Universidad de las Ciencias Informáticas

Facultad 6

Trabajo de Diploma para optar por el Título de

Ingeniero en Ciencias Informáticas.

Título: Herramienta para la recomendación de índices en PostgreSQL.

Autores: Jorge Deinier Sosa Lastres.

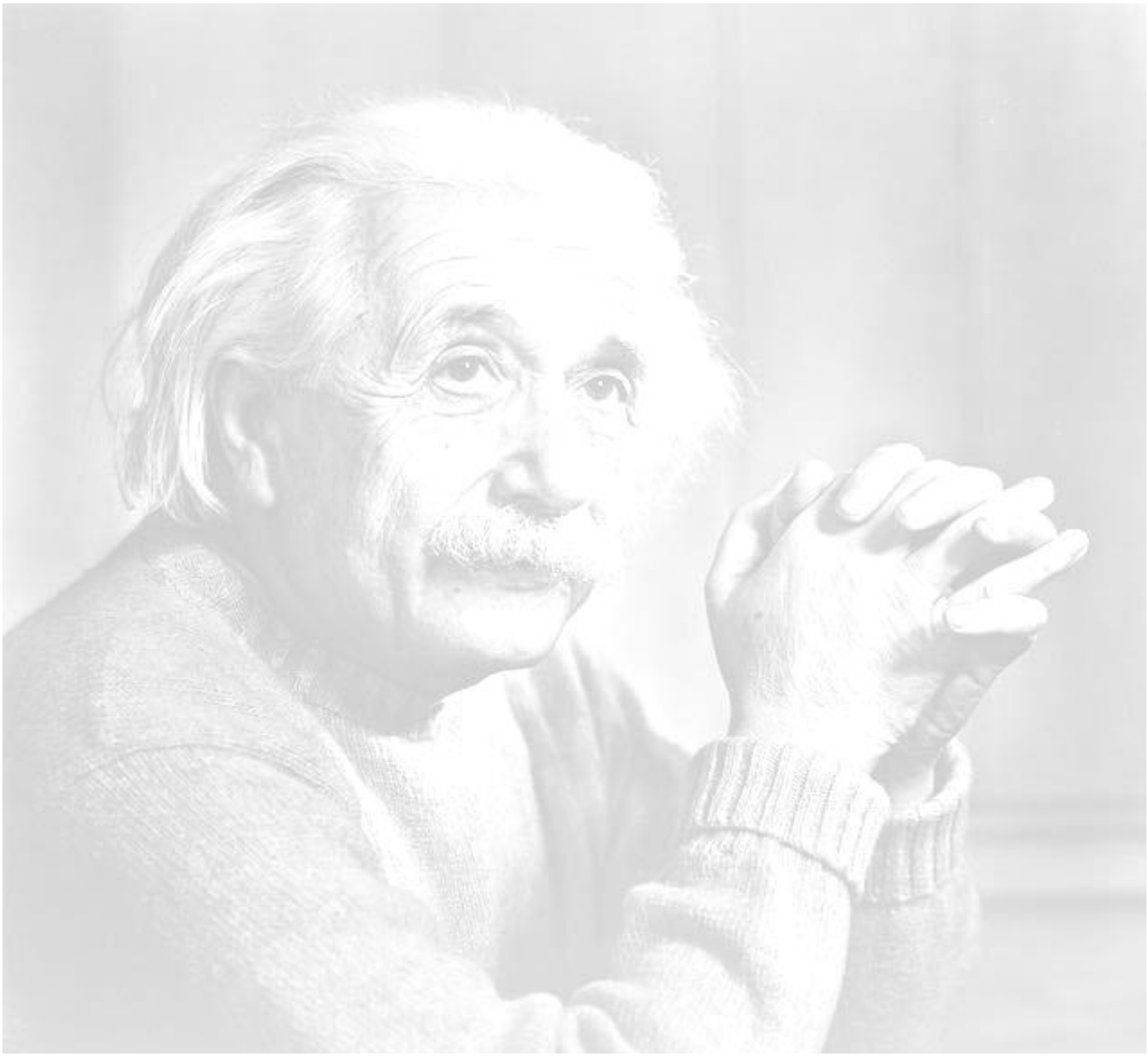
José Manuel Santana Hechavarría.

Tutores: MsC. Anthony R. Sotolongo León.

Ing. Daymel Bonne Solís.

La Habana, Junio 2013.

"Año 55 de la Revolución"



"Los intelectuales resuelven los problemas, los genios los evitan"
Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis, reconociendo a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los _____ días del mes de _____ del año _____.

_____

Jorge Deinier Sosa Lastres

Firma del Autor

_____

José Manuel Santana Hechavarría

Firma del Autor

_____

MsC. Anthony R. Sotolongo León

Firma del Tutor

_____

Ing. Daymel Bonne Solís

Firma del Tutor

DATOS DE CONTACTO

Síntesis del Tutor: MsC. Anthony R. Sotolongo León.

- **Centro:** Universidad de las Ciencias Informáticas. Habana, Cuba.
- **Especialidad de graduación:** Ingeniero Informático
- **Categoría docente:** Asistente
- **Años de experiencia en el tema:** 3
- **Años de graduado:** 6
- **E-mail:** asotolongo@uci.cu

Síntesis del Tutor: Ing. Daymel Bonne Solís.

- **Centro:** Universidad de las Ciencias Informáticas. Habana, Cuba.
- **Especialidad de graduación:** Ingeniero en Ciencias Informáticas
- **Categoría docente:** -
- **Años de experiencia en el tema:** 3
- **Años de graduado:** 3
- **E-mail:** dbonne@uci.cu

AGRADECIMIENTOS

Jorge Deinier:
Agradezco: A mis tutores Anthony y Bonne por su guía y sus consejos durante el desarrollo de este trabajo, y por la paciencia y el tiempo que nos dedicaron.
A los amigos y compañeros que de una u otra forma colaboraron conmigo: A Pablo por los iconos y a Osman por sus consejos, y por prestarme ambos su ayuda siempre que la necesite en estos 5 años, a Eudel y Ernesto Rodríguez por las librerías para la aplicación, y a Marcos Michel por el asesoramiento técnico con Ubuntu.
A los compañeros con los que compartí en estos 5 años de carrera: A Darlon, Chávez, Julio, Osvaldo, Luis Ángel, René, Yuliet, Félix, Tomás, Yosbel. A Dago y Ruberlando aunque ya no estén aquí, y a todos los demás que no he mencionado.
A mi compañero de tesis José Manuel por su colaboración y esfuerzo en todos estos meses.
A toda mi familia por su apoyo, a mi hermano Daniel, a mi sobrina Danisleydis y a mi cuñanda Ayetza, y en especial a mis padres Orlando y Graciela por su apoyo y su dedicación no solo en estos 5 años, sino durante toda mi vida.
Y otro agradecimiento especial a la memoria de mi abuelo Erasmo, por todo el cariño que me dió en vida.

José Manuel:
Quiero agradecer a todas aquellas personas que de alguna forma u otra han hecho posible mi lugar hoy aquí.
A mis amigos de siempre, Alexey y Alejandro Milanés así como a mi hermano Marcos por su amistad incondicional y El Yuni por ser siempre un gran amigo.
También agradezco sinceramente por brindarme su amistad, a Manuel Alejandro, Yosmany, Victor, Ramiro, Ortiz, Danae, El Ruso, Ariel, Ivo Poveda, Osmar, Ale, Dayatnis, Yan, Yariel, Janier, Cesar, Diosbel, Osmel, El Chuchi, El Merco, Daylig, Tahimy, Rebeca, Fernand, Pancho, Sergio, Julio y Habana.
A mis tutores por el apoyo incondicional en cada momento, especialmente al profe Anthony que siempre estuvo disponible en todo momento que fuimos a molestarlo. **A mi compañero de tesis** Jorge por compartir conmigo cada momento agotador durante el desarrollo de la tesis.
A mis abuelos Marlenes y Manolo que desde mi niñez han estado conmigo en todos mis momentos y que con su amor y dedicación han sabido guiarme por el camino correcto, los amo mucho.
A mi abuela Mami por ser la mujer, madre y abuela más dedicada de este mundo, gracias por tu ejemplo de persona y por tu apoyo siempre, un beso.
A mi mamá por ser la mujer más luchadora y emprendedora que conozco, y por defenderme y darlo todo cuando realmente lo he necesitado, por soportar mis malacrianzas, sabes que te quiero mucho, para ti e Isa un beso bien grande.
A mi papá y a mi tío Manuel Ángel por ser ejemplo de sencillez, bondad y respeto y por hacer suyos mis problemas, los amo mucho, gracias.
A mis tíos Norita y Andrés, por apoyarme y quererme sin fin toda mi vida. Gracias por su dedicación y amor.
A mi abuelo Papi y mi tío Niño, a mi flaca que adoro, a mis hermanitos Karen y Manolito, Hiromis, Diana, a mis primitas Magela y María de los Ángeles, un beso grande a todos.
A mi tía Loyman, mi tío Enrique y a la Cosi que aun estando lejos, los tengo presentes siempre.
A dos personas que hicieron en estos 5 años mis días más felices, Yacdelis y Sabrina, las amo.
En fin, a toda esta linda y unida familia que hoy disfruto, mi eterno agradecimiento por haber hecho de mí lo que hoy soy. Los amo mucho y Gracias por EXISTIR.

DEDICATORIA

Jorge Deinier:

"A mi madre, por todo."

José Manuel:

"A toda mi familia, en especial a mi abuela Marlenes."

RESUMEN

El departamento de PostgreSQL, perteneciente al centro DATEC de la Universidad de las Ciencias Informáticas, brinda varios servicios a clientes de toda la universidad, entre los cuales se encuentran la administración de bases de datos y la consultoría. Ante la necesidad de optimizar el tiempo de ejecución de las consultas en el gestor de bases de datos PostgreSQL, se determinó que una de las soluciones factibles consiste en la utilización adecuada de *índices* parciales. En el presente trabajo se desarrolló la herramienta para la recomendación de índices en PostgreSQL. Se analizaron diferentes técnicas de la inteligencia artificial para generar un resultado con mayor exactitud en cada respuesta del sistema. Se identificaron las funcionalidades y se realizó el diseño e implementación de la aplicación. La herramienta desarrollada posibilita obtener los índices parciales que dada una consulta, ayudan a reducir el tiempo de ejecución de la misma.

Palabras Claves: PostgreSQL, índices, consultas, inteligencia artificial.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

A medida que las Tecnologías de la Información y las Comunicaciones (TIC) se han ido desarrollando, en Cuba se han realizado esfuerzos para promover el desarrollo de la informática, de forma tal que permita satisfacer las necesidades de información y conocimiento de todas las personas. Como parte de este proceso surge 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 cuatro departamentos entre los que se encuentra PostgreSQL, el cual tiene como objetivo fundamental contribuir al desarrollo y uso de las tecnologías de bases de datos libres.

A raíz de la creación de los sistemas informáticos surgió la necesidad de almacenar grandes cantidades de información, desde entonces las bases de datos se han convertido en una herramienta necesaria dentro del mundo de la informática, siendo de vital importancia en la recuperación y el almacenamiento de la información.

Para la gestión de los grandes volúmenes de datos con los que trabajan las bases de datos actuales se hizo necesario crear un sistema para administrar los mismos, dando paso a la creación de los SGBD (Sistemas Gestores de Bases de Datos). Los SGBD son aplicaciones de software muy específicos, dedicados a servir de interfaz entre la base de datos, el usuario y las aplicaciones que utilizan, brindando una gran organización, seguridad e integridad de los datos [1].

Existen SGBD de código abierto y de software privativo, dentro de los de código abierto se encuentra PostgreSQL. Dicho gestor está considerado como el gestor de bases de datos de código abierto más avanzado del mundo, proporcionando un gran número de características que normalmente sólo se encontraban en bases de datos comerciales tales como DB2 u Oracle [2].

Las fortalezas de PostgreSQL están basadas en las características que posee, algunas de estas son que incorpora la herencia, tipos de datos y funciones. Cuenta además con características que aportan potencia y flexibilidad adicional: Restricciones, Disparadores, Reglas e Integridad Transaccional. Además aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas. También hace uso del modelo cliente/servidor [3].

Todas éstas funcionalidades pueden resultar costosas desde el punto de vista computacional (dígase uso de memoria, así como de procesadores), lo cual repercute en el tiempo de ejecución de las mismas, por lo que se hace necesario el uso de estrategias que conlleven a un manejo óptimo de los recursos de software y hardware disponibles.

En el proceso de optimización de consultas en PostgreSQL existen varios tipos de configuraciones, tales como: Configuración de Software, de Hardware y SQL; dentro de esta última se encuentran el Análisis Algebraico y los *Índices*.

Una característica muy útil y conocida de PostgreSQL es que brinda la posibilidad de crear índices. Un índice es esencialmente un subconjunto ordenado de las columnas de una tabla, con una entrada que apunta a la tupla correspondiente [3].

Trabajando con un conjunto indexado se puede realizar un procesamiento más rápido para cada solicitud de consulta, ya que se elimina la necesidad de búsqueda en toda la tabla, optando en su lugar por concentrarse en sólo una porción relativamente pequeña de ésta.

La indexación, técnica que optimiza el acceso a los datos, también es caracterizada por acelerar las operaciones de selección que se estén realizando sobre las tablas. Existen varios tipos de índices que se ajustan a diferentes necesidades de las consultas que se realizan en las bases de datos, cada uno utiliza un algoritmo de selección diferente. La selección de estos depende de características específicas de los campos de las tablas con las que se estén trabajando.

El departamento de PostgreSQL dentro de los servicios que brinda a sus clientes, se encuentran la administración de bases de datos y la consultoría. Los problemas más frecuentes que poseen los clientes que hacen uso de los servicios de consultoría radican en la correcta selección de los índices en las tablas de sus bases de datos. Por tal motivo resulta un proceso engorroso realizar la selección correcta de los mismos, ya que puede variar el tiempo de ejecución de una consulta en dependencia de la cantidad de datos que contengan las tablas que estén siendo analizadas.

Es válido acotar que el uso indiscriminado de los índices podría crear una sobrecarga innecesaria de memoria, y lejos de acelerar el proceso de optimización de consultas, estaría contribuyendo a minimizar el mismo.

Por lo que, a partir de los problemas anteriormente planteados se define como **problema científico** ¿Cómo facilitar el proceso de selección de índices para las consultas en PostgreSQL?

Siendo el **objeto de estudio** el proceso de utilización de índices en las bases de datos, enmarcado en el **campo de acción** proceso de selección de índices en PostgreSQL.

El **objetivo general** de la presente investigación es: Desarrollar la herramienta para la recomendación de índices en PostgreSQL.

Para el desarrollo del mismo se desglosan los siguientes **objetivos específicos**:

- ✓ Realizar el análisis de los mecanismos o técnicas de selección de índices en PostgreSQL.
- ✓ Diseñar la herramienta para la selección de índices en PostgreSQL.
- ✓ Implementar la herramienta para la selección de índices en PostgreSQL.
- ✓ Validar la herramienta para la selección de índices en PostgreSQL.

Para dar cumplimiento a los objetivos específicos planteados se planificaron las siguientes **tareas de la investigación**:

- ✓ Caracterización de los diferentes tipos de índices.
- ✓ Caracterización de las técnicas de optimización de la rama de la inteligencia artificial.
- ✓ Caracterización de las técnicas y metodología a utilizar en el desarrollo de la investigación.
- ✓ Selección de las tecnologías a utilizar.
- ✓ Desarrollo de los artefactos pertenecientes a la metodología utilizada.
- ✓ Implementación de las clases necesarias para realizar el parseo de código SQL.
- ✓ Implementación de los módulos para realizar los tipos de análisis que brindará la aplicación.
- ✓ Definición y diseño de casos de pruebas.
- ✓ Validación de la implementación en base a las pruebas realizadas.

El documento posee la siguiente estructura y contenido:

Capítulo 1: Fundamentación teórica.

Se analizan herramientas de administración que trabajan con PostgreSQL. Además se realiza un análisis minucioso del trabajo con índices en el mismo gestor para la optimización de las consultas. También se lleva a cabo un análisis de las diferentes técnicas de optimización que existen en la rama de la inteligencia artificial. Se definen las herramientas, tecnologías y metodología a utilizar en el desarrollo del trabajo.

Capítulo 2: Descripción de la solución.

Se muestran las diferentes características del sistema propuesto. Se define el modelo de dominio para lograr un mejor entendimiento del sistema. Se identifican las historias de usuarios. Se define la arquitectura del sistema. Son planificadas las iteraciones en las cuales serán implementadas cada historia de usuario y se estima el tiempo que durará la implementación de cada historia de usuario a través de las tareas de la ingeniería. Se implementan las funcionalidades identificadas.

Capítulo 3: Validación y prueba.

Se realizan pruebas al sistema para validar su correcto funcionamiento. Se especifican las pruebas a las que fue sometida la herramienta para la recomendación de índices en PostgreSQL en cada una de las iteraciones. Este proceso guía la identificación y corrección de fallos cometidos en las historias de usuarios así como su verificación y materialización, lo que contribuye a elevar la calidad de los productos desarrollados y la seguridad en los programadores para efectuar modificaciones. El resultado de las pruebas realizadas será mostrado mediante una tabla para un mejor entendimiento.

Capítulo 1

Fundamentación teórica

INTRODUCCIÓN

En el presente capítulo se realiza un análisis de los conceptos fundamentales relacionados con las bases de datos, se hace una investigación acerca de la optimización de las consultas SQL a través de la utilización de índices. Se analizan diferentes técnicas de optimización pertenecientes al entorno de la inteligencia artificial. Se muestran además características de las diferentes tecnologías y herramientas seleccionadas para el desarrollo de la herramienta de recomendación de índices en PostgreSQL.

1.1 Bases de datos

El término de bases de datos (BD) fue mencionado por primera vez en California-USA en un simposio en 1963, las mismas se definen como un conjunto de información relacionada y estructurada. Desde el ámbito informático se definen las BD 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 estos datos [3].

Desde el surgimiento de las BD se han elaborado distintos conceptos por especialistas en el tema, esto ha implicado la formulación de varias definiciones que abarcan la esencia de lo que constituye una BD. Uno de los conceptos de base de datos más completo es el expuesto por el colectivo de autores del libro “Database Fundamentals” el cual expresa que: “Una base de datos es un repositorio de datos, diseñado para soportar el almacenamiento, la recuperación y el mantenimiento de datos de manera eficiente [4].

Existen múltiples tipos de BD para adaptarse a diferentes requerimientos de la industria del software. Una BD puede ser especializada para almacenar archivos binarios, documentos, imágenes, videos, datos relacionales, datos multidimensionales, datos transaccionales, datos analíticos, o datos geográficos para nombrar unos pocos. Los datos pueden ser almacenados en varias formas, tales como formas tabulares y formas jerárquicas. Si los datos se almacenan en forma de tabla, entonces se llama BD relacional. Cuando los datos están organizados en una forma de estructura de árbol, se llama BD jerárquica [5].

1.2 Índices en las bases de datos

Un índice de una tabla desempeña la misma función que el índice de un libro: permite encontrar datos rápidamente; en el caso de las tablas, localiza registros. Un índice se puede decir que es un subconjunto ordenado de las columnas de una tabla, con una entrada que apunta a la tupla correspondiente. La indexación es una técnica que optimiza el acceso a los datos y mejora el rendimiento de las consultas, acelerando las funciones de lectura o de acceso a datos así como las operaciones que se realizan sobre las BD; es muy útil cuando la tabla contiene miles de registros. Un índice se define sobre las columnas de las tablas que se utilicen repetidamente en las consultas, sobre aquellos campos por los cuales se realizan operaciones de búsqueda con frecuencia. Una tabla puede ser indexada por campos de tipo *numérico* o de tipo *caracter*, también se puede indexar por un campo que contenga valores **NULL**, excepto las *llaves primarias*. Su uso mejora el rendimiento de las bases de datos, permitiendo recuperar las filas específicas mucho más rápido de lo que podía hacerse sin él. Es posible usar índices con los comandos **SELECT**, **UPDATE** y **DELETE** vinculados a las condiciones de búsqueda, además en concatenaciones de tablas. Un índice definido en una columna que forma parte de una condición de combinación también puede mejorar significativamente el rendimiento de las consultas [6].

1.3 PostgreSQL

El Sistema Gestor de Bases de Datos (SGBD) Objeto Relacional conocido como PostgreSQL está derivado del paquete Postgres escrito en Berkeley. PostgreSQL, es el SGBD de código abierto más avanzado en la actualidad, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones y funciones definidas por el usuario). Cuenta también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, Perl, Tcl y Python) [7].

PostgreSQL ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos básicos en una vía en la que los usuarios pueden extender fácilmente el sistema [7]:

- Clases.
- Herencia.
- Tipos.
- Funciones.

Otras características que aportan potencia y flexibilidad adicional son:

- Restricciones.
- Disparadores.
- Reglas.
- Integridad transaccional.

Estas características colocan a éste gestor en la categoría de las bases de datos identificadas como objeto-relacionales [7]. Por causa de estas razones el centro DATEC ha establecido como su SGBD a PostgreSQL.

PostgreSQL está distribuido bajo licencia BSD[1] con su código fuente disponible libremente. Es considerado el SGBD código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiar a otras bases de datos comerciales.

PostgreSQL utiliza un modelo cliente/servidor y usa multi-procesos en vez de multi-hilos para garantizar la estabilidad del sistema, un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando [3].

Debido a su licencia liberal, PostgreSQL se puede utilizar, modificar y distribuir, por cualquiera, de forma gratuita y, para cualquier propósito, ya sea privado, comercial, o académico [3].

A continuación algunas de las características más importantes soportadas por PostgreSQL.

Generales:

- Es una base de datos 100% ACID[2].
- Replicación asincrónica/sincrónica.
- Copias de seguridad en caliente.
- Completa documentación (en español e inglés).

[1] Berkeley Software Distribution es una licencia de software de libre permisiva, la cual no prohíbe el uso del código fuente en software no libre.

[2] Una BD es ACID cuando ésta cumple con un conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción. Si un SGBD es ACID quiere decir que el mismo cuenta con las funcionalidades necesarias para que sus transacciones tengan las características ACID. ACID es un acrónimo de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

- Licencia BSD.
- Disponible para Linux y UNIX en todas sus variantes (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) y Windows 32/64bit.

Programación / Desarrollo:

- Funciones/procedimientos almacenados en numerosos lenguajes de programación, tales como PL/pgSQL (similar al PL/SQL de Oracle), PL/Perl, PL/Python y PL/Tcl.
- Numerosos tipos de datos y posibilidad de definir nuevos tipos. Además de los tipos estándares en cualquier base de datos, se tienen disponibles, entre otros, tipos geométricos, de direcciones de red, de cadenas binarias, UUID, XML, matrices.
- Soporta el almacenamiento de objetos binarios extensos (gráficos, videos, sonido, etc.).

SQL [8]:

- Llaves primarias y foráneas.
- Columnas auto-incrementales.
- Consultas recursivas.
- Vistas.
- Disparadores comunes, por columna, condicionales.
- Reglas.
- Herencia de tablas.
- Índices compuestos, únicos, parciales y funcionales en cualquiera de los métodos de almacenamiento disponibles (B-tree, Hash, Gist o Gin) [3].

Como se ha podido apreciar una de las características de PostgreSQL es que utiliza índices con el objetivo de optimizar el tiempo de ejecución de las consultas en una BD, por lo que, en el departamento de PostgreSQL de la UCI ha surgido la necesidad de implementar la herramienta que se propone desarrollar.

1.4 Índices en PostgreSQL

PostgreSQL ofrece la posibilidad de crear índices. La creación de un índice en una tabla que contenga un gran número de datos puede tomar mucho tiempo. Por defecto, PostgreSQL permite que otra transacción externa lea (**SELECT**) de la tabla donde se está creando un índice mientras este se crea, pero las

escrituras (**INSERT**, **UPDATE** y **DELETE**) se bloquean hasta que la construcción del índice haya terminado. En un entorno donde sea necesario mantener activo el servidor de bases de datos y realizar acciones de escritura en todo momento esto sería inaceptable. Por eso existe otra variante con la que es posible permitir escrituras en paralelo mientras se crea un índice.

PostgreSQL proporciona varios tipos de índice: B-tree, Hash, GIST y GIN. Cada uno utiliza un algoritmo diferente que se adapte mejor a distintos tipos de consultas. De forma predeterminada, el comando **CREATE INDEX** crea B-tree, que se ajusta a las situaciones más comunes.

Los índices del tipo **B-tree** pueden gestionar consultas usando operadores de igualdad o de rango de datos. En particular, el planificador de consultas de PostgreSQL considera el uso de un índice **B-tree** cada vez que en una columna indexada se usa comparación con uno de estos operadores: <, <=, =>, =, >.

Un índice **Hash** sólo puede manejar simples comparaciones de igualdad. El planificador de consulta considera su uso cada vez que una columna indexada está involucrada en una comparación con el operador **=**. (No se admiten búsquedas usando operadores **IS NULL**).

Los índices **GIST** no son solo un tipo de índice, sino una infraestructura dentro de la cual muchas de las estrategias de indexación diferentes pueden ser implementadas. Los operadores en que pueden ser utilizados varían dependiendo de la estrategia de indexación (Clases de *Operador*). Soporta consultas usando: **<<**, **&<**, **&>**, **>>**, **<<|**, **&<|**, **|&>**, **|>>**, **@>**, **<@**, **~=**, **&&**.

GIN (índices invertidos) pueden manejar los valores que contienen más de una clave, por ejemplo los arreglos. Al igual que **GIST**, puede apoyar diferentes estrategias de indexación definidos por el usuario. A modo de ejemplo, la distribución estándar de PostgreSQL incluye clases de operador **GIN** para los arreglos de una dimensión, que apoyan indexados de consultas utilizando los siguientes operadores: **<@**,**@>**, **=**, **&&**.

1.4.1 Definición de un índice

Cuando se crea una tabla en PostgreSQL es habitual crear índices por los campos que se creen serán más usados para búsquedas. Normalmente, se hace con la siguiente sintaxis:

CREATE INDEX nombre_índice **ON** tabla(campo);

Bien, eso puede bastar en ciertos casos, pero si campo es de tipo texto y se quieren hacer búsquedas usando **LIKE**, se podrían encontrar sorpresas [9]:

EXPLAIN ANALYZE SELECT * **FROM** tabla **WHERE** campo **LIKE** '%valor%';

Esto dirá que se está ejecutando un **SCAN** de toda la tabla. El índice no se está usando para búsquedas tipo **LIKE**. Para que lo use, se necesita crear el índice de forma ligeramente distinta [9]:

CREATE INDEX nombre_índice **ON** tabla(campo text_pattern_ops); (si campo es de tipo **TEXT**)
CREATE INDEX nombre_índice **ON** tabla(campo varchar_pattern_ops); (si campo es de tipo **VARCHAR**)

Si se repite el *Explain Analyze*[3] se puede observar que ahora si se usa el índice. Otro caso típico que se puede encontrar es con la normalización de los datos, o sea, cómo trabajar acentos, eñes, minúsculas y mayúsculas para que el usuario encuentre lo que busca y de esta forma una búsqueda 'proyect' encuentre cosas como 'Caja PROYECTOR', 'Proyectó 1' o 'Proyecté una película'. Para ello se pueden guardar los datos tal como vienen, y crear un índice normalizado, usando una función PL/pgSQL de tipo inmutable. Por ejemplo [9]:

CREATE OR REPLACE FUNCTION str_normalize (value TEXT) **RETURNS** text **AS** $$
BEGIN
RETURN lower(translate(value, áàéèîióòúùäëïöüÁÀÉÈÎÍÓÒÚÙÄËÏÖÜñÑçÇ" ?¿¡[]`{},:;=&%$#|!\oa<>', 'aaeeiioouuaeiouAAEEIIOOUUAEIOUnNcC '));
END;
$$ **LANGUAGE** plpgsql;

CREATE INDEX nombre_índice **ON** tabla(str_normalize(campo) text_pattern_ops);

Ahora se puede comprobar que se permiten hacer búsquedas normalizadas lo más optimizado posible:
EXPLAIN ANALYZE SELECT * **FROM** tabla **WHERE** str_normalize(campo) **LIKE** str_normalize('%valor%');

[3] Explain Analyze es una funcionalidad de PostgreSQL que permite analizar las consultas para observar el tiempo de respuesta o ejecución de las mismas.

Ejemplos de creación de índices utilizando PostgreSQL:

- Para crear un índice en la columna título de la tabla filme:

CREATE INDEX titulo_idx **ON** filme (titulo);

- Para crear un índice con un orden específico, donde aparezcan primero los nulos:

CREATE INDEX titulo_idx_nulls_low **ON** films (titulo **NULLS FIRST**);

1.4.2 Ventajas y desventajas de los índices

El uso de los índices posee muchas ventajas y solo algunas desventajas. A continuación se detallan las mismas.

Al utilizar índices se evitan los siguientes problemas:

1. Sobrecarga de CPU.
2. Sobrecarga de disco y concurrencia.
3. Permiten mayor rapidez en la ejecución de las consultas tipo **SELECT** y **WHERE**.
4. Serán una ventaja para aquellos campos que no contengan datos duplicados, sin embargo si es un campo con valores que se repiten continuamente (Ej. Masculino/Femenino) no es aconsejable su uso.

También como es lógico traen consigo desventajas, tales como:

1. Son una desventaja en aquellas tablas en las que se utiliza frecuentemente operaciones de escritura (**INSERT**, **DELETE** y **UPDATE**), esto sucede porque los índices se actualizan cada vez que se modifica una columna.
2. Por último hay que tener en cuenta que ocupan espacio en la memoria del disco duro y en determinadas ocasiones incluso más espacio que los propios datos.

Es necesario recalcar que no es recomendable usar los índices sobre tablas que contengan baja cantidad de datos.

Estas ventajas y desventajas se deben tener en cuenta a la hora de trabajar con los índices en cualquier SGBD, pero a resumidas cuentas, es recomendable el uso y creación de los mismos, ya que poseen gran importancia para facilitar el trabajo en cualquier BD. En ocasiones puede resultar complicada su utilización, pues no siempre se comportan como se espera, porque los SGBD utilizan diferentes técnicas para la decisión de su utilización.

A raíz del análisis realizado respecto a los índices en PostgreSQL se determina que el proceso de selección de los mismos es muy engorroso, ya que depende tanto de características muy diversas implícitas en cada BD, como de la parte humana implicada en el proceso de selección (los usuarios autorizados). Debido a que la necesidad de la problemática del presente trabajo de diploma radica en la optimización de las consultas mediante el uso de los índices que mejor desempeño presenten en la BD, es evidente que se está en presencia de un problema de optimización. Para dar solución a éste tipo de problemas son muy utilizadas las técnicas basadas en la inteligencia artificial, las cuales plantean dos vías de solución: las que ofrecen una solución exacta basadas en heurísticas tradicionales, y las que ofrecen una solución aproximada conocidas como técnicas metaheurísticas. Como la selección de índices en PostgreSQL es un proceso muy variable, se hace muy difícil el planteamiento de una heurística que ofrezca una solución exacta del mismo, por lo que las técnicas que más se ajustan para dar solución a éste proceso son las técnicas metaheurísticas.

1.5 Técnicas metaheurísticas

En la resolución de problemas de optimización o de cálculos de alta complejidad se utilizan técnicas de optimización que pueden ser consideradas exactas o aproximadas en dependencia de la solución que propenden. Las técnicas o métodos de optimización que facilitan una solución aproximada se denominan técnicas metaheurísticas.

Las técnicas metaheurísticas son estrategias basadas en la inteligencia artificial para dar solución a problemas a los cuales no se les encuentra una solución factible mediante la aplicación de las técnicas de optimización basadas en heurísticas tradicionales. La definición de los métodos metaheurísticos plantea que: “Los metaheurísticos son métodos aproximados diseñados para resolver problemas de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Los metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos [10].”

1.5.1 Clasificación de las técnicas metaheurísticas

Existen diversas formas de clasificación para las técnicas metaheurísticas: basadas en la naturaleza (algoritmos bioinspirados) o no basadas en la naturaleza, basadas en memoria o sin memoria, con función objetivo estática o dinámica, entre otras. No obstante la clasificación más utilizada es la que divide las metaheurísticas en dos grupos: *basadas en trayectoria* y *basadas en población* [11].

1.5.2 Metaheurísticas basadas en trayectoria

Estas técnicas parten de un punto inicial y van actualizando la solución presente mediante la exploración del vecindario, formando una trayectoria. La búsqueda finaliza cuando se alcanza un número máximo de iteraciones, se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso. A continuación se describen algunas de las técnicas metaheurísticas basadas en trayectoria:

El Enfriamiento Simulado, es una de las primeras metaheurísticas creadas, simula el proceso de recocido de los metales y del cristal. En cada iteración se elige una solución S1, a partir de la solución actual S0. Si S1 es mejor que S0, S1 sustituye a S0 como solución actual. Si S1 es peor que S0, se continúa aceptando pero se le asigna una determinada probabilidad. El algoritmo permite elegir soluciones peores a la actual para evitar caer en un óptimo local [12].

La Búsqueda Tabú, es una de las metaheurísticas más utilizadas en problemas de optimización. La Búsqueda Tabú se basa fundamentalmente en la utilización de un historial de búsqueda, que permite ejecutar su estrategia de análisis y exploración de diferentes regiones del espacio de búsqueda. Este historial o memoria se implementa como una lista tabú. En cada iteración se elige la mejor solución entre las permitidas y se añade a la lista tabú, donde se mantienen las soluciones recientes que se excluyen de las siguientes iteraciones [13].

La Búsqueda Local Iterada, se basa en que en cada iteración, a la solución actual se le aplica un cambio o modificación que da lugar a una solución intermedia. A esta nueva solución se le aplica una heurística base para mejorarla que suele ser un método de búsqueda local. Este nuevo óptimo local obtenido por el método de mejora puede ser aceptado como nueva solución actual si pasa una prueba de aceptación [14].

1.5.3 Metaheurísticas basadas en población

Las técnicas metaheurísticas basadas en población trabajan con un conjunto de individuos que representan otras tantas soluciones. Su eficiencia y resultado depende fundamentalmente de la forma con la que se manipula la población en cada iteración. Seguidamente se describen algunas de las técnicas metaheurísticas basadas en población:

Los sistemas basados en Colonias de Hormigas, se inspiran en el comportamiento de las hormigas cuando buscan comida: inicialmente, las hormigas exploran el área cercana al hormiguero de forma aleatoria. Cuando una hormiga encuentra comida, la lleva al hormiguero. En el camino, la hormiga va

depositando una sustancia química denominada feromona que guía al resto de hormigas a encontrar la comida. El rastro de feromona sirve a las hormigas para encontrar el camino más corto entre el hormiguero y la comida. Este rastro es simulado mediante un modelo probabilístico [15].

Los Algoritmos Basados en Nubes de Partículas o Particle Swarm Optimization, son técnicas metaheurísticas inspiradas en el comportamiento del vuelo de las bandadas de aves o el movimiento de los bancos de peces. La toma de decisión por parte de cada individuo o partícula se realiza teniendo en cuenta una componente social y una componente individual, mediante las que se determina el movimiento de esta partícula para alcanzar una nueva posición.

Los Algoritmos Evolutivos, este grupo de técnicas se inspiran en la capacidad de la evolución de seres o individuos para adaptarlos a los cambios de su entorno. Cada individuo representa una posible solución. El funcionamiento básico de estos algoritmos es el siguiente: La población se genera de forma aleatoria. Cada individuo de la población tiene asignado un valor de bondad con respecto al problema considerado, por medio de una función de aptitud, capacidad, adaptabilidad o estado, también denominada con bastante frecuencia por la palabra inglesa "fitness". El valor de aptitud de un individuo es la información que el algoritmo utiliza para realizar la búsqueda. La modificación de la población se efectúa mediante la aplicación de tres operadores: selección, recombinación y mutación. En estos algoritmos se pueden distinguir la fase de selección (explotación de buenas soluciones) y la fase de reproducción (búsqueda de nuevas regiones). Se debe de mantener un equilibro entre estas dos fases. La política de reemplazo permite la aceptación de nuevas soluciones que no necesariamente mejoran las existentes [16].

Los algoritmos evolutivos se pueden clasificar en las siguientes categorías:

Programación Evolutiva, Estrategias Evolutivas, y los Algoritmos Genéticos (constituyen una de las técnicas más conocidas y fueron introducidos por Holland) [17].

1.6 Algoritmos genéticos

Los Algoritmos Genéticos (AG) fueron descubiertos en 1975 por John Holland, de la Universidad de Michigan. Los AG son algoritmos de optimización, es decir, tratan de encontrar la mejor solución a un problema dado entre un conjunto de soluciones posibles. Los mecanismos de los que se valen los AG para llevar a cabo esa búsqueda pueden verse como una metáfora de los procesos de evolución biológica. Los AG son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y

optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acuerdo con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin en 1859. Por imitación de este proceso, los AG son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende, en buena medida, de una adecuada codificación de las mismas [18].

1.6.1 Aplicaciones de los algoritmos genéticos

Los AG han encontrado diversas aplicaciones en las más disimiles ramas de la ciencia que no se restringen solamente al campo de la informática, a continuación se muestran algunas de las aplicaciones de los mismos.

Optimización: Se trata de un campo especialmente abonado para el uso de los AG, por las características intrínsecas de estos problemas. No fue en vano la fuente de inspiración para los creadores de estos algoritmos. Los AG se han utilizado en numerosas tareas de optimización, incluyendo la optimización numérica, y los problemas de optimización combinatoria.

Programación automática: Los AG se han empleado para desarrollar programas para tareas específicas, y para diseñar otras estructuras computacionales tales como el autómata celular, y las redes de clasificación.

Aprendizaje máquina: Los AG se han utilizado también en muchas de estas aplicaciones, tales como la predicción del tiempo o la estructura de una proteína. Han servido asimismo para desarrollar determinados aspectos de sistemas particulares de aprendizaje, como pueda ser el de los pesos en una red neuronal, las reglas para sistemas de clasificación de aprendizaje o sistemas de producción simbólica, y los sensores para robots.

Economía: En este caso, se ha hecho uso de los AG para modelar procesos de innovación, en el desarrollo de estrategias de puja[4], y en la aparición de mercados económicos.

[4] Término usado para nombrar el procedimiento de mejora de la tasa o del precio efectuado tanto para la compra como para la venta. Puede verse también como el importe mínimo en el que se aumenta o disminuye el precio de compra-venta de una acción. Se llama así a la cantidad que ofrece un licitador en una subasta.

Sistemas inmunes: Ha resultado útil el empleo de esta técnica a la hora de modelar varios aspectos de los sistemas inmunes naturales, incluyendo la mutación somática durante la vida de un individuo y el descubrimiento de familias de genes múltiples en tiempo evolutivo.

Ecología: En la modelización de fenómenos ecológicos tales como las carreras de armamento biológico, la coevolución de parásito-huésped, la simbiosis, y el flujo de recursos.

Genética de poblaciones: En el estudio de preguntas del tipo "¿Bajo qué condiciones será viable evolutivamente un gen para la recombinación?"

Evolución y aprendizaje: Los AG se han utilizado en el estudio de las relaciones entre el aprendizaje individual y la evolución de la especie.

Sistemas sociales: En el estudio de aspectos evolutivos de los sistemas sociales, tales como la evolución del comportamiento social en colonias de insectos, la evolución de la cooperación y la comunicación en sistemas multi-agentes [19].

1.7 Selección de la técnica metaheurística a utilizar

El proceso de selección de índices en PostgreSQL depende tanto de características específicas de los campos de la tabla a la que hace referencia la consulta que se esté ejecutando, como de la experiencia y el conocimiento de los usuarios (generalmente administradores de BD) para determinar los índices más adecuados y contribuir a un funcionamiento óptimo en la BD mediante la reducción del tiempo de ejecución de estas consultas.

Con el objetivo de facilitar este proceso se desea elaborar una herramienta que, utilizando técnicas de la inteligencia artificial, recree este proceso de selección de forma automática, brindando al usuario recomendaciones de los índices más óptimos a utilizar. Estas técnicas son también conocidas como técnicas metaheurísticas. Debido a las características específicas de la situación problémica: *donde se parte de un conjunto de individuos que representarían las posibles soluciones de la selección adecuada de los índices en una consulta determinada, y los mismos conformarían la población inicial*, se hace necesario utilizar técnicas metaheurísticas basadas en población. Como se analizó anteriormente existen varias técnicas metaheurísticas basadas en población para dar solución a problemas de optimización. Uno de los aspectos a tener en cuenta es el teorema de *Non Free Lunch* para algoritmos de optimización y

búsqueda, que plantea que ningún algoritmo, por sofisticado que sea, tiene un desempeño superior a ningún otro en todos los problemas posibles [20].

Otro aspecto importante es que los individuos de las poblaciones a emplear estarán conformados sobre la base de distintas combinaciones de los tipos de índices a seleccionar, y posteriormente van a estar involucrados en un proceso de recombinación para determinar la combinación de índices más óptima. Por lo que la técnica metaheurística que más se ajusta para dar solución al problema planteado es la de los *Algoritmos Genéticos*. También se reafirma ésta selección en el artículo "Introducción a los algoritmos genéticos y sus aplicaciones" de la autora Piedad Tolmos, donde expresa que "*Los algoritmos genéticos se han utilizado en numerosas tareas de optimización, incluyendo la optimización numérica, y los problemas de optimización combinatoria*".

Cabe destacar que se hace necesaria la implementación de dos módulos diferentes, pues es preciso optimizar tanto índices *tipados* como *no tipados*. Para esto han sido desarrollados muy a propósito, los AG, los cuales permiten a través del proceso de codificación, establecer diversas codificaciones que se adecúan a la solución más óptima de manera independiente. Y, a pesar de que las operaciones de cruce y mutación varían dependiendo del tipo de codificación, la línea general de resolución del problema confluiría de acuerdo a los preceptos establecidos en la idea esencial del AG, manteniendo de esta forma uniformidad en el proceso.

1.8 Características de los algoritmos genéticos

Los AG poseen una serie de características que facilitan y agilizan cualquiera de los usos que se le estén dando. Seguidamente se muestran algunas de las principales características de los AG, así como requisitos a tener en cuenta para hacer uso de los mismos.

Requisitos para utilizar algoritmos genéticos

- Población de individuos: Representan soluciones al problema a resolver.
- Operadores genéticos: Selección natural, evolución genética y cambios adaptativos.
- Selección: Simula la selección natural.
- Cruce o Recombinación: Simula la evolución genética.
- Mutación: Simula los cambios de adaptación de las especies.

- ❖ Función de bondad (fitness): Determina la adaptación del individuo (solución) al medio (problema) [21].

Codificación de los algoritmos genéticos

Durante los primeros años el tipo de representación utilizado era siempre binario, debido a que se adaptaba perfectamente al tipo de operaciones y al tipo de operadores que se utilizaban en un AG. Sin embargo, las representaciones binarias no son siempre efectivas por lo que se empezaron a utilizar otro tipo de representaciones. En general, una representación debe ser capaz de identificar las características constituyentes de un conjunto de soluciones, de forma que distintas representaciones den lugar a distintas perspectivas y por tanto distintas soluciones. Se pueden considerar tres tipos básicos de representaciones [22]. A continuación se muestran los tipos de representación anteriormente mencionados:

- Representación binaria: Cada gen es un valor 1 ó 0.

1	**0**	**1**	**1**	**0**	**1**

- Representación entera: Cada gen es un valor entero.

2	**5**	**3**	**0**	**-1**	**4**

- Representación real: Cada gen es un valor real.

2,6	**7**	**0**	**-1,2**	**6,5**	**6,9**

Tamaño de población

Debe de ser suficiente para garantizar la diversidad de las soluciones, y además tiene que crecer más o menos con el número de bits del cromosoma, aunque nadie ha aclarado cómo tiene que hacerlo. Por supuesto, depende también del ordenador en el que se esté ejecutando [23].

Población inicial

La población inicial de un AG puede ser creada de muy diversas formas, desde generar aleatoriamente el valor de cada gen para cada individuo, luego utilizar una función ávida o generar alguna parte de cada individuo, hasta aplicar una búsqueda local [22].

Función de evaluación

La función de evaluación mide la adaptación de cada uno de los individuos, es decir, la calidad de cada solución del problema, y depende fundamentalmente de la representación elegida, puesto que la única información que puede evaluar esta función va a ser la contenida en los genes según la representación utilizada. Cuando se utilice un algoritmo genético para resolver un problema de optimización, la función de evaluación será la función a optimizar, **f(x)**, aunque dicha función puede ser modificada para penalizar de cierta forma la violación de restricciones (por ejemplo para problemas difusos) o para fomentar la diversidad [24].

1.8.1 Operador de selección

El operador de selección es el encargado de transmitir y conservar aquellas características de las soluciones que se consideran valiosas a lo largo de las generaciones. El principal medio para que la información útil se transmita es que aquellos individuos mejor adaptados (con mejor valor de función de evaluación) tengan más probabilidades de reproducirse. Sin embargo, es necesario también incluir un factor aleatorio que permita la reproducción a individuos que aunque no estén muy bien adaptados, puedan contener alguna información útil para posteriores generaciones con el objetivo de mantener así cierta diversidad en cada población. Pueden considerarse tres tipos de operadores de selección, aunque existen múltiples operadores distintos a estos. Existen además innumerables combinaciones de aspectos de estos tres mecanismos fundamentales.

- ***Ruleta o Selección Proporcional:*** Con este método la probabilidad que tiene un individuo de reproducirse es proporcional a su valor de función de evaluación, es decir, a su adaptación. Una vez calculadas estas probabilidades, la selección de los individuos para reproducirse es aleatoria según estos valores.

- ***Selección por Ranking:*** consiste en calcular las probabilidades de reproducción atendiendo a la ordenación de la población por el valor de adaptación en vez de atendiendo simplemente a su valor de

adecuación. Estas probabilidades se pueden calcular de diversas formas, aunque el método habitual es el ranking lineal.

- ***Selección por Torneo:*** Reporta un coste computacional muy bajo debido a su sencillez. Se selecciona un grupo de t individuos (normalmente t = 2, torneo binario) y se genera un número aleatorio entre 0 y 1. Si este número es menor que un cierto umbral K (usualmente 0,75), se selecciona para reproducirse al individuo con mejor adaptación, y si este número es menor que K, se selecciona, por el contrario, al individuo con peor adaptación [24].

Al analizar las distintas técnicas de selección antes expuestas, se determina la selección por ranking como la más apropiada a utilizar en la implementación del algoritmo genético para la herramienta que se desea elaborar. Para llegar a esta conclusión es necesario tener en cuenta que al utilizar este proceso aumentan las posibilidades de evitar la convergencia prematura (la posibilidad de que las mejores soluciones copen la población en las primeras iteraciones del algoritmo), manteniendo la diversidad de individuos en la población seleccionada para el cruce. Esto contribuye a recorrer todo el espacio de búsqueda con el algoritmo para encontrar nuevas soluciones en etapas tempranas del mismo. De igual forma se evita incurrir en el caso opuesto, o sea, que se seleccionen pocas copias de los mejores individuos de la población. Este proceso es controlado mediante la utilización de una variable denominada “presión de selección” (un valor real entre 1 y 2), para evitar que la diversidad en la población degenere la búsqueda disminuyendo las probabilidades de cruce entre las mejores soluciones.

1.8.2 Operador de cruce

Después de terminado el proceso de selección de los individuos de la generación actual, estos se someten a un proceso de recombinación para producir los nuevos individuos que se incluirán en la próxima generación. Al estar basado en un algoritmo evolutivo, es posible indicar que el cruce constituye una estrategia de reproducción sexual. Su importancia para la transición entre generaciones es elevada puesto que las tasas de cruce con las que se suele trabajar rondan el 90 %. Los diferentes métodos de cruce podrían operar de dos formas diferentes. Si se opta por una estrategia destructiva los descendientes se insertarán en la población temporal aunque sus padres tengan mejor ajuste (trabajando con una única población esta comparación se realizará con los individuos a reemplazar). Por el contrario utilizando una estrategia no destructiva la descendencia pasará a la siguiente generación únicamente si supera la bondad del ajuste de los padres (o de los individuos a reemplazar).

La idea central del cruce se basa en que, si se toman dos individuos correctamente adaptados al medio y se obtiene una descendencia que comparta genes (valores que conforman el cromosoma del individuo) de ambos, exista la posibilidad de que los genes heredados sean precisamente los causantes de la bondad (valor de aptitud) de los padres. Al compartir las características buenas de dos individuos, la descendencia, o al menos parte de ella, deberá tener una bondad mayor que cada uno de los padres por separado, o sea, se basa en el principio de que si algunos de los genes de los padres son buenos, y el valor de aptitud de los mismos es elevado, los hijos que deben salir de un cruce entre ellos deben ser mejores que los propios padres. Se debe tener en cuenta que este planteamiento no se cumple para todos los cruces entre dos individuos, y si el cruce no agrupa las mejores características en uno de los hijos y la descendencia tiene un peor ajuste que los padres, no significa que se está dando un paso atrás.

Optando por una estrategia de cruce no destructiva se garantiza que pasen a la siguiente generación los mejores individuos. Si aún con un ajuste peor, se opta por insertar a la descendencia, y puesto que los genes de los padres continuarán en la población (aunque dispersos y posiblemente levemente modificados por la mutación) en posteriores cruces se podrán volver a obtener estos padres, recuperando así la bondad previamente perdida [23]. Existen muchos algoritmos que utilizan operadores de cruce. Algunos de los más importantes son:

Cruce de 1 punto

Es la más sencilla de las técnicas de cruce. Una vez seleccionados dos individuos se cortan sus cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola. Se intercambian las colas entre los dos individuos para generar los nuevos descendientes. De esta manera ambos descendientes heredan información genética de los padres, tal y como puede verse en la figura 1 [23].

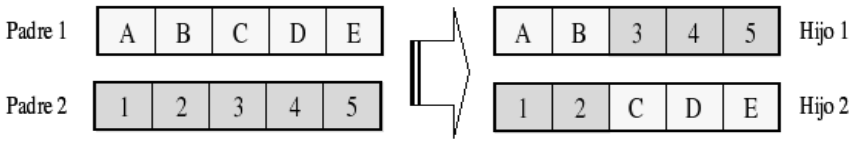


Figura 1: Cruce de 1 punto

Cruce de 2 puntos

Se trata de una generalización del cruce de 1 punto. En vez de cortar por un único punto los cromosomas de los padres como en el caso anterior se realizan dos cortes. Deberá tenerse en cuenta que ninguno de estos puntos de corte coincida con el extremo de los cromosomas para garantizar que se originen tres segmentos. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre (ver figura 2) [23].

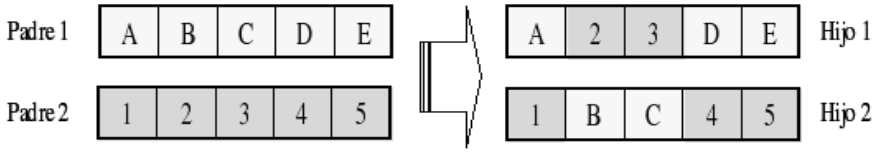


Figura 2: Cruce de 2 puntos

Generalizando se pueden añadir más puntos de cruce dando lugar a algoritmos de cruce multipunto. Aunque se admite que el cruce de 2 puntos aporta una sustancial mejora con respecto al cruce de un solo punto, el hecho de añadir un mayor número de puntos de cruce reduce el rendimiento del AG. El problema principal de añadir nuevos puntos de cruce radica en que se hace más fácil que los segmentos originados sean corruptibles, es decir, que por separado quizás pierdan las características de bondad que poseían conjuntamente. Sin embargo no todo es desventaja y añadiendo más puntos de cruce se consigue que el espacio de búsqueda del problema sea explorado más a fondo [23].

Cruce uniforme

El cruce uniforme es una técnica completamente diferente de las vistas hasta el momento. Cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno u otro padre. Aunque se puede implementar de muy diversas formas, la técnica implica la generación de una máscara de cruce con valores binarios. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre. Si por el contrario hay un 0 el gen se copia del segundo padre. Para producir el segundo descendiente se intercambian los papeles de los padres, o bien se intercambia la interpretación de los unos y los ceros de la máscara de cruce.

Tal y como se puede apreciar en la figura 3, la descendencia contiene una mezcla de genes de cada uno de los padres. El número efectivo de puntos de cruce es fijo pero será por término medio L/2, siendo L la

longitud del cromosoma (número de alelos en representaciones binarias o de genes en otro tipo de representaciones) [24].

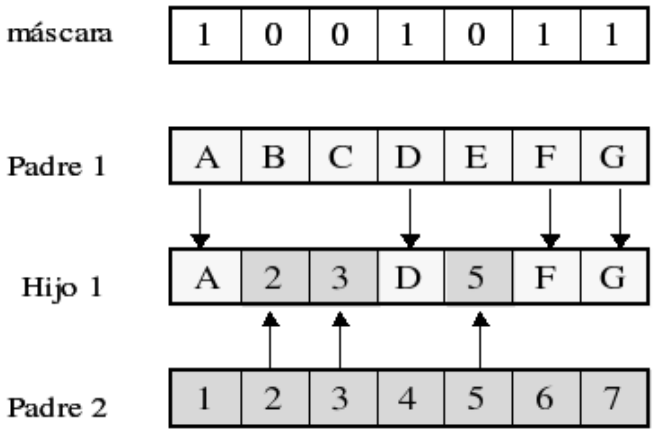


Figura 3: Cruce uniforme

1.8.3 Operador de mutación

La idea detrás de los operadores de mutación es reproducir las mutaciones genéticas producidas en cada generación (provenientes de errores de copia o transferencia en el ADN, etc.) como una de las principales herramientas de la evolución natural. En los AG el operador de mutación tiene varios objetivos, algunos de los mismos son:

- ✓ Preservar la diversidad genética de la población evitando la convergencia prematura.
- ✓ Explorar áreas posiblemente no abordadas del espacio de búsqueda (y cercanas a una buena solución).
- ✓ Quitar al AG de un máximo local si se produjo convergencia prematura.

La mutación se utiliza en bajo porcentaje (entre el 1 y el 5 % en codificación binaria o finita, hasta 10 o 15 % en codificación real) debido al peligro de que opere sobre la única copia disponible de una buena solución y la arruine. En general esto no sucede, ya que las buenas soluciones reciben varias copias y es poco probable que se muten todas. Sin embargo, existen casos especiales como los *Niching Genetic Algorithms* donde existen muchas posibilidades que esto ocurra, por lo que en general estos no utilizan mutación. Un porcentaje excesivo de mutación provoca que la búsqueda se convierta en aleatoria (debido a que gran cantidad de soluciones son mutadas al azar en cada generación). Un porcentaje demasiado

bajo puede provocar convergencia prematura, o que ciertas zonas del espacio de búsqueda no sean exploradas. Sin embargo, este efecto es en general menos notable que el de la *Selección* o *Crossover* [25].

1.8.4 Algoritmo genético canónico

Los AG son empleados para resolver una amplia gama de problemas, por lo cual existe una gran variedad de los mismos cada cual ajustado a las necesidades particulares del problema a resolver. Sin embargo todos tienen su origen en el algoritmo genético canónico (o simple) del cual asimilan como base su funcionamiento para readaptarlo a necesidades específicas. A continuación se explica el funcionamiento del algoritmo genético canónico.

1. Se parte de una población de N individuos generada aleatoriamente.
2. Se escogen N individuos de la generación actual para la generación intermedia (según el criterio escogido).
3. Se emparejan los individuos y para cada pareja:
   a) Con una probabilidad *Pcruce* se aplica el operador de cruce a cada pareja de individuos y se obtienen dos nuevos individuos que pasarán a la nueva generación, con una probabilidad (1 – *Pcruce*). Se mantiene la pareja original en la siguiente generación.
   b) Con una probabilidad *Pmutacion* se mutan los individuos cruzados.
4. Se substituye la población actual con los nuevos individuos, que forman la nueva Generación.
5. Se itera desde el segundo paso hasta que la población converge (la función de *fitness* de la población no mejora) o pasa un número específico de generaciones.

La probabilidad de cruce influirá en la variedad de la nueva generación, cuanto más baja sea, más parecida será a la generación anterior y harán falta más iteraciones para converger. Si ésta es muy alta cada generación será muy diferente, por lo que puede degenerar en una búsqueda aleatoria.
La mutación es un mecanismo para forzar la variedad en la población, pero una probabilidad de mutación demasiado alta puede dificultar la convergencia [26].

1.9 Metodología de desarrollo de software

Las metodologías de desarrollo durante el crecimiento de creación de software a nivel mundial, han contribuido a documentar una inmensa cantidad de información sobre las políticas, procesos y

procedimientos para lograr una mejor calidad del producto final. Una metodología de desarrollo de software es un conjunto ordenado de pasos a seguir para desarrollar un software de alta calidad que cumpla con las necesidades del usuario. Entre los principales objetivos de las metodologías de desarrollo de software se encuentran:

- Cumplir con los requisitos iniciales del problema.
- Minimizar las pérdidas de tiempo en el proceso de desarrollo.
- Minimizar riesgos.
- Incrementar las posibilidades de éxito en el desarrollo de los productos.

1.10 Extreme Programming (XP)

La metodología XP es adecuada para proyectos pequeños y medianos, su ciclo de vida es iterativo e incremental, cada iteración dura pocas semanas, además de no producir demasiada documentación acerca del diseño o la planificación. XP surgió como respuesta y posible solución a los problemas derivados del cambio en los requerimientos. Es una metodología basada en la simplicidad, la comunicación y la reutilización del código de desarrollo (la propiedad del código es colectiva). Provee ventajas como:

- Brinda una programación organizada.
- Posee menor taza de errores.
- Satisface al programador.
- Los clientes tienen el control sobre las prioridades, por lo que es la más utilizada en la implementación de nuevas tecnologías donde los requerimientos cambian rápidamente [27].

XP fue desarrollada por Kent Beck quien 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 es sabido que el cambio va a suceder; el problema es la incapacidad de adaptarse a dicho cambio cuando éste tiene lugar” [27].

XP es además la metodología definida por la línea de producción PostgreSQL del centro DATEC como metodología de desarrollo de software a utilizar, por lo que su utilización para el desarrollo de la herramienta sería el más adecuado.

1.11 Lenguajes y herramientas de desarrollo de software

A continuación se detallan los lenguajes que se utilizarán en el desarrollo de la aplicación, así como las herramientas para la implementación del código y el análisis de la ingeniería propuesta por la metodología XP.

1.11.1 Lenguaje de modelado

Para modelar el sistema se utilizará el Lenguaje Unificado de Modelado (UML). El mismo 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 [28]. Está compuesto por diferentes elementos gráficos que se combinan para formar los diagramas. A continuación las principales características que posee:

- Contiene corrección de errores viables en todas las etapas.
- Es aplicable para tratar asuntos de escala inherentes a sistemas complejos.
- Trabaja en tiempo real.
- Hace uso del modelo cliente-servidor.
- Los modelos permiten la comunicación con el cliente en todas las etapas.
- El principal beneficio de UML es que unifica distintas notaciones previas.

En la actualidad 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.

1.11.2 Herramienta case

Durante el ciclo de vida de desarrollo de un software las herramientas CASE 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 presente trabajo la herramienta CASE Visual

Paradigm (VP), en su versión 8.0, la cual 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 VP ayuda a la construcción de aplicaciones de calidad y contribuye a minimizar los costes del proyecto. VP 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. También facilita el modelado colaborativo con *Subversion*. Proporciona la transformación de diagramas entidad-relación en tablas de BD e ingeniería inversa de BD. Además posee un potente generador de informes.

1.11.3 Lenguaje de programación

El lenguaje de programación seleccionado fue Java, ya que es un lenguaje orientado a objetos, desarrollado por Sun Microsystems a principio de los años 90´s. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. El lenguaje Java posee entre sus ventajas el hecho de ser multiplataforma. El mismo se ha extendido, cobrando cada día más importancia tanto en el ámbito de Internet como en el de la informática en general [29]. Java permite la integración de múltiples frameworks que ofrecen muchas facilidades: persistencia, seguridad, logs, internacionalización, BPM, pruebas, tareas asincrónicas, servicios web, pantallas con elementos ricos de interfaz y relativa facilidad de uso (múltiples implementaciones de JSF por Apache MyFaces, Oracle ADF y RichFaces).

1.11.4 IDE NetBeans 7.2.1

NetBeans IDE es un entorno de desarrollo integrado (IDE) modular basado en estándares. El mismo fue desarrollado con el lenguaje de programación Java. El proyecto de NetBeans consta de un IDE de código abierto con gran variedad de funciones escritas con el lenguaje de programación Java y una plataforma enriquecida para aplicaciones de cliente, que se puede utilizar como marco genérico para crear cualquier tipo de aplicación [29].

1.12 Conclusiones del capítulo

En el desarrollo del presente capítulo se obtuvieron las siguientes conclusiones:

- Se realizó una investigación relacionada con las consultas de las BD en PostgreSQL, permitiendo conocer la necesidad de optimizar el tiempo de respuesta de las mismas mediante el uso de los índices.
- Se realizó un análisis de las técnicas metaheurísticas que serán utilizadas en el proceso de optimización para la selección de índices, determinándose los AG como la más adecuada para el desarrollo de la aplicación.
- Se analizaron las herramientas que serán utilizadas para el desarrollo de la aplicación: IDE NetBeans 7.2.1, de código abierto con gran variedad de funciones y posee una plataforma para aplicaciones de cliente que permite crear cualquier tipo de aplicación; lenguaje de programación Java, multiplataforma y facilita la integración de múltiples marcos de trabajo que ofrecen muchas facilidades, persistencia y seguridad; metodología XP, para lograr un mayor entendimiento del negocio y generar menor cantidad de artefactos, convirtiéndose así en la más cómoda y ágil para trabajar.

Capítulo 2

Descripción de la solución

INTRODUCCIÓN

En el presente capítulo se describen las principales características que tendrá la herramienta. Se muestran los requisitos, se definen las historias de usuario y se realiza un diagrama de clases del diseño que se utiliza como complemento de la metodología definida, para lograr un mejor entendimiento del componente. Además se planifican las iteraciones en las cuales serán implementadas cada historia de usuario y se realiza una estimación de la duración de cada tarea de la ingeniería para de esta forma lograr una implementación correcta y con éxito.

2.1 Modelo de dominio

Dada la propuesta de la implementación de la herramienta para la recomendación de índices en PostgreSQL se propone la definición de un modelo de dominio o modelo conceptual para lograr una mayor comprensión del negocio del problema, donde se muestran los principales conceptos a utilizar en el desarrollo.

El artefacto modelo conceptual o de dominio 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 [30].

El modelo de dominio se describe mediante diagramas de UML, específicamente mediante diagramas de clases. A pesar de que este modelo no forma parte de la metodología escogida, se decide realizar el mismo debido a que brinda un mejor entendimiento para la implementación de la herramienta. A continuación la representación del modelo de dominio propuesto para el desarrollo de la herramienta (ver figura 4).

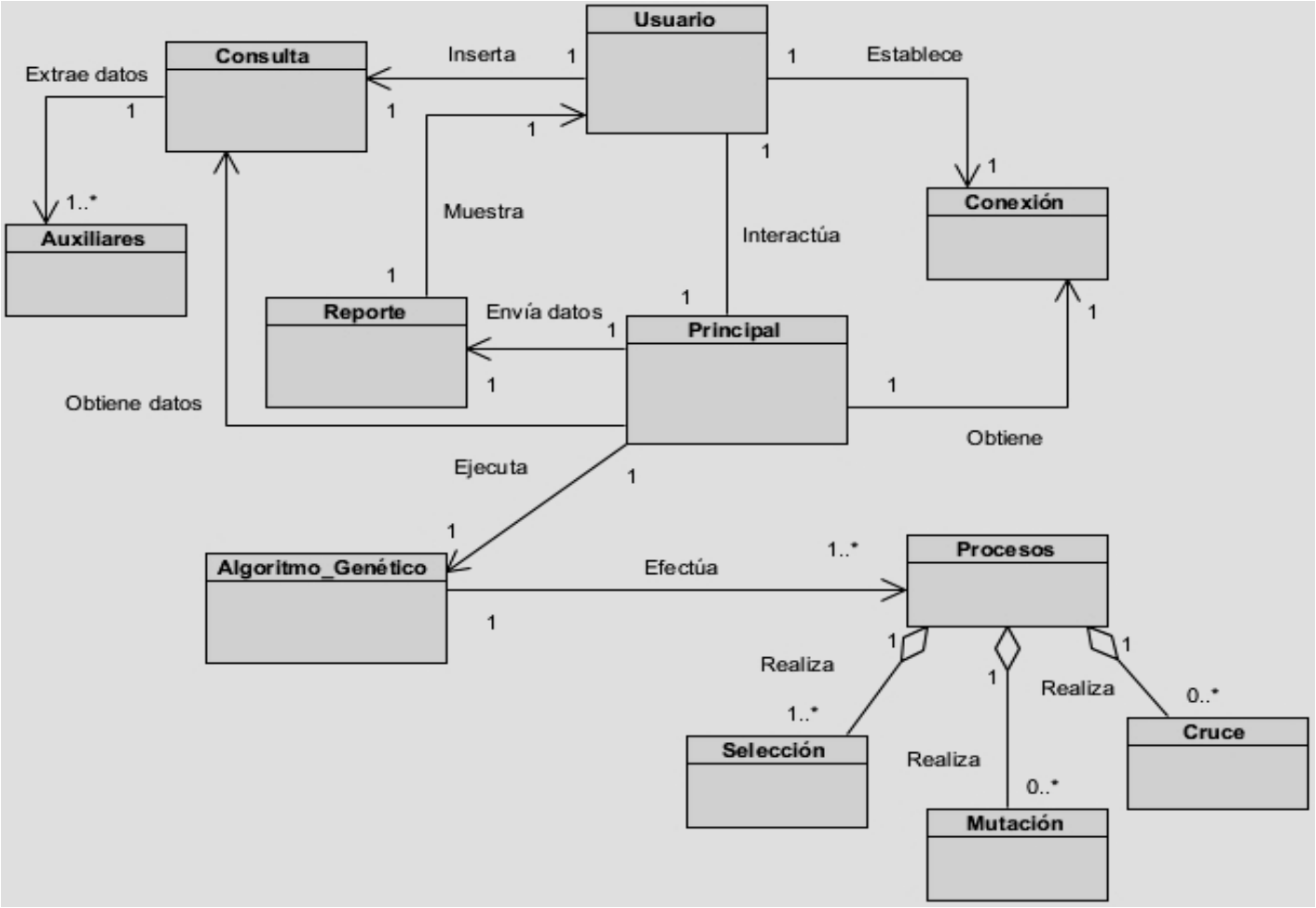


Figura 4: Modelo de dominio

La descripción del modelo de dominio representado en la figura 4 es la siguiente: el usuario interactúa con la ventana principal del sistema, la cual le permitirá establecer la conexión. Una vez realizado este proceso el usuario insertará una consulta en lenguaje SQL y el sistema extraerá de la misma los datos necesarios para que el usuario seleccione el tipo de análisis que desea realizar (dígase análisis para índices *tipados* o *no tipados*). Luego de seleccionado el análisis deseado por parte del usuario, el sistema ejecuta el algoritmo genético correspondiente mediante los distintos procesos (selección, cruce y mutación). Si al finalizar la ejecución del algoritmo se obtiene una solución óptima, el sistema mostrará al usuario un reporte que contendrá la misma.

2.2 Propuesta del sistema

La herramienta para la recomendación de índices en PostgreSQL permitirá a los usuarios insertar las consultas que desea analizar con el objetivo de optimizar las mismas mediante la creación de índices. El procedimiento para un correcto funcionamiento de la misma se describe a continuación:

1. El usuario deberá introducir los datos solicitados (nombre de BD, servidor, puerto, usuario y contraseña) para establecer la conexión con el servidor de BD PostgreSQL. Para acceder al resto de las funcionalidades de la aplicación el usuario necesita realizar el procedimiento anterior.
2. Seguidamente se debe introducir la consulta que se desea optimizar. Solo se analizarán consultas de tipo **SELECT** o que comiencen con la cláusula **WITH** del **SELECT**. El sistema extrae los datos necesarios (esquema, tabla, columna, condición y combinación de índices posibles) dentro de las condiciones, para crear un listado de los posibles índices parciales o posibles respuestas.
3. A continuación se procede a elegir el tipo de análisis que se desea realizar, dependiendo de esta selección el sistema ejecuta el algoritmo genético correspondiente (*ver Anexo 1 "Ejemplo de ejecución de AG para índices tipados"*). En caso de encontrar una solución óptima se devuelven los datos pertenecientes al individuo de esta solución. Los datos se interpretan de la siguiente manera: El objeto individuo contiene la información concerniente al cromosoma del individuo y el valor de aptitud del mismo.

   El cromosoma está conformado por una cadena de números (en caso del análisis para índices no *tipados* esta combinación de números estará conformada por el alfabeto binario, mientras que en el análisis para los índices *tipados* las combinaciones de dígitos se realizarán por números enteros entre 0 y 4 en dependencia de las restricciones de los índices posibles a crear en cada columna), la posición de cada valor se corresponde con el listado de las condiciones extraídas de la consulta. Mientras que el valor de aptitud está dado por la función objetivo, el cual representa la diferencia entre el tiempo de ejecución de la consulta sin los índices recomendados menos el valor del tiempo de ejecución de la consulta en caso de que la misma utilice al menos uno de los índices recomendados, en caso contrario el valor de aptitud asumirá por defecto un número negativo con un valor absoluto extremadamente alto, esto permite diferenciar las buenas soluciones de las malas. Los datos necesarios para calcular este valor se obtienen ejecutando la función EXPLAIN ANALYZE de PostgreSQL.

Como parte de la solución además se genera una cadena binaria denominada “óptimos locales”, cuyos valores correspondientes al digito “1” representan los índices que están siendo realmente utilizados por el planificador de consultas de PostgreSQL, la posición de estos valores coincide con la posición de los valores correspondientes al cromosoma, tal como se demuestra en el siguiente ejemplo:

Para la consulta:
“***SELECT*** *title, language_id, f.last_update, category_id* ***FROM*** *film f inner* ***JOIN*** *film_category as fc* ***ON*** *f.film_id = fc.film_id* ***WHERE*** *f.film_id < 50* ***AND*** *(fc.category_id > 9 or f.rental_duration = 3)* ***AND*** *f.length < 71* ***OR*** *f.fulltext = 'academi' and fc.film_id < 21*;”

Se realizaron ambos análisis obteniéndose los resultados que se muestran en la figura 5:

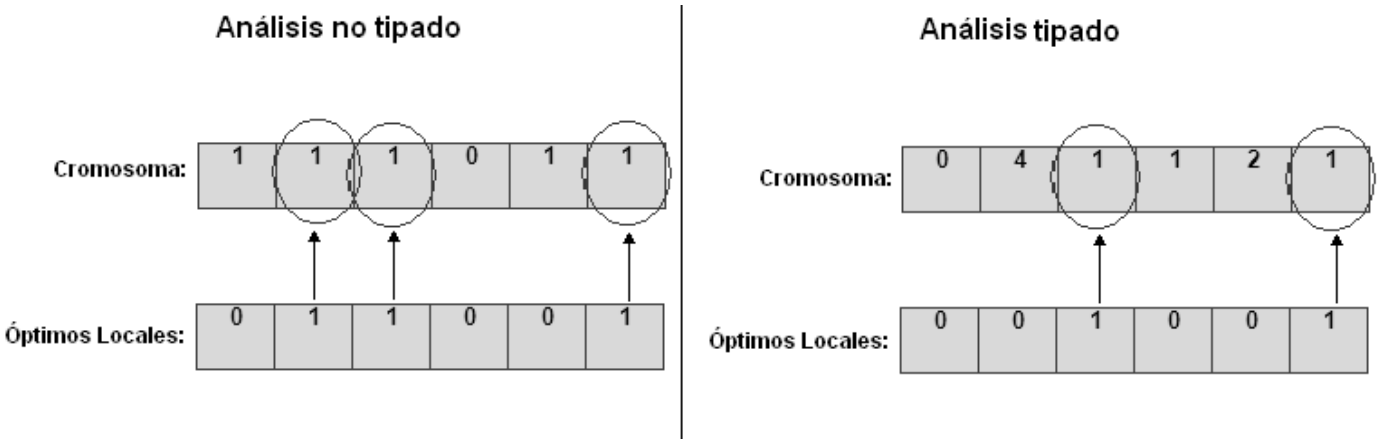


Figura 5: Ejemplo de solución de ambos análisis

En la figura 5 se muestra el cromosoma perteneciente al análisis *tipado*, donde cada número representa el tipo de índice a crear (0: no posee índices; 1: índice Btree; 2: índice Hash; 3: índice Gist; 4: índice Gin). Teniendo en cuenta que la máscara real señala la posición en la cadena del cromosoma correspondiente a los índices que están siendo realmente utilizados, se recomendaría al usuario crear los índices subrayados en la figura 5. Para el análisis *no tipado* el procedimiento para interpretar la solución sería el mismo con la diferencia de que los posibles índices a crear, al no poseer un tipo específico asumirían por defecto Btree, por lo que la cadena del cromosoma estaría conformada solamente por ceros y unos (0: no posee índices; 1: índice Btree).

La aplicación brindará al usuario la posibilidad de realizar una salva de los índices existentes en la BD y restaurar los mismos en caso de producirse algún error en el proceso. Además se le permitirá al usuario exportar el código de los índices de la solución obtenida en formato ".sql" y ".pdf".

2.3 Historias de usuarios

Las historias de usuario (HU) son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico, en cualquier momento las HU pueden 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 pocas semanas [31].

Para el presente trabajo de diploma se obtuvieron un total de 8 HU las cuales serán implementadas. Las HU identificadas son:

1- Establecer conexión.
2- Verificar consulta SQL.
3- Mostrar atributos candidatos.
4- Realizar salvas del código de los índices.
5- Realizar análisis índices *Btree*.
6- Realizar análisis todos los índices.
7- Restaurar índices.
8- Mostrar resultados de los análisis realizados.

Estas HU alcanzan un grado de importancia para el desarrollo de muy alta, alta y media, no se encuentra ninguna que la prioridad en el negocio sea baja puesto que es de suma importancia la realización exitosa de cada una de ellas. A continuación se muestran en las siguientes tablas las HU más significativas (ver tablas 1 y 2).

Tabla 1. Historia de usuario: Realizar análisis índices *Btree*

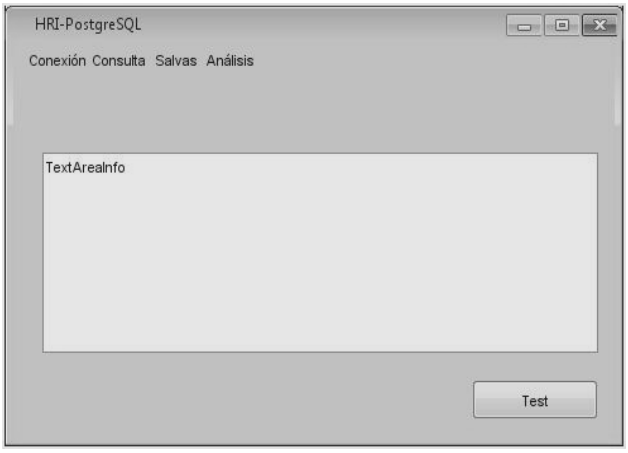
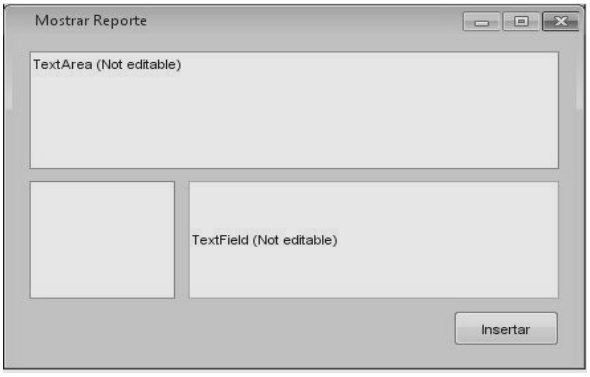
Historia de Usuario	
Número: 5	**Nombre de la Historia de Usuario:** Realizar análisis índices *Btree*.
Cantidad de modificaciones a la Historia de Usuario: 0	
Usuario: usuario	**Iteración asignada:** 1
Prioridad en negocio: Muy alta	**Puntos estimados:** 3
Riesgo en desarrollo: Muy alta	**Puntos reales:** 3
Descripción: Se realiza la codificación binaria de los cromosomas o individuos que conformarán la población. Se crea la población inicial de forma aleatoria y se ejecuta el algoritmo genético en sus diferentes procesos: selección, cruce y mutación. Se ejecutan estos procesos hasta que se alcance una solución con un nivel de optimalidad adecuado o se alcance un número máximo de iteraciones.	
Prototipo de interfaces: 	

Tabla 2. Historia de usuario: Mostrar resultados de los análisis realizados

<table>
<tr><td colspan="3">Historia de Usuario</td></tr>
<tr><td>Número: 8</td><td colspan="2">Nombre de la Historia de Usuario: Mostrar resultados de los análisis realizados.</td></tr>
<tr><td colspan="3">Cantidad de modificaciones a la Historia de Usuario: 0</td></tr>
<tr><td>Usuario: usuario</td><td colspan="2">Iteración asignada: 2</td></tr>
<tr><td>Prioridad en negocio: Alta</td><td colspan="2">Puntos estimados: 3</td></tr>
<tr><td>Riesgo en desarrollo: Alta</td><td colspan="2">Puntos reales: 2.5</td></tr>
<tr><td colspan="3">Descripción: Se muestran los resultados de los análisis realizados en base a la consulta insertada por el usuario. Se ofrecen las distintas soluciones encontradas en dependencia del análisis realizado. Además de mostrar los resultados del Explain Analyze que permite al usuario establecer una comparación entre el resultado de la consulta con los índices creados y sin ellos. Se brinda la posibilidad al usuario de insertar los resultados obtenidos en la base de datos.</td></tr>
<tr><td colspan="3">Prototipo de interfaces:
 
</td></tr>
</table>

2.4 Lista de reserva del producto

La lista de reserva del producto agrupa los requisitos funcionales organizados por prioridad según la complejidad en el negocio y los requisitos no funcionales con una breve descripción de cada uno de ellos, con la estimación de cada uno de los requisitos para su implementación por semanas y el rol que realizó la estimación del mismo.

La etapa de definición de requisitos es una tarea de suma importancia a la hora de desarrollar un sistema. Los *requisitos funcionales* son capacidades o condiciones que el sistema debe cumplir [28] y los *requisitos no funcionales* especifican propiedades del sistema, como restricciones del entorno o de la implementación, rendimiento, facilidad de mantenimiento, extensibilidad y fiabilidad [32]. Cuanto más alto sea el grado de cumplimiento y aceptación por el cliente de los requisitos, más alta será la calidad del sistema desarrollado. A continuación se muestra el listado de los requisitos obtenidos, a través de la lista de reserva del producto (ver tabla 3).

Tabla 3. Lista de reserva del producto

Ítem *	Descripción	Estimación	Estimado por
Prioridad: Muy Alta			
1	Establecer conexión	1 semana	Analista
2	Verificar consulta SQL	2 semanas	Analista
3	Realizar análisis índices Btree	3 semanas	Analista
4	Realizar análisis todos los índices	3 semanas	Analista
Prioridad: Alta			
5	Mostrar resultados de los análisis realizados	3 semanas	Analista
6	Realizar salvas del código de los índices	1 semana	Analista
7	Restaurar índices	2 semanas	Analista
Prioridad: Media			
8	Mostrar atributos candidatos	2 semanas	Analista
Prioridad: Baja			
Requisitos No Funcionales			
1	**Apariencia o interfaz externa:** Se implementará una interfaz gráfica que permitirá la interacción del usuario con las funcionalidades que brinda el sistema. Las ventanas de interfaz visual no podrán ser redimensionadas por el usuario, con el objetivo de lograr mayor ecuanimidad en el proceso y seriedad.		Analista
2	**Software:** Se implementara una herramienta que utilizará sistema operativo multiplataforma, librería JDBC Driver 4.0, SGBD PostgreSQL 9.1.		Analista
3	**Hardware:** Se necesita 1 GB de memoria RAM como mínimo, 250 MB de espacio libre en el disco duro para su instalación y el micro a 2.40 GHz.		Analista
	Usabilidad: La herramienta contará con una interfaz amigable con un		

4	control de flujo que restringirá las acciones del usuario, a realizar solamente las operaciones en el orden de ejecución correcto. La utilización de iconos sugerentes permitirá al usuario familiarizarse con la herramienta. Además cuenta con un manual de ayuda que facilita el manejo de la aplicación.		Analista
5	**Portabilidad:** La aplicación estará disponible en sistemas operativos de software libre y de software privativo.		Analista
6	**Confidencialidad:** Solo tendrá acceso a trabajar con la herramienta el usuario que se encuentre registrado en el servidor de bases de datos de PostgreSQL al cual se desee conectar.		Analista
7	**Integridad:** Se realizará una salva de los índices almacenados en el sistema para prevenir, en caso de alguna interrupción abrupta del mismo, que se pierda la información de los índices almacenados en la BD. Mientras se esté realizando el proceso de análisis, no habrá pérdida de ninguno de los datos, pues los mismos mantendrán su integridad.		Analista
8	**Disponibilidad**: Se podrá hacer uso de la aplicación siempre que esté instalada la misma y el sistema posea acceso a la BD.		Analista
9	**Restricciones del diseño y la implementación:** Para el diseño e implementación de la aplicación se tuvieron en cuenta las siguientes restricciones: El lenguaje de programación Java no permite la sobrecarga de operadores ni la herencia múltiple. Además se deben tener en cuenta restricciones referentes a la capacidad de almacenamiento de los tipos de variables a utilizar, ya que en Java los valores enteros (int) solo admiten números dentro del rango desde -2^{31} hasta $2^{31}-1$ (-2147483648 a 2147483647).		Analista

2.5 Tareas de la ingeniería

Una vez descritas las HU, se procede a describir cada una de las tareas que se van a elaborar dentro de cada HU. Cada una contendrá los objetivos que se deben cumplir, el tiempo en que tardará en realizar la tarea y el responsable asignado.

Las tareas de la ingeniería se consideran como las entradas de trabajo para el equipo de programadores. Son la ficha que contiene el número de identificador de la tarea, el identificador de la HU con la que está relacionada, el nombre de la tarea, la fecha de inicio, la fecha de fin y la descripción. A continuación se muestra un ejemplo de las 24 tareas de ingeniería asociadas a las 8 HU mencionadas anteriormente (ver tabla 4).

Tabla 4. Ejemplo de Tareas de Ingeniería

Tarea de Ingeniería	
Número Tarea: 3	**Número Historia de Usuario:** 2
Nombre Tarea: Obtener subconsulta	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0.5 semanas
Fecha Inicio: 5/02/2013	**Fecha Fin:** 15/02/2013
Programador Responsable: Jorge Deinier Sosa Lastres	
Descripción: Se analiza el código para listar las subconsultas que se encuentran dentro de la consulta introducida por el usuario y obtener los datos necesarios para realizar el análisis.	

2.6 Plan de iteraciones

La planificación de la etapa de implementación del sistema será realizada luego de ser descritas las HU. En este plan se especifica exactamente cuáles HU serán implementadas para cada iteración [33]. Para el desarrollo de la aplicación se definieron 3 iteraciones las cuales se detallan a continuación en la tabla 5:

Tabla 5. Plan de iteraciones del producto

Iteración	Descripción de la iteración	Orden de la HU a implementar	Duración total
Iteración 1	Se implementan las HU que tienen la prioridad en el negocio **MUY ALTA**.	1,2,5,6	9 semanas
Iteración 2	Se implementan las HU que tienen la prioridad en el negocio **ALTA**.	3,8	6 semanas
Iteración 3	Se implementan las HU que tienen la prioridad en el negocio **MEDIA**.	4,7	2 semanas

2.7 Diseño del sistema

Para el diseño de aplicaciones informáticas la metodología XP no requiere la presentación del sistema mediante diagramas de clases utilizando notación UML. En su lugar se usan otras técnicas como las tarjetas CRC (contenido, responsabilidad y colaboración). No obstante el uso de estos diagramas puede aplicarse siempre y cuando influyan en el mejoramiento de la comunicación, no sea un peso su mantenimiento, no sean extensos y se enfoquen en la información importante [34].

2.7.1 Tarjetas CRC

Las tarjetas CRC trabajan con una metodología basada en objetos, estas garantizan que el equipo completo contribuya en la tarea del diseño. Las mismas se dividen en 3 partes:

Clase: Representa una colección de objetos similares.
Responsabilidades: Describen las funciones que debe realizar una clase.
Colaboraciones: Describen las demás clases con las que trabaja una clase en conjunto para llevar a cabo sus responsabilidades.

En la investigación del presente trabajo se identificaron un total de 31 tarjetas CRC. A continuación se muestra un ejemplo de la tarjeta CRC *Parseo_SQL* en la tabla 6, la cual fue generada para obtener los datos necesarios de las consultas y luego realizar los procesos de optimización mediante los AG, el resto de las tarjetas se pueden consultar en el expediente de proyecto asociado a este trabajo.

Tabla 6. Tarjeta CRC Parseo_SQL

Tarjeta CRC	
Clase: Parseo_SQL	
Responsabilidades	**Colaboraciones**
• Listar las subconsultas dentro de la consulta principal. • Listar las condiciones a evaluar en los algoritmos genéticos. • Obtener el nombre de las columnas que intervienen en las condiciones. • Obtener el nombre de las tablas de las columnas que intervienen en las condiciones. • Obtener el nombre de los esquemas de las tablas que intervienen en las condiciones.	• Conexión • Controladora • Metodos_Auxiliares • Metodos_Subconsulta • Metodos_Parseo • Metodos_Condiciones • Consulta • Columna • Tabla

2.7.2 Diagrama de clases

El Diagrama de Clases es el diagrama principal para el análisis y diseño. Representa las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones. Aunque la metodología XP no define un diagrama de clases del sistema, se presenta a continuación el diagrama realizado para un mejor entendimiento de la aplicación, puesto que éste se puede utilizar siempre y cuando contribuya al mejoramiento de la comunicación y comprensión del

sistema, ya que es una guía esencial para los desarrolladores a la hora de implementar. El diagrama cuenta con 31 clases, está compuesto por las relaciones de herencia, composición y agregación.

En la figura 6 se muestra el diagrama de clases con las principales clases de la herramienta asociada al presente trabajo. El diagrama se elaboró siguiendo el patrón arquitectónico de 3 capas definido para la aplicación. En la figura se evidencian las relaciones entre las clases de la capa de información (*Algoritmo_Genetico*, *Selección*, *Mutacion*, *Cruce*, *Parseo_SQL*, *Conexion*, *Reporte* y *Consulta*), y su interacción a través de las clases *Algoritmo_Genetico* y *Parseo_SQL* con la capa de negocio, representada en la clase *Controladora*, esta clase contiene la lógica del negocio y sirve de intermediaria entre la capa de información y la capa de presentación. La capa de presentación está representada en este ejemplo por la clase *Principal_Visual*.

Nota: Para ver el diagrama de clases completamente, diríjase al expediente de proyecto asociado al presente trabajo.

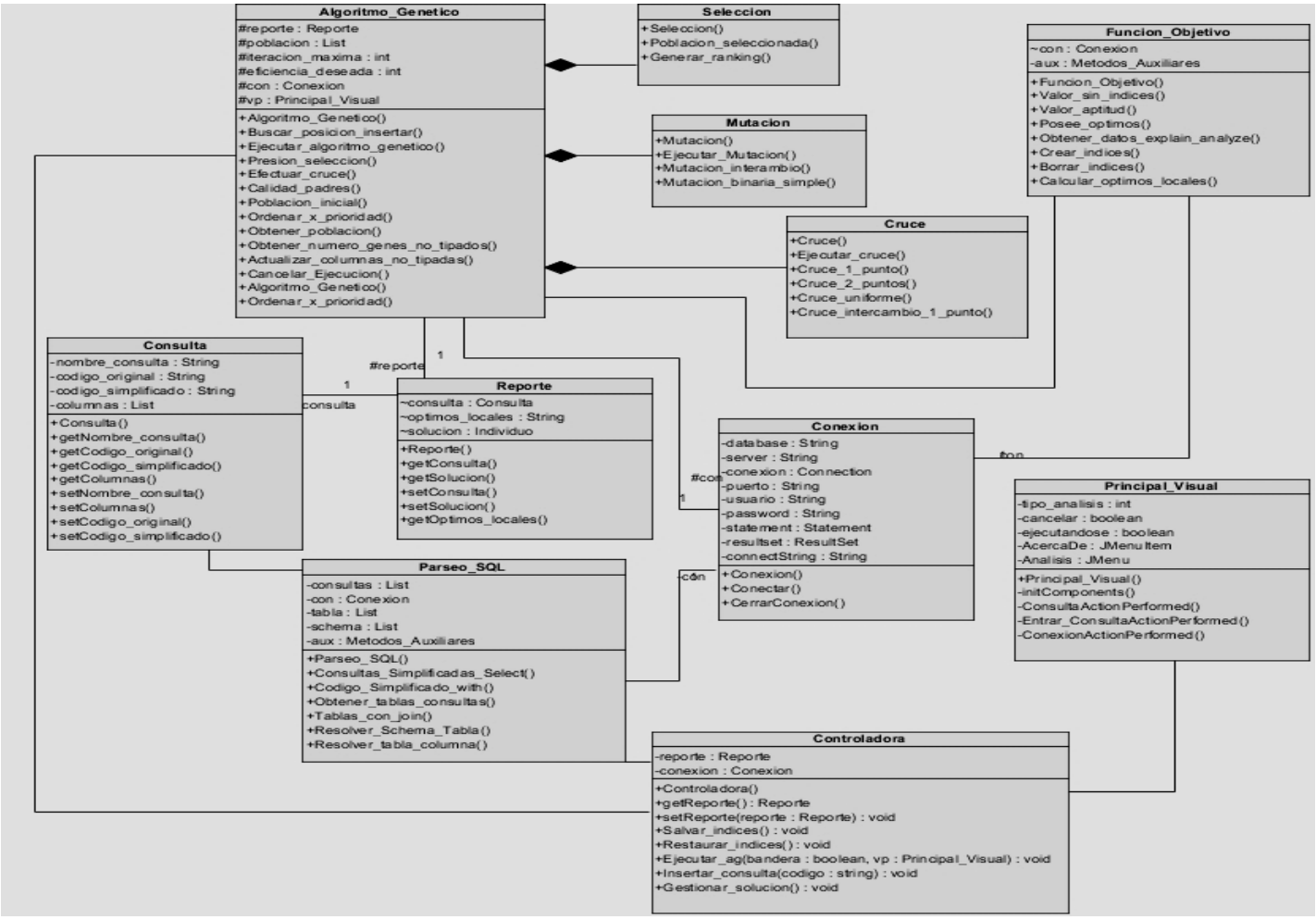


Figura 6: Diagrama de clases

2.8 Patrones de arquitectura

El diseño en la metodología XP aspira a lograr cumplir con los siguientes aspectos:

- ✓ Lograr el conocimiento de los patrones de arquitectura, no solo las soluciones sino también apreciar cuándo se usan y se evoluciona hacia los mismos.
- ✓ Alcanzar el resultado de cómo comunicar el diseño a las personas que necesitan entender el mismo, con el uso de código y diagramas.

Cuando se construye un software como producto empresarial o comercial se llevan a cabo varias técnicas, de manera que el desarrollo se haga en forma ordenada y así poder asegurar un avance continuo del proyecto y un producto final de calidad [35]. Para facilitar y dar cumplimiento a los aspectos antes planteados se utilizan patrones arquitectónicos.

Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software [36]. En el presente trabajo se utiliza el patrón programación en capas para lograr que el sistema quede perfectamente organizado y de esta forma obtener un orden lógico en la programación del mismo. A continuación una breve explicación de cada una de las capas del sistema.

Capa de presentación (o interfaz del usuario): Esta capa contiene las interfaces visuales elaboradas en Java, las cuales permiten al usuario interactuar con el sistema. La capa de presentación se comunica únicamente con la capa de negocio.

Capa de negocio: En esta capa se encuentra la lógica del negocio, es la intermediaria entre la primera capa y la tercera. En la capa de negocio es donde se reciben las peticiones del usuario y se envían las respuestas tras realizar el proceso. Se establecen todas las reglas y parámetros que deben cumplirse.

Capa de información: En esta capa se evidencian todas las clases que poseen alguna información necesaria, pero que ellas de por sí solas no funcionan. Estas clases contienen toda la información que la capa de negocio necesita para realizar las operaciones (ver figura 7).

Figura 7: Modelo en capas

2.9 Patrones de diseño

Los patrones de diseño son soluciones simples a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan. Los patrones de diseños brindan gran ayuda a la hora de resolver problemas de diseño ya que sugieren clases y objetos. Proponen además interfaces entre objeto, ejemplo de esto son las clases y operaciones abstractas. Permiten la reutilización del código. Los patrones tienen la facilidad de implementarse de varias formas siempre que se mantengan estables las interfaces [37]. A continuación se presentan los patrones empleados en la creación de la herramienta para la Recomendación de Índices en PostgreSQL. Se hizo uso de los patrones *GRASP* y *GOF*.

2.9.1 Patrones GRASP

Patrones GRASP: Acrónimo de General Responsability Assignment Software Patterns (Patrones de Software para la asignación General de Responsabilidad). Describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades [38].

Dentro de los patrones GRASP a utilizar en el presente trabajo se encuentran: Experto, Creador, Alta Cohesión y Controlador.

Experto:

Indican que la responsabilidad de la creación de un objeto debe recaer sobre la clase que conoce toda la información necesaria para crearlo. Es aquella clase cuya responsabilidad es llevar consigo toda la información y ser experta en el tema para no depender de ninguna otra. En éste trabajo se evidencia este patrón en la clase *Principal_Visual*, la cual se encarga de asignar responsabilidades a las clases *Conexion_Visual* e *Insertar_Consulta_Visual* (ver figura 8).

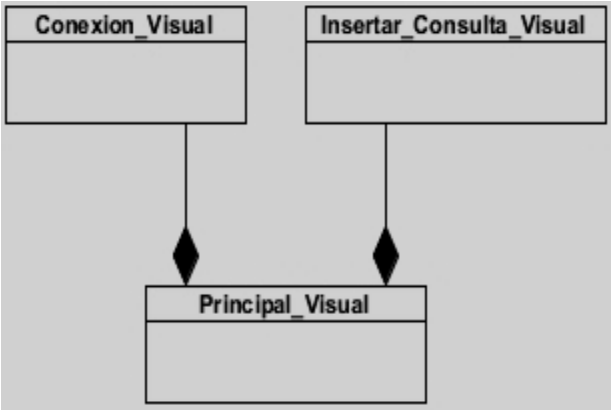


Figura 8. Patrón experto

Creador:

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Un ejemplo de este patrón en el presente trabajo se muestra en la clase *Algoritmo_Genetico*, la cual crea objetos de las clases *Cruce* y *Seleccion*, para realizar sus responsabilidades (ver figura 9).

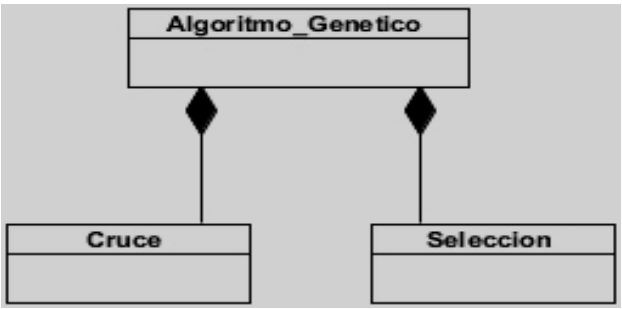


Figura 9. Patrón creador

Controlador:

El patrón controlador es el encargado de controlar un evento del sistema. Es aquel que sirve para intermediar entre la interfaz y el algoritmo. En éste trabajo se refleja este patrón en la clase *Controladora*, que contiene la lógica del negocio y es la encargada de intermediar entre la clases *Insertar_Consulta_Visual* y *Parseo_SQL* (ver figura 10).

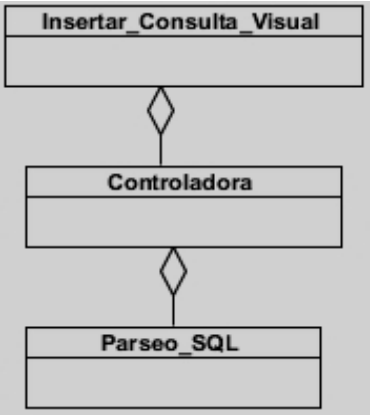


Figura 10. Patrón controlador

Alta cohesión:

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. El alto nivel de cohesión es presentado por las clases que tienen responsabilidades y colaboran con otras para llevar a cabo las tareas. Un ejemplo de este patrón se evidencia en la clase *Parseo_SQL*, la cual para poder realizar la responsabilidad de extraer los datos de la consulta insertada por el usuario necesita de otras clases que contienen métodos que son de vital importancia para poder realizar ésta tarea (ver figura 11).

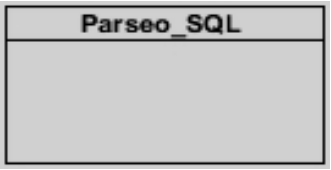


Figura 11. Patrón alta cohesión

2.9.2 Patrones GOF

Patrones GOF: Es la abreviación del grupo Gang of Four, compuesto por Erich Gamma, Richard Helm, Ralph Jhonson y John Vlisides, quienes en su publicación "Design Patterns", describen 23 patrones de diseño comúnmente utilizados y de gran aplicabilidad en problemas de diseño usando modelamiento UML. Estos patrones se agrupan en las siguientes categorías: Creacionales, estructurales y de comportamiento [39].

Comportamiento:
Los patrones de comportamiento facilitan y definen la comunicación e iteración entre los objetos de un sistema y que estos estén lo menos mezclados posible.

Observador (Observer): Este patrón tiene la responsabilidad de que al cambiar el estado de un objeto todas las dependencias del mismo se actualicen. Este patrón se evidencia en la herramienta, puesto que al ejecutar alguno de los tipos de análisis, se actualizan en la consola los datos correspondientes al algoritmo genético que se esté ejecutando. Para desarrollar este patrón se implementaron las interfaces "*Observador"* y "*Sujeto"* donde el sujeto estaría representado por la clase "*Algoritmo_Genetico*" la cual actualiza los datos del observador que estaría representado por la clase "*Principal_Visual*".

2.10 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código [34]. A continuación se presenta un fragmento del estándar definido para la implementación de la herramienta.

Identación
La unidad de identación es de 4 espacios. El uso de la tabulación debe ser evitado pues no existe un estándar que determine con precisión el ancho que va a producir la tabulación.

Comentarios
Es conveniente plasmar información que pueda ser leída tiempo después por personas (posiblemente la misma persona que programó) que necesitan 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. Se debe reservar los comentarios de bloques para la documentación formal o para comentar porciones de código.

Longitud de línea

Evitar líneas con más de 100 caracteres. Cuando una sentencia sobrepase una línea simple del editor, ésta deber ser separada en otras. 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.

Declaración de variables

Cada variable debe de ser declarada en una línea:

```
private Metodos_Auxiliares auxiliar;
private List<Consulta> consultas;
```

El nombre de las variables debe de comenzar con letras minúsculas y cada palabra relevante por la que esté compuesta debe ser con letra minúscula y separadas por un guion bajo.
Ejemplo: consulta, explain_analize.

Declaración de funciones

No debe haber espacio entre el nombre de la función y el paréntesis izquierdo, ni entre éste 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 las variables añadiendo que deben comenzar con letra mayúscula.

```
Function_ejemplo(a, b) {
        var = a + b;
        return var;
}
```

Identificadores

Los identificadores pueden estar formados por cualquiera de las 26 letras minúsculas o mayúsculas (A … Z, a … z), los 10 dígitos (0 … 9) y el caracter 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 de dólar "$" o la barra invertida "\" en los identificadores.

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.

Espacios en blanco

Las líneas en blanco facilitan la lectura determinando secciones de código lógicamente relacionados. La utilización de los espacios en blancos se define a continuación:

- No se debe utilizar un espacio en blanco entre el identificador de una función y el paréntesis que abre la lista de parámetros. Esto 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 operando y operador.
- No se debe utilizar el espacio para separar un operador unario de su operando.
- Cada punto y coma ";" en una sentencia de control debe ser seguido por un espacio.
- Debe dejarse un espacio luego de cada coma ",".

A continuación, en la figura 12, se presenta un ejemplo del estándar aplicado en la implementación de la herramienta para la recomendación de índices en PostgreSQL para la funcionalidad *Crear_indice*. El objetivo de este método es crear una sentencia con la estructura de la declaración de un índice en PostgreSQL.

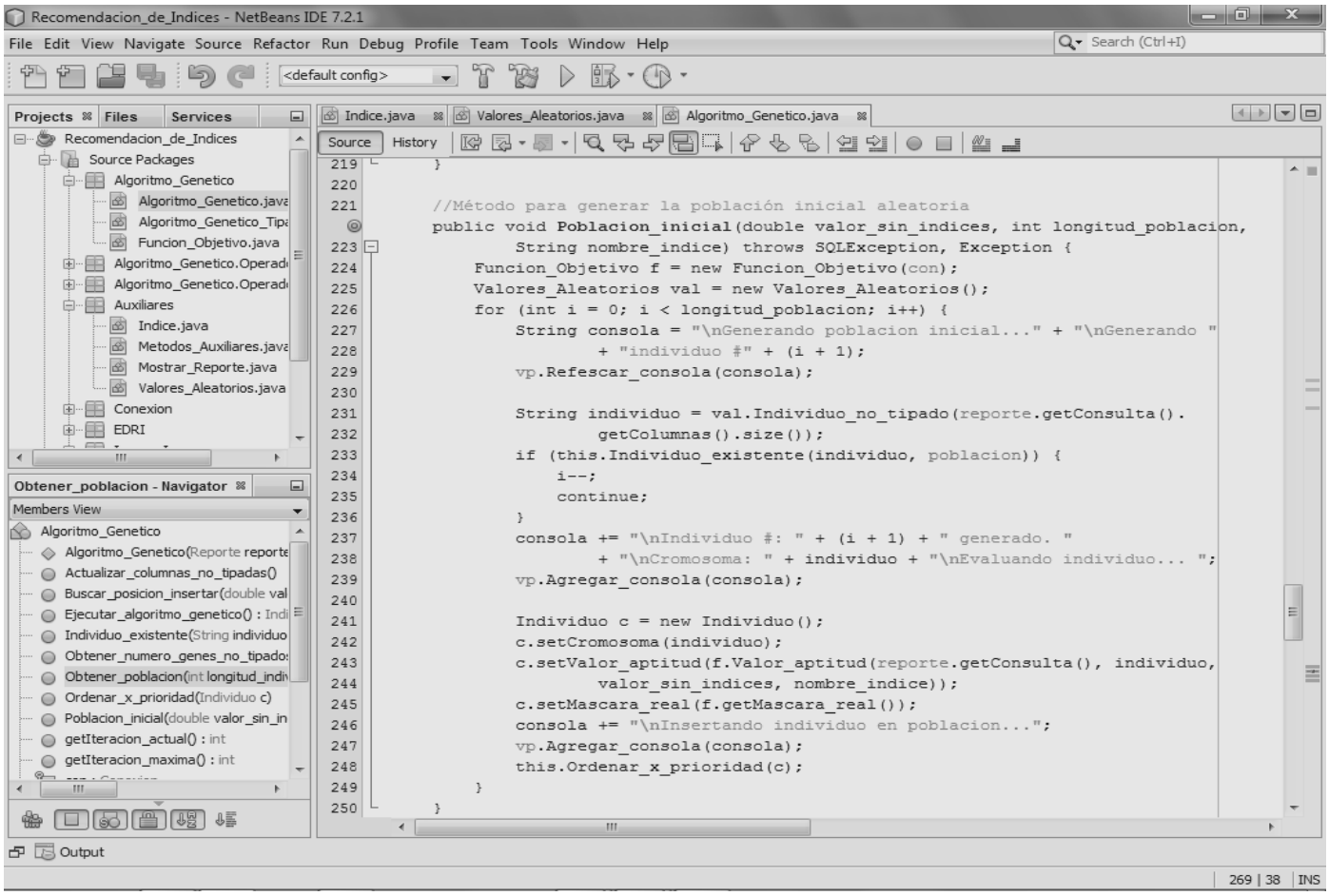


Figura 12. Ejemplo de estándar de codificación

2.11 Conclusiones del capítulo

En éste capítulo han sido analizados todos los elementos que describen las características y diseño del sistema.

- Se elaboró una propuesta con las perspectivas de solucionar el problema identificado, para eso fueron desarrolladas las fases de exploración y diseño propuestas por la metodología utilizada en las que se definieron un total de 24 tareas de la ingeniería agrupadas en 8 HU.
- Se generó además el plan de iteraciones para establecer de ésta forma la iteración en la que se desarrollará cada HU.
- Se describen un total de 31 tarjetas CRC como parte del diseño en la metodológica XP.

Capítulo 3

Validación y prueba

INTRODUCCIÓN

En este capítulo se definen un grupo de normas para la validación de los productos y/o artefactos, logrando que los mismos puedan ser probados para la verificación de su correcto funcionamiento. Se analizan las estrategias de pruebas que definen XP y la técnica que se utilizará para diseñar los casos de pruebas que guiarán la validación de la aplicación, además de mostrarse los resultados arrojados por éstas pruebas.

3.1 Validación del sistema

La validación y comprobación del funcionamiento del sistema antes que pueda ser liberado, permite aumentar su calidad, reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. Permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones.

3.2 Pruebas

La fase de pruebas es una de las fases fundamentales del desarrollo de una aplicación. El objetivo de cada una de las pruebas no es el de prevenir errores, sino detectarlos basándose en técnicas y estrategias empleadas en cada una de las pruebas. Hay multitud de conceptos (y palabras claves) asociadas a las pruebas, clasificarlas es difícil, pues no son mutuamente disjuntas y sí muy entrelazadas. Dentro de las estrategias de pruebas existentes pueden mencionarse las pruebas de unidades, las pruebas de integración, las pruebas de sistema, las pruebas de aceptación, entre otras.

3.2.1 Estrategias de pruebas

Uno de los pilares fundamentales de la metodología XP, es proporcionar al cliente la posibilidad de concretar las funcionalidades de las HU y verificarlas en el proceso de pruebas. Las mismas son la manera de comprobar el código que se vaya implementando. Las pruebas de software consisten en la verificación dinámica del comportamiento de un programa en un conjunto finito de casos de prueba. De

forma adecuada selecciona el dominio de ejecuciones generalmente infinito, contra el comportamiento esperado [40].

Existen distintas estrategias de pruebas, durante el estudio de éste trabajo de diploma se ha investigado acerca de las pruebas correspondientes a la metodología de desarrollo de software empleada en el presente trabajo. XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final.

Pruebas unitarias

Una prueba unitaria es la verificación de un módulo (unidad de código) determinado dentro de un sistema. Son llevadas a cabo por los programadores encargados de cada módulo. Aseguran que un determinado módulo cumpla con un comportamiento esperado en forma aislada antes de ser integrado al sistema [41].

Las pruebas unitarias son una de las piedras angulares de XP. Estas pruebas deben ser definidas antes de realizar el código. Todo código liberado debe pasar correctamente las pruebas unitarias, ya que es lo que posibilita que funcione la propiedad colectiva del código. Los programadores deben realizar estas pruebas cuando:

- La interfaz de un método de la aplicación no es clara.
- La implementación es complicada.
- Para probar entradas.
- En condiciones inusuales.
- Luego de alguna modificación.

Pruebas de aceptación

Las pruebas de aceptación, al igual que las de sistema, se realizan sobre el producto terminado e integrado. A diferencia de las pruebas de sistema, las de aceptación están concebidas para que sea un usuario final quien detecte los posibles errores [42].

Estas pruebas son definidas por el cliente para cada HU, y tienen como objetivo asegurar que las funcionalidades del sistema cumplan con todo lo que se espera de ellas. Las pruebas de aceptación

corresponden a una especie de documento de requerimientos en XP, ya que marcan el camino a seguir en cada iteración, indicándole al equipo de desarrollo hacia donde tiene que ir y en qué puntos o funcionalidades debe poner el mayor esfuerzo y atención. Estas pruebas no se realizan durante el desarrollo, pues sería impresentable al cliente, sino que se realizan sobre el producto terminado o pudiera ser una versión del producto o una iteración pactada previamente con el cliente.

Las pruebas de aceptación definen dos tipos de pruebas: *Alfa* y *Beta*, para posibilitar el descubrimiento de errores que solo el usuario final podría detectar. La prueba *Alfa* es aplicada por los usuarios finales en el área de trabajo del desarrollador en un entorno controlado, donde se registran los errores y problemas detectados. La prueba *Beta* a diferencia de la *Alfa* se aplica en el lugar de trabajo de los usuarios finales, en un entorno no controlado por el desarrollador, donde el usuario final es quien registra los problemas detectados y se lo informa a los desarrolladores, luego los ingenieros del software lo modifican y erradican los errores encontrados para preparar la liberación del producto.

Se decide utilizar para el desarrollo de la herramienta, la estrategia de prueba aceptación de tipo *Alfa*, ya que las pruebas de aceptación son más importantes que las unitarias, puesto que representan la satisfacción del cliente con el producto desarrollado así como el final de una iteración y el comienzo de la siguiente. Es más efectivo que el cliente dé una aceptación del producto que desea, a que el producto lo pruebe el programador. Estas pruebas conllevan al cliente a precisar lo que la aplicación debe hacer en determinadas circunstancias, por esto el cliente es la persona más adecuada para diseñar las pruebas.

3.2.2 Técnica de prueba seleccionada

Cualquier proyecto que se trace una estrategia de prueba debe contar con métodos y técnicas para la aplicación de cada una de estas pruebas. Se decide utilizar la técnica de caja negra ya que provee una serie de ventajas que posibilitan mayor facilidad para el desarrollo de los casos de pruebas.

Pruebas funcionales o Caja Negra

También conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una “Caja Negra” cuyo comportamiento sólo puede ser determinado estudiando las entradas y salidas obtenidas a partir de ellas [43] (ver figura 13).

Las pruebas de caja negra pretenden demostrar que las funciones del software son operativas y que funcionan correctamente aceptando de forma adecuada la entrada de datos y produciendo una salida correcta. Este tipo de prueba permite demostrar al cliente que la aplicación puede satisfacer las necesidades del mismo.

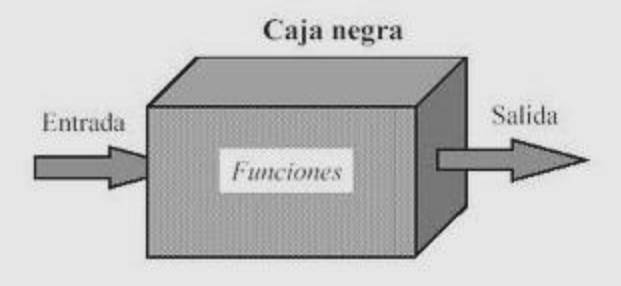


Figura 13. Técnica caja negra

Se decide aplicar la técnica de caja negra, debido a que es más importante presentarle al cliente las funcionalidades de la aplicación en correcto funcionamiento y así dejarlo conforme con el producto.

3.2.3 Casos de pruebas basados en historias de usuarios

Los casos de prueba se desarrollan usando técnicas de caja negra en la actividad *Diseño de los Casos de Prueba*. Esta actividad incluye diseñar las pruebas e identificar los datos de prueba. Para cada funcionalidad a probar, se crea una matriz que muestra su correspondencia con los casos de prueba, conociéndose de esta forma qué ítem cubre cada caso de pruebas [43].

A través de los casos de pruebas es posible probar si la aplicación realmente funciona, en ellos se describen los diferentes escenarios y se indica la respuesta correcta que el sistema debe mostrar en cada escenario. Se debe dejar bien claro en cada caso de prueba el flujo central de la aplicación, que no es más que los pasos a seguir para trabajar en la aplicación a la hora de probar cada HU. A continuación se muestra un ejemplo de los casos de pruebas utilizados para la realización de las pruebas a la herramienta para la recomendación de índices en PostgreSQL (ver tablas 7 y 8). Se desarrolló un caso de prueba para cada HU, los mismos fueron: *Establecer conexión*, *Verificar consulta SQL*, *Realizar análisis índices Btree*, *Realizar análisis todos los índices*, *Mostrar atributos candidatos*, *Mostrar resultados de los análisis*

realizados, *Realizar salvas del código de los índices* y *Restaurar índices*. Si se desea analizar el resto de las pruebas realizadas, debe remitirse al expediente de proyecto asociado a éste trabajo.

Tabla 7. Caso de prueba "Realizar análisis índices Btree"

Escenario	Descripción	Variable 1	Variable 2	Respuesta del sistema	Flujo central
EC 1.1 Realizar análisis de índices Btree con datos correctos y se obtiene un resultado óptimo.	En este escenario se introducen los datos necesarios para realizar el análisis no tipado.	V (50)	V (90)	El sistema muestra en la consola un mensaje indicando que se ha obtenido un resultado que mejora el rendimiento de la consulta insertada y le brinda al usuario la opción "Mostrar Reporte"	El usuario Selecciona la opción "Realizar análisis no tipado" en la ventana principal de la aplicación, el sistema muestra un panel compuesto por una consola y barra de progreso, dos campos de textos para insertar las variables, y los botones "Ejecutar", "Cancelar" y "Mostrar Reporte". El usuario selecciona la opción ejecutar.
EC 1.2 Realizar análisis de índices Btree con datos correctos y no se obtiene un resultado óptimo.		V (20)	V (80)	El sistema devuelve un mensaje indicando que no se han obtenido resultados óptimos para la consulta analizada en esa ejecución.	
EC 1.3 Realizar análisis de índices Btree con datos incorrectos.	En este escenario se introducen los incorrectos para realizar el análisis.	I (1500)	V (99)	El sistema muestra en la consola un mensaje indicando que se han introducido datos incorrectos.	
		V (55)	I (125)		

Tabla 8. Descripción de las variables asociadas al caso de prueba

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Iteraciones	JSpinner	No	Debe introducirse un número entero entre 1 y 1000.
2	Eficiencia	JSpinner	No	Debe introducirse un número entero entre 1 y 100.

Base de datos de prueba

Para la realización de las pruebas se utilizó como base de datos de prueba a “Pagila”, que es un puerto de la base de datos de ejemplo “Sakila” la cual está disponible para MySQL. Pagila está destinada a proporcionar un esquema estándar que se puede utilizar en ejemplos para libros, tutoriales, artículos, ejemplos de código, etc. La misma ha sido utilizada a nivel internacional para realizar pruebas en el SGBD PostgreSQL.

Ejemplo de uso

SELECT * **FROM** film **WHERE** fulltext **=** ‘Titanic’;

Consultas de prueba

Para la realización de los casos de pruebas se utilizaron una serie de consultas elaboradas por los desarrolladores y los tutores del presente trabajo de diploma.

Ejemplo de consultas

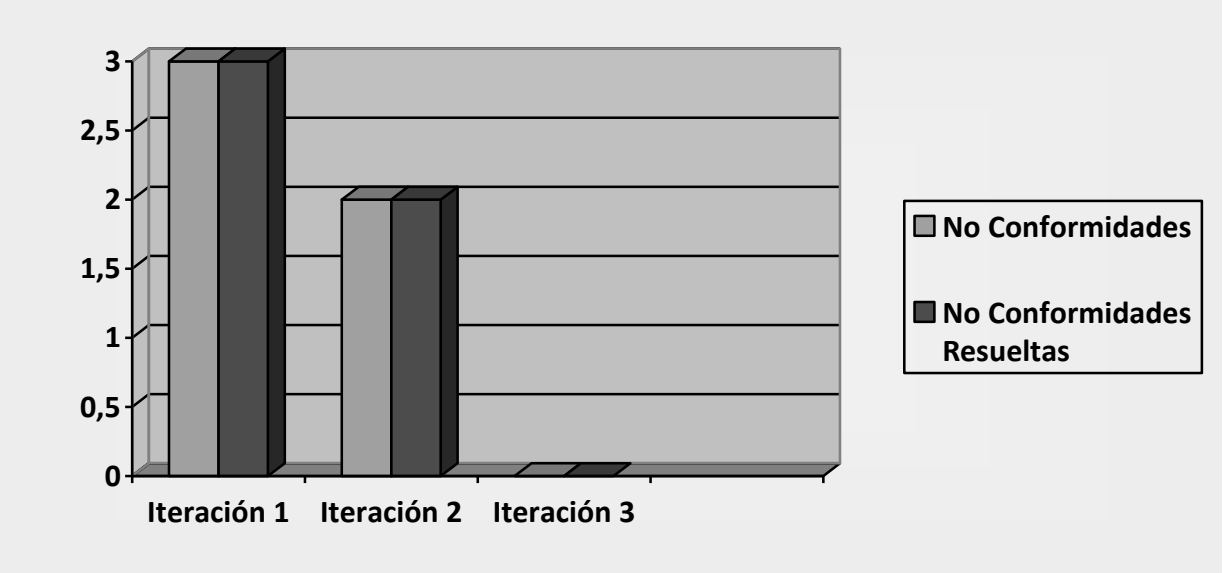
- ***SELECT*** *first_name* ***FROM*** *actor as a* ***WHERE*** *a.actor_id* ***IN*** *(****SELECT*** *a.actor_id* ***FROM*** *film_actor f* ***WHERE*** *f.film_id=945);*
- ***SELECT*** *title, language_id, f.last_update, category_id* ***FROM*** *film f inner* ***JOIN*** *film_category as fc* ***ON*** *f.film_id = fc.film_id* ***WHERE*** *f.film_id < 50* ***AND*** *(fc.category_id > 9 or f.rental_duration = 3)* ***AND*** *f.length < 71* ***OR*** *f.fulltext = 'academi' and fc.film_id < 21;*
- ***SELECT*** *title, language_id, f.last_update, category_id* ***FROM*** *film f inner* ***JOIN*** *film_category as fc* ***ON*** *f.film_id = fc.film_id* ***WHERE*** *f.film_id < 50 and (fc.category_id > 9 or f.rental_duration = 3) and f.length < 71* ***OR*** *f.fulltext = 'academi' or fc.film_id < 21;*

Nota: Para ver las imágenes de los resultados obtenidos al ejecutar las consultas en la aplicación así como la respuesta del sistema para las mismas, remítase al Anexo 2 “Mostrar reporte de la solución obtenida”.

3.2.4 Presentación de los resultados de las pruebas funcionales

Las pruebas han sido aplicadas a las ocho historias de usuarios permitiendo detectar algunos errores. Fueron encontradas en una primera iteración tres no conformidades las cuales fueron resueltas en siete

días. Para una segunda iteración se encontraron dos no conformidades solucionadas completamente en un período de tres días. Finalmente para una tercera iteración no se encontraron no conformidades (ver gráfica 1). El producto ha sido liberado emitiéndose un acta de liberación por los responsables del departamento de PostgreSQL (*ver Anexo 3 "Acta de aceptación del producto"*).



Gráfica 1. Resultado de las pruebas aplicadas al sistema

3.3 Conclusiones del capítulo

Las pruebas de software permiten la verificación de la calidad de un producto. Son utilizadas para identificar posibles fallos de una aplicación.

- Fueron probadas todas las funcionalidades y fueron solucionadas cada una de las no conformidades encontradas en la aplicación.
- Se logró presentar los resultados arrojados en cada iteración logrando obtener una herramienta que responde a todos los requisitos funcionales identificados, la cual cuenta con una alta calidad.

CONCLUSIONES GENERALES

Para la realización del presente trabajo se hizo necesario realizar una investigación acerca de los procesos de creación y selección de índices en PostgreSQL y de las distintas técnicas de optimización de la rama de la inteligencia artificial, permitiendo seleccionar la técnica de algoritmos genéticos como la más adecuada para el desarrollo de la aplicación.

- Se implementó la herramienta para la recomendación de índices en PostgreSQL.
- Se validó la solución mediante la confección de casos de prueba, uno por cada HU demostrando que la aplicación cumple con las funcionalidades identificadas.

RECOMENDACIONES

Al quedar resueltos los objetivos trazados al comienzo del presente trabajo, se plantean las siguientes recomendaciones:

- Continuar el desarrollo de la herramienta con el objetivo de incorporar otras funcionalidades y características como: establecer el tipo de orden ascendente y descendente en el **ORDER BY**, definir operadores de clases en caso de necesitarlos, incluir funciones en tipos de datos que permitan búsquedas *case-insensitive* (mayúscula y minúscula).
- Implementar en versiones posteriores de la herramienta el mismo principio de funcionamiento de ésta, pero incluir en la recomendación de índices otras cláusulas que contengan subconsultas y en ellas utilicen índices, tales como las de tipo **INSERT**, **UPDATE** y **DELETE**.

REFERENCIAS BIBLIOGRÁFICAS

1. **Pérez Valdés, Damian.** [En línea] 27 de Octubre de 2007. http://www.maestrosdelweb.com/editorial/%C2%BFque-son-las-bases-de-datos/.

2. **González, Carlos D.** [En línea] http://www.usabilidadweb.com.ar/bases_prof_exp.php.

3. **PostgreSQL.** Sitio Oficial de PostgreSQL. [En línea] 2010. http://www.postgresql.org.es/sobre_postgresql.

4. **Sharma, Neeraj, y otros.** *Database Fundamentals.* s.l. : IBM Corporation, 2010.

5. **Chu, Stephen y Thomson.** *Introducing databases.* Conrick, M. : s.n., 2006. Vol. 13. ISBN 0-17-012731-1, p. 69.

6. **Palma, Heisler.** Desarrollo Web. [En línea] 28 de Agosto de 2012. http://www.desarrolloweb.com/articulos/intro-indices-mysql.html.

7. **Lockhart, Thomas.** *Tutorial de PostgreSQL.* 1998.

8. **Grupo de usuarios PostgreSQL de Argentina;.** ArPUG. [En línea] 18 de Junio de 2012. http://www.arpug.com.ar/trac/wiki/PgAdmin.

9. **Comunidad KMKey en Español;.** KMKey. [En línea] 22 de Julio de 2009. http://kmkey-es.blogspot.com/2009/07/configuracion-avanzada-de-indices.html.

10. **Ibrahim H., Osman y Kelly, James P.** *Meta-Heuristics: An Overview.* 1996.

11. **Burset M, Guigó R.** *Evaluation of gene structure prediction programs.* 1996.

12. **Kirkpatrick, Scott, Gelatt, D. y Vecchi, Mario P.** *Optimization by simulated annealing.* s.l. : Science, 1983. págs. 671-680. Vol. 220. 4598.

13. **Glover, Fred.** *Future paths for integer programming and links to artificial intelligence.* 1986. págs. 533-549. Vol. 13.

14. **Stützle, Thomas G.** Local Search Algorithms for Combinatorial Problems. *Darmstadt University of Technology PhD Thesis.* 1998.

15. **Dorigo, Marco.** *Optimization, learning and natural algorithms.* Milano : Politecnico de Milano, 1992.

16. **Back, Thomas, Fogel, David B. y Michalewicz, Zbigniew.** *Handbook of evolutionary computation.* s.l. : IOP Publishing Ltd, 1997.

17. **Ackermann, Thomas, Andersson, Göran y Söder, Lennart.** *Distributed generation: a definition.* 2001. págs. 195-204. Vol. 57.

18. **Gil Lodoño, Natyhelem.** *Algoritmos Genéticos.* s.l. : Universidad Nacional de Colombia, 2006.

19. **Rodríguez-Piñero, Piedad Tolmos.** *Introducción a los algoritmos genéticos y sus aplicaciones.* s.l. : Universidad Rey Juan Carlos, Servicio de Publicaciones, 2003.

20. **Wolpert, David H. y Macready, William G.** *No free lunch theorems for optimization.* 1997. págs. 67-82. Vol. 1.

21. **Álvarez Macías, José.** *Curso de Doctorado en Ingenieria de Control Automática y Electrónica.* 2006. Tercer Ciclo.

22. **Goldberg, David E.** *Genetic algorithms in search, optimization, and machine learning.* 1989.

23. **Pose, Marcos Gestal.** *Introducción a los Algoritmos Genéticos.* s.l. : Departamento de Tecnologías de la Información y las Comunicaciones Universidad de Coruña, 2000.

24. **Gallego, Mariano Luque, y otros.** *Algoritmos genéticos para la resolución de problemas de Programación por Metas Entera.* [ed.] ISSN: 1575605X. Asociación Española de Profesores Universitarios de Matematicas aplicadas a la Economía y la Empresa. Málaga : s.n., 2002. pág. 42. Vol. 10.

25. **Dr. Will, Adrian.** *Algoritmos Genéticos y Optimización Heurística.* 2003.

26. **Béjar, Javier.** *Inteligencia Artificial (IA). Resolución de problemas. Algoritmos de búsqueda.* Catalunya (España) : Departamento sw Lenguajes y Sistemas e Ingeniería en Informática, 2006.

27. **Valverde Martínez, David María.** Blog de David Valverde. *Introducción a la Programación Extrema (XP).* [En línea] 6 de Septiembre de 2007. http://www.davidvalverde.com/blog/introduccion-a-la-programacion-extrema-xp/.

28. **Giraldo Gómez, Gloria Lucia.** *Ingeniería de software clase 5, Actores y sus roles. Modelo de Dominio.* s.l. : Escuela de Sistemas. Universidad Nacional de Colombia, 2006.

29. **NetBeans.** Sitio Oficial del NetBeans. [En línea] 8 de Diciembre de 2009. https://netbeans.org/community/releases/68/relnotes_es.html.

30. *Modelo Conceptual / Modelo de Dominio.* **Carolina Martínez, Prof. Ivette.** Caracas : Universidad Simon Bolivar. Ingeniería del Software, 2000.

31. **Letelier, Patricio y Pnadés, María del Carmen.** *Metodologías ágiles para el desarrollo de software, Extreme Programming (XP).* sn. 2006. Vol. 5.

32. **Rodríguez Guadarrama, Addiel y Mazorra, Thaymí.** *Análisis y diseño de una herramienta web para la gestión de la información.* Habana : Universidad de las Ciencias Informáticas (UCI), 2010.

33. **Barbosa Guerrero, Lugo Manuel.** *Arquitectura de Software como eje temático de investigación.* 2006.

34. **Hugo y Serrano Cuayahuitl, Victor.** *Pruebas de unidad, Estándares de codificación y generación automática de código.* Universidad Autónoma de Tlaxcala. s.l. : Facultad de Ciencias básicas, ingeniería y tecnología.

35. **Vargas Del Valle, Ricardo J.** *Programación en Capas.* 2010.

36. **Llorente, César de la Torre.** *Guía de Arquitectura N-Capas orientada al dominio con .NET.* 2010. 978-84-936696-8.

37. **Laman, Craig.** *Introducción al análisis y diseño orientado a objeto.* México: s.n. : PRENTICE HALL, 1999. 970-17-0261.

38. **Hurtado Bustamante, Diana Paola.** *Patrones Grasp.* s.l. : Universidad del Valle: s.n., 2011.

39. **Abran, Alain.** *Guide to the software Engineering Body of Knowledge.* s.l. : SWEBOK, 2004. 0-7695-2330-7.

40. **Malforá, Dayvis y otros.** *Testing en eXtreme Programing.* 2006.

41. **Suárez, Pablo y Fontela, Carlos.** *Documentación y pruebas ante el paradigma de objetos.* 2003.

42. **Juristo, Natalia, Moreno, Ana M. y Vegas, Sira.** *Técnicas de evaluación de software.* 2006.

43. **Pérez, Beatriz.** *ProTest-Proceso de Testing Funcional.* Puebla, Mexico : s.n., 2006. 970-94770-0-5.

BIBLIOGRAFÍA

1. **Abran, Alain.** *Guide to the software Engineering Body of Knowledge.* s.l. : SWEBOK, 2004. 0-7695-2330-7.

2. **Ackermann, Thomas, Andersson, Göran y Söder, Lennart.** *Distributed generation: a definition.* 2001. págs. 195-204. Vol. 57.

3. **Alfonso, Hugo, y otros.** *Metaheurísticas aplicadas a problemas de optimización.* La Pampa (Argentina) : Facultad de Ingeniería - Universidad Nacional de La Pampa, 2010.

4. **Alonso, Sergioso, y otros.** *La Metaheurística de Optimización Basada en Colonias de Hormigas: Modelos y Nuevos Enfoques.* Granada (España): s.n., 2001. 18071.

5. **Álvarez Macías, José.** *Curso de Doctorado en Ingenieria de Control Automática y Electrónica.* 2006. Tercer Ciclo.

6. **Back, Thomas, Fogel, David B. y Michalewicz, Zbigniew.** *Handbook of evolutionary computation.* s.l. : IOP Publishing Ltd, 1997.

7. **Barbosa Guerrero, Lugo Manuel.** *Arquitectura de Software como eje temático de investigación.* 2006.

8. **Béjar, Javier.** *Inteligencia Artificial (IA). Resolución de problemas. Algoritmos de búsqueda.* Catalunya (España) : Departamento sw Lenguajes y Sistemas e Ingeniería en Informática, 2006.

9. **Burset M, Guigó R.** *Evaluation of gene structure prediction programs.* 1996.

10. **Cardemil, Andrés y Durán, Guillermo.** *Un Algoritmo Tabu Search para el Traveling Tournament Problem* l. : Facultad de Ciencias Exactas y Naturales, 2004, Revista Ingeniería de Sistemas, Vol. XVIII.

11. **Carolina Martínez, Prof. Ivette.** *Modelo Conceptual / Modelo de Dominio.* Caracas : Universidad Simon Bolivar. Ingeniería del Software, 2000.

12. **Chu, Stephen y Thomson.** *Introducing databases.* Conrick, M. : s.n., 2006. Vol. 13. ISBN 0-17-012731-1, p. 69.

13. **Comunidad KMKey en Español;.** KMKey. [En línea] 22 de Julio de 2009. http://kmkey-es.blogspot.com/2009/07/configuracion-avanzada-de-indices.html.

14. **Dorigo, Marco.** *Optimization, learning and natural algorithms.* Milano : Politecnico de Milano, 1992.

15. **Dr. Will, Adrian.** *Algoritmos Genéticos y Optimización Heurística.* 2003.

16. **Ezequiel Rozic, Sergio.** *Bases de Datos.* Buenos Aires (Argentina) : MP Ediciones SA, 2004. ISBN 987-526-213-7.

17. **Gallego, Mariano Luque, y otros.** *Algoritmos genéticos para la resolución de problemas de Programación por Metas Entera.* [ed.] ISSN: 1575605X. Asosiación Española de Profesores Universitarios de Matematicas aplicadas a la Economía y la Empresa. Málaga : s.n., 2002. pág. 42. Vol. 10.

18. **Gil Lodoño, Natyhelem.** *Algoritmos Genéticos.* s.l. : Universidad Nacional de Colombia, 2006.

19. **Giraldo Gómez, Gloria Lucia.** *Ingeniería de software clase 5, Actores y sus roles. Modelo de Dominio.* s.l. : Escuela de Sistemas. Universidad Nacional de Colombia, 2006.

20. **Glover, Fred.** *Future paths for integer programming and links to artificial intelligence.* 1986. págs. 533-549. Vol. 13.

21. **Goldberg, David E.** *Genetic algorithms in search, optimization, and machine learning.* 1989.

22. **González, Carlos D.** [En línea] http://www.usabilidadweb.com.ar/bases_prof_exp.php.

23. **Grupo de usuarios PostgreSQL de Argentina;.** ArPUG. [En línea] 18 de Junio de 2012. http://www.arpug.com.ar/trac/wiki/PgAdmin.

24. **Herrera, Francisco.** *Introducción a los Algoritmos Metaheurísticos.* Granada (España) : Dpto. Ciencias de la Computación e I.A., 2009. 18071.

25. **Hugo y Serrano Cuayahuitl, Victor.** *Pruebas de unidad, Estándares de codificación y generación automática de código.* Universidad Autónoma de Tlaxcala. s.l. : Facultad de Ciencias básicas, ingeniería y tecnología.

26. **Hurtado Bustamante, Diana Paola.** *Patrones Grasp.* s.l. : Universidad del Valle: s.n., 2011.

27. **Ibrahim H., Osman y Kelly, James P.** *Meta-Heuristics: An Overview.* 1996.

28. **Joskowicz, Ing. José.** *Reglas y Prácticas en eXtreme Programming.* 2008.

29. **Juristo, Natalia, Moreno, Ana M. y Vegas, Sira.** *Técnicas de evaluación de software.* 2006.

30. **Kirkpatrick, Scott, Gelatt, D. y Vecchi, Mario P.** *Optimization by simulated annealing.* s.l. : Science, 1983. págs. 671-680. Vol. 220. 4598.

31. **Laman, Craig.** *Introducción al análisis y diseño orientado a objeto.* México: s.n. : PRENTICE HALL, 1999. 970-17-0261.

32. **Letelier, Patricio y Pnadés, María del Carmen.** *Metodologías ágiles para el desarrollo de software, Extreme Programming (XP).* sn. 2006. Vol. 5.

33. **Llorente, César de la Torre.** *Guía de Arquitectura N-Capas orientada al dominio con .NET.* 2010. 978-84-936696-8.

34. **Lockhart, Thomas.** *Tutorial de PostgreSQL.* 1998.

35. **Malforá, Dayvis y otros.** *Testing en eXtreme Programing.* 2006.

36. **Melián , Belén, Moreno, José A. y Pérez, J. Marcos.** *Metaheurísticas: una visión global.* 2003. págs. 7-28. ISSN: 1137-3601.

37. **NetBeans.** Sitio Oficial del NetBeans. [En línea] 8 de Diciembre de 2009. https://netbeans.org/community/releases/68/relnotes_es.html.

38. **Palma, Heisler.** Desarrollo Web. [En línea] 28 de Agosto de 2012. http://www.desarrolloweb.com/articulos/intro-indices-mysql.html.

39. **Pérez, Beatriz.** *ProTest-Proceso de Testing Funcional.* Puebla, Mexico : s.n., 2006. 970-94770-0-5.

40. **Pérez Valdés, Damian.** [En línea] 27 de Octubre de 2007. http://www.maestrosdelweb.com/editorial/%C2%BFque-son-las-bases-de-datos/.

41. **Pose, Marcos Gestal.** *Introducción a los Algoritmos Genéticos.* s.l. : Departamento de Tecnologías de la Información y las Comunicaciones Universidad de Coruña, 2000.

42. **PostgreSQL.** Sitio Oficial de PostgreSQL. [En línea] 2010. http://www.postgresql.org.es/sobre_postgresql.

43. **Pressman, Roger S.** *Ingeniería del Software: Un enfoque práctico.* Séptima.

44. **Rodríguez Guadarrama, Addiel y Mazorra, Thaymí.** *Análisis y diseño de una herramienta web para la gestión de la información.* Habana : Universidad de las Ciencias Informáticas (UCI), 2010.

45. **Rodríguez-Piñero, Piedad Tolmos.** *Introducción a los algoritmos genéticos y sus aplicaciones.* s.l. : Universidad Rey Juan Carlos, Servicio de Publicaciones, 2003.

46. **Sharma, Neeraj, y otros, y otros.** *Database Fundamentals.* s.l. : IBM Corporation, 2010.

47. **Stützle, Thomas G.** Local Search Algorithms for Combinatorial Problems. *Darmstadt University of Technology PhD Thesis.* 1998.

48. **Suárez, Pablo y Fontela, Carlos.** *Documentación y pruebas ante el paradigma de objetos.* 2003.

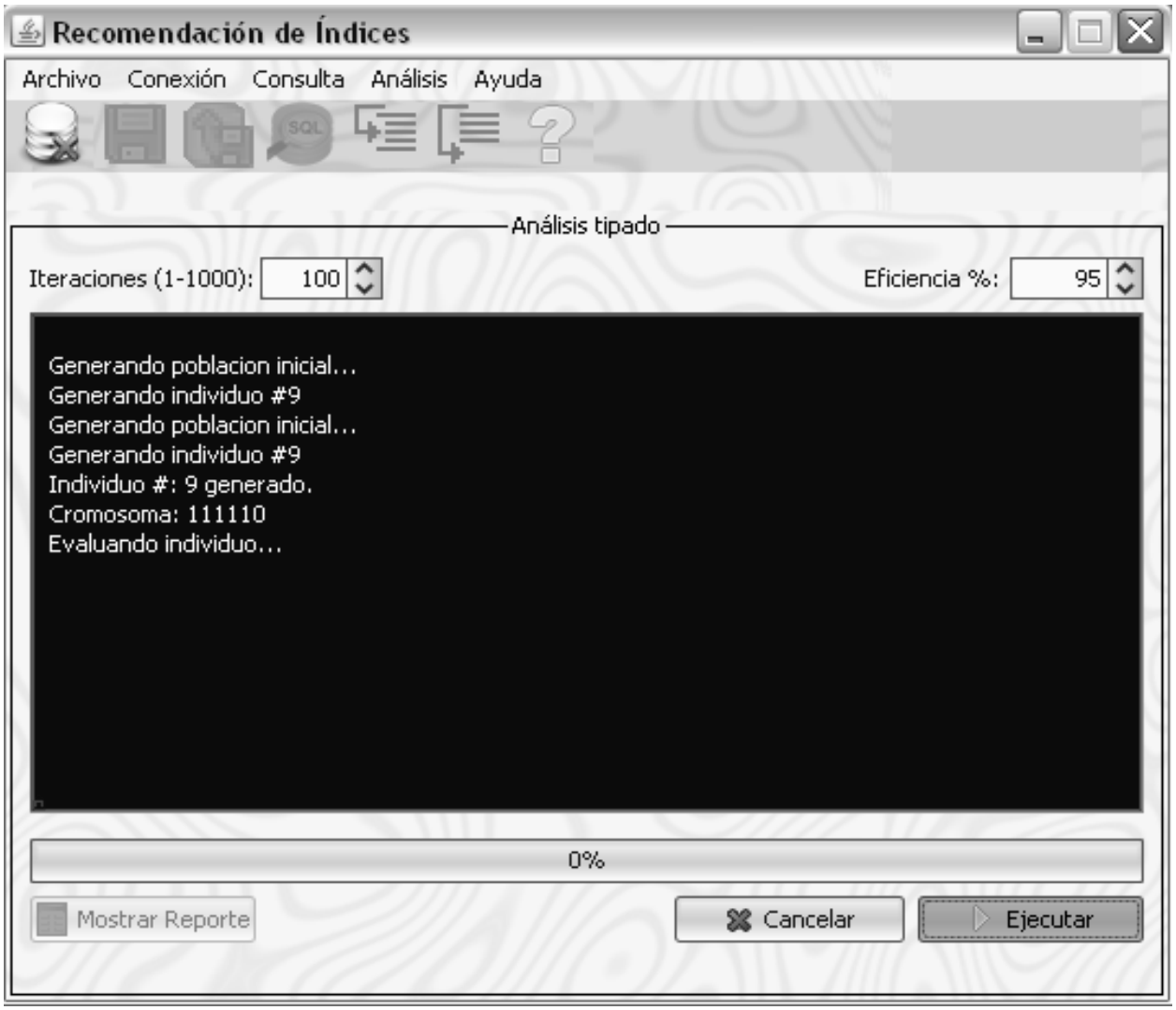
49. **Valverde Martínez, David María.** Blog de David Valverde. *Introducción a la Programación Extrema (XP).* [En línea] 6 de Septiembre de 2007. http://www.davidvalverde.com/blog/introduccion-a-la-programacion-extrema-xp/.

50. **Vargas Del Valle, Ricardo J.** *Programación en Capas.* 2010.

51. **Wolpert, David H. y Macready, William G.** *No free lunch theorems for optimization.* 1997. págs. 67-82. Vol. 1.

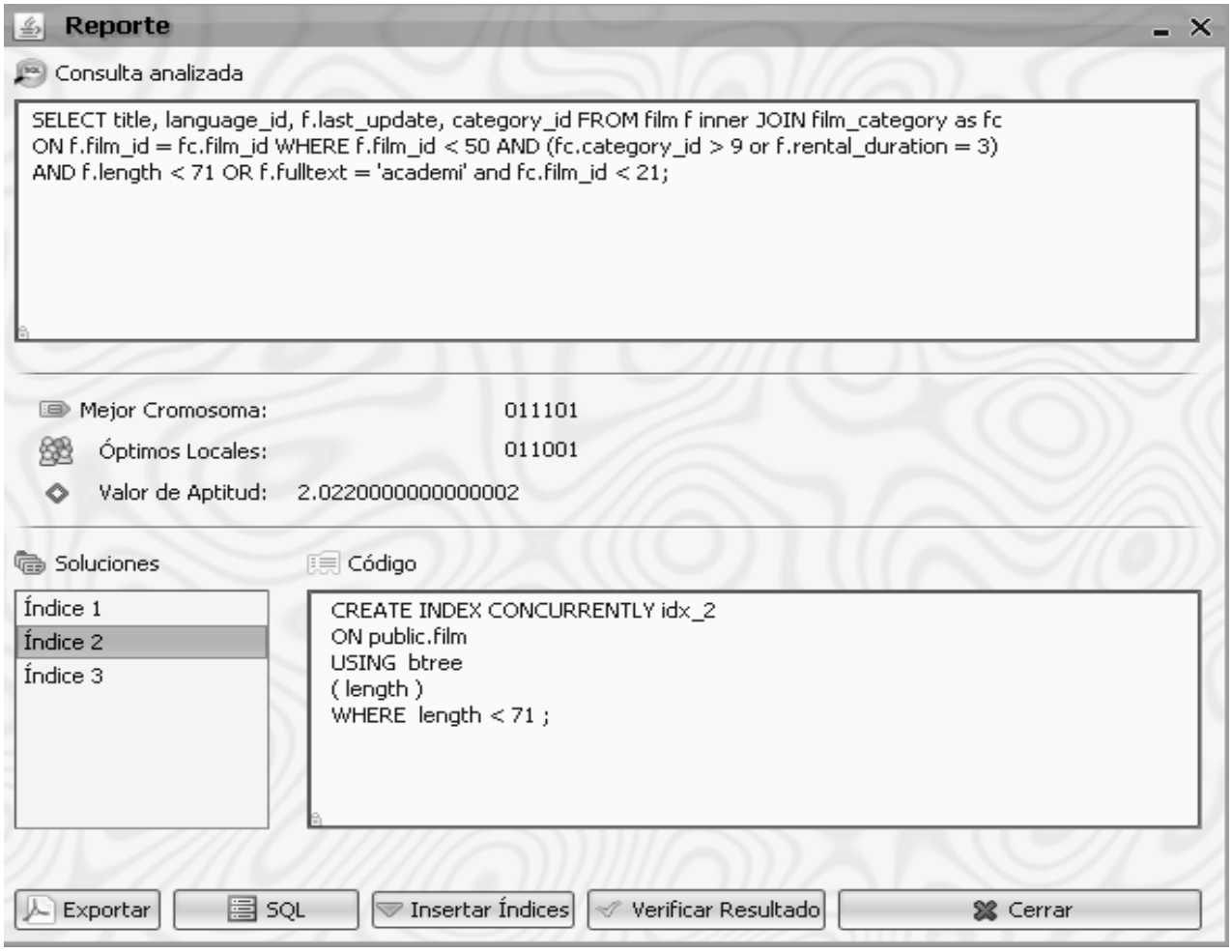
ANEXOS

Anexo 1 Ejemplo de ejecución de AG para índices tipados.



Se ejecuta el algoritmo genético para índices tipados después de haber introducido los datos correspondientes al número de iteraciones y el coeficiente de eficiencia asociado.

Anexo 2. Mostrar reporte de la solución obtenida.



Se muestra un reporte con la solución obtenida después de haber sido ejecutado el algoritmo genético. El valor "Mejor cromosoma" muestra la mejor combinación de índices obtenida al finalizar la ejecución. El valor "Óptimos Locales" representa la posición de los índices en la cadena de mejor cromosoma que realmente están siendo utilizados por el planificador de consultas de PostgreSQL. El "Valor de aptitud" muestra la diferencia entre el tiempo de ejecución de la consulta sin índices y la consulta utilizando la combinación de índices recomendada. Por último se muestra el código de los índices que se recomiendan crear, seleccionados a través de los valores antes expuestos.

Anexo 3. Acta de aceptación del producto.

UCI
Universidad de las Ciencias Informáticas

Centro de Tecnologías de Gestión de Datos
DATEC

La Habana, 7 de Junio del 2013
"Año 54 de la Revolución".

ACTA DE ACEPTACIÓN

De una parte, el Centro de Tecnologías de Gestión de Datos, en lo sucesivo DATEC, de la Universidad de las Ciencias Informáticas, representado en este acto por: MsC Anthony Rafael Sotolongo León, y de **otra parte** los estudiantes: Jorge Deimier Sosa Lastres y José Manuel Santana Hechavarría.

Primero: Que en cumplimiento de los requisitos funcionales han sido efectuadas las implementaciones correspondientes.

CONSIDERANDO: Que los hitos realizados han sido desarrollados con la calidad requerida y bajo las condiciones pactadas y aprobadas por **Las Partes**.

CONSIDERANDO: Que los hitos que se han ejecutado cumplen con los requerimientos establecidos.

POR TANTO: Las Partes acuerdan formalizar mediante la presente Acta, la aceptación del producto: Herramienta para la recomendación de índices en PostgreSQL

Y para que así conste, se extiende la presenta **Acta** en dos (2) ejemplares, rubricados por **Las Partes**.

Jorge Deimier Sosa Lastres

José Manuel Santana Hechavarría

Entregan

MSc. Anthony Rafael Sotolongo León

UCI Universidad de las Ciencias Informáticas Decanato Facultad 6

Recibe

GLOSARIO DE TÉRMINOS

Indices tipados: En el presente trabajo se denominan índices tipados los índices a los cuales se les define un tipo específico (Btree, Hash, Gist, o Gin).

Indices no tipados: En el presente trabajo se denominan índices no tipados los índices a los cuales no se les define un tipo específico, estos asumen por defecto Btree.

Análisis tipado: Consiste en la ejecución del AG para los índices tipados.

Análisis no tipado: Consiste en la ejecución del AG para los índices no tipados.