidos por el código de la aplicación, y los elimina para limpiar la memoria de *objetos basura* y permitir la creación de otros nuevos. Gracias a este recolector se evitan errores de programación muy comunes como intentos de borrado de objetos ya borrados, agotamiento de memoria por olvido de eliminación de objetos inútiles o solicitud de acceso a miembros de objetos ya destruidos.

- *Seguridad de tipos:* El CLR facilita la detección de errores de programación difíciles de localizar comprobando que toda conversión de tipos que se realice durante la ejecución de una aplicación .NET se haga de modo que los tipos origen y destino sean compatibles.

- *Aislamiento de procesos:* El CLR asegura que desde código perteneciente a un determinado proceso no se pueda acceder a código o datos pertenecientes a otro, lo que evita errores de programación muy frecuentes e impide que unos procesos puedan atacar a otros. Esto se consigue gracias al sistema de seguridad de tipos antes comentado, pues evita que se pueda convertir un objeto a un tipo de mayor tamaño que el suyo propio, ya que al tratarlo como un objeto de mayor tamaño podría accederse a espacios en memoria ajenos a él que podrían pertenecer a otro proceso. También se consigue gracias a que no se permite acceder a posiciones arbitrarias de memoria.

- *Tratamiento de excepciones:* En el CLR todo los errores que se puedan producir durante la ejecución de una aplicación se propagan de igual manera: mediante excepciones. Esto es muy diferente a como se venía haciendo en los sistemas Windows hasta la aparición de la plataforma .NET, donde ciertos errores se transmitían mediante códigos de error en formato Win32, otros mediante HRESULTs y otros mediante excepciones.

El CLR permite que excepciones lanzadas desde código para .NET escrito en un cierto lenguaje se puedan capturar en código escrito usando otro lenguaje, e inclu-

ye mecanismos de depuración que pueden saltar desde código escrito para .NET en un determinado lenguaje a código escrito en cualquier otro. Por ejemplo, se puede recorrer la pila de llamadas de una excepción aunque ésta incluya métodos definidos en otros módulos usando otros lenguajes.

- *Soporte multihilo:* El CLR es capaz de trabajar con aplicaciones divididas en múltiples hilos de ejecución que pueden ir evolucionando por separado en paralelo o intercalándose, según el número de procesadores de la máquina sobre la que se ejecuten. Las aplicaciones pueden lanzar nuevos hilos, destruirlos, suspenderlos por un tiempo o hasta que les llegue una notificación, enviarles notificaciones, sincronizarlos, etc.

- *Distribución transparente:* El CLR ofrece la infraestructura necesaria para crear objetos remotos y acceder a ellos de manera completamente transparente a su localización real, tal y como si se encontrasen en la máquina que los utiliza.

- *Seguridad avanzada:* El CLR proporciona mecanismos para restringir la ejecución de ciertos códigos o los permisos asignados a los mismos según su procedencia o el usuario que los ejecute. Es decir, puede no darse el mismo nivel de confianza a código procedente de Internet que a código instalado localmente o procedente de una red local; puede no darse los mismos permisos a código procedente de un determinado fabricante que a código de otro; y puede no darse los mismos permisos a un mismo códigos según el usuario que lo esté ejecutando o según el rol que éste desempeñe. Esto permite asegurar al administrador de un sistema que el código que se esté ejecutando no pueda poner en peligro la integridad de sus archivos, la del registro de Windows, etc.

- *Interoperabilidad con código antiguo:* El CLR incorpora los mecanismos necesarios para poder acceder desde código escrito para la plataforma .NET a código escrito previamente a la aparición de la misma y, por tanto, no

preparado para ser ejecutando dentro de ella. Estos mecanismos permiten tanto el acceso a objetos COM como el acceso a funciones sueltas de DLLs preexistentes (como la API Win32)

Como se puede deducir de las características comentadas, el CLR lo que hace es gestionar la ejecución de las aplicaciones diseñadas para la plataforma .NET. Por esta razón, al código de estas aplicaciones se le suele llamar código gestionado, y al código no escrito para ser ejecutado directamente en la plataforma .NET se le suele llamar código no gestionado.

2.4.2. Microsoft intermediate language (MSIL)

Todos los compiladores que generan código para la plataforma .NET no generan código máquina para CPUs x86 ni para ningún otro tipo de CPU concreta, sino que generan código escrito en el lenguaje intermedio conocido como Microsoft Intermediate Language (MSIL) El CLR da a las aplicaciones las sensación de que se están ejecutando sobre una máquina virtual, y precisamente MSIL es el código máquina de esa máquina virtual. Es decir, MSIL es el único código que es capaz de interpretar el CLR, y por tanto cuando se dice que un compilador genera código para la plataforma .NET lo que se está diciendo es que genera MSIL.

MSIL ha sido creado por Microsoft tras consultar a numerosos especialistas en la escritura de compiladores y lenguajes tanto del mundo académico como empresarial. Es un lenguaje de un nivel de abstracción mucho más alto que el de la mayoría de los códigos máquina de las CPUs existentes, e incluye instrucciones que permiten trabajar directamente con objetos (crearlos, destruirlos, inicializarlos, llamar a métodos virtuales, etc.), tablas y excepciones (lanzarlas, capturarlas y tratarlas).

Ya se comentó que el compilador de C# compila directamente el código fuente a MSIL, que Microsoft ha desarrollado nuevas versiones de sus lenguajes Visual Basic (Visual Basic.NET) y C++ (C++ con extensiones gestionadas) cuyos compiladores generan MSIL, y que ha desarrollado un intérprete de JScript (JS-

cript.NET) que genera código MSIL. Pues bien, también hay numerosos terceros que han anunciado estar realizando versiones para la plataforma .NET de otros lenguajes como APL, CAML, Cobol, Eiffel, Fortran, Haskell, Java, Mercury, ML, Mondrian, Oberon, Oz, Pascal, Perl, Python, RPG, Scheme y Smalltalk.

La principal ventaja del MSIL es que facilita la ejecución multiplataforma y la integración entre lenguajes al ser independiente de la CPU y proporcionar un formato común para el código máquina generado por todos los compiladores que generen código para .NET. Sin embargo, dado que las CPUs no pueden ejecutar directamente MSIL, antes de ejecutarlo habrá que convertirlo al código nativo de la CPU sobre la que se vaya a ejecutar. De esto se encarga un componente del CLR conocido como compilador JIT (Just-In-Time) o jitter que va convirtiendo dinámicamente el código MSIL a ejecutar en código nativo según sea necesario. Este jitter se distribuye en tres versiones:

- jitter normal: Es el que se suele usar por defecto, y sólo compila el código MSIL a código nativo a medida que va siendo necesario, pues así se ahorra tiempo y memoria al evitarse tener que compilar innecesariamente código que nunca se ejecute. Para conseguir esto, el cargador de clases del CLR sustituye inicialmente las llamadas a métodos de las nuevas clases que vaya cargando por llamadas a funciones auxiliares (stubs) que se encarguen de compilar el verdadero código del método. Una vez compilado, la llamada al stub es sustituida por una llamada directa al código ya compilado, con lo que posteriores llamadas al mismo no necesitarán compilación.

- jitter económico: Funciona de forma similar al jitter normal solo que no realiza ninguna optimización de código al compilar sino que traduce cada instrucción MSIL por su equivalente en el código máquina sobre la que se ejecute. Esta especialmente pensado para ser usado en dispositivos empotrados que dispongan de poca potencia de CPU y poca memoria, pues aunque genere código más ineficiente es menor el tiempo y memoria que necesita para compilar. Es más, para ahorrar memoria este jitter puede descargar

código ya compilado que lleve cierto tiempo sin ejecutarse y sustituirlo de nuevo por el stub apropiado. Por estas razones, este es el jitter usado por defecto en Windows CE, sistema operativo que se suele incluir en los dispositivos empotrados antes mencionados.
Otra utilidad del jitter económico es que facilita la adaptación de la plataforma .NET a nuevos sistemas porque es mucho más sencillo de implementar que el normal. De este modo, gracias a él es posible desarrollar rápidamente una versión del CLR que pueda ejecutar aplicaciones gestionadas aunque sea de una forma poco eficiente, y una vez desarrollada es posible centrarse en desarrollar el jitter normal para optimizar la ejecución de las mismas.

- prejitter: Se distribuye como una aplicación en línea de comandos llamada ngen.exe mediante la que es posible compilar completamente cualquier ejecutable o librería que contenga código gestionado y convertirlo a código nativo, de modo que posteriores ejecuciones del mismo se harán usando esta versión ya compilada y no se perderá tiempo en hacer la compilación dinámica.

La actuación de un jitter durante la ejecución de una aplicación gestionada puede dar la sensación de hacer que ésta se ejecute más lentamente debido a que ha de invertirse tiempo en las compilaciones dinámicas. Esto es cierto, pero hay que tener en cuenta que es una solución mucho más eficiente que la usada en otras plataformas como Java, ya que en .NET cada código no es interpretado cada vez que se ejecuta sino que sólo es compilado la primera vez que se llama al método al que pertenece. Es más, el hecho de que la compilación se realice dinámicamente permite que el jitter tenga acceso a mucha más información sobre la máquina en que se ejecutará la aplicación del que tendría cualquier compilador tradicional, con lo que puede optimizar el código para ella generado (por ejemplo, usando las instrucciones especiales del Pentium III si la máquina las admite, usando registros extra, incluyendo código inline, etc.) Además, como el recolector de basura de .NET mantiene siempre compactada la memoria dinámica las reservas de memoria se harán más rápido, sobre todo en aplicaciones que no agoten la memoria y,

por tanto, no necesiten de una recolección de basura. Por estas razones, se pretende que futuras versiones de sus jitters puedan incluso conseguir que el código gestionado se ejecute más rápido que el no gestionado.

2.4.3. Metadatos

En la plataforma .NET se distinguen dos tipos de módulos de código compilado: ejecutables (extensión .exe) y bibliotecas de enlace dinámico (extensión .dll generalmente) Ambos son ficheros que contienen definiciones de tipos de datos, y la diferencia entre ellos es que sólo los primeros disponen de un método especial que sirve de punto de entrada a partir del que es posible ejecutar el código que contienen haciendo una llamada desde la línea de comandos del sistema operativo. A ambos tipos de módulos se les suele llamar ejecutables portables (PE), ya que su código puede ejecutarse en cualquiera de los diferentes sistemas operativos de la familia Windows para los que existe alguna versión del CLR.

El contenido de un módulo no sólo MSIL, sino que también consta de otras dos áreas muy importantes: la cabecera de CLR y los metadatos:

- La cabecera de CLR es un pequeño bloque de información que indica que se trata de un módulo gestionado e indica es la versión del CLR que necesita, cuál es su firma digital, cuál es su punto de entrada (si es un ejecutable), etc.
- Los metadatos son un conjunto de datos organizados en forma de tablas que almacenan información sobre los tipos definidos en el módulo, los miembros de éstos y sobre cuáles son los tipos externos al módulo a los que se les referencia en el módulo. Los metadatos de cada modulo los genera automáticamente el compilador al crearlo.

2.5. Mono

Mono es un proyecto libre y compatible patrocinado por Ximian, que consiste en desarrollar una plataforma de desarrollo libre y basada en Linux compatible con

Microsoft .NET. Su objetivo es permitir que los desarrolladores de GNU/Linux desarrollen aplicaciones multiplataforma basadas en .NET. El proyecto Mono implementará varias tecnologías desarrolladas por Microsoft que han sido enviadas al ECMA[18] para su estandarización.

El proyecto Mono también ha despertado gran interés por desarrollar componentes, librerías y plataformas basadas en C#. Mono ya no está limitado a implementar la plataforma .NET, sino que también contiene otros componentes. Algunos de los componentes de la plataforma Mono han sido desarrollados por el equipo Mono, y algunos otros han sido incorporados de otros proyectos de software libre, de los que los más importantes son:

- Un compilador para el lenguaje C# y Visual Basic.Net.
- Un entorno de ejecución virtual: Un compilador JIT (Just-In-Time = justo-a-tiempo, esto es, que compila el código justo antes de ser ejecutado), un compilador AOT (AOT=ahead-of-time, antes-de-tiempo , esto es, que compila a código nativo un archivo y de esta forma no necesita la compilación JIT cada vez que se ejecute el programa), gestión automática de memoria, un interprete (mint), motor multiproceso.
- Una máquina virtual para los bytecodes del Lenguaje Intermedio Común (CLI).
- Una implementación de la biblioteca de clases de .NET: manipulación XML, Remoting, Reflection.Emit, Xslt, etc.
- Biblioteca de clases multiplataforma para el acceso a bases de datos: PostgreSQL, MySQL, DB2, TDS, Sybase, Oracle, ODBC y Gnome-GDA.
- Biblioteca de clases UNIX: Mono.Posix.
- Biblioteca de clases GNOME: la familia Gtk#.

[18] Asociación Europea de Fabricantes de Computadoras.

Por lo comentado anteriormente puede existir un gran parecido a Java y la máquina virtual de Java, pero existe una gran diferencia ya que los bytecodes de Java (CLI en Mono) solo se generan a partir del lenguaje Java. En Mono se pueden generar CLI a partir de varios lenguajes tales como: C++, C, Fortran, Eiffel, Lisp, Java, C# y Visual Basic ver (figura 2.8).

Esto permite una independencia total del lenguaje, debido que se puede programar bibliotecas en C# y utilizarlas desde Python sin ninguna dificultad.

Mono multilenguaje

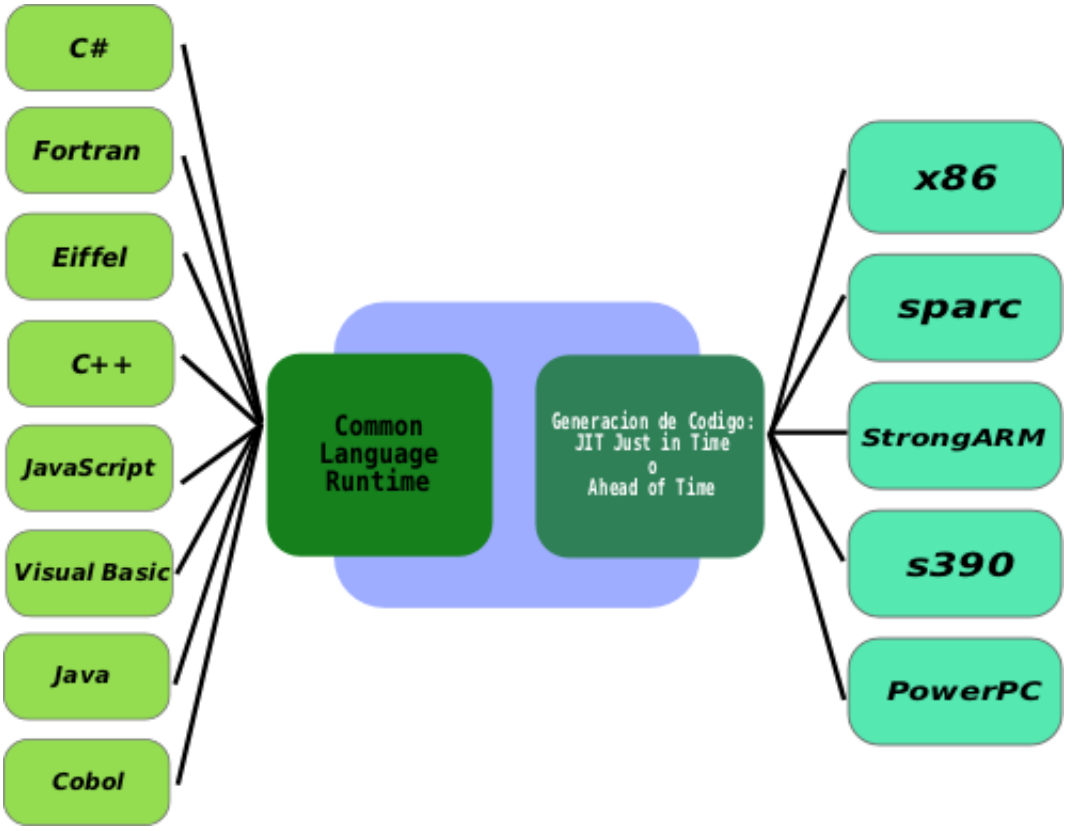

Figura 2.8: Fuente : Presentación Miguel Icaza.

2.6. GTK+

GTK+ significa GIMP Toolkit (conjunto de herramientas GIMP[19]). GIMP Toolkit es una biblioteca utilizada para desarrollar aplicaciones que tengan una interfaz gráfica de usuario (GUI, Graphical User Interface). Esta biblioteca se utiliza ampliamente hoy en día para desarrollar aplicaciones GUI para Linux. GIMP fue desarrollado con la biblioteca GTK+ y proporciona un ejemplo de aplicación con interfaz gráfica desarrollada profesionalmente. La biblioteca GTK+ puede tambien utilizarse para desarrollar aplicaciones en otras plataformas para las que se disponga de una version de dicha biblioteca. GTK+ es una biblioteca orientada a objetos escrita en C que puede utilizarse en aplicaciones escritas en diversos lenguajes. La lista de lenguajes permitidos incluye Ada95, C++, C#, Eiffel, Objective C, Pascal, Perl, Python y muchos otros.

GTK+ depende de la siguientes bibliotecas ver (figura 2.9):

1. GLib biblioteca de propósito general, no especifica para la creación de GUIs. GLib provee principalmente usos para tipos de datos, macros, tipo de conversiones, cadenas, entre otros.

2. Pango biblioteca para el manejo de texto internacional, centrándose alrededor del objeto PangoLayout, que representa el párrafo en el texto. Pango provee el motor para GtkText, GtkLabel, GtkEntry y otro widgets que manejan texto.

3. ATK es el kit de herramientas de accesibilidad. Provee un set de interfaces genéricas que permiten conexión con dispositivos de accesibilidad que interactúan con los usuarios discapacitados.

4. GdkPixbuf es una pequeña biblioteca que permite manipular imágenes.

5. GDK es una abstracción que permite a GTK+ usar múltiple sistema de ventanas. GDK provee dibujo y fácil manejo sobre X11, Windows y sobre dis-

[19] Graphical Image Manipulation (Manipulación de Imágenes Gráficas).

positivos de framebuffer sobre Linux.

Arquitectura bibliotecas GTK+.

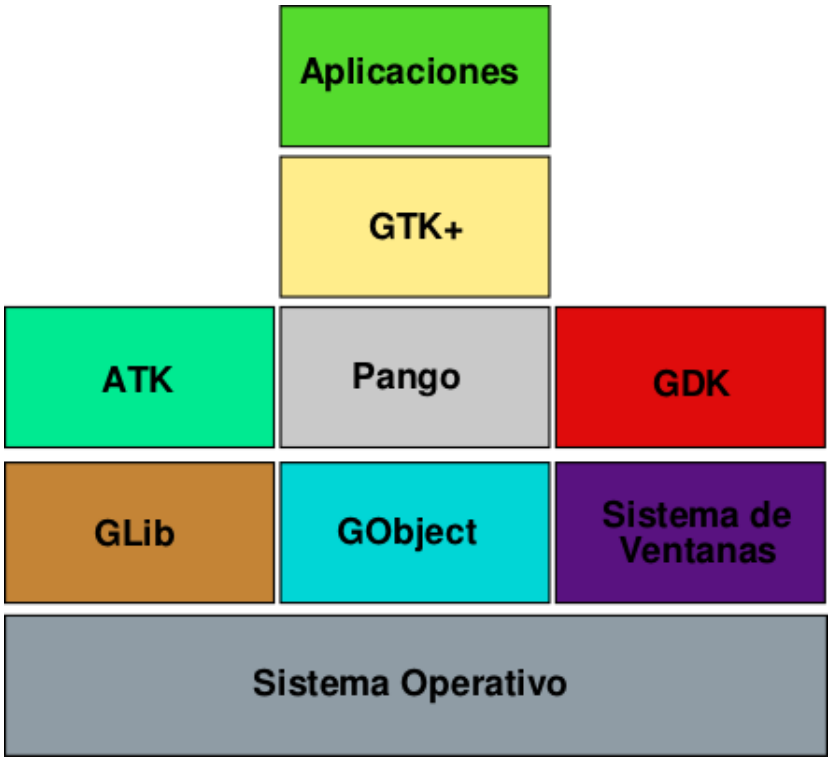

Figura 2.9: Fuente: Adaptación de Diagrama de Eric Harlow.

Capítulo 3

Análisis / Diseño

3.1. Descripción de la metodología utilizada

Para llevar a cabo un proyecto es muy importante contar con métodos que guíen y ayuden a obtener un producto de calidad.

Durante el desarrollo del proyecto se utilizarán procesos iterativos, inspirados en RUP, de esta forma resulta más factible implementar una pequeña parte del sistema, que implementar toda la aplicación de una sola vez.

El proceso de desarrollo se dividirá en 5 etapas que son:

- Requerimientos
- Análisis / Diseño
- Implementación
- Aplicación de Pruebas

3.1.0.1. Requerimientos

Esta fase corresponde a la recepción de requisitos del usuario, mediante el modelo de casos de uso. Se realizará también un análisis de riesgos por cada caso

de uso, que servirá para comprender el grado de complejidad que presenta cada caso y precisar el número de veces que deberá ser iterado para disminuir su riesgo.

3.1.0.2. Análisis / Diseño

En esta etapa se presenta todo el desarrollo del proyecto, se analizan los casos de uso mediante diagramas de colaboración para mostrar la interacción y la estructura de los objetos. Esta etapa comprende también el diseño de casos de uso, mediante diagramas de actividad, modelando así el comportamiento de las actividades realizadas, terminando en una síntesis representada por interfaces lógicas del sistema.

3.1.0.3. Construcción e implementación

Esta etapa contempla la construcción de los módulos del software y la puesta en marcha que se utilizará. Además, se elaboran las interfaces finales, que están asociadas a los casos de uso desarrollados en la fase anterior, detallando para ello la función de los códigos y tablas relacionadas a cada interfaz.

3.1.0.4. Pruebas

Esta fase comprende la ejecución de las pruebas que fueron elaboradas anteriormente. Implica, además, conocer el comportamiento del sistema y reconocer los errores y defectos que éste presente.

3.2. Estudio de factibilidad

3.2.1. Factibilidad operacional

3.2.1.1. Descripción

Consiste en la capacidad que tienen los usuarios para interactuar con la aplicación, por lo tanto, resulta de gran importancia analizar cuan eficiente es el com-

portamiento de cada individuo frente a él.

3.2.1.2. Análisis

Los usuarios que utilizarán la aplicación son en la mayoría gente con alto conocimiento de computación, dado a que la orientación de la aplicación es para Administradores de Base de Datos, aunque no se descarta el uso por parte de usuarios comunes.

3.2.1.3. Conclusión

Basados en los antecedentes previos del análisis de factibilidad operacional con respecto a los conocimientos de los usuarios de la aplicación, se puede concluir que la aplicación es operativamente factible.

3.2.2. Factibilidad técnica

3.2.2.1. Descripción

El estudio de Factibilidad Técnica permitirá determinar la disponibilidad de recursos técnicos para desarrollar la aplicación. Dichos recursos contemplan el hardware y el software necesario para desarrollar e implementar la aplicación.

3.2.2.2. Análisis

Los requerimientos técnicos para el desarrollo de esta aplicación son :

Debian GNU/Linux	Sistema Operativo
Lyx	Documentación
PostgreSQL	Motor de Base de Datos
MonoDevelop	Codificación

Cuadro 3.1: Requerimientos técnicos

Con respecto al Hardware:

- Equipo con procesador pentium III de 500Mhz, o superior, 256 Mb en RAM.

3.2.2.3. Conclusión

Dado que la mayoría del software necesario para el desarrollo de la apliación son de libre distribución y que se cuenta con el Hardware, la construcción de la herramienta es técnicamente factible.

3.2.3. Factibilidad económica

3.2.3.1. Descripción

El estudio de Factibilidad Económica permitirá determinar los costos en los cuales se debe incurrir para financiar el desarrollo y ejecución de la aplicación, analizando la relación costo / beneficio relacionada con el proyecto.

3.2.3.2. Análisis

Para la instalación de la aplicación no será necesario invertir en nuevos equipos computacionales. En cuanto al software, no se incurrirá en gastos adicionales ya que se construirá bajo software de libre distribución, la cual no tienen costo.

3.2.3.3. Conclusión

Por los antecedentes anteriormente expuesto se puede concluir que el sistema es económicamente factible.

3.2.3.4. Conclusión estudio factibilidad

De acuerdo a los resultados arrojados por cada uno de los estudios realizados, es posible determinar que el Estudio de Factibilidad es positivo, es decir, que es posible seguir adelante con las demás etapas de desarrollo.

3.3. Requerimientos

3.3.1. Datos de entrada

Base de Datos: Estos datos son necesarios para establecer la conexión con la BD.

- Nickname
- Base de Datos
- Host
- UserName
- Password

Consulta de SQL: Datos necesarios para ver árbol gráfico de la consulta.

- Consulta SQL.

Índices: Datos necesarios para la creación del índice.

- Nombre
- Tabla
- Columna
- Método de Acceso
- Único
- Comentarios

3.3.2. Salidas de la aplicación

Consultas

- Árbol Gráfico.
- Atributos del Query.
- Explain.
- Explain Analyze.
- Recomendaciones de índices.

3.3.3. Requerimientos funcionales

Los requerimientos se han clasificado de la siguiente forma:

Evidente: Se dice que un requerimiento es evidente cuando debe realizarse y el usuario debería saber que se ha realizado.

Oculto: Se dice que un requerimiento es oculto cuando debe realizarse, aunque no es visible para los usuarios.

R_F_#	Función	Categoría
R_F_1	Ver Árbol Gráfico	evidente
R_F_2	Ver atributos de la consulta	evidente
R_F_3	Mostrar Explain	evidente
R_F_4	Mostrar Explain Analyze	evidente
R_F_5	Ofrecer recomendaciones de índices	evidente
R_F_6	Generar archivo XML	oculto
R_F_7	Recuperar Estadística	oculto

Cuadro 3.3: Requerimientos: Funcionales

A_#	Atributo	Categoría
A_01	interfaz atractiva, opciones simples y pantallas amistosas, que permitan que cualquier tipo de usuario pueda hacer uso del sistema sin grandes dificultades, para así facilitar la aceptación del sistema por parte de los usuarios finales.	Facilidad de Uso
A_02	El sistema debe desplegar mensajes de advertencia, éxito o fallo según corresponda a la operación que se este llevando a cabo.	Metáfora de Interfaz
A_03	Esperar 5 segundos como máximo para cualquier solicitud que se pida a la aplicación.	Tiempo de Respuesta

Cuadro 3.5: Requerimientos: atributos de la aplicación

3.3.4. Atributos de la aplicación

3.3.5. Metas de la aplicación

Se refiere a lo que se desea lograr con la aplicación y lo que se espera de él. A continuación se mencionan los aspectos de mayor relevancia que se quieren lograr:

- Elaboración gráfica de la planificación de la consulta.
- Recomendar uso de índices.
- Permitir la creación de índices.
- Impresión del explain.
- Coloreo de sintaxis de las consultas SQL.

3.4. Realización de casos de uso

3.4.1. Diagrama de casos de uso

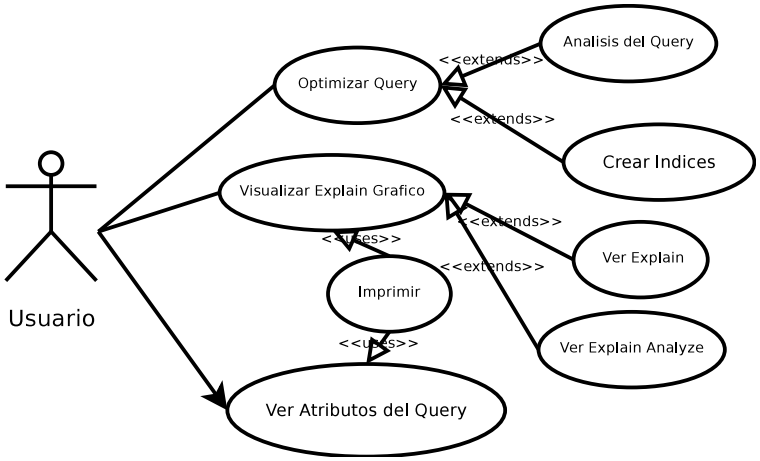

Figura 3.1: Diagrama de casos de uso

3.4.2. Actores

3.4.2.1. Identificación de actores

El actor identificado es : *Usuario*

3.4.2.2. Descripción de actores

Usuario : Se refiere a la persona que interactuará con la aplicación. Este actor puede realizar todas las acciones de la aplicación sin restricción.

3.4.3. Descripción de casos de uso

Casos de Uso	:	Visualizar Explain Gráfico

Descripción	:	El usuario ve el árbol gráfico de una consulta SQL y los datos asociados a ella.
Pre-condición	:	Para que un Usuario pueda ver el explain gráfico, es necesario que exista una consulta SQL.
Flujo Básico	:	1.El usuario ingresa una consulta SQL. 2.El caso de uso verifica que la consulta SQL este correcta. 3.La aplicación despliega el árbol gráfico de la consulta. 4.La instancia del caso de uso finaliza.
Flujo Alternativo	:	1. La de aplicación debe informarle al Usuario que uno de los datos esta incorrecto. 2. La Aplicación no debe permitir la entrada de código SQL, ni la generación del Explain gráfico.
Post-condición	:	El caso de uso termina cuando se genera el Explain gráfico.

3.4.3.0.1. Visualizar explain gráfico

Casos de Uso	:	Ver Explain
Descripción	:	La aplicación despliega al usuario la salida del Explain en modo texto, tal como lo hace PostgreSQL.
Pre-condición	:	Para que un Usuario pueda ver el explain , es necesario que exista una consulta SQL.
Flujo Básico	:	1.El usuario ingresa una consulta SQL. 2.El caso de uso verifica que la consulta SQL este correcta. 3.La aplicación despliega el Explain en texto de la consulta SQL. 4.La instancia del caso de uso finaliza.
Flujo Alternativo	:	1. La aplicación debe informarle al Usuario que uno de los datos esta incorrecto.
Post-condición	:	El caso de uso termina cuando se genera el Explain en modo texto.

3.4.3.0.2. Ver explain

Casos de Uso	:	Ver Explain Analyze
Descripción	:	1.El módulo es invocado por el caso de uso Optimización de Query.
Pre-condición	:	Para que un Usuario pueda ver el Explain Analyze, es necesario la ejecución previa del caso de uso ver Explain Gráfico.
Flujo Básico	:	1.El usuario invoca el caso de uso ver Explain Analyze. 2.El caso de uso verifica que la consulta SQL este correcta. 3.La aplicación despliega el Explain Analyze en texto de la consulta SQL. 4.La instancia del caso de uso finaliza.
Flujo Alternativo	:	1. La aplicación debe informarle al Usuario que uno de los datos esta incorrecto.
Post-condición	:	El caso de uso termina cuando se genera el Explain Analyze en modo texto.

3.4.3.0.3. Ver explain analyze

3.4.3.1. Optimizar query

Casos de Uso	:	Optimizar Query
Descripción	:	Este caso de uso presenta al usuario una alternativa y mejoras en el uso de índices.
Pre-condición	:	Para que la aplicación pueda recomendar el uso de índices, es necesario tener una consulta SQL.
Flujo Básico	:	1.El usuario ingresa una consulta SQL. 2.El caso de uso verifica que la consulta SQL este correcta. 3.EL caso de uso utiliza al módulo Análisis del Query para la elección de la solución más adecuada a la consulta realizada, recomendando un índice para reducir el tiempo respuesta de la información obtenida. 4.La instancia del caso de uso finaliza.
Flujo Alternativo	:	1. La aplicación debe informarle al Usuario que uno de los datos esta incorrecto.
Post-condición	:	El caso de uso termina cuando la aplicación despliega una recomendación de índice para la consulta.

Casos de Uso	:	Análisis del Query
Descripción	:	Este caso de uso analiza la consulta SQL y las tablas implicadas, con el objetivo de recomendar el uso adecuado de índices.
Pre-condición	:	Para que la aplicación pueda hacer un análisis del query, es necesario la ejecución previa del caso de uso Optimización del Query.
Flujo Básico	:	1.El módulo es invocado por el caso de uso Optimización de Query. 2.El caso de uso verifica que la consulta SQL este correcta 3.Análisis de consulta y evaluación del mejor caso para la generación del índice. 4.La instancia del caso de uso finaliza.
Flujo Alternativo	:	1. La aplicación debe informarle al Usuario que uno de los datos esta incorrecto.
Post-condición	:	El caso de uso termina cuando se ha evaluado y generado la solución más óptima.

3.4.3.1.1. Análisis del query

Casos de Uso	:	Crear Índices
Descripción	:	Este caso de uso permite al usuario crear índices.
Pre-condición	:	Para que la aplicación permita la creación de índices, es necesario la ejecución previa del caso de uso Optimización de Query.
Flujo Básico	:	1.El usuario invoca el caso de uso crear índices. 2.La aplicación le solicita al usuario datos como : -Nombre Índice -Método de Búsqueda -Tabla -Columnas 3.Crea el índice. 4.La instancia del caso de uso finaliza.
Flujo Alternativo	:	1. La aplicación debe informarle al Usuario que uno de los datos esta incorrecto.
Post-condición	:	El caso de uso termina cuando el índice es creado con éxito, de lo contrario informa del error.

3.4.3.1.2. Crear índices

3.4.3.2. Ver atributos del query

Casos de Uso	:	Ver Atributos del Query
Descripción	:	Este caso de uso muestra al usuario: -tipo de escaneo. -tiempo transcurrido antes de mostrar las filas. -costo total estimado. -número estimado de filas retornadas. -el promedio (bytes) de filas retornadas.
Pre-condición	:	Para que la aplicación pueda mostrar los datos es necesario tener una consulta SQL al cual ejecutarle el EXPLAIN.
Flujo Básico	:	1. El usuario invoca el caso de uso Ver Atributos del Query ingresando una consulta SQL. 2. El caso de uso verifica los datos y procesa la consulta . 3. El caso de uso despliega en pantalla la respuesta a la consulta SQL ingresada. 4. La instancia del caso de uso finaliza.
Flujo Alternativo	:	El caso 2 puede que los datos estén incorrectos, lo cual la aplicación informara el error.
Post-condición	:	El caso de uso termina cuando se despliegan los atributos de la consulta SQL o cuando la consulta es incorrecta.

3.4.3.3. Imprimir

Casos de Uso	:	Imprimir
Descripción	:	Este caso de uso imprime el árbol gráfico, los atributos de una consulta SQL o ambos.
Pre-condición	:	Para que la aplicación pueda imprimir los datos es necesario contar con el árbol gráfico o atributos de una consulta SQL.
Flujo Básico	:	1. El usuario invoca el caso de uso. 2. El caso de uso procesa los datos necesarios para imprimir. 3. El caso de uso imprime los datos solicitados por el usuario. 4. La instancia del caso de uso finaliza.
Flujo Alternativo	:	El paso 1 tiene la opción de cancelar.
Post-condición	:	El caso de uso termina cuando se imprimen los datos solicitados o cuando la operación imprimir es cancelada por el usuario.

3.5. Análisis de casos de uso

3.5.1. Diagramas de colaboración

3.5.1.1. Ver explain gráfico

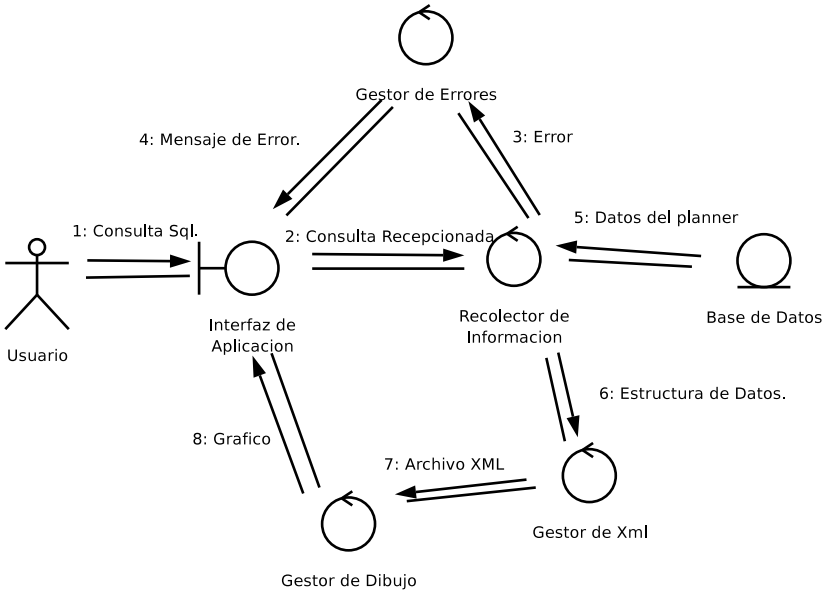

Figura 3.2: Ver explain gráfico

3.5.1.1.1. Descripción El usuario ingresa la consulta SQL a través de la Interfaz de Aplicación (1). Se recepciona la consulta (2), el módulo Recolector de Información se encargará de enviar los datos a la BD mediante funciones (5), la BD devolverá los datos correspodientes al *planner* (6) (la información necesaria para generar el explain gráfico) . El Recolector de Información recibe los datos del planner (6) y por medio de funciones llena una estructura de datos con la información requerida. El módulo Gestor de XML genera un archivo XML, (7) a partir de la estructura. El archivo XML (8) se procesa por el Gestor de Dibujo quien es el encargado de crear y desplegar el árbol gráfico de la consulta SQL (9).

En caso de que los datos estén incorrectos, se procede a mandar un código de error al (3) Gestor de Errores, este dependiendo del error muestra en la Interfaz de Recepción de Consulta, los mensajes correspondientes señalando al usuario que los datos que ingreso están incorrectos (4).

3.5.1.2. Optimizar query

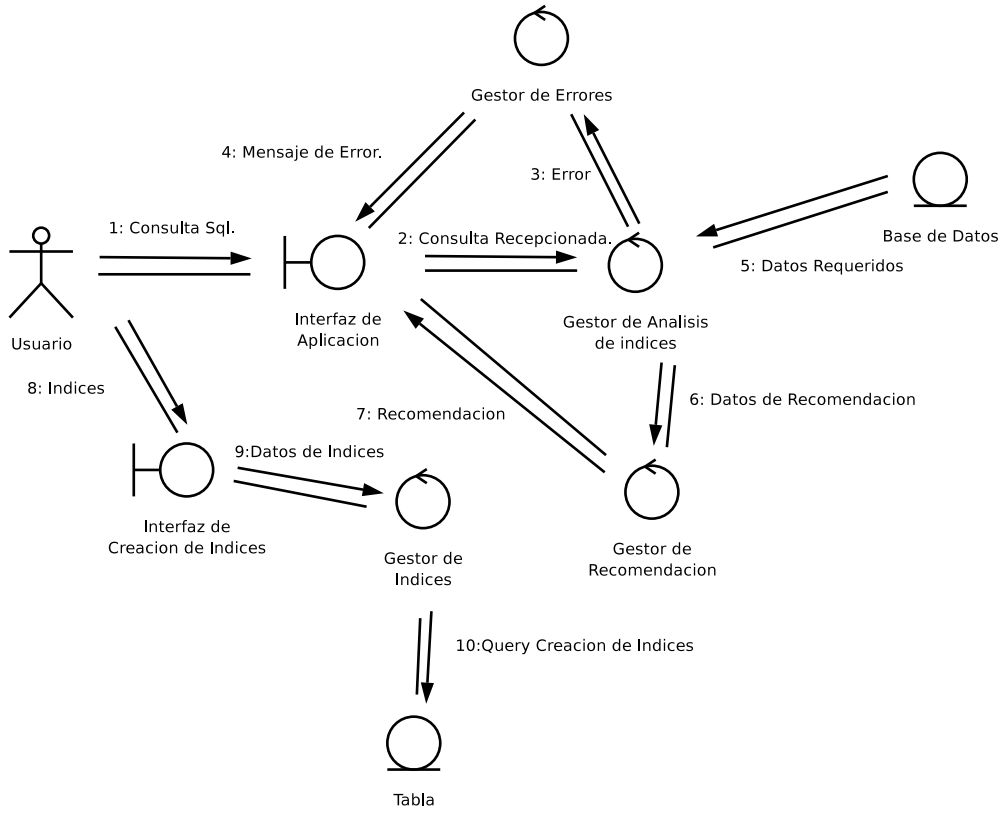

Figura 3.3: Optimizar query

3.5.1.2.1. Descripción El usuario ingresa la consulta SQL a través de la Interfaz de Aplicación (1). Se recepciona la consulta (2), el módulo Gestor de Análisis de Índices se encargará de interactuar con la BD mediante funciones (5),(6) para obtener los datos. El módulo Gestor de Recomendación se encargará de ofrecer al usuario la recomendación de creación del índice(7).

El usuario solicita crear un índice a través de la Interfaz de Creación de Índices (9), el módulo Gestor de índices será el encargado de generar el Query (11) que creará el índice, sobre la tabla seleccionada por el usuario.

En caso de que los datos estén incorrectos, se procede a mandar un código de error al (3) Gestor de Errores, este dependiendo del error muestra en la Interfaz de Recepción de Consulta, los mensajes correspondientes señalando al usuario que los datos que ingreso están incorrectos (4).

3.5.1.3. Ver atributos del query

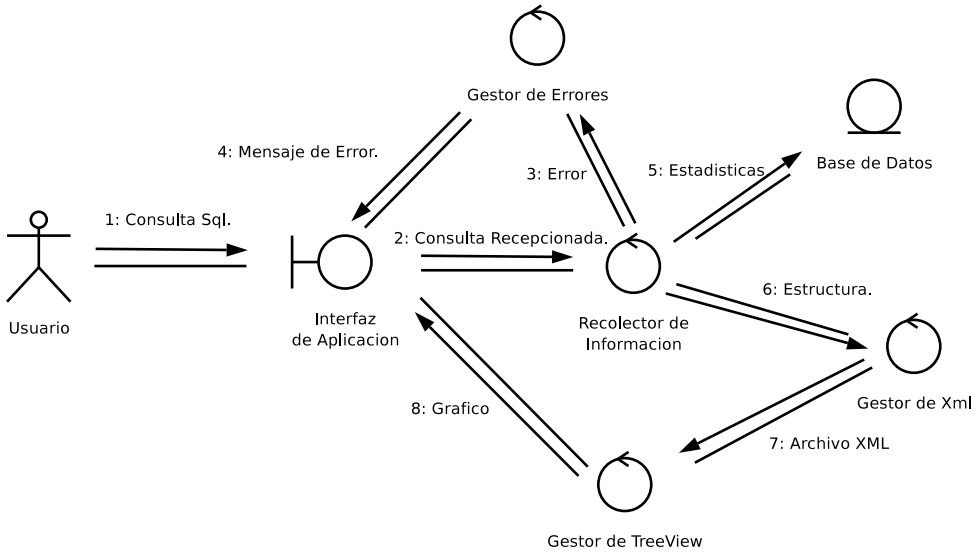

Figura 3.4: Ver atributos del query

3.5.1.3.1. Descripción El usuario ingresa la consulta SQL a través de la Recepción de Consultas (1). Se recepciona la consulta (2), la consulta recepcionada es enviada al Gestor de Recolector de Información que se encargara de (5) obtener los datos de la base de datos. Si esta es errónea se le informa del error al Gestor de Errores (3) el cual le informará de esto al usuario(4).

Al no ocurrir ningún error en los datos obtenidos por la consulta el Recolector de Información (5) se encargará de estructurar estos datos para entregárselos al Gestor de XML (6).

El Gestor de XML genera un archivo XML con los datos que le fueron entregados y se lo entrega al Gestor de Vista de Arbol[1](7) para que cree la interfaz gráfica que será mostrado al usuario(8).

En caso de que los datos estén incorrectos, se procede a mandar un código de

[1]TreeView vista gráfica de árbol en GTK+

error al (3) Gestor de Errores, este dependiendo del error muestra en la Interfaz de Recepción de Consulta, los mensajes correspondientes señalando al usuario que los datos que ingreso están incorrectos (4).

3.5.2. Análisis de casos de uso.

3.5.2.1. Clase de interfaz.

Interfaz de aplicación.

- **Descripción:**

Se utiliza para que la aplicación pueda obtener la consulta a la cual se le desea crea un índice o graficar.

- **Responsabilidad:**

Permite el ingreso de una consulta SQL al sistema.

- **Atributo:**

Consulta SQL.

Creación de índice.

- **Descripción:**

Se utiliza para que la aplicación pueda obtener los datos necesarios para la creación de un índice.

- **Responsabilidad:**

Permite la captura de los datos necesarios para la creación de un índice.

- **Atributo:**

Nombre del Índice.	Tipo de Búsqueda.
Tablas .	Columnas.

Cuadro 3.14: Atributos de la interfaz de creación de índice

3.5.3. Clases de control.

Recolector de información:

Se relaciona con la interfaz Recepción de Consulta. Responsable de reunir la información necesaria para crear la estructura de datos que necesita el Gestor de XML para realizar su labor.

Gestor de errores:

Se relaciona con la interfaz Recepción de Consulta e Interfaz de Aplicación. Responsable de informarle al usuario de los errores que se presenten durante la ejecución del sistema.

Gestor de XML:

Se relaciona con el Recolector de Información. Responsable de la creación del archivo XML de los datos obtenidos, archivo necesario para la realización de la labor de los gestores de Dibujo y TreeView.

Gestor de dibujo:

Se relaciona con el Gestor de XML. Responsable de dibujar los gráficos obtenidos de los datos entregados por el Recolector de Información.

Gestor de análisis de índices:

Se relaciona con la Interfaz de Aplicación. Responsable de recolectar todas las información necesaria para recomendar un índice.

3.6. Diseño y especificación de la interfaz

Como uno de los objetivos de la aplicación es que se integre plenamente con el escritorio GNOME[2], debe considerar los siguientes aspectos:

- Las pantallas deben ser agradables a la vista.
- Deben cumplir con los estándares de HIG[3].

A continuación se describirá y mostrará el diseño de algunas pantallas:

3.6.1. Pantalla de conexión a base de datos

Descripción Es la instancia inicial para acceder a la aplicación además de ser la encargada de conectar a un equipo cliente con un servidor de base de datos remoto. Esta consta de los siguientes atributos:

- Nickname: Nombre de la Conexión o de la sesión que ha guardado el usuario.
- Base de Datos : En esta entrada de texto se especifica el nombre de la base de datos a la cual se desea conectar.
- Host: Entrada de texto que refiere a la dirección IP o DNS del servidor donde se encuentra hospedada la base de datos a la que se desea acceder.
- Port: Entra de texto que se refiere al puerto que se empleará para realizar la conexión con el servidor de base de datos.
- UserName: Entrada de texto que hace referencia al nombre o login del usuario registrado en el motor de base de datos para acceder a la base de datos.

[2]GNU Network Object Model Enviroment.

[3]Guía de Interfaz Humana, estándar que indica como deben estar organizadas las pantallas en GNOME.

- Password: Hace referencia a la clave de acceso del usuario que desea acceder a la base de datos.

Figura 3.5: Pantalla de conexión a base de datos

3.6.2. Pantalla principal

Descripción Esta pantalla es la principal, donde el usuario podrá ingresar una consulta SQL (la entrada de texto tiene la particularidad de tener coloreado de sintaxis), el gráfico de la consulta, atributos, entre otros. Se solicita datos como :

- Consulta : Consulta SQL a la cual se le aplicarán todas la acciones.

Descripción de la barra de herramientas.

- Nuevo: Botón encargado de iniciar un nuevo plan de ejecución gráfico.
- Abrir: Botón encargado de abrir una ventana de búsqueda para seleccionar un archivo que contenga instrucciones SQL.
- Guardar: Botón encargado de almacenar las consultas analizadas en un archivo.
- Ejecutar: Botón encargado de la ejecución gráfica de una consulta SQL seleccionada por el usuario.
- Parar: Botón encargado de detener la ejecución gráfica de una consulta SQL seleccionada por el usuario.
- Acercar: Botón encargado de aumentar la imagen de la ejecución gráfica de la consulta SQL seleccionada por el usuario.
- Alejar: Botón encargado de disminuir la imagen de la ejecución gráfica de la consulta SQL seleccionada por el usuario.
- Ajustar: Botón encargado de ajustar la imagen de la ejecución gráfica de la consulta SQL seleccionada por el usuario a la pantalla.
- Imprimir: Botón encargado de imprimir los resultados obtenidos de la ejecución gráfica de la consulta SQL seleccionada por el usuario a la pantalla.

Descripción Notebook.

- Graphic: Área encargada de mostrar la imagen de la ejecución gráfica de la consulta SQL seleccionada por el usuario a la pantalla.
- Cost Query: Encargada de desplegar la información de la consulta SQL seleccionada por el usuario.

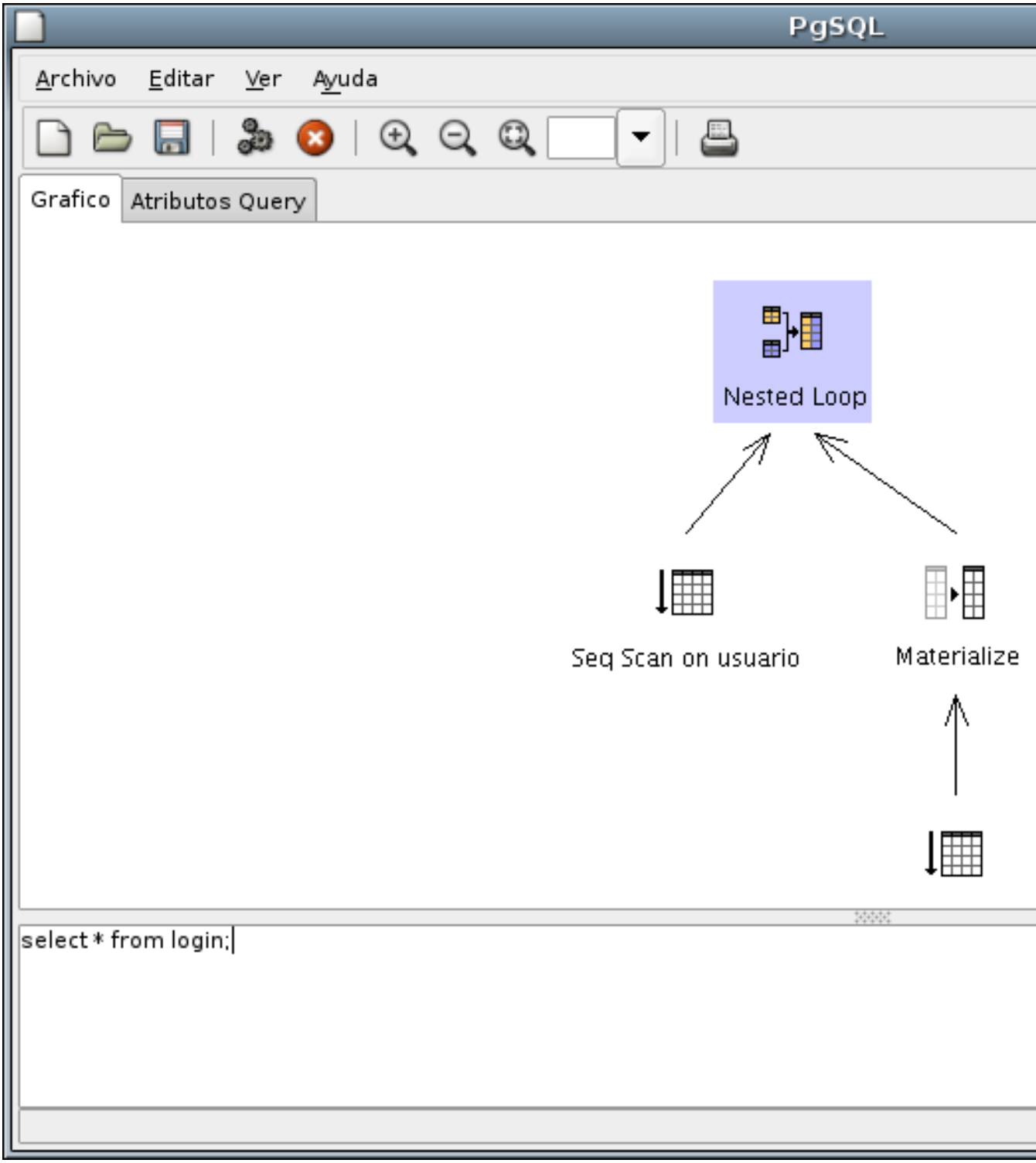

Figura 3.6: Pantalla principal

3.6.3. Pantalla creación de índices

Notebook: Propiedades

Descripción Pantalla donde el usuario tiene la posibilidad de crear un índice a un determinada tabla :

- Nombre: Encargado de recibir el nombre del nuevo índice.
- Tabla: Encargado de recibir el nombre de la tabla donde sera creado el índice.
- Columna: Encargado de recibir el nombre de la columna sobre la que trabajara el índice.
- Método de Acceso: Encargado de recibir la selección del método de búsqueda con que trabajara el índice. Los cuales pueden ser:
  - B-Tree
  - R-Tree
  - Hash
- Único: Encargado de especificar que este índice sera el único que trabajara sobre la columna de la tabla antes mencionada. (Habilitar la opción UNIQUE[4] del índice).
- Comentarios: Encargado de recibir comentarios, descripciones u observaciones sobre el índice creado.

Notebook: SQL.

Descripción: Genera el código SQL según las especificaciones hechas en Propiedades.

[4] Verificar por valores duplicados en la tabla al momento de crear el índice.

Figura 3.7: Pantalla creación de índices

3.7. Pruebas

Son los mecanismos para asegurar que el producto final funcione correctamente y que satisfaga los requerimientos propuestos, antes de ponerlo en operación

3.7.1. Datos de Prueba

Pre-Condición: Existe una base de datos creada mediante instrucciones SQL, que cumple con todas la especificaciones (tablas atributos) del Modelo Entidad Relación, y que además se encuentra poblada para la realización de las pruebas. Existe una tabla llamada login y no existe ni la tabla pepe ni tampoco la columna.

3.7.1.1. Pruebas de unidad o módulos

Módulo a probar	Datos de prueba	Resultado obtenido	Resultado esperado
Ver Explain gráfico	select * from login;	Gráfico de la consulta.	Gráfico de la consulta.
	selec * from login;	Error de sintaxis cerca de ”selec”	Gráfico de la consulta.
	select pepe from login;	Error columna no existe.	Gráfico de la consulta.
	select * from pepe;	Error la relacion ”pepe” no existe.	Gráfico de la consulta.

Módulo a probar	Datos de prueba	Resultado obtenido	Resultado esperado
Ver Atributos del query.	select * from login;	Atributos de la consulta en vista de árbol	Atributos de la consulta en vista de árbol
	selec * from login;	Error de sintaxis cerca de ”selec”	Atributos de la consulta en vista de ábol
	select pepe from login;	Error columna no existe.	Atributos de la consulta en vista de ábol
	select * from pepe;	Error la relacion ”pepe” no existe.	Atributos de la consulta en vista de ábol

Módulo a probar	Datos de prueba	Resultado obtenido	Resultado esperado
Ver Recomendaciones de índices	select * from login;	La aplicación responde bien a la prueba.	La aplicación responde con un ofreciemiento de creación de índice.
Imprimir	selec * from login;	Dialogo de impresión con la salida del explain en texto y de la consulta echa por el usuario.	Dialogo de impresión con la salida del explain en texto y de la consulta echa por el usuario.
Ver explain texto.	select pepe from login;	Ventana con la salida del explain en modo texto.	Ventana con la salida del explain en modo texto.
Ver explain analyze texto.	select * from pepe;	Ventana con la salida del explain analyze en modo texto.	Ventana con la salida del explain analyze en modo texto.
Interfaz de creacion de índices.	Datos de índices. - Nombre - Tabla - Columna - Unico - Comentarios	Indice creado.	Indice creado.

3.7.1.2. Pruebas de integración

A medida que se avanzó en la etapa de programación de los distintos módulos que integran la aplicación, se probaron y se fueron integrando a medida que los resultados de las pruebas fueron las esperadas. Algunas de las pruebas más significativas y representativas:

Módulos integrados	Resultado esperado	Resultado Obtenido
Ver explain gráfico. Ver atributos del query.	Representacion grafica de la consulta y los atributos de la consulta en forma de árbol.	La aplicación entrega los datos de forma satisfactoria
Ver Recomendaciones de índices. Interfaz de creacion de índices.	La aplicación le ofrecerá al usuario la creación de un índice, estos datos serán de utilidada para la interfaz de creación de índices.	La aplicación entrega los datos de forma satisfactoria.
Ver explain texto Ver explain analyze texto Imprimir	Las salidas de explain, explain analyze y la consulta serán imprimidas.	La aplicación entrega los datos de forma satisfactoria.

3.7.1.3. Plan de pruebas

Como es una aplicación open-source las pruebas reales y a fondo serán echas por la comunidad que decida descargar la aplicación.

Prueba de especificación Duración Encargado	1 día (2 horas) Desarrollador.
Prueba de unidad Duración Encargado	3 días (2 horas por día) Desarrollador
Prueba de Integración Duración Encargado	3 días (2 horas por día) Desarrollador
Prueba de Validación Duración Encargado	2 días (2 horas por día) Desarrollador
Prueba de Sistema Duración Encargado	2 días (2 horas por día) Desarrollador

Conclusiones

Cada día, aumenta el ínteres por el uso y desarrollo de software libre, su masificación a nivel mundial, robutez y confiabilidad son razones de sobra para emprender un proyecto de este índole.

El objetivo de este proyecto de titulo, ha sido por lo tanto crear una aplicación gráfica, que permita ver de forma gráfica el árbol de una consulta SQL y sus atributos además de ofrecer recomendaciones de optimizaciones al usuario.

El desarrollo de PGSql Analyze, se enfocó hacia el uso de nuevas tecnologías open-source, el motivo de la elección es que para ellas no existe restricción para ver el código fuente, existe gran cantidad de foros de ayuda e información disponible en la red, ademas de una gran comunidad de desarrolladores, esto fue de gran ayuda para la investigación y desarrollo de la aplicación.

En lo personal, fue un gran desafío utilizar, herramientas, técnicas y tecnologías desconocidas. Por otra parte valorar el tiempo que cada proyecto requiere, pues no se trata de programar sin una planificación previa; se deben analizar riesgos, costos y beneficios, especificar un diseño para agilizar la programación.

Como desarrollador de este proyecto, las conclusiones finales sobre el desarrollo de la aplicación apuntan principalmente a que siempre uno tiene que tratar de tomar el riesgo y no quedarse con los conocimientos previos, ya que según mi parecer es una buena alternativa de conocer y aprender mucho más, es por esto que existe una gran satisfacción y orgullo por lo logrado durante estos meses.

Índice alfabético

Bibliografía

[The Official GNOME 2 Developer's Guide]	Matthias Warkus, The Official GNOME 2 Developer 's Guide, 2004
[El Lenguaje de Programacion C-Sharp]	José Antonio González Seco, El Lenguaje de Programación C#.
[Microsoft SQL Server 7.0 Performance Tunning]	Steve Adrien DeLuca, Marcilina Garcia, Jaime A. Reding y Edward Whalen, Microsoft SQL Server 7.0 Performance Tunning.
[Mono a Developer's Notebook]	Edd Dumbill, Niel M. Bornstein, Mono A Developer 's Notebook.
[Programmer's Manual]	Programmer's Manual, http://developers.postgresql.org/docs/postgres/index.h
[PostgreSQL Documentation]	PostgreSQL Documentation, http://www.postgresql.org/docs/index.html

[FAQ Postgresql] FAQ Postgresql, http://www.PostgreSQL.org/docs/faq-english.html

[Bruce Momjian] PostgreSQL Internals Trough Pictures, Bruce Momjian, diciembre 2001

[1] PostgreSQL Hardware Performance Tuning, Bruce Momjian, 16 enero 2003

[Eric Harlow] Desarrollo de aplicaciones Linux con GTK+ y GDK, Eric Harlow.

[Mono Website] Mono Website, http://www.mono-project.com

[Mono Hispano] Mono Hispano, http://www.mono-hispano.org

[Wikipedia] Wikipedia (Enciclopedia Libre), http://es.wikipedia.org

[Miguel Icaza's Weblog] Miguel Icaza's WebLog, http://primates.ximian.com/~miguel/archive/2004/Ma 30.html, Introducing Mono for Developers.

[2] B-Tree definición, www.inf.udec.cl/~andrea/cursos/retrieval/indexes.pdf