Universidad de las Ciencias Informáticas

Facultad 6

Plugin diseñador de bases de datos v2.0 para la herramienta de administración de bases de datos HABD.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autores: Wilber Turruella Velázquez.

Raúl Marinas Quesada.

Tutor: Ing. Flavio E. Roche Rodríguez.

La Habana 2013

"Año 55 de la Revolución"

"Creer en los jóvenes es ver en ellos además de entusiasmo, capacidad; además de energía, responsabilidad; además de juventud, pureza, heroísmo, carácter, voluntad, amor a la patria ¡fe en la patria !, ¡amor a la Revolución, fe en la Revolución, confianza en sí mismo !, convicción profunda de que la juventud puede, que la juventud es capaz, convicción profunda de que sobre los hombros de la juventud se pueden depositar grandes tareas".

Fidel Castro Ruz.

DECLARACIÓN DE AUTORÍA

Declaro ser autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año_____.

_____	_____
Wilber Turruella Velázquez	Raúl Marinas Quesada
Firma del Autor	Firma del Autor

_____

Ing. Flavio Enrique Roche Rodríguez

Firma del tutor.

Datos de contacto

Tutor: Ing. Flavio Enrique Roche Rodríguez.

Universidad de las Ciencias Informáticas, La Habana, Cuba

Categoría docente:

Años de experiencia en el tema: 2.

Años de graduado: 2.

e-mail: feroche@uci.cu

De Wilber:

Agradecerle a mis padres por darme la vida y por la crianza que me han dado. A dios por darme la posibilidad de existir y por guiar mis pasos. Agradecerle a mi sol, a mi madre por estar en todos los momentos de mi vida, por luchar junto a mi y por brindarme todo su amor, hoy vemos el fruto de tanto sacrificio que hemos vivido juntos, gracias mamita. A mi padre, estés donde estés, sé que estás orgulloso de mi, gracias Bilin. A mis abuelitos Elvia y Ramón, sin ustedes no soy nadie, gracias por tanto amor y por formar nuestra familia, los amo. A mis hermanos Gume y Yilber por siempre comprenderme, cuidarme, por respetarme a pesar de ser el menor, simplemente gracias. A todos mis tios, Juaquin, por ser el que más admiro de todos, gracias Juaqui por estar siempre al tanto de mi vida. A todas mis tias. Agradecerles también a mis primos Alianet, Aimé, Falli, Yeidis, Mika, Damaris, Denia, Yelenis, Eduardo, Adriana, Eric, Adrián. Pipi y Ania.

Mis agradecimientos a mis estrellitas queridas, a los que he visto nacer, crecer y que tan feliz me hacen, mis sobrinos Solangel, Soraida y el pequeño William, gracias por alegrar mi vida. A todas las amistades que he formado en estos años en la universidad especialmente Manuel, Soto, Dalied, Ortiz, Ali, Beatriz, Carlos Julio y tata Yennia. Gracias Yenisey y Yanier por permitirme formar parte de sus vidas y escogerme para ser el padrino del fruto de su amor, la pequeña Dalila. A la tia Nury, por ayudarme en los momentos más difíciles que he vivido en la UCI, muchas gracias mi tia. A Pichardo por ser mi segunda madre y darme todo su amor. A mi amigo Angel por soportarme en estos años y por siempre brindarme su mano. A la familia Utria, especialmente Odalis, Odairis y Negra por toda la dulzura y amor que me entregaron siempre, las llevo en mi corazón. Agradecerle a una persona muy especial en mi vida, a mi compañero Adrián por siempre estar cuando más lo he necesitado y por su ayuda incondicional. A mi tutor Flavio por toda su ayuda y por confiar siempre en mí. A la profe Vilmavis por su comprensión y confianza, gracias profe. A todos y todas y aquellas persona que mencioné mil gracias.

De Raúl:

La realización del presente trabajo diploma implicó mucho esfuerzo y dedicación, no hubiera sido posible sin el apoyo un grupo de personas que día tras día nos dieron ánimo para seguir adelante.

Agradezco en primer lugar a mis padres. A ellos, gracias, por entregarlo todo para construir mi futuro, por ser mis guías de toda la vida, por enseñarme que con trabajo y esfuerzo podemos lograr todas nuestras metas, por educarme de la forma que lo han hecho, por darme su amor y su cariño, en fin por ser la razón que me impulsa a seguir adelante. Gracias a mi hermano por siempre estar apoyándome en todos mis pasos, por su entrega en todo momento. A mi abuela, mi otra madre, por su paciencia y entrega, por depositar en mí toda la confianza y el amor que un ser humano puede llevar consigo, por estar presente en las buenas y las malas y gracias por existir.

Agradezco a mi dúo de tesis por el esfuerzo realizado para lograr que este trabajo saliera adelante.

Gracias a mis tíos y primos, por ayudarme en todo momento, por darme aliento para seguir adelante. A mis amigos por apoyarme cuando lo he necesitado, por creer en mí y por estar siempre al tanto de la marcha del trabajo, por todo su interés en verme convertido en ingeniero. A mi tutor por todo el apoyo y dedicación a este trabajo, por todo el esfuerzo realizado día a día, por los señalamientos y recomendaciones.

Agradezco a todos aquellos que de una forma u otra hicieron posible mi llegada al final de este largo y complicado camino que ha sido esta carrera, a todos los profesores que tuve, a mis compañeros de todos los apartamentos por los que he pasado, a mis compañeros de brigada, los nuevos y los viejos, a las amistades que he hecho durante la carrera y a mis compañeros del proyecto, sin su apoyo en estos cinco años, sin su ayuda, mucha o poca, no hubiese sido posible llegar hasta aquí..

De Wilber

Dedico especialmente este Trabajo de Diploma a mi mamita Mariluz y a mis abuelos Elvia y Ramón.

A la memoria de personas especiales que pasaron por mi vida y hoy ya no están:

Primeramente a mimi (Mirellita) por ser esa estrellita que me guía.

A mis abuelos Elda y Williams.

A mi amigo Maikel Luis Fernández que siempre pidió me graduara por él, pues sabía que no podría contra su enfermedad, hoy lo hago por los dos.

Por último y no menos importante sino todo lo contario a mi padre William que la vida me lo arrebató cuando más lo necesitaba pero sé que siempre está conmigo.

De Raúl:

Dedico este trabajo a mis padres, a mi abuela y a mi hermano, a quienes jamás encontraré la forma de agradecer el amor, aliento, comprensión y apoyo brindado en los momentos buenos y malos de mi vida, este logro es dedicado a ellos y quiero que sepan que mis esfuerzos son inspirados en cada uno de ustedes.

Gracias por demostrarme que todo en la vida se puede lograr.

Gracias por ser parte de lo que más amo.

Gracias por estar en mi vida.

….Gracias….

Resumen

El constante desarrollo de la industria informática trae consigo que el usuario requiera de aplicaciones que hagan las acciones más sencillas, con el menor costo de tiempo y esfuerzo posible. En la actualidad las empresas dedicadas a la producción de *software* en nuestro país tienen entre sus principales objetivos desarrollar productos y servicios informáticos de alta calidad. Para alcanzar estos resultados, lograr un lugar y un reconocimiento en el mercado mundial, numerosas empresas y entidades cubanas enmarcan su trabajo en la implementación de herramientas que ayuden a facilitar el trabajo por parte de los usuarios. Una de estas instituciones es la Universidad de las Ciencias Informáticas (UCI), donde se desarrollan aplicaciones netamente libres, ejemplo de ello es la Herramienta de Administración de Bases de Datos HABD, que fue creada en el curso 2010-2011 en el departamento PostgreSQL, perteneciente al Centro de Tecnologías de Gestión de Datos. Dicha herramienta permite la incorporación de numerosas funcionalidades gracias a la arquitectura que posee. En el curso 2011-2012 se le agregó la funcionalidad que permite realizar el diseño gráfico de las bases de datos para el gestor PostgreSQL. Con el objetivo de explotar las potencialidades de HABD y que la misma preste un mayor número de funcionalidades, se decidió en el presente trabajo de diploma realizar una versión 2.0 del *plugin* que permita efectuar el diseño para los gestores MySQL, SQLite y PostgreSQL.

Palabras clave: bases de datos, gestor de bases de datos, HABD, herramienta de administración de bases de datos, *plugin*.

Índice

Índice de Figuras

Índice de Tablas

Introducción

Dentro del sector tecnológico, los sistemas de información se han convertido en un eslabón fundamental en numerosas empresas. Estos han evolucionado aceleradamente por la necesidad de almacenar grandes cantidades de información de forma fácil, rápida y segura. Los grandes cúmulos de datos que existen en la actualidad no pueden ser procesados por el hombre manualmente debido a la complejidad que estos requieren, surgiendo de esta forma lo que hoy se conoce como bases de datos.

Debido a la gran evolución de las base de datos, estas tienen mayor calidad, flexibilidad y manejo desde la creación de los sistemas gestores de base de datos (SGBD). De esta forma, la gestión de datos se libró de los grandes ordenadores centrales, para así distribuirse según las necesidades de los usuarios, permitiéndole crear y mantener sus bases de datos, utilizando herramientas para transformar lo lógico en conjuntos de datos.

A través de los años el desarrollo de los SGBD lo han tenido importantes compañías como ORACLE, Informix, entre otras, pero a veces se hace un poco difícil su obtención al ser empresas privadas, lo que trae consigo que se dificulte acceder a sus productos.

Debido a esta situación, Cuba encamina su trabajo en el desarrollo de nuevas alternativas que permitan obtener resultados dentro de la industria informática. Se desarrollan aplicaciones informáticas para resolver los conflictos existentes en las diferentes ramas de la sociedad, además se ha creado organismos y entidades, con el objetivo de dar soluciones a dichos problemas y lograr un mayor desarrollo tecnológico.

Una de las instituciones enmarcadas en el desarrollo de soluciones informáticas, tanto para empresas nacionales como internacionales, es la Universidad de las Ciencias Informáticas (UCI), un centro enfocado en alcanzar un lugar cimero en la industria informática a nivel mundial. La universidad está formada por varios centros de desarrollo de *software*. Uno de estos es el Centro de Tecnologías de Gestión de Datos (DATEC), el cual a su vez está compuesto por varios departamentos, siendo PostgreSQL uno de ellos. En dicho departamento se desarrollan aplicaciones completamente libres que facilitan el trabajo con los principales gestores de bases de datos que existen.

La herramienta de administración de bases de datos HABD fue creada en dicho departamento, la cual surge a partir de las investigaciones realizadas sobre los inconvenientes encontrados en herramientas similares que existen en el mundo. HABD posee una arquitectura enfocada a *plugin* permitiendo agregar un gran número de funcionalidades, evitando así que los usuarios necesiten de varias aplicaciones para resolver un único problema.

En el año 2012, se desarrolló el *plugin* “Diseñador de bases de datos” para la herramienta HABD, el cual permite llevar a cabo la confección de los modelos de datos para bases de datos PostgreSQL. El desarrollo de este *plugin* permite a los usuarios hacer un mayor uso de la herramienta HABD. La integración de este *plugin* a HABD ha facilitado a los usuarios su trabajo en el diseño de las bases de datos, permitiendo disminuir la redundancia de los datos.

Sin embargo, el mismo tiene como principal inconveniente su imposibilidad de exportar el modelo de datos de otros gestores como MySQL y SQLite; lo cual limita su utilización solo al gestor de base de datos PostgreSQL. Por estas limitantes se obliga a los usuarios que usan HABD a utilizar más de una aplicación informática para llevar a cabo el diseño de bases de datos en estos gestores.

Por la situación problemática antes planteada surge el siguiente **problema a resolver**: ¿Cómo diseñar bases de datos relacionales en la herramienta de administración de bases de datos HABD para los gestores PostgreSQL, MySQL y SQLite?

Con el desarrollo de esta investigación se pretende dar solución al problema antes mencionado. Para alcanzar estos resultados se plantea como **objeto de estudio:** el proceso de modelado de bases de datos relacionales, enmarcado en el **campo de acción** herramientas para el diseño de bases de datos.

Por la problemática antes expuesta y con el objetivo de dar solución a la interrogante surgida se define como **objetivo general** del trabajo: Desarrollar un *plugin* para la herramienta de administración de bases de datos HABD, que permita el diseño de bases de datos para los gestores PostgreSQL, MySQL y SQLite lo cual permitirá extender sus potencialidades. Para dar cumplimiento al objetivo general se desglosan los siguientes **objetivos específicos**:

1. Definir el marco teórico a partir de un estudio del estado del arte alrededor del proceso y herramientas para el modelado de bases de datos relacionales.

2. Analizar los diseñadores de bases de datos existentes en el mundo.
3. Diseñar el *plugin* de bases de datos v2.0 para la herramienta HABD.
4. Implementar las funcionalidades del *plugin* diseñador de bases de datos v2.0 para los gestores PostgreSQL, MySQL y SQLite.
5. Probar el *plugin* diseñador de bases de datos v2.0 para la herramienta HABD.

Para dar cumplimiento a los objetivos antes mencionados se definen las siguientes **tareas de la investigación**:

1. Revisión bibliográfica de herramientas para el diseño de bases de datos existentes en el mundo.
2. Identificación y descripción de las funcionalidades del *plugin*.
3. Definición de la nueva arquitectura del *plugin.*
4. Elaboración del modelo de diseño del *plugin.*
5. Implementación de las funcionalidades previas del diseñador de bases de datos en MySQL y SQLite.
6. Implementación de los tipos de datos para PostgreSQL, MySQL y SQLite.
7. Implementación de las restricciones para PostgreSQL, MySQL y SQLite.
8. Implementación de la creación de índices para PostgreSQL, MySQL y SQLite.
9. Implementación de la creación de vistas para PostgreSQL, MySQL y SQLite.
10. Identificación de los métodos de prueba para la validación del *plugin.*
11. Elaboración de los casos de pruebas basados en Historias de Usuario.

Para el desarrollo del presente trabajo se hizo uso de los siguientes métodos científicos:

Métodos teóricos:

Histórico-Lógico: Posibilitó efectuar un análisis histórico sobre la evolución de los diseñadores de bases de datos desde sus inicios hasta la actualidad.

Analítico-Sintético: Se utilizó para analizar y sintetizar toda la información adquirida en la investigación sobre los diseñadores de bases de datos, seleccionar los conceptos, las definiciones y las teorías más importantes relacionadas con el tema.

Métodos empíricos:

Observación: Con este método se efectuó un seguimiento de la evolución de los diseñadores de bases de datos.

La presente investigación está dividida en: Resumen, Introducción, tres Capítulos, Conclusiones, Recomendaciones, Bibliografía referenciada, Bibliografía consultada y Anexos.

Capítulo 1: "Fundamento teórico", se presenta el marco teórico conceptual, con la recopilación de los principales conceptos y definiciones que sustentan la investigación. Se describen las herramientas y tecnologías a utilizar en el desarrollo de la solución.

Capítulo 2: "Características del sistema propuesto", se describe la propuesta de solución. Se realiza el modelo de dominio. Se describen la lista de reserva del producto, historias de usuarios, tareas de ingeniería, patrones, estándares de código e interfaces de la aplicación.

Capítulo 3: "Validación y pruebas", se detallan las diferentes pruebas realizadas para validar el correcto funcionamiento del *plugin* desarrollado.

CAPÍTULO 1

FUNDAMENTO TEÓRIC0

Introducción

En el presente capítulo se abordan los conceptos referentes al dominio del problema, los cuales son imprescindibles para comprender el objetivo y el alcance del trabajo. Se efectúa una revisión bibliográfica con relación a las principales herramientas de diseño de bases de datos existentes y sus principales características. Además se selecciona la metodología a utilizar para la solución y se realiza la descripción de las características de cada una de las herramientas utilizadas para el desarrollo de la solución propuesta, proporcionando fundamentos que justifican su selección.

1.1. Conceptos asociados al objeto de estudio

En el presente epígrafe se realiza una explicación de los conceptos fundamentales asociados al objeto de estudio y al campo de acción de la investigación, con el fin de obtener una mayor comprensión del tema, logrando una solución que cumpla con los objetivos trazados.

1.1.1. Bases de Datos

Según Matos [1] es un conjunto de datos interrelacionados entre sí, almacenados con carácter más o menos permanente en la computadora. O sea, que una base de datos puede considerarse una colección de datos variables en el tiempo.

Sin embargo para Date [2] una base de datos es un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada.

De acuerdo a las definiciones expuestas anteriormente, los autores de la presente investigación asumirán que una base de datos es un conjunto de datos estructurados apropiadamente y relacionados entre sí

cuya finalidad general es almacenar información y permitir a los usuarios recuperar y actualizar esa información basada en peticiones. Las bases de datos no solamente contienen datos, sino que también cuentan con una descripción de la misma.

1.1.2. Sistema Gestor de Bases de Datos (SGBD)

Por el constante crecimiento de los datos y la forma concurrente de acceso a estos por parte de los usuarios, surgen los sistemas gestores de bases de datos (SGBD), con el objetivo de lograr un mayor manejo y administración de los mismos.

Según Sicilia [3] un SGBD es entendido como un *software* que proporciona servicios para la creación, el almacenamiento, el procesamiento y la consulta de la información almacenada en bases de datos de forma segura y eficiente. Un SGBD actúa como un intermediario entre las aplicaciones y los datos.

En la actualidad debido al desarrollo tecnológico y la creciente competencia en el mercado, han surgido numerosos SGBD, los cuales son libres y propietarios, a continuación se muestran ejemplos de cada uno de ellos:

Algunos SGBD propietarios

- Oracle [4].
- DB2 [5].
- InterBase [6].
- SQL Server [7].
- dBase [8].

Algunos SGBD libres

- PostgreSQL [9].
- FireBird [10].
- SQLite [11].
- MySQL [12].

En la actualidad dos de los sistemas gestores de bases de datos y explotados por sus potentes características son MySQL y SQLite. A continuación se hace referencia a la investigación realizada sobre las principales propiedades de estos:

MySQL

MySQL es un Sistema de Gestión de Bases de Datos relacional, una idea originaria de la empresa MySQL AB establecida inicialmente en Suecia en 1995 y cuyos fundadores son David Axmark, Allan Larsson, y Michael "Monty" Widenius.

MySQL se usa para realizar proyectos sencillos, donde no se requiere almacenar gran cantidad de información, ha demostrado ser muy rápido y eficiente. Actualmente posee licencia propietaria debido a que fue comprada por la SunMicrosystems, las versiones de MySQL que hayan sido liberadas en fechas anteriores a esa adquisición, serán consideradas con licencia libre. Esto supone una gran ventaja si se tiene en cuenta los precios que hay que pagar para obtener un gestor de bases de datos como Oracle o SQL Server. Otra de las ventajas es que el mismo es multiplataforma, por lo cual funciona lo mismo en servidores con sistema operativo Linux, Unix y Windows.

Las principales características de este gestor de bases de datos son:

- Aprovecha la potencia de sistemas multiprocesadores, gracias a su implementación multihilo.
- Soporta gran cantidad de tipos de datos para las columnas.
- Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP, entre otros).
- Gran portabilidad entre sistemas.
- Soporta hasta 32 índices por tabla.
- Gestión de usuarios y contraseñas, manteniendo un buen nivel de seguridad en los datos.
- Escrito en C y en C++.
- Proporciona sistemas de almacenamiento transaccional y no transaccional.

MySQL surgió como una necesidad de un grupo de personas sobre un gestor de bases de datos rápido, por lo que sus desarrolladores fueron implementando únicamente lo que precisaban, intentando hacerlo funcionar de forma óptima [13].

SQLite

SQLite es un pequeño pero potente sistema gestor de bases de datos, como MySQL, Oracle, dBase o FileMaker. Las principales ventajas respecto a otros sistemas de gestión de bases de datos son:

- Es un proyecto de dominio público.
- Tamaño muy reducido, menos de 300Kb, que lo hace perfecto para ser utilizado junto con otras aplicaciones.
- Las bases de datos se guardan en forma de ficheros, por lo que es posible trasladar sin problemas una base de datos (o fichero) a cualquier dispositivo que tenga instalado SQLite.
- Es multiplataforma.
- Estabilidad. Después de más de 10 años de desarrollo, muchas aplicaciones como Firefox, OpenOffice o incluso Android confían en esta herramienta para gestionar sus datos [14].

SQLite es *Software* Libre, de código fuente de dominio público y licencia GPL. Una de las principales características por las que se destaca SQLite es por su completo soporte para tablas e índices en un único archivo por bases de datos, soporte transaccional, rapidez, escaso tamaño y su completa portabilidad.

Es muy rápido y la ventaja fundamental es que permite utilizar un amplio subconjunto del lenguaje estándar SQL. Combina el motor y la interfaz de la base de datos en una única biblioteca, y almacena los datos en un único archivo de texto plano facilitando la portabilidad de los datos, y solamente tiene la restricción del espacio de disco asignado al usuario en el servidor.

SQLite dispone de una completa interfaz orientada a objetos, con distintas funciones que facilitan la manipulación de datos. Funciones muy similares a las que se pueden manejar con MySQL.

1.1.3. Herramienta de administración de bases de datos (HABD)

Con el fin de evitar la utilización de varias herramientas de administración de bases de datos y facilitar el trabajo de los usuarios, en el departamento PostgreSQL se desarrolló en el curso 2010-2011 la herramienta HABD. Esta herramienta brinda una interfaz amigable y una arquitectura basada en *plugin*, permitiendo que una vez creado un componente para solucionar una tarea específica exista un bajo

acoplamiento o dependencia con otros componentes, de manera que permita realizar modificaciones al código sin necesidad de transformar y afectar el funcionamiento completo del sistema. Provee mayor calidad en el *software*, dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización. De esta forma se le pueden agregar gran número de funcionalidades evitando que los usuarios necesiten de varias aplicaciones para resolver un único problema. En la actualidad la herramienta HABD cuenta con un total de 9 *plugins*, los cuales pueden trabajar de forma simultánea o de forma independiente de acuerdo a las necesidades del usuario que esté utilizando la aplicación, a continuación se mencionan cada uno de los *plugins*:

- Editor de Consultas.
- Diseñador de Base de Datos.
- Editor de Funciones y Disparadores.
- Monitorización.
- Recuperación de entidades.
- CRUD-PG[1].
- Particionado de Tablas.
- Normalización.
- Generador de Datos.

1.1.4. *Plugin*

Debido al constante desarrollo, muchas aplicaciones ya existentes carecen de funcionalidades de gran importancia, con el fin de mantener el correcto funcionamiento y aprovechamiento de las mismas, se crean complementos que amplían las funcionalidades de estas, para poder lograrlo han surgido los *plugins*.

[1]Herramienta para generar funciones automáticas mediantes procedimientos almacenados o funciones programadas en diferentes lenguajes como son PL/pgSQL y SQL.

El Dr. Harvey M. Deitel los define como: Pequeñas aplicaciones que añaden alguna función a otros programas, habitualmente de mayor tamaño. Un programa puede tener uno o más *plugin*. Son muy utilizados en los programas navegadores para ampliar sus funcionalidades [15].

Otro de los conceptos de *plugin* más utilizados es el definido por escritor Paul J que los define como módulos de *hardware* o *software* que añaden características o servicios específicos a un sistema más grande [16].

Según los conceptos vistos anteriormente, se puede definir que los *plugins* son complementos que se crean para brindar beneficios a los desarrolladores que trabajan con aplicaciones, para de esta forma ampliar sus capacidades.

El uso de *plugins* se ha convertido en una gran tendencia. Muchas empresas productoras de *software* crean aplicaciones basadas en esta tecnología aprovechando sus ventajas, tales como:

- Bajo acoplamiento: Permite modificar el código sin afectar el funcionamiento de otros componentes ni del sistema en general.
- Simplificación de pruebas: Es posible probar cada *plugin* o funcionalidad nueva antes de probar el sistema.
- Mejoras continuas: Aún terminado y desplegado el *software* es posible seguir trabajando en el mismo agregando nuevas funcionalidades.

1.1.5. Diseñadores de bases de datos

Un diseñador de bases de datos es una herramienta que permite realizar el diseño gráfico visualmente de los diagramas Entidad Relación de las bases de datos. Este proceso tiene como ventaja para el usuario poder realizar el diseño de la bases de datos de forma más rápida y sencilla, evitando el menor margen de error posible. Otro de los procesos que se lleva a cabo en la generación del modelo de datos de la bases de datos, es una separación entre los modelos físicos y lógicos, manteniendo entre estos una total integración.

Durante mucho tiempo, el diseño de bases de datos fue considerado una tarea para expertos. Sin embargo, se ha progresado mucho en este aspecto y se considera una disciplina estable, con métodos y técnicas propias. Una de las tareas más importantes a la hora de la creación de una aplicación web es la construcción de las bases de datos. Existen dos factores importantes a tener en cuenta para ello: la selección de un adecuado sistema gestor de bases de datos y un buen diseño de la misma, elementos de

vital importancia ya que se debe evitar que falle alguno de ellos, pues puede traer consigo que la aplicación de muchos problemas durante el transcurso de su vida.

A continuación se hace referencia a la investigación realizada sobre los principales diseñadores de bases de datos embebidos o no en las herramientas de administración de bases de datos más importantes que existen en la actualidad.

Diseñadores de bases de datos embebidos dentro de herramientas de administración de bases de datos

PgAccess

PgAccess cuenta con una interfaz gráfica para diferentes gestores de bases de datos tales como MySQL, SQLite, SQL Server, Oracle y PostgreSQL permitiendo a los usuarios realizar la gestión de las tablas, editarlas, definir consultas, secuencias y funciones. El diseñador de esta herramienta ofrece la posibilidad de ejecutar VACUUM y recuperar información sobre las tablas. La principal desventaja que tiene esta herramienta es que se encuentra desactualizada y ya no recibe ningún tipo de soporte, por lo que no resulta factible utilizarla para realizar el proceso de diseño [17].

PgAdmin

PgAdmin es una potente herramienta de administración de bases de datos que tiene como principal características que cada vez que se realiza alguna modificación en un objeto, escribe la(s) sentencia(s) SQL correspondiente(s), lo que hace que, además de una herramienta muy útil, sea a la vez didáctica. También incorpora funcionalidades para realizar consultas, examinar su ejecución [como el comando *explain*[2]] y trabajar con los datos. A pesar de sus bondades cuenta con la desventaja que no posee un diseñador de bases de datos gráfico que posibilite realizar el proceso de producción de un modelo de datos de las bases de datos, imposibilitándose su uso para dar solución al problema propuesto [18].

[2] Permite examinar la ejecución de la consulta y trabajar con los datos

Diseñadores de bases de datos que no están embebidos en herramientas de administración

Visual Paradigm

Visual Paradigm es una herramienta CASE que permite realizar el diseño de las bases de datos de una forma dinámica, realizar el diseño de sus tablas y llenar las mismas con sus atributos, brinda la posibilidad de relacionar las tablas entre sí, los diagramas obtenidos son exportados en un script. Permite las relaciones de uno a mucho, uno a uno y mucho a mucho. Aunque todo apunta que esta herramienta es muy potente para realizar el diseño de las bases de datos cuenta con la desventaja que no es posible realizar la herencia entre tablas en gestores que la soporten, además no utiliza tipos de datos como XML y Moneda, correspondientes al gestor PostgreSQL. Otra desventaja con la que cuenta esta herramienta es que es privativa y aunque la universidad paga por su uso, no se puede estimar el tiempo que se pagará por el mismo, debido a esto no es posible su utilización para realizar el diseño de las bases de datos [19].

PgDesigner

PgDesigner es un programa de código abierto para las bases de datos de diseño gráfico en Oracle, DB2, PostgresSQL, HSQLDB, SQLite, SQLServer. El código fuente está escrito en el lenguaje Gambas, y actualmente sólo se ejecuta en sistema operativo Linux [20].

Es una potente herramienta que brinda la posibilidad de crear las relaciones entre las tablas visualmente de forma sencilla y rápido. Permite salvar el modelo realizado y cargar el script generado. Aunque es una excelente herramienta tiene como principal desventaja que no cuenta con una documentación sobre su instalación. Además necesita librerías Gambas, las cuales deben instalarse como paquetes adicionales para poderlo ejecutar, condición no factible para la solución del problema planteado.

ERWIN

Es una herramienta que se utiliza para el diseño de bases de datos que tiene como principal característica la realización de un diseño productivo, generación y mantenimiento en aplicaciones. Tiene la ventaja que automatiza el proceso de diseño de una forma inteligente.
Esta herramienta permite diseñar las bases de datos de alto desempeño .Separa los modelos físicos y lógicos. Posibilita realizar ingeniería inversa y hacia delante. Genera automáticamente tablas, vistas, índices, reglas de integridad referencial [llaves primarias, llaves foráneas], valores por defecto y

restricciones de campos y dominios. Soporta principalmente bases de datos relacionales SQL y bases de datos que incluyen Oracle, SQL Server, Sybase [21].
Esta herramienta podría utilizarse para realizar el diseño de bases de datos pero existe el inconveniente que es privativa, trayendo consigo que no se pueda utilizar para el diseño de las bases de datos en HABD.

ER/Studio

Es una herramienta de bases de datos que facilita diseñar, generar y mantener aplicaciones de bases de datos de calidad y alto rendimiento. Desde un modelo lógico de sus requerimientos de información y reglas del negocio que definen su bases de datos, hasta un modelo físico optimizado por las características específicas de su bases de datos de destino, ER/Studio le permite visualizar la estructura adecuada, los elementos clave y un diseño optimizado de su bases de datos.

Es una herramienta de modelado de datos fácil de usar y multinivel para el diseño de bases de datos a nivel físico y lógico. ER/Studio está equipado para crear y manejar diseños de bases de datos funcionales y confiables. Ofrece fuertes capacidades de construcción automática de bases de datos, documentación y fácil creación de reportes [22].

Esta herramienta tiene como principal desventaja que es privativa y no es multiplataforma, solo trabaja sobre los diferentes sistemas operativos de *Windows*.

***Plugin* diseñador de bases de datos para la herramienta de administración de bases de datos HABD**

En el año 2012 se implementó el *plugin* para diseñar bases de datos PostgreSQL, permite realizar los modelos de las bases de datos, la ejecución y generación de los script. El diseñador de bases de datos de la herramienta HABD brinda la posibilidad de realizar gráficamente el proceso completo de cada uno de los modelos que contienen los datos y tablas para una base de datos, esta herramienta permite realizar la gestión de las tablas dando la posibilidad de eliminarlas, crearlas, visualizarlas y modificarlas. Brinda la opción de insertar, eliminar, modificar y visualizar los atributos de las tablas, trabajar la herencia, especificando las restricciones necesarias. Maneja de forma dinámica el trabajo con los esquemas y tablespace. Una vez creado el modelo, el diseñador accede a exportarlo en formato mdl y cargarlo en caso de que sea necesario. Todas las características antes mencionadas evidencian que es perfecto

diseñador de bases de datos, pero el mismo tiene como desventaja que solo permite realizar el diseño de bases de datos PostgreSQL, por lo que se decide realizar una nueva versión del mismo que posibilite realizar el diseño además de PostgreSQL para los gestores MySQL y SQLite [23].

El estudio hecho anteriormente arrojó como conclusión que ninguna de las herramientas analizadas brindan la solución a la interrogante planteada ya que no pueden integrarse como *plugin* a la herramienta de administración de bases de datos HABD, que permita realizar el diseño de bases de datos para los gestores MySQL, PostgreSQL y SQLite. Es por ello que se decide realizar una herramienta que cumpla las expectativas del proyecto y que ayude a solucionar el problema planteado anteriormente. A pesar de no haber seleccionado ninguno de los diseñadores analizados se tomaron en cuenta características importantes para la realización del sistema como pueden ser: comprobar que no existan atributos de igual nombre entre dos tablas y la forma de insertar datos visualmente en una tabla, así como el tratamiento y manejo de los índices de estas.

1.1.6. Metodologías de Desarrollo de *Software*

Una metodología es un conjunto de pasos o acciones definidas para guiar un proceso determinado. En el caso del desarrollo del *software*, su definición no se aleja de este marco, a esto se le añade la definición de las etapas, los roles y las tareas para la generación de los artefactos finales de productos informáticos [24].

En correspondencia con el concepto anterior se puede definir que la metodología de desarrollo de *software* es un conjunto de técnicas, procedimientos, herramientas, y un soporte documental que ayuda al equipo de desarrollo a realizar un nuevo *software,* guiando su ciclo de vida, proceso de vital importancia para garantizar la satisfacción del cliente, lo que justifica su extendida utilización en la actualidad. Durante el proceso de guiar al desarrollador en la construcción del *software*, las metodologías definen artefactos, roles e indican detalladamente las actividades que se van a realizar mediante prácticas y técnicas recomendadas.

Estas se clasifican en dos grandes grupos debido a sus características, unas son ágiles (ligeras) y las otras son tradicionales (pesadas), las ágiles prestan mayor importancia en lograr que el producto de *software* se realice con la calidad requerida, mientras que en las tradicionales se enfocan más en definir

detalladamente los procesos, tareas y herramientas a utilizar, además requiere una extensa documentación, pretendiendo prever todo desde un inicio.

1.1.7. Metodología Extreme Programming (XP)

XP se clasifica dentro del grupo de las metodologías ágiles, la cual utiliza un enfoque orientado a objetos con el objetivo de lograr un producto en el menor tiempo posible, que sea fácil de usar y fiable. XP se basa en:

- Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos con el objetivo de detectar futuros errores.
- Re fabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo.
- Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

El uso de **XP** tiene muchas ventajas que permiten su empleo en el desarrollo del *plugin*, ya que está basada en realizar pruebas a los principales procesos con el objetivo de detectar futuros errores, posibilitando que los requisitos puedan ser modificados o cambiados, permitiéndole mayor adaptabilidad al producto, pues es especialmente adecuada para proyectos pequeños. Concibe que la propiedad del código es colectiva, cualquiera puede desarrollar, mejorar, simplificar, cualquier necesidad del proyecto, siempre usando sistemas tipo CVS (Controlador de versiones) para evitar la duplicación de trabajo. En esta metodología el cliente está presente en todo momento y colabora con el proyecto como un miembro más del mismo, existiendo de esta forma una comunicación efectiva en tiempo real entre el cliente y los desarrolladores, fortaleciendo el trabajo del proyecto en equipo [25].

Se decide trabajar con XP como metodología, la cual guiará el proceso de desarrollo del *software*, ya que la misma se adecua perfectamente a las características que presenta el proyecto, como son: requisitos muy cambiantes, existe un alto riesgo técnico, posee un equipo pequeño y poco tiempo de desarrollo.

1.2. Tecnologías y herramientas

Con el objetivo de lograr un buen desempeño en el desarrollo de la implementación del *plugin,* se realizó un estudio acerca de las tecnologías y herramientas existentes. Seguidamente se abordan elementos sobre las herramientas seleccionadas para su diseño e implementación.

1.2.1. Lenguaje de modelado

Con la evolución de la industria del *software* se ha hecho necesario desarrollar herramientas para el modelado de sistemas, que faciliten de esta forma el trabajo de analistas y diseñadores de *software*.

El lenguaje de modelado está formado por un conjunto de símbolos que permitirán modelar parte de un diseño de *software* orientado a objetos. Se utiliza extensivamente en combinación con una metodología de desarrollo para avanzar de una especificación inicial a un plan de implementación, mostrar y comunicar dicho plan a todo un equipo de desarrolladores. El uso de un lenguaje de modelado es más sencillo que la auténtica programación, pues existen menos medios para verificar efectivamente el funcionamiento adecuado del modelo.

Lenguaje unificado de modelado (Unified Modeling Language o UML)

UML es un lenguaje de modelado que se utiliza para definir un sistema, detallar los artefactos del mismo, documentar y construir. Es un lenguaje fácil y descriptivo, ya que permite modelar aplicaciones en cualquier dominio, contiene generación automática de código, ajustando su utilización para todas aquellas personas que no tengan conocimientos de lenguajes de programación. Su objetivo principal y más importante es resolver los requisitos guiados por casos de usos. Permite crear todos los tipos de diagramas de clases, código inverso y generar código desde diagramas. Incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios realizados por sus compañeros.

Entre sus principales características están las siguientes:

- Visualiza, comprende y mejora sus procesos de negocio con la más completa herramienta de modelado de procesos de negocio *Business Process Modeling Notation* (BPMN).
- Posee una excelente interoperabilidad en una única plataforma de desarrollo.

- Genera la documentación del proyecto en varios formatos como HTML, MS Word y PDF.
- Ingeniería inversa, *Java*, C++.
- Generador de informes para la generación de documentos.
- Distribución automática de diagramas [26].

1.2.2. Herramienta CASE (*Computer-Aided Software Engineering*) para el modelado

Las Herramientas CASE (*Software* asistido por Computadora, del inglés *Computer Aided Software Engineering*) son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un *software*. Estas se pueden ver como la unión de las herramientas automáticas de *software* y las metodologías de desarrollo de *software* formales [27].

En la actualidad existen un gran número de herramientas CASE, tal es el caso de *Umbrello*, BOUML, ArgoUML, Eclipse UML, *Rational Rose* y *Visual Paradigm*. Esta última es una herramienta muy poderosa y profesional que soporta el ciclo de vida del desarrollo de un *software*. Tiene características gráficas muy cómodas que facilitan la realización de los diagramas de modelado que sigue el estándar UML, que son: Diagramas de Clase, Casos de Uso, Comunicación, Secuencia, Estado, Actividad, Componentes.

Visual Paradigm 8.0

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El *software* de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas y proyectos UML [28].

Presenta varias ventajas entre las que se destacan:

- Poderosa herramienta de generación de .pdf/.html a partir de diagramas UML.
- Sincronización entre el código fuente y el modelo en tiempo real.

- Soporte para toda la notación UML.

Se decide utilizar Visual Paradigm como herramienta CASE para realizar el diseño de los diagramas de clases, las clases conceptuales del modelo de dominio y el diagrama de paquetes, ya que mediante esta se obtiene un mejor entendimiento de la implementación y diseño de la aplicación, además de ser la herramienta a utilizar definida por el propio proyecto.

1.2.3. Lenguaje de programación

Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que son interpretadas por las computadoras, diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, es un modo práctico para que los seres humanos puedan dar instrucciones a una computadora. También la palabra programación se define como el proceso de creación de un programa de computadora, mediante la aplicación de procedimientos lógicos [29].

1.2.4. Lenguaje de Programación C++

En la actualidad, el C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. El lenguaje C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original [30].

Es un lenguaje procedural [orientado a algoritmos] y orientado a objetos, como lenguaje procedural se asemeja al C y es compatible con él. Como lenguaje orientado a objetos se basa en una filosofía completamente diferente, que exige del programador un completo cambio de mentalidad. El lenguaje C++ depende del *hardware*, es uno de los más potentes porque permite programar a alto y bajo nivel. Una de las propiedades de C++ es la reutilización del código en forma de librerías de usuario.

Ventajas del lenguaje C++

- Generar programas eficientes, gracias a la posibilidad de la gestión de memoria que posee mediante punteros.

- Es un lenguaje muy robusto en cuanto a la creación de sistemas complejos se refiere.

- Posee la programación orientada a objetos, lo que permite al programador diseñar aplicaciones desde un punto de vista más como una comunicación entre los objetos, que en una secuencia estructurada de código.

- Por ser un lenguaje compilado, su velocidad de ejecución es más rápida que la de otros lenguajes como por ejemplo Java que utiliza una máquina virtual para poder ejecutarse.

- La posibilidad de poder ser compilado en una variedad de computadoras, con pocos cambios (portabilidad)[31].

Se decide seleccionar el lenguaje C++ porque el *plugin* que se va a desarrollar necesita ser integrado a la herramienta de administración de base de datos HABD, que fue implementada en este lenguaje, además de contar con las características y ventajas antes mencionadas.

1.2.5. Entorno de Desarrollo Integrado (IDE)

Un IDE (*Integrated Development Environment*), en español, Entorno Integrado de Desarrollo es un programa informático compuesto por un conjunto de herramientas de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los entornos de desarrollo pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Entorno de Desarrollo Integrado Qt Creator 2.3.1

Qt Creator es un IDE para desarrollar aplicaciones en C++ de manera sencilla y rápida pues posee un avanzado editor de código C++, característica que determina su empleo en el desarrollo del *plugin*. Es basado en la librería QT y posee herramientas para la administración y construcción de proyectos, depurador visual, GUI (Interfaz gráfica de usuario) integrada, diseñador de formularios y auto-completado de código. Cuenta con las siguientes características principales:

- Editor avanzado para C++.

- Ayuda sensible al contexto integrado.

- Soporta los lenguajes: C#/.NET Lenguajes (Mono), Python: PyQt y PySide, Ada, Pascal, Perl, PHP y Ruby.
- Diseñador de formularios (GUI) integrado.
- Herramientas para la administración y construcción de proyectos.
- Completado automático.
- Depurador visual [32].

Está completamente integrado con toda la documentación Qt y ejemplos a través de la ayuda Qt *Plugin*.

Se utiliza como entorno de desarrollo integrado Qt Creator, ya que es el IDE en el que fue desarrollada la herramienta HABD, posee un elevado completamiento de código y, es entorno de desarrollo predefinido por el proyecto.

1.2.6. Framework de desarrollo

En el desarrollo de *software,* un *framework* es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos concretos, en base a la cual otro proyecto de *software* puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Framework QT 4.8

El *framework* QT es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica, como herramientas de la consola y servidores. Utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios lenguajes de programación a través de *bindings*[3]. Funciona en las principales plataformas, y tiene un amplio apoyo. La API (La interfaz de programación de aplicaciones) de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte

[3] Es una adaptación de una biblioteca para ser usada en un lenguaje de programación.

de red y una API multiplataforma unificada para la manipulación de archivos, además de estructuras de datos tradicionales. Esta biblioteca se encuentra distribuida bajo los términos de GNU *Lesser General Public License*, Qt es *software* libre y de código abierto [33].

Se decide utilizar el *Framework* QT 4.8 porque es el *framework* en el que se desarrolló la herramienta de administración de base de datos HABD a la que será integrado el *plugin*. También posee una Interfaz de Aplicación de Programación (API) muy fácil de utilizar, que permite una mejor productividad y calidad del trabajo. Además de ser el *framework* escogido por el departamento y el proyecto.

1.2.7. Herramienta para el control de versiones

Subversion es un sistema de control de versiones libre y de código fuente abierto. Maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros.

Para el desarrollo de un sistema es de mucha trascendencia emplear un correcto control de versiones. La herramienta que se seleccionó para esta actividad en la realización de la solución es Subversion 1.5.4 por el conjunto de razones que se muestran a continuación [34].

Subversion (SVN 1.5.4)

Sistema de control de versiones libre y de código fuente abierto. Posee como elemento fundamental un repositorio, el cual constituye un almacén central de datos. Este guarda información en forma de árbol de archivos.Soporta cualquier cantidad de conexiones simultáneas al repositorio y pudiendo cada uno de los usuarios leer o escribir en esos archivos. Al escribir datos, un cliente pone a disposición de otros la información; así mismo al leer datos, el cliente recibe información de otros. Subversión es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros [35].

Las principales características de Subversion son:

- Mantiene versiones no sólo de archivos, sino también de directorios.
- También se mantienen versiones de los metadatos asociados a los directorios.
- Además de los cambios en el contenido de los documentos, se mantiene la historia de todas las operaciones de cada elemento, incluyendo la copia, cambio de directorio o de nombre.
- Atomicidad de las actualizaciones. Una lista de cambios constituye una única transacción o actualización del repositorio. Esta característica minimiza el riesgo de que aparezcan inconsistencias entre distintas partes del repositorio.
- Posibilidad de elegir el protocolo de red. Además de un protocolo propio (svn), puede trabajar sobre http (o https).
- Soporte tanto de ficheros de texto como de binarios.
- Mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.

Se decide utilizar como Sistema de Control de Versiones Subversion en su versión SVN-1.5.4 ya que nos brinda la posibilidad de gestionar las modificaciones durante el desarrollo del *software*. Permite que varias personas trabajen sobre los mismos ficheros, implementa un sistema virtual de fichero versionado que sigue los cambios en todos los árboles de directorios. Además expresa las diferencias entre ficheros usando un algoritmo de diferenciación binario, que funciona exactamente igual, tanto en ficheros de textos como en ficheros binarios, además es el sistema de control establecido por el proyecto.

Conclusiones del capítulo

Los conceptos abordados en el capítulo tales como bases de datos, sistema gestor de bases de datos, herramienta de administración de bases de datos HABD, *plugin* y diseñadores de bases de datos, permitieron alcanzar un mayor entendimiento sobre el negocio que se implementará. Para guiar el proceso de desarrollo fue seleccionada la metodología *Extreme Programing* (XP). Del estudio de las herramientas y tecnologías existentes se determinó utilizar Visual Paradigm 8.0 como herramienta CASE, empleando, el lenguaje de modelado UML, el *framework* QT 4.8, el IDE de desarrollo QT Creator 2.3.1, como lenguaje

de programación C++. Se decidió utilizar el controlador de versiones Subversion. Los principales diseñadores de base de datos estudiados, por las limitaciones que presentan, no serán tomados en cuenta en la solución, aunque si se tomaron de ellos algunas características para incorporarlas en el *plugin* a desarrollar, tales como el proceso de modelado de las bases de datos y la generación de los *script*.

CAPÍTULO 2

CARACTERÍSTICAS DEL SISTEMA PROPUESTO

Introducción

En el siguiente capítulo se presenta la propuesta de un *Plugin* para el diseño de bases de datos PostgreSQL, MySQL y SQLite. Se hace una especificación de las características que tendrá el mismo, además de sus requisitos funcionales y no funcionales. Se definen las historias de usuario y se propone un diagrama de clases del diseño que se utiliza como complemento de la metodología definida, para lograr un mejor entendimiento del componente.

2.1. Descripción del *plugin* diseñador de bases de datos v2.0 para HABD

El *plugin* diseñador de bases de datos para integrarse a HABD, presenta una interfaz visual sencilla y amigable para quienes interactúen con la herramienta. Ofrece la posibilidad de realizar gráficamente el proceso de modelado de las bases de datos de los gestores PostgreSQL, MySQL y SQLite. La utilización de esta herramienta permite gestionar las tablas de la base de datos, llevando a cabo la creación, modificación, eliminación y visualización de las mismas, así como el trabajo de la herencia. De la misma forma es posible el manejo de los atributos de las tablas, permitiendo realizar las acciones de crear, eliminar, modificar y visualizar los mismos. También es posible la creación, eliminación, modificación y visualización de restricciones según las necesidades del usuario. Le facilita el trabajo al usuario con los índices, *tablespace*[4], vistas y esquema, para que el mismo los adecue según sus requisitos. Se estableció,

[4] Los *tablespace* son referencias a ubicaciones físicas del almacenamiento de bases de datos y/o de los objetos que éste contiene.

con el objetivo de lograr que el modelo creado cuente con un mejor diseño que cuando el usuario decida crear un atributo o una tabla no permitirle poner espacios y si comienza con un dígito se le notificará al usuario una alerta de lo que está escribiendo, logrando de esta manera que las bases de datos se conformen sin errores ni inconsistencias. También la aplicación brinda la posibilidad de generar el script de la base de datos, lo cual le permitirá al usuario cargarlo en el gestor deseado y en la ubicación que determine. Una vez creado el modelo, el diseñador accede a exportarlo en formato mdl y cargarlo en caso de que sea necesario, también tiene la opción de exportarlo en formato jpg[5] para tener una mejor visualización del modelo creado en cualquier otra parte.

2.2. Modelo de dominio

La metodología XP, es seleccionada para guiar el proceso de desarrollo del *plugin*, no tiene definido una técnica para determinar el negocio, por lo que es necesario realizar este proceso de una manera entendible para llevar a cabo su implementación. Una de las técnicas más utilizadas es el modelo de dominio, el cual facilita el entendimiento del *software* que se desea desarrollar; constituyendo uno de los artefactos más importantes para el proyecto.

El Modelo de Dominio es una representación visual estática del entorno, un diagrama con los objetos que existen (reales) relacionados con el proyecto. Ayuda a comprender los conceptos que utilizan los usuarios, conceptos con los que trabajan y con los que deberá trabajar nuestra aplicación.

[5] Son las siglas de ***Joint Photographic Experts Group***, el nombre del grupo que creó este formato. **jpg** es un formato de compresión de imágenes, tanto en color como en escala de grises, con alta calidad.

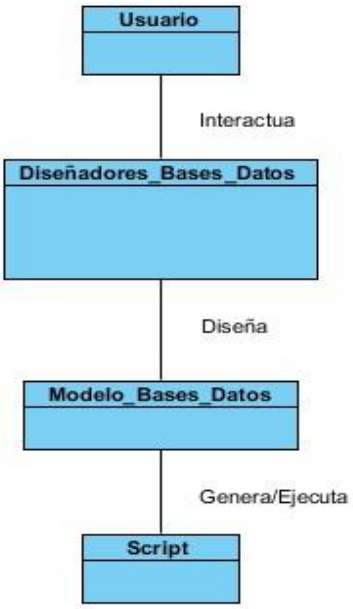


Figura 1: Diagrama Modelo de Dominio.

Usuario: Persona que interactúa con la herramienta para realizar el diseño de las bases de datos.

Diseñadores de bases de datos: Herramientas embebidas dentro o no, de herramientas de administración de bases de datos que se emplean para realizar el diseño de las bases de datos.

Modelo de Bases de Datos: Es una descripción de algo conocido como contenedor de datos (algo donde se guarda información), así como los métodos para almacenar y recuperar información de estos contenedores. Los modelos de datos no son objetos físicos, son abstracciones que permiten la implementación de un sistema eficiente de base de datos.

Script: Modelo generado por el diseñador de bases de datos, el cual contiene las tablas, *tablespace*, esquemas y todas las propiedades del modelo de la base de datos diseñada.

2.3. Historias de Usuario

Las historias de usuarios son artefactos generados en XP, tienen la misma finalidad que los casos de uso en la metodología de desarrollo RUP (Proceso Unificado de Desarrollo, en español), pero con algunas diferencias: constan de tres o cuatro líneas escritas por el cliente en un lenguaje no técnico, no se debe hablar ni de posibles algoritmos para su implementación ni de diseños de base de datos adecuados. Son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [36].

Para el presente trabajo se obtuvieron un total de 15 historias de usuarios (HU) que se desarrollarán en tres iteraciones. A continuación se muestran dos HU, el resto podrán encontrarse en el documento Plantilla de Historias de Usuarios del Expediente de proyecto.

Tabla 1: Seleccionar el gestor de bases de datos.

Historia de Usuario	
Número 1	**Nombre de la Historia de Usuario:** Seleccionar el gestor de bases de datos.
Cantidad de modificaciones a la Historia de Usuario: 0	
Usuario: Wilber Turruella Velázquez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alto	**Puntos estimados:** 1 semanas
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semanas
Descripción: Permite al usuario escoger el gestor de bases de datos para realizar el diseño.	
Observaciones: Para poder realizar el diseño el usuario deberá seleccionar primero un gestor, de lo contrario no podrá realizar ninguna operación.	

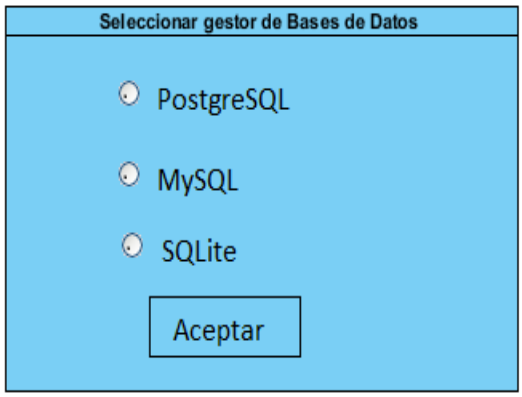
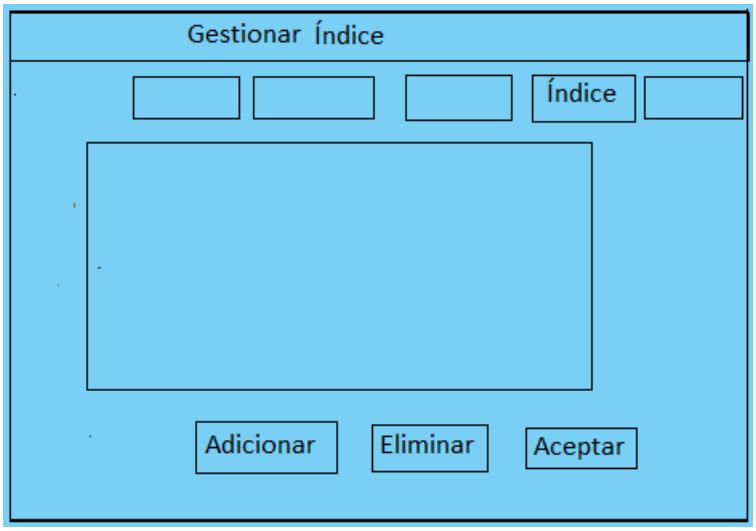
Prototipo de interfaces:


Tabla 2: Gestionar Índice.

	Historia de Usuario
Número 13	**Nombre de la Historia de Usuario:** Gestionar Índice.
Cantidad de modificaciones a la Historia de Usuario: 0	
Usuario: Wilber Turruella Velázquez	**Iteración asignada:** 2
Prioridad en negocio: Alto	**Puntos estimados:** 0.8 semanas
Riesgo en desarrollo: Alto	**Puntos reales:** 0.8 semanas
Descripción: El usuario puede crear, eliminar, modificar y visualizar el índice con los campos asociados al mismo, así como el tipo de índice.	
Observaciones:	

Prototipo de interfaces:



2.4. Lista de reserva del producto

La lista de reserva del producto es una tabla que contiene los requisitos funcionales que debe cumplir la aplicación que se desea realizar, ordenados según la prioridad de implementación, ubicados en Muy Alta, Alta, Media y Baja, en esta última aparecen los requisitos no funcionales del sistema. Indica la estimación de cada uno de ellos, su implementación por semanas y quien hizo la estimación.

Tabla 3: Lista de reserva del producto.

Ítem *	**Descripción**	**Estimación**	**Estimado por**
Prioridad: Muy Alta			
1	Seleccionar Gestor de bases de datos.	0.5	Analista.
2	Crear Tabla.	0.5	Analista.
3	Eliminar Tabla.	0.5	Analista.
4	Modificar Tabla.	0.5	Analista.
5	Visualizar Tabla.	0.5	Analista.
6	Visualizar Sentencia SQL Tabla.	0.5	Analista.

7	Crear Atributo.	0.5	Analista.
8	Eliminar Atributo.	0.5	Analista.
9	Modificar Atributo.	0.5	Analista.
10	Visualizar Atributo.	0.5	Analista.
11	Establecer Relación.	0.5	Analista.
12	Eliminar Relación.	0.5	Analista.
Prioridad: Alta			
13	Crear Restricción.	0.5	Analista.
14	Eliminar Restricción.	0.5	Analista.
15	Modificar Restricción.	0.5	Analista.
16	Visualizar Restricción.	0.5	Analista.
17	Visualizar Sentencia SQL Restricción.	0.5	Analista.
18	Generar Script.	0.5	Analista.
19	Ejecutar Script.	0.5	Analista.
20	Exportar Modelo.	1	Analista.
21	Cargar Modelo.	1	Analista.
22	Generar Imagen.	0.5	Analista.
Prioridad: Media			
23	Crear *Tablespace*.	0.2	Analista.
24	Eliminar *Tablespace.*	0.2	Analista.
25	Modificar *Tablespace.*	0.2	Analista.
26	Visualizar *Tablespace.*	0.2	Analista.
27	Visualizar Sentencia SQL *Tablespace.*	0.2	Analista.
28	Crear Esquema.	0.2	Analista.
29	Eliminar Esquema.	0.2	Analista.
30	Modificar Esquema.	0.2	Analista.
31	Visualizar Esquema.	0.2	Analista.
32	Crear Índice.	0.2	Analista.
33	Eliminar Índice.	0.2	Analista.
34	Modificar Índice.	0.2	Analista.

35	Visualizar Índice	0.2	Analista.
36	Crear Vista.	0.2	Analista.
37	Eliminar Vista.	0.2	Analista.
38	Modificar Vista.	0.2	Analista.
39	Visualizar Vista.	0.2	Analista.

Prioridad: Baja

Usabilidad.

Cualquier usuario que posea conocimientos básicos en el trabajo con los gestores PostgreSQL, MySQL y SQLite podrá hacer uso de la aplicación para realizar el diseño de una base de datos.

Rendimiento.

La herramienta que se propone debe ser rápida y eficaz realizando el diseño de sus bases de datos con calidad.

Portabilidad.

Será un sistema multiplataforma, lo que permitirá poder disponer del mismo en cualquier sistema operativo.

***Software*.**

Para poder hacer uso de este *plugin* se debe tener instalado las librerías de QT, el compilador de C++, los gestores PostgreSQL o MySQL o SQLite.

Apariencia o interfaz externa.

Al integrarse el *plugin* a la herramienta HABD contará con una interfaz amigable para el usuario, facilitándole al usuario interactuar de una forma sencilla para realizar el diseño de su base de datos.

Disponibilidad.

Se podrá hacer uso de la aplicación siempre que estén instaladas todas las librerías necesarias.

Restricciones de diseño e implementación:

El diseñador de bases de datos debe estar listo para el despliegue en un tiempo aproximado de diez meses y para el desarrollo del mismo se usará el lenguaje de programación C++.

2.5. Tarea de la Ingeniería

El equipo de desarrollo evalúa cada HU y las divide en Tareas de la Ingeniería (TI), donde cada una de estas representa una característica del sistema [37]. A continuación se muestran dos TI, mostrándose el encargado de programarla y una descripción breve de lo que se necesita. En la Tabla 4 se muestra la TI correspondiente a seleccionar el gestor de bases de datos y en la Tabla 5 se muestra la de establecer relación entre tablas, encontrándose el resto de ellas en el documento correspondiente a las TI del expediente de proyecto.

Tabla 4: Seleccionar el gestor de bases de datos.

Tarea de Ingeniería	
Número Tarea: 1	**Historia de Usuario:** 1
Nombre Tarea: Seleccionar el gestor de bases de datos.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 1
Fecha Inicio: 12/12/2012	**Fecha Fin:** 18/12/2012
Programador Responsable: Wilber Turruella Velázquez.	
Descripción: El usuario debe seleccionar el gestor de bases de datos al cual se le realizará el modelo de datos.	

Tabla 5: Realizar relación visual de las tablas.

Tarea de Ingeniería	
Número Tarea: 12	**Historia de Usuario:** 4
Nombre Tarea: Diseñar relación visual de las tablas.	
Tipo de Tarea: Desarrollo.	**Puntos Estimados:** 0.5
Fecha Inicio: 1/05/2013	**Fecha Fin:** 28/05/2013
Programador Responsable: Wilber Turruella Velázquez.	
Descripción: El usuario debe seleccionar el tipo de relación a establecer entre las dos tablas.	

2.6. Plan de iteraciones

Después de contar con una definición de las historias de usuarios debe crearse un plan de iteraciones, indicando cuáles se desarrollarán en cada iteración del programa. El objetivo de este artefacto es tener una planificación del trabajo, donde los desarrolladores y el cliente establecen los tiempos de implementación de las historias de usuarios y la prioridad con que serán desarrolladas.

El plan de iteraciones está compuesto por iteraciones que no superan las tres semanas. Este posee una arquitectura del sistema que puede ser establecida en la primera iteración y utilizada durante la trayectoria del proyecto. Para que se efectúe la creación de esta arquitectura se deben escoger las historias correctas que la exijan, pero como el cliente es el que decide que o cuales historias se implementarán en cada iteración, muchas veces la implementación de esta arquitectura no es posible. El sistema estará listo para su utilización al completarse la última iteración.

En el transcurso de la elaboración del plan de iteraciones se deben tomar en cuenta diferentes elementos, tales como:

- Velocidad del proyecto.
- Pruebas de aceptación no superadas de la iteración precedente.
- Tareas no terminadas de la iteración precedente.
- Historias de usuarios no abordadas [38].

Ya identificadas las historias de usuarios se establecieron cinco iteraciones para el desarrollo de la aplicación. A continuación se detallan en el plan de iteración las HU a desarrollar en cada iteración, así como el tiempo estimado para ello.

Tabla 6: Plan de Iteraciones.

Iteración	**Descripción de la iteración**	**Orden de la HU a implementar**	**Duración total**
1	En esta iteración se desarrollaran historias de usuarios de prioridad muy alta, estas serán las encargadas de llevar a cabo el proceso de selección del gestor de bases de datos, así como la gestión de las tablas.	1-2	3 Semanas

2	El objetivo fundamental de esta iteración es llevar a cabo la gestión de los atributos y las relaciones entre tablas. Todas estas historias de usuarios son de prioridad muy alta.	3-4	3 Semanas
3	Esta iteración tiene como objetivo la implementación de historias de usuario de prioridad alta, en la misma se llevará a cabo la gestión de las restricciones, así como generar y ejecutar script.	5-6-7-8	3.5 Semanas
4	Esta iteración tiene como objetivo la implementación de historias de usuario de prioridad alta, en la misma se llevará a cabo las historias de usuario exportar y cargar modelo así como generar imagen.	9-10-11	2.5 Semanas
5	Esta iteración tiene como objetivo la implementación de historias de usuario de prioridad media, en la misma se llevará a cabo la gestión de los tablespaces, esquemas, índices y las vistas.	12-13-14-15	3.2 Semanas

2.7. Modelo de diseño

En la fase de diseño de la metodología XP se sugiere tener diseños simples y sencillos, para conseguir un mejor entendimiento e implementación consiguiendo reducir el tiempo y esfuerzo a la hora de desarrollar. Esta fase incluye las tarjetas clase-responsabilidades-colaboración (CRC) y diagrama de clases.

2.7.1. Tarjetas Clase, Responsabilidad y Colaboración

Las tarjetas Clase, Responsabilidad y Colaboración (CRC) son una técnica que permiten diseñar el sistema en conjunto. Permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Posibilitan que el equipo completo contribuya en la tarea del diseño representando un objeto o clase de agrupamiento. La clase a la que pertenece el objeto se puede escribir en la parte de arriba de la tarjeta, en una columna a la izquierda se escriben las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad.

- **Clase**: Cualquier persona, cosa, evento, concepto, pantalla o reporte. Además consiste en crear una tarjeta para cada clase.
- **Responsabilidades**: Son todos los servicios que proporciona para todos los contratos que soporta la lista de servicios que una instancia de una clase puede pedir a una instancia de otra.
- **Colaboraciones**: Aquellas clases con las que trabaja en conjunto para llevar a cabo sus responsabilidades [39].

En el proceso de diseño del *plugin* se crearon 13 Tarjetas CRC en total. A continuación se muestran unas de las más significativas:

Tabla 7: Tarjeta CRC 6.

Tarjeta CRC 6	
Clase: DBTable	
Responsabilidades	**Colaboraciones**
Permite gestionar las tabla de la bases de datos.	DBObjetc. Database.

Tabla 8: Tarjeta CRC 8.

Tarjeta CRC 8	
Clase: DBView.	
Responsabilidades	**Colaboraciones**
Permite gestionar las vistas.	DBObject. Database.

Tabla 9: Tarjeta CRC 10.

Tarjeta CRC 10	
Clase: DBÍndex.	
Responsabilidades	**Colaboraciones**
Permite gestionar los índices.	DBObject. DBTable. DBColumn.

2.7.2. Diagrama de clases

A pesar de que la metodología escogida para guiar el proceso de desarrollo de la aplicación no incluye un diagrama de clases, en la solución propuesta se realiza el mismo para lograr de una forma más sencilla y organizada el trabajo, obteniéndose como resultado una descripción más detallada del *plugin* por parte del equipo de desarrollo. Los diagramas de clases se utilizan para representar la estructura estática de un sistema incluyendo una colección de elementos, tales como, clases y relaciones [38].
A continuación se muestra el diagrama de clases perteneciente al *plugin.*

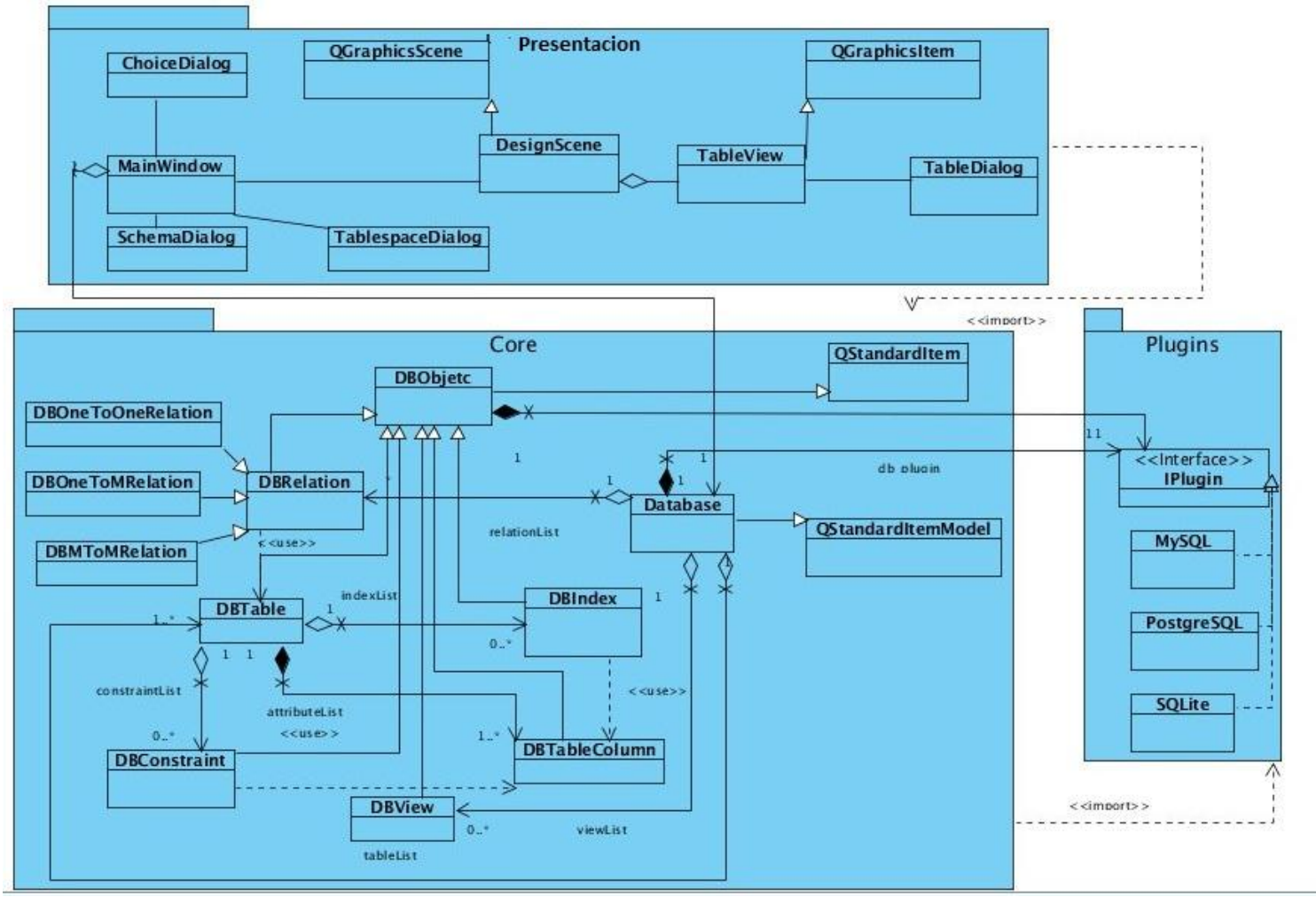


Figura 2: Diagrama de clases del diseño.

2.8. Patrón de Arquitectura para el *plugin* diseñador de bases de datos

Un patrón de arquitectura de *software* es un esquema genérico probado para solucionar un problema particular, el cual es recurrente dentro de un cierto contexto. Este esquema se especifica describiendo los componentes, con sus responsabilidades y relaciones [40].

De acuerdo a la definición anterior y con el objetivo de lograr un trabajo organizado en el desarrollo de la aplicación se ha decido utilizar como patrón de programación en capas. A continuación se describen brevemente las capas definidas para la aplicación.

Capa de Presentación: Es la que permite la comunicación del usuario con el sistema, mediante una interfaz sencilla y amigable. Se comunica únicamente con la capa de negocio. Este paquete contiene las clases visuales que trae por defecto el framework QT y las que se implementan en la aplicación, como se evidencia en la Figura 3.

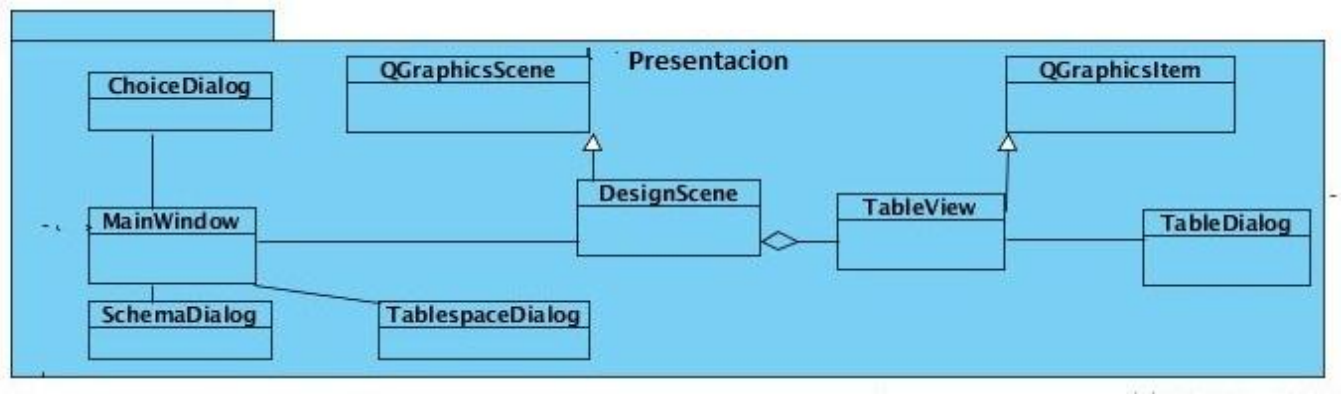


Figura 3: Capa de presentación.

Capa de negocio o Core: Esta capa juega un papel fundamental ya que es la encargada de establecer las reglas que se deben cumplir, pues en la misma se encuentra la lógica del negocio. Funciona como intermediaria entre la primera y la tercera capa ya que es donde se reciben las peticiones hechas por los usuarios y se envían las respuestas mediantes los procesos definidos. En la Figura 4 se muestran las clases que forman parte de esta capa, como son **DBTable:** encargada de gestionar las tablas en el diseño y **DBTableColumna:** encargada de la gestión de los campos de las tablas.

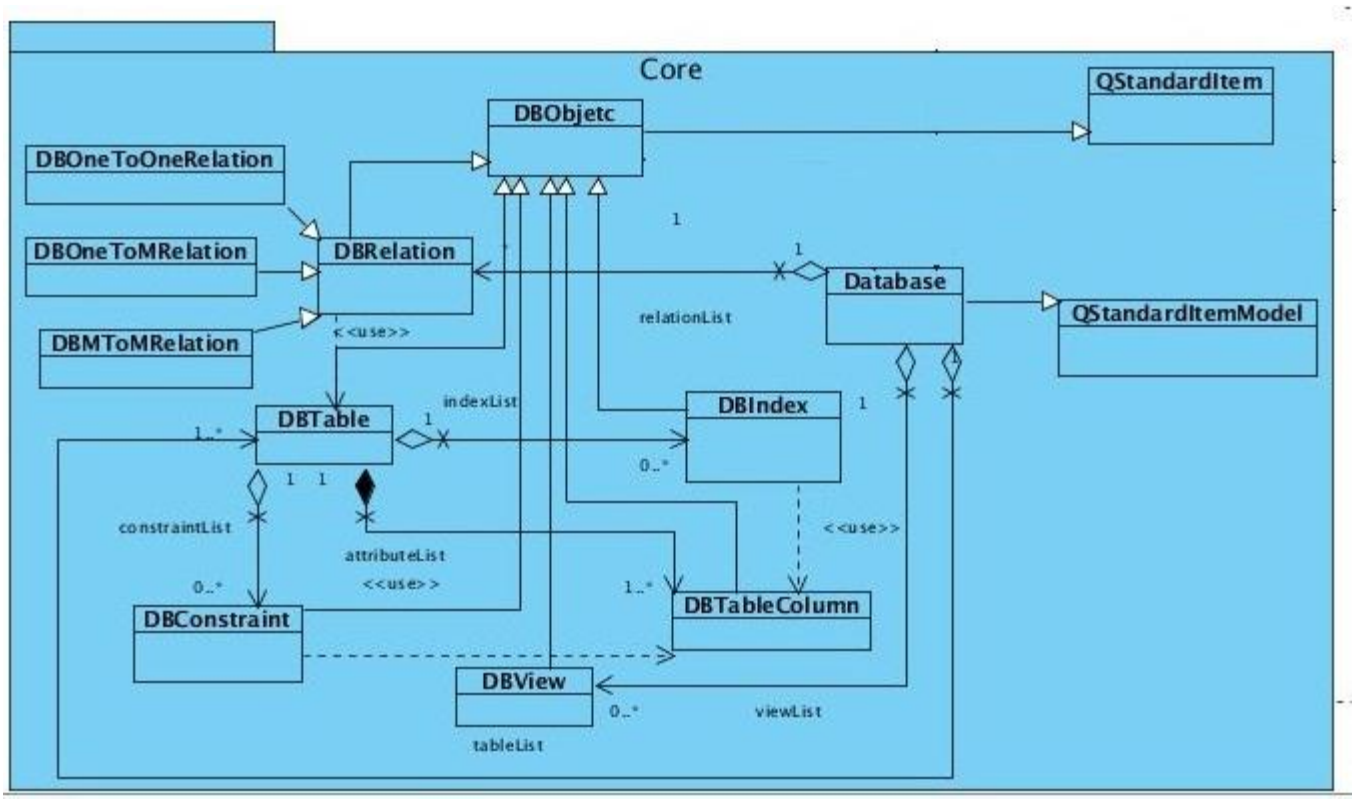


Figura 4 : Capa de Negocio o Core.

Capa de información: Esta capa se encarga de contener aquellas clases que cuentan con informaciones necesarias para realizar funcionalidades específicas, como se puede observar en la Figura 5, existe una clase específica para cada gestor de base de datos que posee toda la información detallada de los mismos. Es importante aclarar que estas clases por sí solas no funcionan.

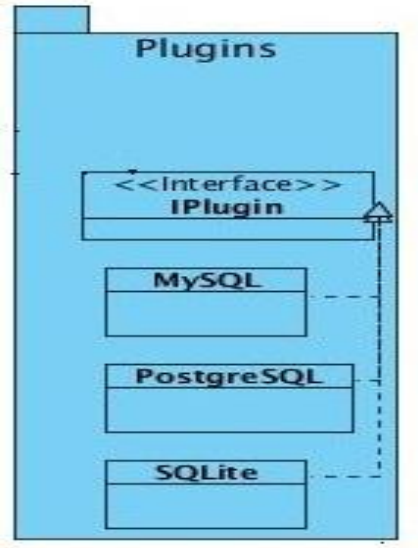


Figura 5: Capa de Información.

2.9. Patrones de Diseño

Los Patrones de Diseño definen cómo construir un *software*, cómo utilizar las clases y los objetos de forma conocida. Proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo. Estos constituyen una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias [41].

A continuación se presentan los patrones empleados en el *plugin*.

Los patrones GRASP (Patrones de *Software* para la asignación General de Responsabilidad) constituyen un apoyo para la enseñanza, que ayuda a entender el diseño de objetos. De los diferentes patrones que ofrece GRASP se ha tenido en cuenta para la modelación del *plugin* diseñador de bases de datos los siguientes:

Patrón Experto: El patrón garantiza que la responsabilidad de la creación de un objeto o la implementación de un método, recaiga sobre la clase que conoce toda la información necesaria para

crearlo lo que contribuye a un adecuado encapsulamiento, favoreciendo la robustez y fácil mantenimiento del sistema. El patrón se evidencia en la clase Database ya que esta es la encargada de contener lo necesario para gestionar las actividades correspondientes al diseño de la base de datos, como se observa en la siguiente figura esta es la responsable del trabajo con las vistas.

Figura 6: Patrón Experto.

Patrón Creador: Se asigna la responsabilidad a una clase de crear cuando, contiene, agrega, compone, almacena o usa otra clase, lo que brinda una alta posibilidad de reutilizar la clase creadora. Un ejemplo donde se pone de manifiesto este patrón de diseño es en la clase **Database**, debido a que la misma tiene la responsabilidad de crear objetos de tipo tabla.

Figura 7: Patrón Creador.

Patrón Bajo Acoplamiento: El bajo acoplamiento está dado por disminuir al máximo la interdependencia entre las clases y el poco impacto al introducirse cambios en los objetos. Este patrón se evidencia en la creación de las clases DBRelation, las cuales modelan las relaciones entre las tablas. Las clases mencionadas contienen como miembros privados únicamente referencias a las clases relacionadas, permitiendo así que el agregar o eliminar relaciones las tablas involucradas no se vean afectadas.

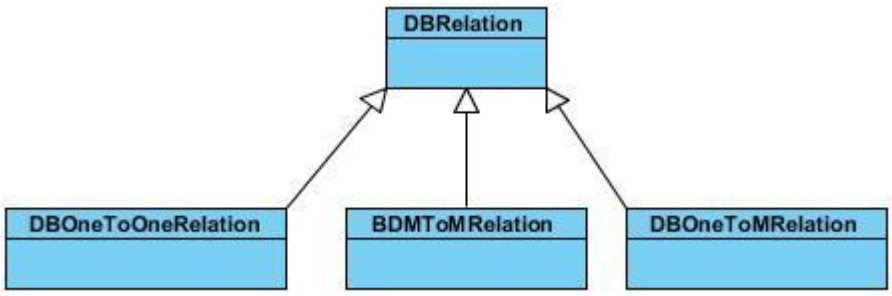


Figura 8: Patrón Bajo acoplamiento.

Alta Cohesión: La alta cohesión está dada por la relación inversa que existe entre la responsabilidad que recae sobre una clase o subsistema, la complejidad y cantidad de trabajo que realiza. El componente *MainWindow* constituye la clase controladora de la capa de Presentación y es la que de una forma u otra maneja todas las funcionalidades que el usuario ejecuta. Está clase delega la responsabilidad de controlar los elementos lógicos del sistema (dígase tablas, relaciones, índices, etc) en la clase *Database*.

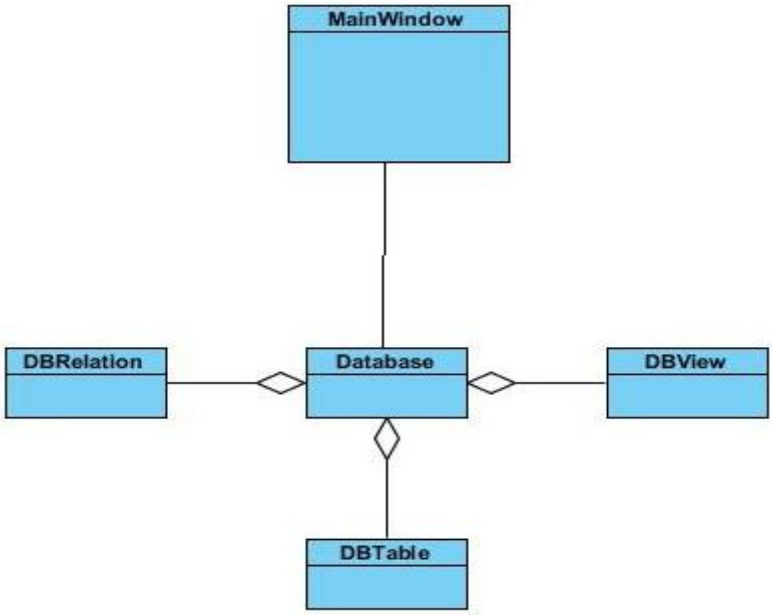


Figura 9: Patrón Alta Cohesión.

Controlador: El patrón controlador es el encargado de controlar un evento del sistema, se utiliza para intermediar entre la interfaz y el algoritmo. El uso de este patrón se evidencia en la clase TablespaceDialog, que es la encargada de recoger los datos entrados por el usuario para la creación de los tablespace.

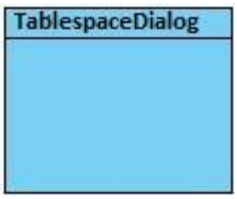


Figura 10: Patrón Controlador.

Los patrones GOF:

Los patrones de diseño del grupo de GOF se clasifican en tres grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

Strategy: Define una familia de algoritmos, encapsula uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan [42].

Este patrón se evidencia, como se muestra en la Figura 11, en la clase IPlugin que es la encargada de crear los objetos del tipo de gestor y la forma de comportarse cada uno de ellos.

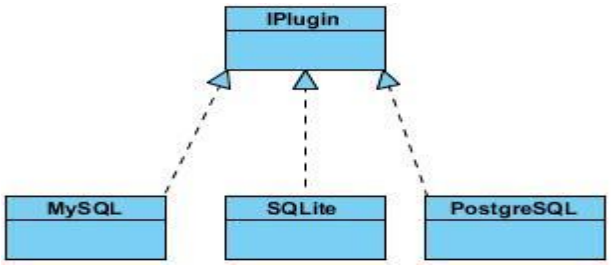


Figura 11: Patrón Strategy.

Singleton: Permite garantizar que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma. Este patrón permite que el acceso a una instancia esté más controlado, se reduce el espacio de nombres (frente al uso de variables globales), permite refinamientos en las operaciones y en la representación, mediante la especialización por herencia de “Solitario”, es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable) [43].

El empleo de este patrón se evidencia en las siguientes clases como se muestra en la Figura 12.

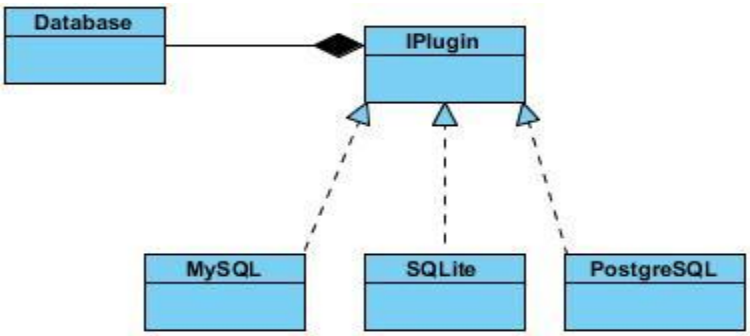


Figura 12: Patrón Singleton.

2.10. Estándares de codificación

Usar técnicas de codificación y realizar buenas prácticas de programación con vista a generar un código de alta calidad, es de gran importancia para el *software* que se desarrolla y para obtener un buen rendimiento del mismo. Además, si se aplica de forma correcta el estándar definido facilita añadir nuevas características, modificar las ya existentes, depurar errores, o mejorar su rendimiento. El propósito de las siguientes reglas y recomendaciones es lograr que los programadores del proyecto PostgreSQL tengan un estilo de código común**.**

Identación

La unidad de identado es de 4 espacios. El uso de la tabulación debe ser evitado porque [tal como se escribía en el siglo pasado] no existe un estándar que determine con precisión el ancho que va a producir la tabulación.

Longitud de la Línea

Deben evitarse las líneas con más de 80 caracteres. Cuando una sentencia sobrepase una línea simple del editor, esta deber ser separada en otras. Realizar la separación después de un operador, preferentemente luego de una coma, pues de esta forma 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 dentada con cinco espacios.

Comentarios

Es muy importante dejar alguna información que pueda ser leída después por aquellas personas (posiblemente usted mismo) que necesitan entender que fue lo que se hizo en el fragmento de código, sirviéndole el comentario para tener un mayor entendimiento de lo implementado. Se debe tener en cuenta que a la hora de escribirlos, los mismos deben tener la mayor claridad posible para lograr un mayor entendimiento del usuario. Es recomendable la utilización de comentarios de una sola línea. Reserve los comentarios de bloques para la documentación formal o para comentar porciones de código.

Declaración de variables

La declaración de cada una de las variables debe realizarse en una línea y ser comentada, además deben aparecer ordenadas alfabéticamente: El nombre de las variables debe de comenzar con letras minúsculas

y cada palabra relevante por la que esté compuesta debe ser con letra mayúscula. A continuación se muestra un ejemplo de código de la clase DBTable.

```
DBTable* DBTable::createRelationMToM(DBTable* secundary, QString name)
{
    DBTable* generatedTable = new DBTable(name);

     QString columnName = this->getColumnId()->getName();
     createRelationOneToM(generatedTable, columnName);
     columnName = secundary->getColumnId()->getName();
     secundary->createRelationOneToM(generatedTable,columnName);

    this->mToMVect.push_back(secundary);
    return generatedTable;
}
```

Declaración de funciones

No debe haber espacio entre el nombre de la función y el paréntesis izquierdo, ni entre este y la lista de parámetros. Debe haber un espacio entre el paréntesis derecho y la llave de comienzo del cuerpo de la función. Las sentencias del cuerpo deben estar en la línea siguiente. La llave que cierra debe estar alineada con la línea de declaración de la función. Los nombres de las funciones se rigen por las mismas características que el de las variables.

Identificadores

Los identificadores pueden estar formados por cualesquiera de las 26 letras minúsculas o mayúsculas (A - Z, a – z), los 10 dígitos (0 – 9) y el 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 dólar “$” o la barra invertida “\” en los identificadores.

Tipos de sentencias

Sentencias Simples

Cada línea debe contener como máximo una sentencia. Debe poner un punto y coma “;” al final de cada sentencia simple. Tenga en cuenta que una sentencia de asignación puede resultar en la asignación de

una función o de un objeto como literal y en todos los casos como sentencia de asignación debe estar finalizada con un punto y coma, como se muestra en el siguiente fragmento de código.

```
PGSQLTable* generatedTable = new PGSQLTable(name, this->tableSpace, this->getSchema());
```

Sentencias Compuestas

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

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

```
PGSQLDatabase::PGSQLDatabase( QString name, TableSpace* tableSpace, Schema* schema, DBManager manager):Database( name, manager )
    {
            this->schema.push_back(schema);
            this->connectionLimits = connectionLimits;
            this->tableSpace = tableSpace;
            this->tableSpaceList.push_back(tableSpace);
       }
```

Etiquetas

Las sentencias etiquetadas son opcionales. Solo estas sentencias deben ser etiquetadas: while, do, for, foreach, switch.

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.

```
TableSpace* PGSQLDatabase::createTableSpace(QString name, QString dir)
{
  return new TableSpace(name, dir);
}
```

Sentencia if

La sentencia if debe ser escrita de esta manera:

if (condition)
{
    statements
}

if (condition)
{
    statements
} **else**
{
    statements
}

if (condition)
{
    statements
}
else if (condition)
{
    statements
}

else

{

statements

}

Estructuras repetitivas

Las estructuras repetitivas deben ser escritas de esta manera:

for (initialization; condition; update)

{

statements

}

while (condition)

{

statements

}

forecah (valor1, valor2)

{

statements

}

Sentencia switch

La sentencia switch debe ser escrita de la siguiente forma:

switch (expression)

{

case expression:

statements

case expression:

statements

default:

statements

}

Espacios en blanco

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

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

Declaraciones de clases

Solo debe existir un fichero con más de una clase declarada.

2.11. Interfaces de la aplicación

A continuación se presentarán una serie de figuras donde se evidencian algunas de las funcionalidades del *plugin.*

En la Figura 13 se muestra la interfaz principal donde el usuario debe seleccionar primeramente el gestor de bases de datos al cual desea realizarle el diseño de su base de datos. En la Figura 13 se muestra como el usuario tiene la posibilidad de seleccionar el gestor de bases de datos al cual desee realizarle el diseño de la base de datos. En la Figura 14 se refleja la funcionalidad de crear una tabla.

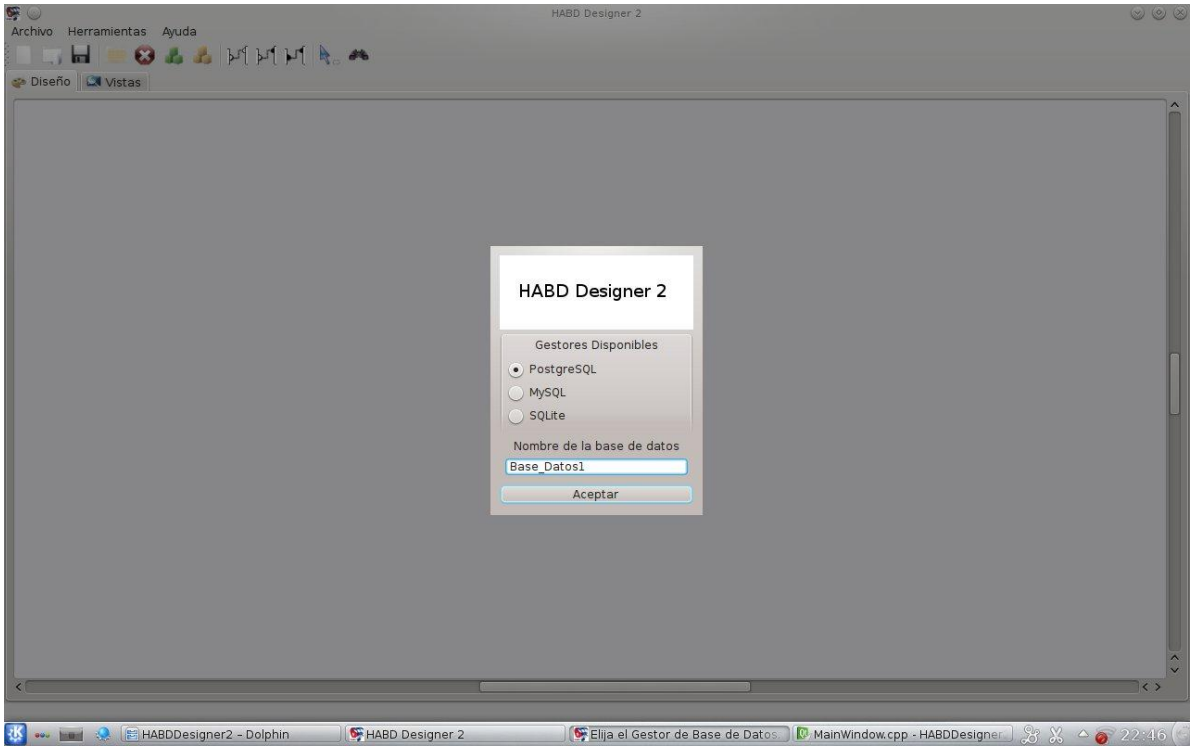


Figura 13. Seleccionar Gestor de Base de Datos.

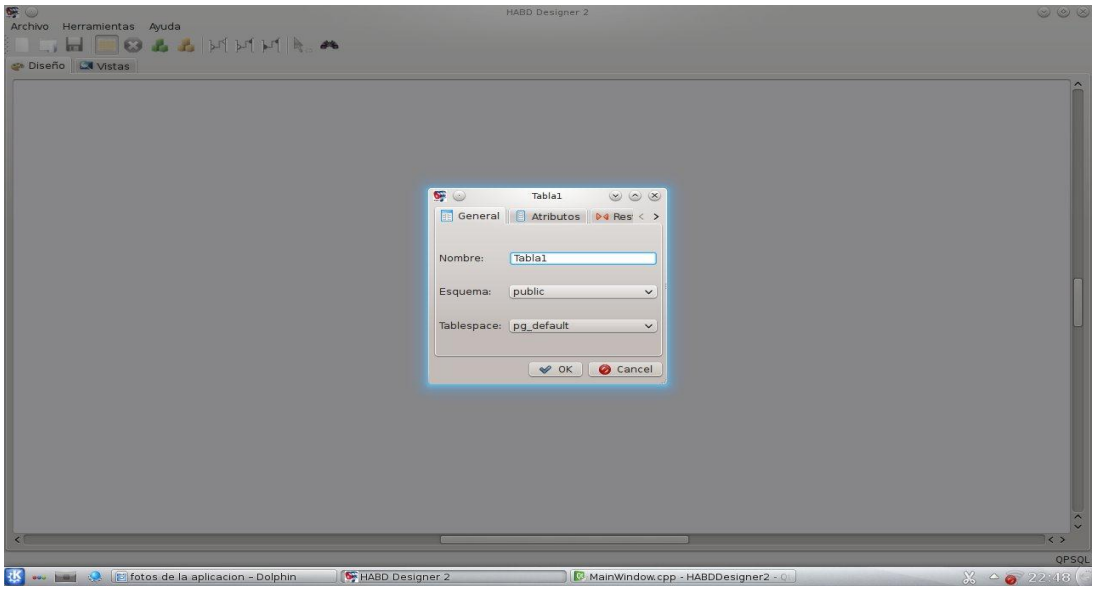


Figura 14. Crear Tabla.

Conclusiones del capítulo

En este capítulo se identificaron 15 Historias de Usuario, en las cuales se agrupan un total de 39 requisitos funcionales, se planificó su implementación en 5 iteraciones comenzando por las Historias de Usuario de mayor prioridad del negocio. Se confeccionaron 13 tarjetas CRC. Se describe la arquitectura a través de los patrones de diseño GRASP y GOF.

CAPÍTULO 3

VALIDACIÓN Y PRUEBA

Introducción

Con el objetivo de determinar si el resultado de un producto es el que se espera, se hace necesario someterlo a pruebas, permitiendo estas la determinación de su calidad, mediante la aplicación de técnicas experimentales o la ejecución de un programa que integra las diferentes fases del ciclo de desarrollo dentro de la ingeniería de *software*, todo esto con el propósito de descubrir errores. En este capítulo se analizan las estrategias de pruebas definidas por la metodología XP, el método que se utilizará para diseñar los casos de pruebas y los resultados arrojados por estas.

3.1. Estrategia de prueba

La etapa de pruebas es una de las fases en el ciclo de vida de los proyectos. No tienen el objetivo de prevenir errores sino de detectarlos. Se efectúan sobre el trabajo realizado y se deben encarar con la intensión de detectar la mayor cantidad de posibles errores [44].

La metodología XP fracciona las pruebas del sistema en dos grandes grupos, las cuales son **unitarias** y **aceptación** o **funcionales**. Las Unitarias se realizan para controlar el funcionamiento de pequeñas porciones de código, generalmente son llevadas a cabo por los mismos programadores ya que son los que conocen el código con mayor detalle, simplificando la tarea de elaborar un conjunto de datos de

prueba para testearlo. Las de aceptación se realizan sobre el producto terminado e integrado, están concebidas para que un usuario final sea quien detecte posibles errores.

Existen dos tipos de pruebas de Aceptación, las cuales son Alfa y Beta, a continuación se describe cada una de ellas:

La prueba alfa: se realiza por un cliente en un entorno controlado por el equipo de desarrollo. Para que tengan validez, se debe primero crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto, se procede a realizar las pruebas y a documentar los resultados [45].

La prueba beta: se realizan en las instalaciones propias de los clientes. Se realizan copias del sistema y es distribuido para cada uno de ellos. Cada usuario realizará sus propias pruebas y documentará los errores que encuentre, así como las sugerencias que crea conveniente realizar para que el equipo de desarrollo tenga en cuenta al momento de analizar las posibles modificaciones. Cuando el sistema tenga un solo cliente, las pruebas de aceptación se hacen de común acuerdo con éste y se definen los aspectos a probar y la forma de informar resultados.

El estudio antes realizado permitió aplicar al *plugin* la estrategia de prueba de aceptación de tipo alfa, ya que estas presentan mayor importancia que las unitarias, debido a que representan la satisfacción del cliente con el producto desarrollado, el final de una iteración y el comienzo de la siguiente.

3.2. Método y técnica seleccionada

Las pruebas unitarias utilizan la técnica de caja blanca o de caja negra, mientras que las de aceptación las de caja negra. A continuación se especifica el método seleccionado correspondiente al tipo de prueba a aplicar:

Caja Negra: Se refiere a las pruebas que se llevan a cabo sobre la interfaz del *software* donde los casos de prueba pretenden demostrar que las funciones del *software* son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. Se centran principalmente en los requisitos funcionales del *software*. Durante las

iteraciones, las HU seleccionadas, serán traducidas a pruebas. En ellas se especifican, desde la perspectiva del cliente, los escenarios para probar que una HU ha sido implementada correctamente. El objetivo final de las pruebas es garantizar que los requerimientos han sido cumplidos y que el sistema es aceptable [46].

Métodos de las pruebas de Caja Negra:

- Técnica de prueba basada en gráficos**:** En la técnica se debe entender los objetos que se modelan en el *software* y las relaciones que conectan a estos, tales como objetos de datos, objetos de programa como módulos o colecciones de sentencias del lenguaje de programación.
- Partición equivalente: Divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. En otras palabras, este método intenta dividir el dominio de entrada de un programa en un número finito de clases de equivalencia, de tal modo que se pueda asumir razonablemente que una prueba realizada con un valor representativo de cada clase es equivalente, a una prueba realizada con cualquier otro valor de dicha clase.
- Análisis de Valores Límites: El análisis de valores límite (AVL) es una técnica de diseño de casos de prueba que completa a la partición equivalente. En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL lleva a la elección de casos de prueba en los extremos de la clase [47].

El estudio antes concluido arrojó como resultado que se decide utilizar para determinar que el sistema desarrollado cumple con las funciones propuestas, las pruebas de caja negra y empleando la técnica de partición equivalente.

3.3. Casos de pruebas basados en Historias de Usuario

Para desarrollar un *software* de calidad y libre de errores, los casos de prueba son muy importantes. Un caso de prueba bien diseñado tiene gran posibilidad de llegar a resultados más fiables y eficientes. El diseño de las pruebas especifica los detalles del método de prueba para una característica del *software* e identifica las pruebas correspondientes.

Un caso de prueba es un conjunto de acciones con resultados y salidas previstas basadas en los requisitos de especificación del sistema [48].

Se pueden obtener varios casos de prueba para determinar que una funcionalidad es completamente satisfactoria. Debe existir al menos un caso de prueba para cada HU. Se definieron 15 casos de pruebas correspondientes a las 15 HU, los cuales se realizan con el objetivo de determinar que una funcionalidad ha sido implementada satisfaciendo las necesidades del cliente y demostrar que sus funciones son operativas. A continuación se muestra el caso de prueba asociado a la historia de usuario Gestionar Tabla, en el mismo se evidencian los escenarios Insertar tabla con datos correctos y datos incorrectos.

Tabla 10: Caso de prueba realizado a la aplicación.

Escenario	**Descripción**	**V1**	**V2**	**V3**	**V4**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Crear Tabla con datos correctos.	El sistema muestra la interfaz que permite crear la tabla.	V V V V	V V V V	V I V I	V I I V	Se crea la tabla con los datos introducidos.	1- El sistema muestra una interfaz que permite introducir el nombre de la tabla, los atributos, las restricciones y los índices. 2- El usuario introduce los datos de forma correcta. 3- El sistema crea la tabla con los datos introducidos.
EC 1.2 Crear Tabla con datos incorrectos. (“”,ñ,111)	El sistema muestra la interfaz que permite crear la tabla.	V I I I	I V I I	I I V I	I I I V	El sistema muestra un alerta de error especificando cual parámetro es incorrecto.	1- El sistema muestra una interfaz que permite crear la tabla. 2- El Usuario introduce incorrectamente los datos y hace clic en

							aceptar. 3- El sistema muestra una alerta de error. No se crea la tabla.

Después de haber sido aplicados los casos de pruebas al *plugin* implementado, basados en las 15 HU, en 5 iteraciones de desarrollo, se detectaron en la primera iteración de desarrollo 3 no conformidades, las mismas fueron resueltas en un período de tiempo de 3 días. En la segunda iteración de desarrollo se detectaron 5 no conformidades, siendo solucionadas en 2 días. Al concluir la tercera iteración de desarrollo y aplicar las pruebas, se detectaron 6 no conformidades, siendo resueltas en 4 días. Concluyendo la cuarta iteración de desarrollo se detectó 1 no conformidad, la cual fue resuelta en breve período de tiempo. Finalizada la quinta iteración de desarrollo y última, se detectaron 3 no conformidades, resolviéndose las mismas en 2 días. Es importante destacar que por cada iteración de desarrollo fue necesario aplicar solo dos iteraciones de pruebas. En la Figura 16 se muestra una gráfica con la cantidad de no conformidades que se detectaron por cada iteración.

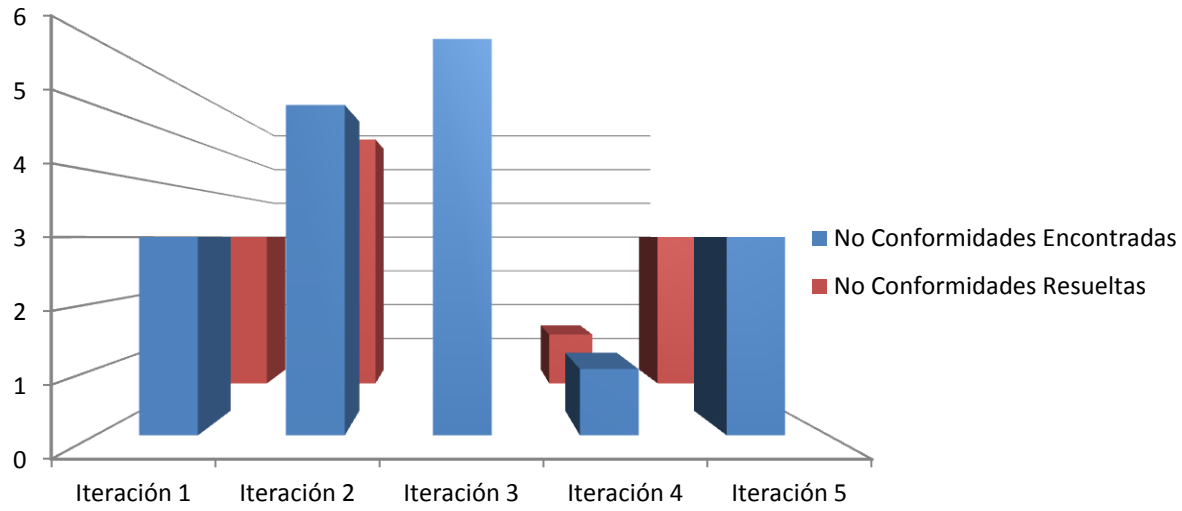


Figura 15: Resultados de las pruebas aplicadas.

Conclusiones del capítulo

El estudio realizado sobre estrategias de pruebas, ayudó a determinar como la más conveniente a aplicar al *plugin* implementado, las pruebas de aceptación, haciendo uso del método de caja negra con la técnica de partición equivalente. Al aplicarle la misma se obtuvieron 18 no conformidades las cuales fueron solucionadas en 2 iteraciones de pruebas en las 5 iteraciones de desarrollo.

Conclusiones generales

Una vez culminada la investigación se puede afirmar que los objetivos y tareas trazadas fueron cumplidos satisfactoriamente, permitiendo:

- El estudio de las tecnologías y herramientas posibilitó seleccionar las más idóneas para el desarrollo del *plugin* capaz de diseñar bases de datos MySQL, SQLite y PostgreSQL, logrando su correcta integración con la herramienta HABD.
- Se diseñaron 13 tarjetas CRC que permitieron agrupar por funcionalidades las clases que guiaron el proceso de implementación del plugin diseñador de bases de datos v2.0.
- Se implementaron los 39 requisitos funcionales agrupados en 15 Historias de Usuario, que permitieron obtener un *Plugin* diseñador de bases de datos v2.0 que integrado a la herramienta HABD permite realizar el diseño de bases de datos MySQL, PostgreSQL y SQLite.
- Se validaron las 15 Historias de Usuario mediante un número igual de casos de pruebas, verificando el correcto funcionamiento de la aplicación.

Recomendaciones

- Implementar una funcionalidad que le permita al usuario deshacer los cambios hechos en la aplicación.
- Implementar una funcionalidad que permita identificar las relaciones entre las tablas según el tipo de relación.

Bibliografía referenciada

1. Matos, R.M., *Diseño de bases de datos.* La Habana: CUJAE, 1997.
2. Date, C.J., *Introducción a los sistemas de bases de datos*2001: Pearson Educación.
3. Sicilia, M.A. *Funciones de los Sistemas Gestores de Bases de Datos*. 2008 2008 [cited 2013; Available from: http://cnx.org/content/m17543/latest/.
4. Oracle Corporation. *About Oracle*. 2006; Available from: http://www.oracle.com/us/corporate/index.html.
5. IBM Corporation. *IBM DB2 database software*. 2012; Available from: http://www-01.ibm.com/software/data/db2/.
6. Borland Software Corporation. *InterBase*. 2009; Available from: www.embarcadero.com/products/interbase.
7. Microsoft. *SQL Server*. 2012; Available from: http://www.microsoft.com/en-us/sqlserver/default.aspx.
8. dBase LLC. *dBase*. 2012; Available from: http://www.dbase.com/.
9. PostgreSQL. *PostgreSQL*. 2012; Available from: http://www.postgresql.org.
10. FirebirdSQL Fundation. *Firebird*. 2012; Available from: http://www.firebirdsql.org/.
11. SQLite. *SQLite*. 2000; Available from: http://www.sqlite.org/.
12. Sun Microsystems, *MySQL.* 1998.
13. Orallo, J.H., *La disciplina de los sistemas de bases de datos.* Historia, situación actual y perspectivas, 2002.
14. Peña Cuellar, G.E. *¿Qué es SQLite?* 2012.
15. Deitel, H.M. and P.J. Deitel, *Cómo programar en C/C++ y Java*2004: Pearson Educación.
16. González, L., *Lenguajes del lado servidor y del lado cliente* in *Proyecto EATS*2007.
17. Langer, S., N. Charboneau, and T. French, *DCMTB: a virtual appliance DICOM toolbox.* Journal of Digital Imaging, 2010. **23**(6): p. 681-688.
18. Nikitin, O., *Storage, processing and visualization data system of drifter observations of surface currents in the World Ocean.* Russ. J. Earth Sci, 2012. **12**.

19. Huettig, F., J. Rommers, and A.S. Meyer, *Using the visual world paradigm to study language processing: A review and critical evaluation.* Acta psychologica, 2011. **137**(2): p. 151-171.
20. PostgreSQL Global Development Group *pgDesigner2*. 2007.
21. Ávila Arteaga, C. and M. Ramírez Hernández, *Análisis y diseño del sistema de control de servicio social de la UAEH.* 2005.
22. Vargas, R. and P. Alvarado *Modelo de base de datos con ER/studio*. 2006.
23. Castellanos, Y.P. and M.A.M. Martínez, *Plugin diseñador de bases de datos para la herramienta de administración de bases de datos HABD*, 2012, Universidad de las Ciencias Informáticas.
24. González Hernández, Y., *PROPUESTA DE METODOLOGIA PARA EL DASARROLLO DE ALMACENES DE DATOS EN DATEC*, 2012, Universidad de las Ciencias Informáticas.
25. Joskowicz, J., *Reglas y prácticas en Extreme Programming.* Universidad de Vigo. España, 2008.
26. Visual Paradigm. *Visual Paradigm web site*. 2007; Available from: http://www.visual-paradigm.com/.
27. Barrios , M. *Herramientas CASE*. 2011.
28. Free Download Manager. *Paradigma visual para UML (Plataforma Java) (Visual Paradigm for UML [Java Platform]) 6.0*. 2007; Available from: http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%5Bcuenta_de_Plataforma_de_Java_14715_p/.
29. Castillo Banco, L. *LENGUAJES DE PROGRAMACION*. 2009.
30. García de Jalón, J., et al., *Aprenda C++ como si estuviera en primero*, 1998. p. 87.
31. AmericaTI.com, *Ventajas y Desventajas: Comparación de los Lenguajes C, C++ y Java*, 2006.
32. Villalobos, R. *Qt Creator, desarrollando aplicaciones rápidamente.* 2009.
33. Alemany Garrido, S., *Introducción a Qt. Programación gráfica en C++ con Qt4*, 2009.
34. Ministerio de Educación, C.y.D. *Sistema de control de versiones: subversion*. 2008.
35. PRESSMAN, R.S., *Ingeniería Del software un enfoque practico. quinta edición. Madrid. MacGraw-Hill*, 2002, interamericana de España.
36. José, E.A.S.P.L. and H. Canós, *Mejorando la gestión de historias de usuario en eXtreme Programming.*
37. de Benito Crosetti, B., *Herramientas para la creación, distribución y gestión de cursos a través de Internet.* Edutec: Revista electrónica de tecnología educativa, 2000(12): p. 2.
38. Zapata, M.A. *Diseño estructural: Diagrama de clases*. 2006.

39. Bedoya, A. *Patrones de Diseño, ¿qué son y para qué sirven?* 2011; Available from: http://geektheplanet.net/5461/patrones-de-diseno-%C2%BFque-son-y-para-que-sirven.xhtml.
40. Rivera Riera, L.A., T. Alulema, and J. Pablo, *Integración de la Metodología Agil XP el Estudio de Windows Communication Foundation como Aternativa Metodológica para el Desasrrollo de Software Orientado a Servicios. Caso Práctico: CIPFIE-ESPOCH.* 2010.
41. Larman, C., *UML y patrones*1999: Pearson.
42. Garcia, J. *Diseño de Software Orientado a Objetos*. 2005.
43. Ros, M.Z., *Patrones en elearning. Elementos y referencias para la formación.* RED. Revista de Educación a Distancia, 2011(27): p. 1-10.
44. Suárez, P. and C. Fontela, *Documentación y pruebas Antes del paradigma de objetos*, 2003.
45. Torres, I.N., *Pruebas de Aceptación y Piloto.* Serie Científica, 2008. **1**(7).
46. Juristo, N., M. Moreno Ana, and S. Vegas, *Técnicas de evaluación de Software.* Visitar http://is. ls. fi. upm. es/udis/docencia/erdsi/Documentacion_Evaluacion_7. pdf, 2004.
47. Cabezas, P. and E. Tatiana, *Desarrollo de un sistema para voto electrónico y emisión de resultados en procesos electorales de la Escuela Politécnica Nacional.* 2010.
48. Aristegui, J.L., *Los casos de prueba en la prueba del software.* Lámpsakos, 2010(3): p. 27-34.

Bibliografía consultada

1. ABRAN, A. y BOURQUE, P. *SWEBOK: Guide to the software engineering Body of Knowledge.* IEEE Computer Society, 2004. ISBN 0769523307.
2. AMBLER, S. *Agile modeling: effective practices for extreme programming and the unified process.* Wiley, 2002. ISBN 047127190X.
3. AmericaTI.com, Ventajas y Desventajas: Comparación de los Lenguajes C, C++ y Java, 2006.
4. BECK, K. y ANDRES, C. *Extreme programming explained: embrace change.* Addison-Wesley Professional, 2004. ISBN 0321278658.
5. COLLINS-SUSSMAN, B.; FITZPATRICK, B.*, et al. Version control with subversion.* O'Reilly Media, 2007. ISBN 0596004486.
6. Castellanos, Y.P. and M.A.M. Martínez, Plugin diseñador de bases de datos para la herramienta de administración de bases de datos HABD, 2012, Universidad de las Ciencias Informáticas.
7. Cristiá Álvarez, Aldo, Ortiz Valmaseda, Marcos Luis y Ortiz Vázquez, Yudisney. Propuesta del gestor de bases de datos cubano basado en PostgreSQL. [en línea] Vol.: 3, No. 9, (2010). [Consulta el 25 de octubre de 2012]. Disponible en http://publicaciones.uci.cu/index.php/SC.
8. Carolina Martínez, Prof. Ivette. *Modelo Conceptual / Modelo de Dominio.* Caracas : Universidad Simon Bolivar. Ingeniería del Software, 2000.
9. DAHAN, U. *Employing the Domain Model Pattern.* 2009, Disponible en: http://msdn.microsoft.com/en-us/magazine/ee236415.aspx.
10. Date, C.J., *Introducción a los sistemas de bases de datos*2001: Pearson Educación.
11. Deitel, H.M. and P.J. Deitel, *Cómo programar en C/C++ y Java*2004: Pearson Educación.
12. Delgado Dapena, Martha D. Definición del modelo del negocio y del dominio utilizando UML.
13. FALCÓN, Y. F. y PAJARES, R. R. Plugin para el particionado de tablas en la herramienta de administración HABD. *Serie Científica*, 2012, vol. 5, nº 8,
14. González, L., *Lenguajes del lado servidor y del lado cliente* in *Proyecto EATS*2007.
15. Garcia, J. Diseño de Software Orientado a Objetos. 2005.
16. GUTIÉRREZ, J. J. *¿Qué es un framework web?* 2006,
17. Garzón Pérez, Teresa. *Sistemas Gestores de Bases de Datos.* 2010. 1988-6047.

18. Gutíerrez J.J, Escalona M.J, Mejías M., Torres, J. Pruebas del sistema en Programación Extrema. Sevilla: s.n.

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. José, E.A.S.P.L. and H. Canós, Mejorando la gestión de historias de usuario en eXtreme Programming.

21. Larman, C., UML y patrones1999: Pearson.

22. LORENA GUERRERO, G.; ROCIO ERAZO, L.*, et al. DESARROLLO DE SOFTWARE DEFINICIÓN GENERAL DEL PROCESO.* 2011, Disponible en: http://artemisa.unicauca.edu.co/~leydierazo/ProyectoSW/ProcesoDeDesarrollo.pdf.

23. Langer, S., N. Charboneau, and T. French, *DCMTB: a virtual appliance DICOM toolbox.* Journal of Digital Imaging, 2010. 23(6): p. 681-688.

24. MULLER, M. M. y TICHY, W. F. Case study: extreme programming in a university environment. En *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on. 2001*. p. 537-544.

25. Matos, R.M., *Diseño de bases de datos.* La Habana: CUJAE, 1997.

26. Mendoza Sánchez, María A. Metodologías de desarrollo de Software [en línea] 7 de junio de 2004. [Consultado el 10 de enero de 2013.] ftp://ucistore.uci.cu/documentacion/Ingenieria%20Software/Metodologias/caracteristicas%20breves%20de%20RUP,%20XP,%20MSF.pdf.

27. Mesa Reyes, Yunior y Vázquez Ortiz, Yudisney. Sistemas de Bases de Datos. Espacio de comunicación e intercambio para la comunidad técnica cubana de PostgeSQL. PostgreSQL. Ciudad de la Habana, Cuba: s.n., 2011. pág. 7. 1994-1536.

28. Nikitin, O., *Storage, processing and visualization data system of drifter observations of surface currents in the World Ocean.* Russ. J. Earth Sci, 2012.

29. Orallo, J.H., *La disciplina de los sistemas de bases de datos.* Historia, situación actual y perspectivas, 2002.

30. PRESSMAN, R. S. *Ingeniería Del software un enfoque practico. quinta edición. Madrid. MacGraw-Hill.* interamericana de España, 2002,

31. PRESSMAN, R. S. y TROYA, J. M. *Ingeniería del software.* McGraw Hill, 1988. ISBN 8476152221.
32. PostgreSQL. [en línea] 30 de septiembre 2011. [Consultado el 5 de febrero de 2013]. Disponible en http://wiki.postgresql.org/wiki/SQL/MED.
33. Peña Cuellar, G.E. *¿Qué es SQLite?* 2012.
34. Ros, M.Z., *Patrones en elearning. Elementos y referencias para la formación.* RED. Revista de Educación a Distancia, 2011(27): p. 1-10.
35. ROCHA, M.; LUSCOMBE, N.*, et al.* Advances in Intelligent and Soft Computing: Preface. *Advances in Intelligent and Soft Computing*, 2012, vol. 154, nº p. V-VI. ISSN 1867-5662.
36. Rodríguez Guadarrama, Addiel y Mazorra, Thaymí. Análisis y diseño de una herramienta web para la gestión de la información. Universidad de la Ciencias Informáticas (UCI). 2010.
37. SQLite. *SQLite*. 2000; Available from: http://www.sqlite.org/.
38. Sicilia, M.A. *Funciones de los Sistemas Gestores de Bases de Datos*. 2008 2008 [cited 2013; Available from: http://cnx.org/content/m17543/latest/.
39. SCHMIDT, D. C.; STAL, M.*, et al. Pattern-oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2.* Wiley, 2000.
40. SCCHENONE, M. Diseño de una metodología ágil de desarrollo de software. *Buenos Aires. Universidad de Buenos Aires*, 2004, nº.
41. SOMMERVILLE, I. *Ingeniería del software.* Pearson Educación, 2005. ISBN 8478290745.
42. Sun Microsystems, *MySQL.* 1998.
43. Sitio Oficial de la Comunidad PostgreSQL [en línea] 21 de enero 2010. [Consultado el 5 de febrero de 2013] http://www.postgresql.org/docs/9.1/static/extend-extensions.html
44. VAZQUEZ ORTÍZ, Y. PostgreSQL Empresarial Cubano: Un sistema diseñado acorde a las necesidades de las entidades del país. 2012, nº Disponible en: http://ccia.cujae.edu.cu/index.php/siia/siia2012/paper/view/2411.
45. VAZQUEZ ORTÍZ, Y. y ORTÍZ VALMASEDA, M. L. Propuesta del gestor de bases de datos cubano basado en PostgreSQL. *Serie Científica*, 2010, nº Disponible en: http://publicaciones.uci.cu/index.php/SC/article/view/314%E2%80%8E.
46. *Vargas, R. and P. Alvarado Modelo de base de datos con ER/studio. 2006.*
47. Visual Paradigm. Visual Paradigm web site. 2007; Available from: http://www.visual-paradigm.com/

48. Yague, Agustin y Garbajosa, Juan, Actas de los talleres de las jornadas del software y bases de datos. Madrid: s.n., 2009. Vols. Vol 3, Num4. Universidad Politécnica de Madrid (UPM).

49. Zamudio, Esmeralda Villegas y Méndez, Alejandra Virrueta. Investigación documental. Metodologías de desarrollo de software. Instituto tecnológico superior de Apatzingán, s.n., 2010.

Glosario de Términos

API: Interfaz de programación de aplicaciones o API (del inglés *Application Programming Interface*).

Esquema: Los esquemas son usados en las bases de datos para separarlas de manera lógica dándonos la opción de tener en un momento determinado corriendo un sistema real y uno de prueba dentro de la misma base pero separados mediante esquemas, podemos también tener en dos esquemas distintos los mismos nombres de tablas sin que esto nos represente un error.

GUI: acrónimo inglés de *Graphical User Interface* (Interfaz Gráfica de Usuario). Es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

HABD: Herramienta de Administración de Bases de Datos.

Hardware: Se refiere a todos los aparatos, tarjetas (circuitos impresos electrónicos), y demás objetos físicos de los que está compuesto un PC. El *hardware* abarca todas las piezas físicas de un ordenador (disco duro, placa base, memoria, tarjeta aceleradora o de vídeo, lectora de CD, microprocesadores, entre otras). Sobre el *hardware* es que corre el *software* que se refiere a todos los programas y datos almacenados en el ordenador.

Host: Es el término usado en informática para referirse a las computadoras conectadas a una red, que proveen y utilizan servicios de ella.

Índices: Un índice es un subconjunto ordenado de las columnas de una tabla, con una entrada que apunta a la tupla correspondiente.

Lenguaje procedural: Es un lenguaje donde el usuario especifica que datos se necesitan y como obtenerlos.

Licencia GPL: La licencia GPL (*General Public License*) obliga a incluir el código fuente en su distribución, siendo imposible cambiar la licencia al programa, al distribuirlo tal cual o modificado.

Multiplataforma: Es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de *software*, que puedan funcionar en diversas plataformas. Una plataforma es una combinación de *hardware* y *software* usada para ejecutar aplicaciones.

Plugin: Módulo de *hardware* o *software* que añade una característica o un servicio específico a un sistema más grande.

***Software*:** Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

Software Libre: es el *software* que, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente.

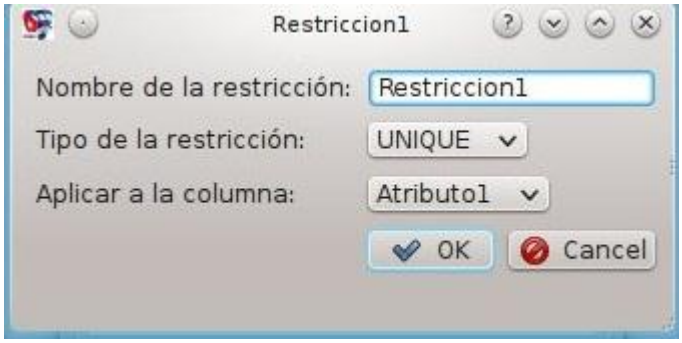
Script: Un *script* es un guión o conjunto de instrucciones. Permiten la automatización de tareas creando pequeñas utilidades. Es muy utilizado para la administración de sistemas UNIX. Son ejecutados por un intérprete de línea de comandos. Usualmente son archivos de texto.

UML: Lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

Vacuum: Es el proceso en el cual se realiza una reorganización de datos a nivel físico.

Anexos

Anexo 1: Interfaz para crear una restricción.



Anexo 2: Interfaz de visualiza la restricción creada.

Anexo 3: Interfaz que la creación de un esquema.