Universidad de las Ciencias Informáticas

FACULTAD 6

Título: *Plugin* Gestor de Tablas para la Herramienta de Administración de Base de Datos HABD

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

Autores: Yudelis Matos Martínez
Osman de Armas Sánchez

Tutores: Ing. Beatriz Lara Osorio
Ing. Mikel Díaz Hernández

La Habana, Cuba, 2013
"Año 55 de la Revolución"


"No existe una manera fácil. No importa cuán talentoso seas, tu talento te va a fallar si no lo desarrollas. Si no estudias, si no trabajas duro, si no te dedicas a ser mejor cada día."
Will Smith

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos 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 _____.

Yudelis Matos Martínez

_____
Firma del Autor

Osman de Armas Sánchez

_____
Firma del Autor

Beatriz Lara Osorio

_____
Firma del Tutor

Mikel Díaz Hernández

_____
Firma del Tutor

Autores:

Yudelis Matos Martínez

e-mail: ymatosm@estudiantes.uci.cu

Osman de Armas Sánchez

e-mail: odearmas@estudiantes.uci.cu

Tutores:

Ing. Beatriz Lara Osorio

Especialidad de Graduación: Ingeniero en Ciencias Informáticas.

Años de experiencia en el tema: 1

Años de graduado: 1

e-mail: blara@uci.cu

Ing. Mikel Díaz Hernández

Especialidad de Graduación: Ingeniero en Ciencias Informáticas.

Años de experiencia en el tema: 1

Años de graduado: 1

e-mail: mdiazhdez@uci.cu

De Yudelís:

Las palabras más hermosas que pudiera decir para agradecer a todas las personas que me apoyaron todo este tiempo no son nada en comparación con el amor y respeto tan grande que siento por cada una de ellas. Espero entonces que estas líneas logren llegar a ustedes con la misma fuerza que causó su apoyo en mi corazón:

A mis padres *Olga y Rafael por siempre estar ahí cuando más los necesité, apoyándome en cada momento.*

A mis hermanos *Adonis, Ángel y Yuniel aunque me paso la vida peleando con ellos y siempre estamos "fajados" no se imaginan lo importante que fueron para mí todo este tiempo.*

A mi nene *Rober, tu amor, consideración, respeto y cariño marcaron mi vida y ayudaron a darme fuerzas para nunca rendirme ante situaciones difíciles, siempre supiste hacerme saber cuál era el camino correcto. Te quiero mucho.*

A mi familia *Yuyi, Cari, el lachi, Delvis, Ramón, mi abue "berecueto" en fin todos mis tíos y primos que nunca dijeron que no a nada aunque pareciera imposible y se preocuparon en todo momento por mí.*

A mi dúo *y compañero Osman, si hubiese que hacer este trabajo de nuevo elegiría volver a ser tu compañera, pues a pesar de todos los obstáculos nunca dudamos y nos apoyamos incondicionalmente.*

A mis amistades *Mari "la nana", gracias por estar en los momentos buenos y malos de la UCI. Por ser el hombro amigo donde de vez en cuando soltaba una lágrima y los consejos que me diste para seguir adelante. A la chula Glei por aguantar las muelas de desahogo aunque no fuera el mejor momento. A Yosbel, de no ser por ti muchas de nosotras no estuviéramos aquí, por los repasos de programación, los buenos consejos y por pasarte los cuatro años que nos conocemos alegrándome los días.*

A Migue, Frank, el Flaco, la lisi, la gorda, Niuvys, Eilen, el poti, fuki, Sheyla, Jennifer, Arianna, Kati, Ari, Ailsa, Yade, Leonel , Yoni, Luis Miguel en fin el piquete más loco de la UCI y los que desafortunadamente ya no están entre nosotros: los momentos más agradables en la universidad los pasé junto a ustedes.

A, Lisi, Oice, Paulina, Delfina y Vidal, a todos mis vecinos, por preocuparse tanto por mí.

***A mís tutores** Beatriz Lara y Mikel Hernández por hacer lo mejor que pudieron por nosotros. En especial Mikel que nos dedicó su tiempo dando muy buenos consejos.*

A los integrantes del departamento PostgreSQL, al tribunal y oponente por los consejos y sugerencias.

A todos los que de una forma u otra cooperaron para convertirme en la persona que soy hoy.

Gracias

De Osman:

A mis padres *por hacer realidad mi sueño, por siempre apoyarme en todas mis decisiones. Por demostrarme que si se puede, por darme tanto amor y cariño, por depositar en mi toda su confianza, por su sacrificio, su dedicación y comprensión, por la educación y los principios que inculcaron en mi.*

A mi familia *por todo el amor y el apoyo que siempre he recibido de ellos. Por preocuparse tanto por mi, en especial a mis abuelos que siempre han estado presente cuando los he necesitado y me han brindado su apoyo en todo momento.*

A mi abuelo *Luis el cual no se encuentra físicamente pero siempre lo llevo en mi corazón y le agradezco ya que me enseño muchas cosas de la vida y me dio fuerzas para seguir siempre adelante y luchar por las cosas que uno quiere.*

A mi novia *la cual ha sabido aguantarme en este tiempo que llevamos juntos, por darme siempre su apoyo y ayudarme a cambiar en la vida como persona.*

A todos mis amigos *por estar conmigo en los buenos y malos momentos, los que empezamos desde primero luchando por seguir y a otros que hoy no se encuentran aquí: Panchy, Alain, Osniel, Yoan, Eudel, Raul Marinas, Jose Antonio, el Poty, el Yabo, Marquito, Ernesto, Raul Alejandro, Aviles, Liandry, Marvel, Jorgito, Frank, Yonnis, Marcos Michel, Michel, Sureny, Yuned, Lienny, Audrey y mi edificio y grupo en general.*

A mi dúo de tesis *por todas las horas dedicadas en la realización de este trabajo, le agradezco enormemente que haya compartido este momento tan especial conmigo del cual hemos aprendido mucho.*

A mis suegros *los cuales se han portado súper bien conmigo y han sabido sobrellevarme en todo momento.*

A todos los que de una forma u otra cooperaron para convertirme en la persona que soy hoy. *Gracias*

De Yudelis:

Por ser la luz que guía mi vida, por dar hasta lo imposible para que su "niña" llegara hasta aquí.

Mamí: *porque eres la persona más especial que existe en este mundo, aunque a veces no sea capaz de demostrarlo como tú te lo mereces, por todas las noches en vela que pasaste pensando y rezando para que todo me saliera bien. Siempre me ha resultado difícil decirte "TE QUIERO", hoy quiero con este triunfo decirte TE AMO, TE ADORO, eres y serás siempre todo para mí.*

Papí: *porque se lo que han significado para ti estos últimos cinco años y lo orgulloso que siempre demostraste estar de mí, por creer en la niña malcría de la casa a la que no se le dice que no a nada y brindarme tu apoyo incondicional , TE QUIERO MUCHO.*

De Osman:

Dedico este trabajo a las personas más importantes en mi vida:

***A mi mamá** que es la mejor madre del mundo, por su ternura, por su amor, por su dedicación, por ser compresiva y tener paciencia con su hijo malcriado que la ama con la vida, por siempre aconsejarme y guiarme por el buen camino, a ti mamita te dedico mi tesis con todo mi amor para que te sientas orgullosa de mi y veas que toda la confianza que depositaste en mi no fue en vano.*

***A mi papá** que es el hombre más importante en mi vida, por enseñarme a luchar por lo que quiero, por los principios y la educación que siempre me ha inculcado, a ti pipo te dedico mi tesis, porque sé que me quieres tanto como yo a ti, porque sé que siempre estarás ahí para mí cuando lo necesite y veas que toda la confianza que depositaste en mi no fue en vano, ahí va nuestro sueño hecho realidad.*

***A mis abuelos**, dios quiera que siempre estén a mi lado, esto también es parte de su esfuerzo y sacrificio en la vida.*

RESUMEN

Las herramientas de administración de base de datos (BD) proveen funcionalidades que permiten a los usuarios el trabajo con BD. Algunas de estas aplicaciones tienen deficiencias en el manejo de los datos y componentes de las tablas por las que están compuestas, resultando complicada la gestión de las mismas. En el departamento *PostgreSQL* del Centro de Tecnología de Gestión de Datos (DATEC) se encuentra en desarrollo la Herramienta de Administración de base de datos (HABD), la cual presenta una arquitectura basada en *plugin.* Esta herramienta cuenta con funcionalidades que permiten gestionar tablas mediante líneas de comando, por lo que se necesita tener conocimientos para construir tablas a través de consultas *SQL*. El presente trabajo tiene como objetivo desarrollar un *plugin* para HABD que permita la gestión de tablas de una BD a través de una interfaz visual, lo que garantiza que la gestión de tablas en HABD se realice de una manera más sencilla y cómoda para el usuario final. Luego de realizar una investigación en busca de herramientas que realizaran este procedimiento, se seleccionó *PgAdmin III* y el *EMS for PostgreSQL* como guía para el desarrollo del trabajo debido a las características que poseen sus editores de tablas. Se identificaron las principales funcionalidades a implementar en el *plugin* y las tecnologías a utilizar. A través de la aplicación, el usuario podrá realizar acciones para crear, eliminar, actualizar, filtrar y modificar los componentes de una tabla, cumpliendo con las expectativas de los usuarios que interactúen con la misma.

Palabras claves: gestión de tablas, HABD, herramientas de administración de base de datos, *plugin*.

Índice de Contenido

Índice de Figuras

Índice de Tablas

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC), posibilitan crear herramientas informáticas para almacenar grandes cantidades de información en diferentes instituciones de la sociedad. Estas herramientas se enfrentan al reto de ofrecer soluciones para almacenar la creciente avalancha de datos que una institución puede generar y evitar la pérdida de estos.

Como alternativa para el trabajo en las entidades del negocio surgen las BD las cuales ayudan a satisfacer los requerimientos de información de una organización (1), pues brindan una forma segura para almacenar, controlar, recuperar, ordenar y analizar los datos. Las BD se clasifican de acuerdo al modelo de administración de datos utilizado. El modelo relacional se utiliza para modelar problemas reales y administrar un conjunto de datos relacionados entre sí.

Para gestionar BD relacionales existen diferentes Sistemas Gestores de Base de Datos (SGBD), que constituyen un conjunto de programas para crear, dar mantenimiento y asegurar la integridad, confidencialidad y seguridad de las BD. Los SGBD ofrecen a las organizaciones una mejor forma para archivar, buscar, gestionar y compartir información (2). Uno de los SGBD relacionales más conocidos es *PostgreSQL*, utilizado en todo el mundo por ser un gestor de código abierto. Al incorporar como características adicionales las restricciones, reglas, herencia y funciones, permite a los usuarios extender las funcionalidades que realiza este sistema y lo convierte en un potente gestor de BD objeto - relacional.

Cuba no se encuentra ajena a los cambios tecnológicos, por lo que promueve el uso de aplicaciones de código abierto. En los últimos años ha realizado grandes esfuerzos en el orden económico con el objetivo de preparar profesionales cada vez más competentes y enfrentar los desafíos del mundo de la informática, ejemplo de ello es la Comunidad Técnica Cubana de *PostgreSQL*. Esta comunidad utiliza el gestor *PostgreSQL* como SGBD para el desarrollo de aplicaciones en los Centros de desarrollo de software del país, lo que contribuye al fortalecimiento de la soberanía tecnológica cubana.

La Universidad de las Ciencias Informáticas (UCI) se destaca en la creación de aplicaciones que hacen uso del SGBD *PostgreSQL*. En ella se encuentra el Centro de Tecnología de Gestión de Datos (DATEC), el cual consta con un departamento denominado PostgreSQL especializado en el trabajo con este gestor, pues lo utiliza como alternativa factible para el desarrollo de soluciones informáticas.

Dentro de las herramientas en desarrollo en el departamento se encuentra la Herramienta de Administración de Base de Datos (HABD), la cual permite a sus desarrolladores incorporar *plugins* para extender sus funcionalidades. Uno de los *plugins* desarrollados es el Editor de Consultas *SQL* (*Structure Query Language* por sus siglas en inglés), que permite efectuar sentencias (*select, insert, delete y update*), directamente a las tablas que el usuario desee gestionar apoyándose en el completamiento de código mientras se escriben dichas sentencias. La gestión se ejecuta a través de líneas de comando, lo que imposibilita realizar cambios específicos a las tablas y sus componentes de manera sencilla y ordenada, convirtiéndolo en un proceso engorroso para los usuarios que interactúan con HABD.

Por lo antes planteado surge el siguiente **problema a resolver**: ¿Cómo visualizar el proceso de gestión de tablas en la herramienta de administración de base de datos HABD?

Para dar solución al problema antes planteado se define como **objeto de estudio**: Proceso de gestión de tablas en BD relacionales, enmarcado en el **campo de acción:** Gestión de tablas en las BD *PostgreSQL* en HABD.

Se define como **objetivo general de la investigación**: Desarrollar un *plugin* con interfaz gráfica que permita el proceso de gestión de tablas en HABD.

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

- Analizar las herramientas para la administración de BD que permiten la gestión de tablas.
- Diseñar el *plugin* Gestor de Tablas para la herramienta HABD.
- Implementar el *plugin* Gestor de Tablas para la herramienta HABD.
- Evaluar el funcionamiento del *plugin* Gestor de Tablas para la herramienta HABD a través pruebas funcionales.

Para dar cumplimiento a los objetivos planteados se trazan las siguientes **Tareas de la Investigación:**

- Revisión bibliográfica sobre herramientas de administración en BD que utilizan el gestor *PostgreSQL*.
- Caracterización de las herramientas de administración de BD existentes en el mundo que permiten la gestión de tablas.

- Caracterización de las metodologías, herramientas y tecnologías a utilizar en el desarrollo del *plugin* para la gestión de tablas.
- Identificación de las funcionalidades del *plugin* a implementar para la herramienta HABD.
- Confección del modelo de diseño a partir de las funcionalidades identificadas.
- Definición y selección de los estándares de codificación a utilizar en la implementación del *plugin*.
- Implementación de las funcionalidades identificadas para el proceso de gestión de tablas en la herramienta HABD.
- Implementación de las funcionalidades para la integración del *plugin* con la herramienta HABD.
- Definición del método de prueba a utilizar en la validación del *plugin* Gestor de Tablas.
- Diseño de los casos de prueba para la validación del *plugin*.
- Aplicación de los casos de pruebas para validar que la aplicación responde a las necesidades del usuario final.

La investigación en el documento está organizada en 3 capítulos conformados de la siguiente manera:

Capítulo 1: Fundamentación Teórica.

En este capítulo se realiza un estudio de los principales conceptos que guían la investigación haciendo énfasis en las herramientas de administración de BD que existen en el mundo y las características de los gestores de tablas que éstas poseen. Se realiza un estudio de las tecnologías en las que se apoyará la realización de este *plugin* de acuerdo a las tendencias actuales, encaminadas hacia el software libre.

Capítulo 2: Planificación y Diseño del *Plugin*.

En el presente capítulo se describen las características propuestas para la elaboración del *plugin*. Se representa a través del modelo de dominio los conceptos más significativos del negocio y las historias de usuario definidas según los requisitos identificados. Se seleccionaron los patrones de diseño y arquitectura para el desarrollo del *plugin*.

Capítulo 3: Implementación y Prueba del *Plugin*.

En este capítulo se define el modelo de implementación de acuerdo al diseño elaborado en el capítulo anterior, guiado por los estándares de codificación seleccionados. Para evaluar el correcto funcionamiento del sistema se describe el método de prueba y se elaboran los diseños de casos de prueba.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el capítulo se realiza un análisis de los distintos conceptos para lograr una mejor comprensión de la investigación. Para describir el marco teórico del trabajo, se elabora un análisis del estado del arte con respecto a las herramientas de administración de BD y sus características para el proceso de gestión de tablas. Se fundamenta la selección de la metodología, herramientas y tecnologías a utilizar en el transcurso del desarrollo del *plugin* Gestor de Tablas para HABD.

1.1 Definición de *plugin*

Según el **Dr. Harvey M. Deitel** los *plugins*: consisten en pequeñas aplicaciones que añaden alguna función a otros programas, habitualmente de mayor tamaño. Un programa puede tener uno o más *plugins* (3).

Ventajas del *plugin:*

- Permiten a los desarrolladores externos colaborar con la aplicación principal extendiendo sus funciones.
- Reducen el tamaño de la aplicación.
- Permiten separar el código fuente de la aplicación a causa de la incompatibilidad de las licencias de software.

Desventajas del *plugin*:

- No se puede ejecutar sino está integrado a una herramienta.
- Se diseñan para funcionalidades específicas.

Teniendo en cuenta lo antes expuesto se considera que los *plugins* son complementos que se adicionan a un programa para aumentar sus funcionalidades, estos se pueden personalizar para adaptarlos al sistema al cual se integren. Dado el hecho que la arquitectura de HABD está basada en *plugin* y que estos brindan gran estabilidad para cualquier aplicación que se desarrolle se decide incorporar el Gestor de Tablas a HABD mediante este complemento.

1.2 Sistemas de Gestión de Base de Datos

Un SGBD está formado por una BD y aplicaciones que permiten a los usuarios realizar operaciones en ellas, como almacenar, manipular y consultar datos pertenecientes a una BD (4).

Para el buen funcionamiento de estos sistemas deben cumplirse los siguientes objetivos (5):

- Definir la BD mediante el Lenguaje de Definición de Datos *DDL* (*Data Definition Language* por sus siglas en inglés): permite especificar la estructura, tipo de datos y las restricciones sobre los datos.
- Separar la descripción y manipulación de los datos: permite flexibilidad para ejecutar consultas y actualización de datos.
- Permitir la inserción, eliminación, actualización o consultas de datos, mediante el Lenguaje de Manejo de Datos *DML* (*Data Model Language* por sus siglas en inglés).
- Proporcionar acceso controlado a la BD: permite la seguridad e integridad de los datos almacenados.
- Proveer interfaces procedimentales y no procedimentales: permite la manipulación de los datos por usuarios pasivos y programadores.
- Independizar la estructura de la organización lógica de los datos: independencia física.
- Permitir una fácil administración de los datos.

Dentro de los SGBD más conocidos y utilizados actualmente se encuentran: *SQL, ORACLE, MySQL, SQL Server, DB2, SLQ/DS, INGRES, INFORMIX, SYBASE, PARADOX, DBASE, ACCESS y PostgreSQL* (6). Donde cada uno cuenta con sus especificaciones de uso, siendo el usuario el encargado de acuerdo a sus requisitos y necesidades de la selección de alguno de ellos para la realización de su trabajo.

PostgreSQL

PostgreSQL es el líder en sistemas de BD de código abierto, con una comunidad mundial de miles de usuarios y contribuyentes, además de contar con docenas de empresas y organizaciones para su desarrollo. El proyecto *PostgreSQL* se basa en 25 años de ingeniería, a partir de la Universidad de California, Berkeley, y tiene un ritmo sin precedentes de desarrollo en la actualidad. El conjunto de características maduras que tiene *PostgreSQL* no sólo rivaliza con sistemas de BD propietarios, sino que los supera en características avanzadas, extensibilidad, seguridad y estabilidad (7).

Soporta gran parte del estándar SQL y ofrece modernas características como: ejecución de consultas complejas y disparadores, creación de vistas, mantiene integridad transaccional, incorpora el control de concurrencia multiversión (técnica de concurrencia donde ninguna tarea o hilo es bloqueado mientras se realiza una operación en la tabla) y puede ser extendido por el usuario añadiendo tipos de datos, operadores, funciones, ventanas, métodos de indexado y lenguajes procedurales (8).

PostgreSQL es el gestor sobre el que está basada la herramienta de administración de base de datos HABD de ahí que el *plugin* al integrarse a la misma también realizará su trabajo sobre dicho gestor.

1.3 Herramientas de Administración de BD

Los sistemas de administración de base de datos *DBMS (Database Management System* por sus siglas en inglés*)*, proveen herramientas que permiten ejecutar tareas de mantenimiento y administración de datos. Tienen definidos funciones para este tipo de actividades como el almacenamiento, la recuperación, actualización y control de la integridad de los datos (9).

Las herramientas de administración de BD se dividen en dos grupos. Las herramientas de código abierto (*PgAdmin III, PGAcces, OpenOffice.org, Mergeant, PGInhaler* (10)) se obtienen sin costo, lo que evita el derroche de grandes sumas de dinero en el pago de las licencias. Permiten estudiar su funcionamiento para la adaptación de las mismas a necesidades propias, distribuir copias así como tratar de realizar mejoras y hacer públicas las mismas. Las herramientas propietarias (*EMS SQL Management for PostgreSQL, DBOne, DB Tools Manager, SyBase Power Designer, Microsoft Access, PgManager, PGExplorer, Aqua Data Studio* (10)) poseen restricciones de uso, imposibilidad de modificaciones y redistribución, así como el elevado monto de dinero que se necesita para la obtención de su licencia, principal causa por la cual muchos usuarios no pueden utilizar estas herramientas.

Herramientas propietarias:

DB Tools Manager: es una herramienta disponible para PostgreSQL y MySQL, permite manejar todos los aspectos de la BD: tablas, *triggers*, funciones, entre otras. Esta herramienta es capaz de gestionar todos los aspectos relativos a la gestión de datos a través de una interfaz de usuario avanzada. Contiene un editor de diseño de consulta para usuarios sin conocimientos de administración de BD (11).

EMS SQL Manager for PostgreSQL: es una herramienta de alto rendimiento para la administración de BD *PostgreSQL*. Soporta las últimas características de *PostgreSQL* incluyendo disparadores, y editor de funciones. Ofrece una amplia gama de herramientas de BD de gran alcance como un diseñador para crear BD en unos pocos clics, *Visual Query Builder* para crear consultas complejas, un editor de tablas potente y otras características útiles para la administración de BD *PostgreSQL* (12). La gestión de tablas se puede realizar a través del árbol de objetos o mediante la ventana que contiene toda la información relacionada con la tabla.

La **Figura 1** muestra cómo el usuario puede observar sobre una misma interfaz todos los componentes relacionados con una tabla (columnas, índices, restricciones, datos, entre otros) lo que garantiza la comodidad en la gestión de las mismas.

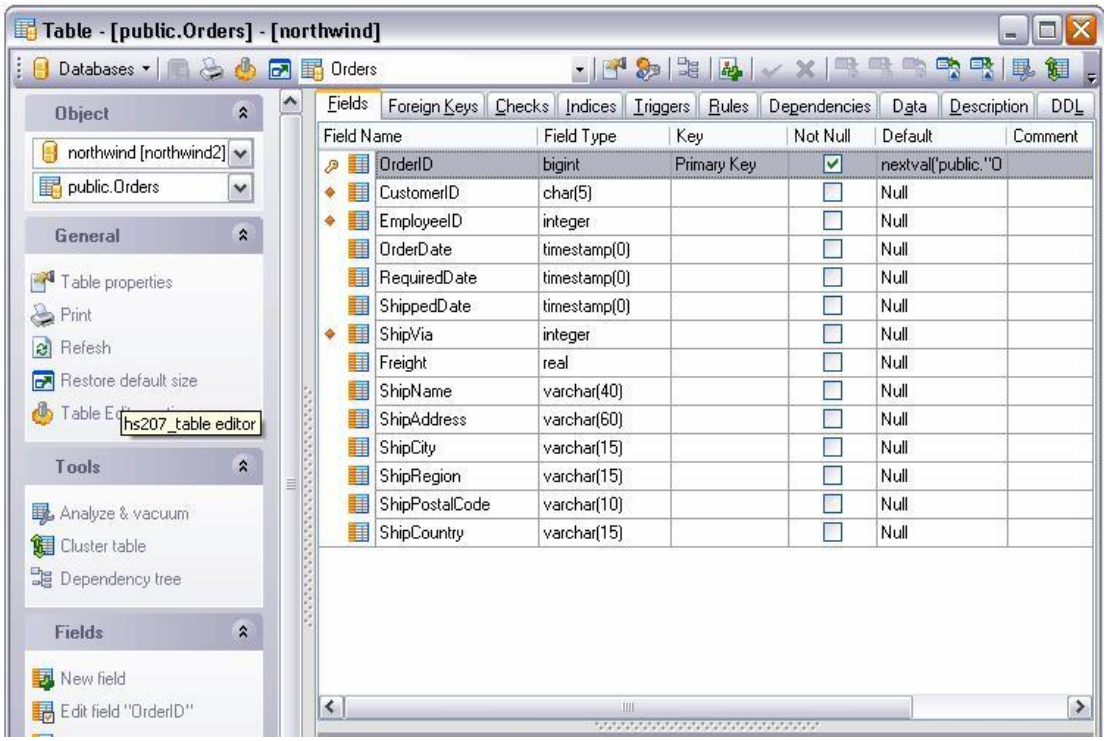


Figura 1. Interfaz visual del editor de tablas del *EMS SQL Management for PostgreSQL.*

Herramientas libres:

PGAcces: permite al usuario interactuar con *PostgreSQL* de una manera similar a muchas aplicaciones de BD, con menús de opciones y diversas herramientas gráficas. El usuario puede evitar el uso de líneas de comandos para la mayoría de las tareas. *PgAccess* no cambia el modo de actuar de *PostgreSQL*, sólo hace más fácil su uso para usuarios familiarizados a interfaces gráficas (13). Además de permitir la edición y adición de BD, tablas, vistas y funciones, permite realizar consultas gráficamente. A pesar de facilitar el trabajo con la gestión de tablas, su principal inconveniente radica en el elevado consumo de memoria, por lo que no es capaz de manejar la información de forma eficiente. Los usuarios necesitan de la tecnología adecuada para obtener buen rendimiento.

***PgAdmin* III:** es una aplicación gráfica para gestionar el SGBD *PostgreSQL*, siendo la más completa herramienta de código abierto. Está escrita en C++ usando la librería gráfica multiplataforma *wxWidgets*, lo que permite su uso en sistemas operativos como *Linux*, *FreeBSD*, *Solaris*, *Mac OS X* y *Windows*. Esta herramienta está diseñada para responder a las necesidades de todos los usuarios, desde escribir consultas *SQL* simples hasta desarrollar BD complejas. La interfaz gráfica soporta todas las características de *PostgreSQL* y facilita la administración. También incluye un editor *SQL* con resaltado de sintaxis, un editor de código de la parte del servidor y un agente para lanzar scripts programados (14).

La **Figura 2** muestra como el editor de tablas en la herramienta *PgAdmin* III facilita al usuario gestionar cada componente de una tabla (propiedades, herencia, columnas, restricciones, privilegios, entre otros) en una misma interfaz.

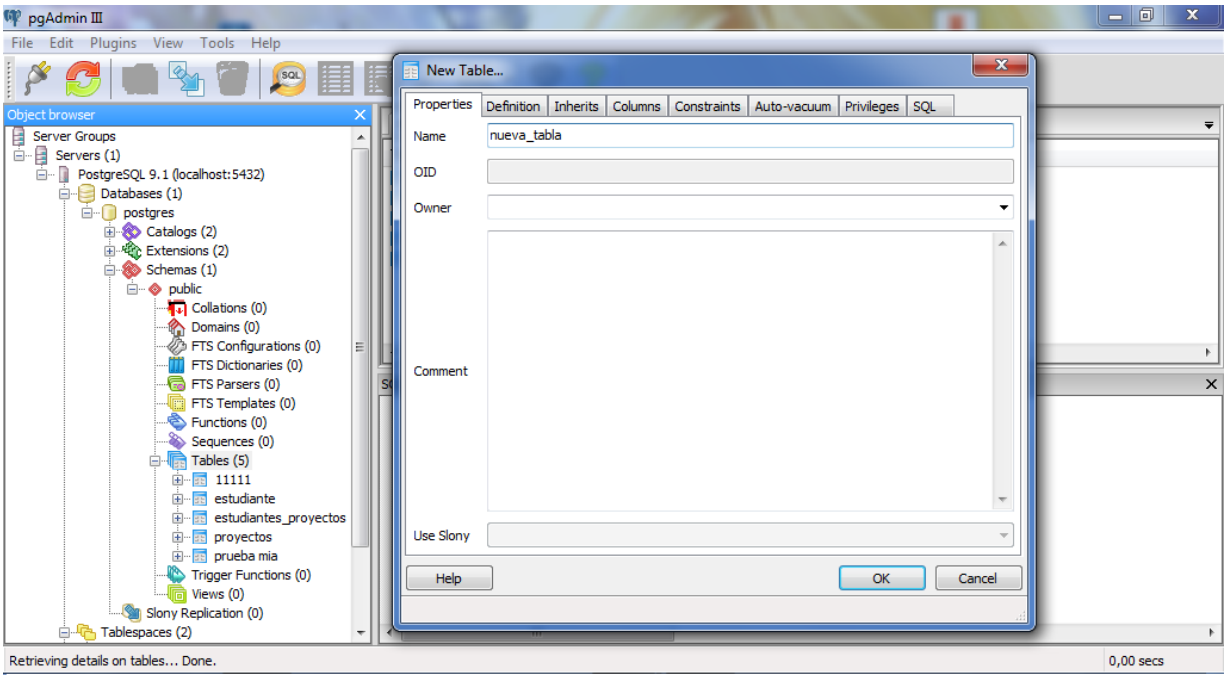


Figura 2. Interfaz visual del editor de tablas del *PgAdmin* III.

La **Tabla 1** representa una comparación entre las herramientas de administración de BD antes mencionadas y HABD, la cual demuestra que todas presentan una solución para gestionar las tablas de una BD. Esto permite a los usuarios interactuar con las mismas de manera sencilla, no siendo así con HABD, de ahí la necesidad impetuosa de elaborar un gestor de tablas para esta herramienta.

Tabla 1. Comparación entre funcionalidades de herramientas de administración de BD existentes y HABD.

Herramientas de Administración de BD	Gestor de consultas SQL	Gestor de funciones y disparadores	Gestor de Tablas
DeZign for Databases	Sí	Sí	Sí
EMS SQL Manager for PostgreSQL	Sí	Sí	Sí
PGAcces	Sí	Sí	Sí

PgAdmin III	Sí	Sí	Sí
HABD	Sí	Sí	No

1.4 Componentes de las tablas en una BD

Las tablas son los objetos principales dentro de una BD, pues permiten almacenar los datos y se utilizan para organizar y agrupar los mismos según características comunes. Una BD puede contener todas las tablas que necesite para organizar los datos. Contienen la información que utilizará el resto de los objetos de una BD.

Una tabla se define mediante una colección de columnas. En ellas, los datos se organizan con un formato de filas y columnas, similar al de una hoja de cálculo. Cada fila representa un registro único y cada columna un campo dentro de un registro (15). Las tablas representan una lista de valores que pueden ser modificadas por el usuario de forma visual lo que facilita el trabajo con ellas.

Composición de una tabla:

- **Columnas:** representan algún atributo del objeto representado por la tabla.
- **Filas:** representan una única repetición del objeto representado por la tabla.
- **Registros**: corresponde a cada fila de una tabla donde se representa un conjunto de datos relacionados.
- **Campos**: corresponde al nombre de la columna, el cual debe ser único y tener un tipo de dato asociado como una de sus principales características.

A los campos, tanto como a las tablas en general se les puede asignar propiedades especiales que proporcionen control adicional sobre la función que realizan en una BD. Entre las propiedades de una tabla se encuentran las restricciones e índices, las cuales se describen a continuación:

Restricciones:

Una restricción es una limitación que obliga el cumplimiento de ciertas condiciones en las tablas de las BD, formando parte fundamental de las mismas. Se dividen en restricciones de columna o restricciones de tabla. La restricción de columna sólo se aplica a la columna seleccionada, mientras que en las tablas se declaran de forma independiente de la definición de las columnas y se pueden aplicar a varias columnas

de la tabla a la vez. Las restricciones proveen un método para implementar reglas que establezcan los datos que pueden ser almacenados en una BD.

Entre los diferentes tipos de restricciones se encuentran (16):

- ***CHECK*:** exigen la integridad del dominio mediante la limitación de los valores que se pueden asignar a una columna. Especifica una condición de búsqueda booleana (se evalúa como *True*, *False* o desconocido) que se aplica a todos los valores que se indican en la columna y rechaza todos los valores que se evalúan como *False*. En una misma columna se pueden especificar varias restricciones *Check*.
- ***UNIQUE*:** exigen la unidad de los valores de un conjunto de columnas. En una restricción de este tipo, dos filas de la tabla no pueden tener el mismo valor en las columnas y admiten que una columna contenga valor *Null*.
- ***FOREIGN KEY*:** se utiliza para establecer y exigir un vínculo o relación entre los datos de las tablas. Una llave foránea de una tabla apunta a una llave primaria de otra tabla. La cláusula ***ON DELETE*** perteneciente a esta restricción controla las acciones que se llevarán a cabo si se intenta eliminar una fila a la que apuntan las llaves foráneas existentes, y la cláusula ***ON UPDATE*** define las acciones que se llevarán a cabo si intenta actualizar un valor de llave primaria a la que apuntan las llaves foráneas. Ambas cláusulas admiten las opciones *NO ACTION, CASCADE, SET NULL y SET DEFAULT.*
- ***PRIMARY KEY*:** una tabla sólo puede tener una restricción *Primary Key* y ninguna columna a la que se aplique esta restricción puede aceptar valores *Null*, debido a que las restricciones *Primary Key* garantizan datos únicos. Se recomienda utilizar una columna pequeña de tipo entero como llave primaria.

Índices:

Los índices surgen con la necesidad de tener un acceso más rápido a los datos. Un índice mejora la velocidad de las operaciones, pues permiten un rápido acceso a los registros de una tabla en una BD a través de punteros que asignan la ubicación de almacenamiento de los datos especificados. Pueden ser creados con cualquier combinación de columnas de una tabla y proporcionan la base tanto para

búsquedas rápidas al azar como para ordenar los registros en las tablas. Mejoran el rendimiento de las aplicaciones de BD y reducen la cantidad de datos que se deben leer al ejecutar una consulta (17).

1.5 Gestión de tablas en herramientas de administración de BD

El proceso de gestión de tablas permite a los usuarios hacer cambios en los componentes de una tabla, modificando la información que de ellas se obtiene. Añade, elimina y actualiza datos, columnas, índices y restricciones para incorporar propiedades a una tabla. Este proceso brinda además la posibilidad de analizar el *SQL* utilizado en una consulta y editarlo si el usuario tiene conocimientos para implementar haciendo uso del lenguaje *DDL*. Permite además crear condiciones para restringir los datos que el usuario inserte, así como indexar las tablas de una BD para agilizar el proceso de búsqueda de datos.

La **Tabla 2** muestra las características y funcionalidades de las gestión de tablas de HABD con respecto a las herramientas *EMS SQL Management for PostgreSQL* y *PgAdmin III,* las cuales contienen las características primordiales para las gestión de tablas, sirviendo de guía para la selección de las funcionalidades a implementar en el *plugin* Gestor de Tablas.

Tabla 2. Características principales de los gestores de tablas.

Características y funcionalidades de la gestión de tablas en las herramientas de administración de BD	***EMS SQL Manager for PostgreSQL***	***PgAdmin III***	Características que presenta HABD actualmente
Sistema multiplataforma	No	Sí	Sí
Editor *SQL*	Sí	Sí(débil)	Sí
Esquema de edición y navegación con apoyo de interfaces gráficas de usuario para tablas	Sí	Sí(débil)	No
Gestionar tablas	Sí	Sí(débil)	Sólo por consultas SQL

Múltiple visualización de tablas	Sí	Sí	No
Modificar datos en las celdas de una tabla	Sí	Sí	No
Generar/Salvar *DDL* de las tablas	Sí	No	No
Gestionar columnas	Sí	Sí	Sólo por consultas *SQL*
Administrar restricciones	Sí	Sí	No
Administrar índices de tablas	Sí	Sí	No
Filtrar datos en una tabla	Sí	Sí	No
Otorgar permisos sobre las tablas	Sí	Sí	No

A partir de las características analizadas y de las necesidades planteadas por los usuarios que utilizan HABD se decide incorporar a dicha herramienta por medio del *plugin* Gestor de Tablas las siguientes propiedades:

- **Esquema de edición y navegación con apoyo de interfaces gráficas de usuario:** permite el proceso de edición de las tablas y sus componentes a través de una interfaz visual.
- **Gestionar tablas:** posibilita crear, modificar, eliminar y actualizar las tablas y sus propiedades.
- **Truncar tablas:** permite eliminar los datos de una tabla.
- **Administrar restricciones:** posibilita crear o eliminar restricciones a las tablas de una BD, las cuales pueden ser de los tipos llave primaria, foránea, de exclusión, entrada y única.
- **Administrar índices:** permite agregar, editar, y eliminar índices a las tablas para la búsqueda de datos.

- **Modificar datos en las celdas de una tabla:** permite editar los datos en las celdas seleccionadas dentro de las tablas.
- **Filtrar datos:** permite ingresar una condición de filtrado, de manera que solo se muestren los datos que cumplan la misma.
- **Generar/Salvar *DDL* de las tablas:** exporta el *DDL* de una tabla.
- **Gestionar columnas:** permite crear, actualizar, modificar y eliminar columnas en las tablas de una BD.
- **Otorgar permisos sobre las tablas:** permite conceder privilegios a los usuarios sobre las tablas.

1.6 Metodologías y herramientas a utilizar en el desarrollo del *plugin*

Las metodologías de desarrollo de software conforman un enfoque para estructurar una aplicación con el objetivo de lograr productos de alta calidad. Están formadas por un conjunto de procedimientos, reglas, técnicas y herramientas que guían a los desarrolladores en el proceso de cómo crear un software. Las herramientas son importantes en el desarrollo de un software, una buena selección de las mismas minimiza los esfuerzos y permite desarrollar las aplicaciones en tiempo, siempre y cuando se haga un buen uso de estas.

A continuación se precisan las herramientas y metodologías a utilizar en el desarrollo del *plugin.* Se enuncian algunas de sus características y ventajas, aunque las mismas se encuentran definidas por el proyecto HABD perteneciente al departamento *PostgreSQL*.

1.6.1 Metodología de desarrollo de software

Las metodologías de desarrollo de software se dividen en ágiles y robustas. Las metodologías ágiles están orientadas a la interacción con el cliente y el desarrollo incremental del software, permiten mostrar versiones funcionales en pequeños intervalos de tiempo para evaluar y sugerir cambios al producto antes de la entrega final. Dentro de este tipo de metodologías se encuentra *Extreme Programming* (XP). Esta metodología se centra en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software y promueve además el trabajo en equipo. Se basa en una realimentación continuada entre el cliente y el equipo de desarrollo con una comunicación fluida entre todos los participantes, también busca

simplificar las soluciones implementadas y valentía para enfrentar los posibles cambios que puedan hacerse al software (18).

La metodología XP está compuesta por varias fases que guían el desarrollo de software:
1ra Fase. Planificación del proyecto: entre las acciones que se ejecutan en esta fase se encuentra definir las historias de usuario con el cliente y crear el plan de iteraciones donde se indica en qué iteración serán implementadas las historias de usuarios previamente definidas.
2da Fase. Diseño: sugiere utilizar diseños sencillos y simples pues de esta manera costará menos tiempo y esfuerzo desarrollarlo posteriormente.
3ra Fase. Codificación: se definen los estándares de codificación que se utilizarán para facilitar la comprensión del código.
4ta Fase. Pruebas: el buen funcionamiento del sistema se lleva a cabo a través de las historias de usuario definidas en la primera fase y sugiere que se realicen pruebas funcionales pues sirven para evaluar las distintas tareas en las que se dividen las historias de usuarios.

¿Por qué utilizar *XP* para el desarrollo del *plugin*?
Debido a las características descritas anteriormente se seleccionó la metodología *XP* como guía para la realización del *plugin* Gestor de Tablas por ser una metodología ágil para el desarrollo de software. La misma consiste en ajustarse a una serie de reglas que se centran en las necesidades del cliente para lograr un producto de buena calidad en poco tiempo y brinda la opción de negociar con el mismo durante el desarrollo de la aplicación. Además es la metodología definida por el proyecto HABD perteneciente al departamento *PostgreSQL*.

1.6.2 Lenguaje de programación
Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar (19). Pueden usarse con diferentes fines: para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Están constituidos por un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Su principal ventaja es que pueden adaptarse fácilmente para ejecutarse en diferentes tipos de equipos.

Lenguaje C++

C++ es un lenguaje de programación que permite programar sistemas operativos, compiladores, aplicaciones de BD, procesadores de texto y juegos (20). Algunas de las particularidades de este lenguaje es que posee una amplia documentación y es orientado a objetos. Posibilita a los desarrolladores programar a alto y bajo nivel. Permite que los niveles de velocidad de ejecución de los programas sean más altos y el consumo de memoria es más pequeño con respecto a otros lenguajes. Además PostgreSQL tiene sus librerías programadas en C++.

De acuerdo a las características antes enunciadas, se selecciona el lenguaje C++ por ser orientado a objetos y un modelo avanzado de programación. Además la herramienta HABD está desarrollada bajo este lenguaje por lo que es conveniente implementar el *plugin* utilizando C++ para ganar en flexibilidad e integridad entre las aplicaciones.

1.6.3 *Framework* de desarrollo

Según **Jerome Lafosse**[1] un *framework* es: un conjunto de bibliotecas, herramientas y normas a seguir que ayudan a desarrollar aplicaciones. Un *framework* está compuesto por varios componentes que interactúan los unos con los otros. Las aplicaciones pueden escribirse de manera más eficaz si se utiliza un *framework* adaptado al proyecto. En proyectos de desarrollo a gran escala y de diseño en equipo, son muy útiles, incluso imprescindibles. Permiten la reutilización de código, la estandarización del desarrollo y la utilización del ciclo de desarrollo (21).

Por tanto se define de forma general a un *framework* como un esquema o patrón para el desarrollo o la implementación de una aplicación que puede definir la estructura de un software completa o centrarse en un aspecto solamente.

***Framework* de desarrollo: *Qt* 4.5**

Una de las características esenciales de *Qt* es su condición de software libre y completamente gratuito. Cuenta con una amplia librería de clases en C++, lenguaje de programación antes seleccionado para la

[1]Ingeniero en Informáticas, obtuvo su diploma en el conservatorio nacional de artes y oficios (CNAM), especialista en las tecnologías web.

implementación del *plugin*. Ofrece una *suite* de aplicaciones para facilitar y agilizar las tareas de desarrollo entre las que se encuentran (22):

- *QtAssistant*: Herramienta para visualizar la documentación oficial de Qt.
- *QtDesigner*: Herramienta para crear interfaces de usuario.
- *QtLinguist*: Herramienta para la traducción de aplicaciones.

1.6.4 Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado *IDE* (*Integrated Development Enviroment* por sus siglas en inglés) es un sistema que facilita el trabajo del desarrollador en la labor de construcción de un software. Está compuesto por una serie de herramientas que utilizan los programadores para implementar el código de una aplicación entre las que se encuentran un editor de texto, un compilador, un intérprete, herramientas para la automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y vínculos para hacer uso de los sistemas de control de versiones.

HABD es una aplicación de escritorio desarrollada en el *IDE QtCreator*, el cual permite crear interfaces de usuario, la comunicación con BD, manejo de cadenas de caracteres, entre otras. Es un *IDE* multiplataforma disponible para sistemas operativos como *Linux*, *Mac OSX*, *Windows*, *Symbian* y *Maemo*. Dentro de las principales características de *QtCreator* se encuentran (23):

- Avanzado editor de código C++.
- Soporta lenguajes como: *C#, .NET, Python, Ada, Pascal, Perl, PHP y Ruby.*
- Interfaz Gráfica de Usuario *GUI* (*Graphic User Interface* por sus siglas en inglés) integrada y un diseñador de formularios.
- Herramientas para la administración de proyectos.
- Depurador visual.
- Resaltado y autocompletado de código.

Por tanto se decide utilizar como *IDE QtCreator* 2.7 pues fue diseñado para hacer que el desarrollo en C++ sea más rápido y fácil pues posee un editor de código para este lenguaje, además fue el *IDE* utilizado en la implementación de HABD.

1.6.5 Lenguaje de modelado *UML*

El lenguaje de modelado unificado *UML* (*Unified Modeling Language* por sus siglas en inglés), es un lenguaje para la especificación, visualización, construcción y documentación de los artefactos de los sistemas de software (24). Permite modelar procesos de negocio y funciones del sistema. Se utiliza para describir un sistema y detallar los artefactos para su posterior construcción. *UML* no es un método de desarrollo, lo que significa que no se utiliza para determinar qué se debe hacer en primer lugar o cómo diseñar el sistema, sino que ayuda a visualizar el diseño y a hacerlo más accesible. Sin embargo no describe un proceso concreto que determine las fases de desarrollo de un sistema, se puede utilizar como lenguaje para definir procesos propios, en dependencia del contexto donde se desea implementar el sistema teniendo como único punto en común los tipos de diagramas que se generen.

1.6.6 Herramientas *CASE* para modelar el sistema

Las herramientas de Ingeniería de Software Asistida por Computadoras *CASE* (*Computer Aided Software Engineering* por sus siglas en inglés), permiten aumentar la productividad en el desarrollo de software, pues facilita crear productos con calidad y generar diagramas de forma automatizada. Entre las más conocidas herramientas *CASE* se encuentra *Visual Paradigm*, pues se puede aplicar en todo el ciclo de vida de un software.

Visual Paradigm es una herramienta multiplataforma que ayuda a generar diagramas para crear aplicaciones con calidad. Soporta múltiples usuarios trabajando en un mismo proyecto, además de facilitar el modelado de BD y el proceso de negocio (25).

Entre las ventajas que brinda Visual Paradigm se destacan:

- Navegación intuitiva entre el código y el modelo.
- Poderoso generador de documentación y reportes UML PDF/HTML/MS Word.
- Soporte completo de notaciones UML.
- Diagramas de diseño automático sofisticado.
- Análisis de texto y soporte de tarjeta CRC.
- Proporciona soportes a varios lenguajes en generación de código e ingeniería inversa a través de plataformas Java, C++, entre otros lenguajes de programación.

Se decide utilizar la herramienta *Visual Paradigm* 8.0 en el desarrollo del *plugin* Gestor de Tablas pues además de contar con las características antes mencionadas, la UCI cuenta con su licencia, lo que permite hacer uso de esta en proyectos productivos para realizar aplicaciones que contribuyen al desarrollo de la universidad.

Conclusiones del capítulo

En la investigación realizada se consultaron diversas fuentes bibliográficas que fundamentan teórica y metodológicamente varias herramientas para la gestión de BD. Se seleccionó *PgAdmin* III y *EMS Manager for PostgreSQL* como guía fundamental en la construcción del *plugin* Gestor de Tablas, las cuales arrojaron un cuadro comparativo que posibilitó seleccionar las principales funcionalidades para la confección del *plugin* guiado por la metodología de desarrollo *XP*. Se especificaron las herramientas a utilizar, definiendo como lenguaje de modelado y de programación *UML* 2.0 y C++ respectivamente, apoyado por el *framework* QT 4.5 y el *IDE* de desarrollo *QtCreator* 2.7 utilizando como herramienta *CASE Visual Paradigm* 8.0.

CAPÍTULO 2: PLANIFICACIÓN Y DISEÑO DEL *PLUGIN*.

Introducción

En el presente capítulo se describen las características propuestas para la elaboración del *plugin*. Se confecciona el modelo de dominio y las historias de usuarios que describen las características del software. Se obtiene la lista de reserva del producto que recoge los requisitos funcionales y no funcionales del sistema propuesto. Con el objetivo de lograr un mejor entendimiento con respecto al diseño de la aplicación se elaboran las tarjetas CRC (Clase, Responsabilidad y Colaboración) y el diagrama de clases. Se especifican los patrones de diseño y arquitectónico a utilizar en el desarrollo del *plugin*.

2.1 Modelo de dominio

Un modelo de dominio se utiliza para el diseño de un software, es una representación de las clases conceptuales del mundo real, no de componentes software. Con la realización del modelo de dominio se pretende unificar el vocabulario entre los usuarios y los desarrolladores, para poder entender el contexto en que se desarrollará un software. (26).

Por tanto se determina que el modelo de dominio es la representación visual de conceptos u objetos relacionados entre sí y agrupados en un dominio. Es el mecanismo fundamental para comprender el problema y establecer conceptos comunes. Se utiliza como alternativa para la comprensión por parte del cliente de la estructura del negocio y se describe mediante diagramas de clases.

A pesar de que la metodología *XP* no genera el modelo de dominio como artefacto, se decide realizar para lograr un mejor entendimiento del negocio, puesto que es un modelo sencillo que facilitará la comprensión del *plugin* que se desea desarrollar.

La **Figura 3** muestra el Modelo de dominio confeccionado para el *plugin* Gestor de Tablas.

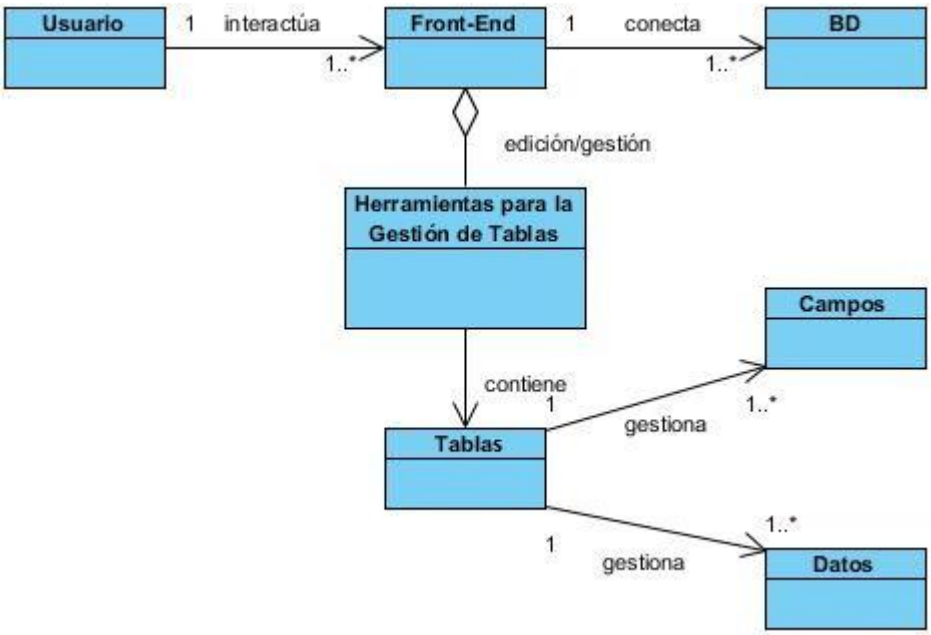


Figura 3. Modelo de dominio del *plugin* Gestor de Tablas.

Usuario: representa a la persona que interactúa con la herramienta y ejecuta todas las operaciones.

Front-End: encargada de administrar los *plugins*, representa la interfaz de la herramienta HABD.

BD: BD sobre la que va a trabajar la herramienta.

Herramientas para la Gestión de Tablas: *plugins* que permiten al *Front-End* gestionar las tablas.

Tablas: módulo que permite realizar acciones para crear, eliminar, mostrar, y modificar las tablas en las BD.

Campos: elemento que se puede crear, modificar, actualizar y eliminar en una BD.

Datos: elemento que se puede crear, modificar, actualizar y eliminar en una BD.

2.2 Propuesta del componente a desarrollar

La herramienta HABD se conecta a una BD PostgreSQL y permite incorporar *plugins* para agregar funcionalidades en ella. Al integrar el *plugin* Gestor de Tablas los usuarios podrán agregar, editar y eliminar tablas, así como añadirles índices, restricciones, datos y columnas a las mismas. El usuario podrá exportar el *SQL* generado en las tablas, el cual se genera de acuerdo a las propiedades insertadas de

forma visual. Permite establecer niveles de privilegios sobre los usuarios que acceden a las tablas. Véase **Figura 4.**

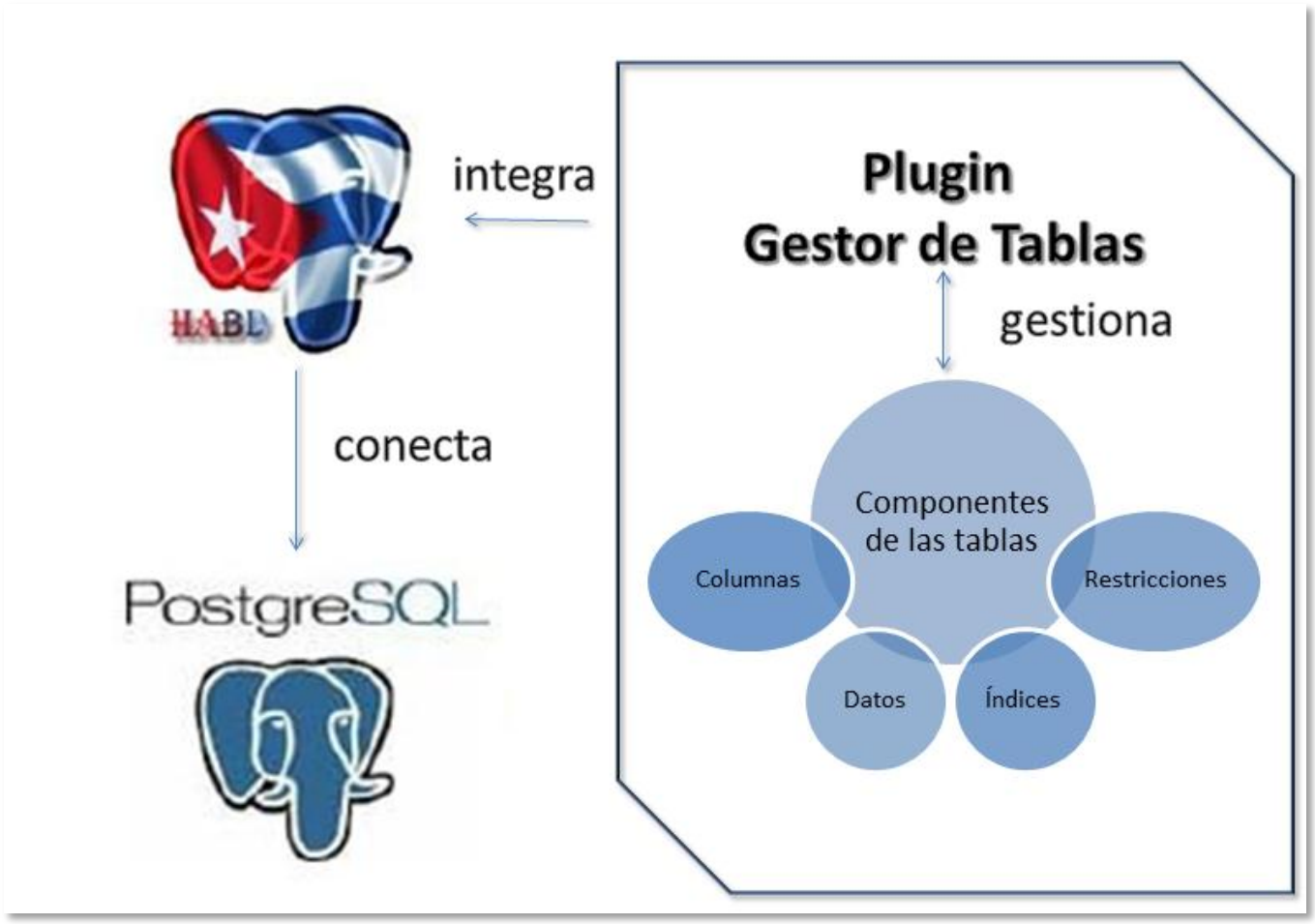


Figura 4. Propuesta de solución para el *plugin* Gestor de Tablas.

2.3 Historias de Usuarios

Las Historias de Usuario (HU) son una parte esencial dentro de la metodología *XP*. Describen las funcionalidades del software en un lenguaje fácil para los desarrolladores y clientes pues representan una breve descripción del comportamiento del sistema. Se realiza una por cada característica principal del software lo que logra remplazar gran documentación de requisitos. A partir de estas se elaboran las posibles pruebas funcionales al sistema. Las HU deben tener un tiempo de estimación menor o igual a las dos semanas para evitar que sean muy complejas. Establecen el nivel de prioridad que alcanzan las

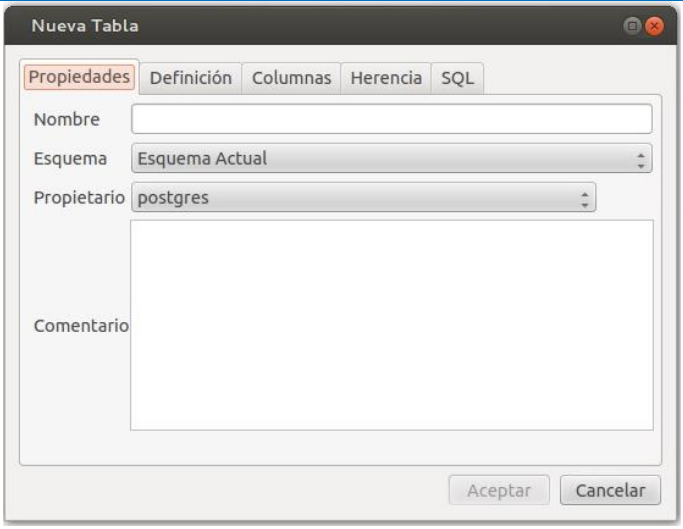
funcionalidades durante el desarrollo del proyecto para indicar cuáles son las más importantes para el resultado final (27).

Para el desarrollo del *plugin* Gestor de Tablas se identificaron nueve HU (Gestionar tablas, Gestionar columnas de tablas, Gestionar datos de tablas, Administrar índices de tablas, Administrar restricciones, Conceder permisos sobre tablas, Salvar *DDL*, Explorador visual de objetos, Truncar tabla). Donde las cinco primeras fueron clasificadas en muy altas, las dos siguientes en altas y las restantes en media, según el nivel de prioridad.

A continuación la **Tabla 3** muestra un ejemplo de la HU "Gestionar tablas" que tiene prioridad muy alta dentro del *plugin*, el resto de las HU se encuentran en el expediente de proyecto (ver Anexo 1).

Tabla 3. Historia de Usuario "Gestionar tablas".

Historia de Usuario	
Número: 5	**Nombre de la Historia de Usuario:** Gestionar tablas
Cantidad de modificaciones a la Historia de Usuario: 0	
Usuario: Yudelis Matos Martínez	**Iteración asignada:** 1ra
Prioridad en negocio: Muy Alta	**Puntos estimados:** 2 semanas
Riesgo en desarrollo: Muy Alto	**Puntos reales:** 2 semanas
Descripción: Crea, elimina, edita y actualiza una tabla.	
Observaciones: El usuario debe llenar correctamente los campos obligatorios para realizar operaciones sobre las tablas.	
Prototipo de interfaces:	



2.4 Lista de Reserva del Producto

La metodología *XP* define dentro de su primera fase como una de sus actividades principales la Lista de Reserva del Producto (LRP), donde se recogen todos los requisitos funcionales y no funcionales del software, los cuales son agrupados según su prioridad para llevar un mejor control de la culminación de los mismos.

A continuación la **Tabla 4** muestra la LRP elaborada para el *plugin* Gestor de Tablas.

Tabla 4. Lista de Reserva del Producto.

Item	Descripción	Estimación	Estimado por
Prioridad: Muy Alta			
1.	Gestionar tablas	2 semanas	Analista
2.	Gestionar campos de tablas	2 semanas	Programador
3.	Administrar índices de tablas	1.6 semanas	Programador
4.	Gestionar datos de tablas	2 semanas	Programador
5.	Administrar restricciones	1.6 semanas	Programador
Prioridad: Alta			

6.	Salvar *DDL*	0.8 semanas	Programador
7.	Conceder permisos sobre tablas	1 semana	Programador
Prioridad: Media			
8.	Truncar tabla	0.6 semanas	Analista
9.	Explorador visual	1 semanas	Programador
Requisitos No Funcionales :			
1.	**Apariencia o interfaz externa:** El *plugin* tendrá una interfaz organizada con opciones de navegación flexibles para ser más factible su uso a los usuarios.		
2.	**Facilidad de Uso:** Para utilizar el *plugin* se necesita de conocimientos básicos de BD y computación.		
3.	**Portabilidad:** El *plugin* podrá ser utilizado en *Linux* o *Windows*, ya que será un sistema multiplataforma.		
4.	**Software:** Para utilizar el *plugin* se debe tener instalado el gestor de BD *PostgreSQL* en la versión 9.1.2, la librería *Qt* libqt4-sql-psql en el caso de la distribución Ubuntu perteneciente al sistema operativo Linux y la herramienta HABD.		
5.	***Hardware*:** Se necesita 256MB de memoria *RAM* como mínimo, 300MB de espacio en disco duro para su instalación y el micro a 300Mhz.		

6. **7.**	**Disponibilidad:** Se podrá hacer uso del *plugin* siempre que esté instalado el mismo las librerías de Qt necesarias. **Soporte:** Se dispondrá de un manual de usuario para los clientes y usuarios finales de la aplicación. **Restricciones de diseño e implementación:** Para el diseño del plugin se utilizará *Visual Paradigm* en su versión 8.0, el lenguaje de programación C++, y como *IDE* de desarrollo *QtCreator* en su versión 2.7.		

La LRP contiene todas las historias de usuarios identificadas para el *plugin,* representada en forma de tabla y comenzando por las de más alta prioridad a implementar. Especifica el tiempo estimado para su realización y el rol que identificó dicha estimación.

2.5 Tareas de la Ingeniería

Las Tareas de la Ingeniería (TI) dividen en escenarios las HU previamente confeccionadas para el desarrollo del sistema, donde a cada HU le puede corresponder más de una TI. Describen con más profundidad las características del sistema y se consideran la entrada de trabajo para el programador (28). Las TI se representan en forma de ficha y contienen el número que la identifica, el identificador de la HU con la que está relacionada, el nombre, tipo, fecha de inicio y fin, equipo responsable y la descripción.

La **Tabla 5** muestra una de las TI correspondiente a la HU Gestionar tablas. Se realizaron un total de 24 TI que se encuentran en el expediente de proyecto (ver Anexo 2).

Tabla 5. Tarea de Ingeniería "Crear tabla".

Tarea de Ingeniería	
Número Tarea: 11	**Número Historia de Usuario:** 5
Nombre Tarea: Crear tabla.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0.5 semanas
Fecha Inicio: 23/02/2013	**Fecha Fin:** 26/02/2013
Programador Responsable: Osman de Armas Sánchez	
Descripción: Permite crear una tabla con sus propiedades.	

2.6 Plan de Iteraciones

Un Plan de Iteración (PI) es una planificación donde los desarrolladores y clientes establecen los tiempos de implementación ideales de las HU. Las iteraciones contarán con una breve descripción de lo que se implementará en ellas, el orden en que se van a implementar las HU y la duración total de cada una. La **Tabla 6** muestra el PI elaborado para la confección del *plugin* Gestor de Tablas

Tabla 6. Plan de Iteraciones.

Iteraciones	Descripción de la iteración	Orden de la HU a implementar	Duración total
1.	En esta iteración se implementarán las HU con prioridad muy alta.	HU 5 HU 6 HU 7 HU 8 HU 9	9.2 semanas
2.	En esta iteración se implementarán las HU con prioridad alta.	HU 2 HU 3	1.8 semanas
3.	En esta iteración se implementarán las HU con prioridad media.	HU 1 HU 4	1.6 semanas

2.7 Diseño del sistema

El diseño de un sistema es importante en el desarrollo de un producto de software, se realiza con el objetivo de garantizar la satisfacción de los usuarios y comprender el sistema a desarrollar.

La metodología de desarrollo *XP* no requiere de la utilización de diagramas de clases como parte del modelo de diseño, en su lugar se utilizan las tarjetas CRC.

2.7.1 Tarjetas CRC

Las tarjetas CRC (Clase, Responsabilidad, Colaboración) se crean en la 2da fase de la metodología *XP* y representan las clases a la que pertenece un objeto determinado. Estas tarjetas se dividen en tres secciones que contienen la información del nombre de la clase, sus responsabilidades y sus colaboradores. En ellas se expresa el diseño del sistema (28).

- **Clase**: representa cualquier persona o evento.
- **Responsabilidades**: representa las actividades que realizan, atributos y métodos.
- **Colaboradores**: representa las clases con las que trabaja en conjunto para llevar a cabo sus responsabilidades.

Para la elaboración del *plugin* se confeccionaron 22 tarjetas CRC, las cuales agrupan todas las funcionalidades del sistema de manera que sean comprensibles por clientes y desarrolladores.

La **Tabla 7** muestra la tarjeta CRC "Table" elaborada para el diseño de la aplicación, las demás quedan adjuntas en el expediente de proyecto (ver Anexo 3).

Tabla 7. Tarjeta CRC "Table".

Tarjeta CRC	
Clase: Table	
Responsabilidades	**Colaboraciones**
Adicionar, actualizar, modificar y eliminar tablas con sus propiedades.	Controller Query Column DateTable Index

2.7.2 Diagrama de clases

Como complemento a las tarjetas CRC se crea el diagrama de clases, que se utiliza para facilitar la comunicación. Se emplea como apoyo adicional, siempre y cuando no sean extensos y se enfoquen en la información importante.

La **Figura 5** muestra el diagrama de clases del *plugin* Gestor de Tablas.

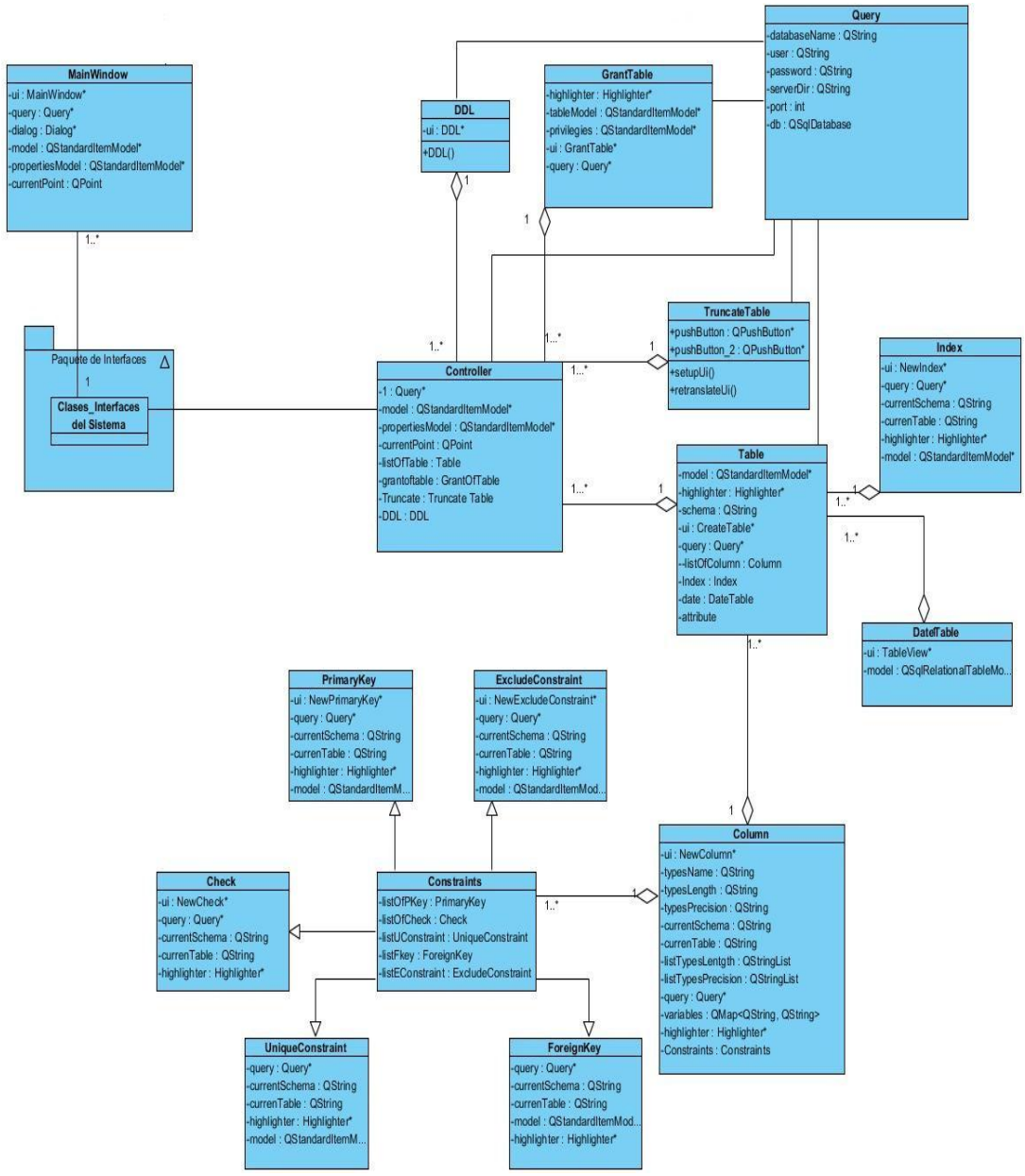


Figura 5. Diagrama de Clases del *plugin* Gestor de Tablas.

2.8 Patrón Arquitectónico

Las técnicas metodológicas desarrolladas con el fin de facilitar la programación se engloban dentro de la llamada Arquitectura de Software o Arquitectura lógica. Se refieren a un grupo de abstracciones y

patrones que nos brindan un esquema de referencia útil para guiarnos en el desarrollo de software dentro de un sistema informático (29).

Para el desarrollo del *plugin* la arquitectura a utilizar será Modelo - Vista - Controlador (MVC) el cual se divide en las siguientes partes:

- **Modelo:** define las reglas del negocio y recoge todas las funcionalidades del sistema.
- **Vista:** responsable de recibir los datos del modelo y mostrarlo al usuario.
- **Controlador:** ejecuta la lógica del modelo y las vistas.

La **Figura 6** muestra como queda reflejado el uso del patrón MVC en las clases del *plugin* Gestor de Tablas.

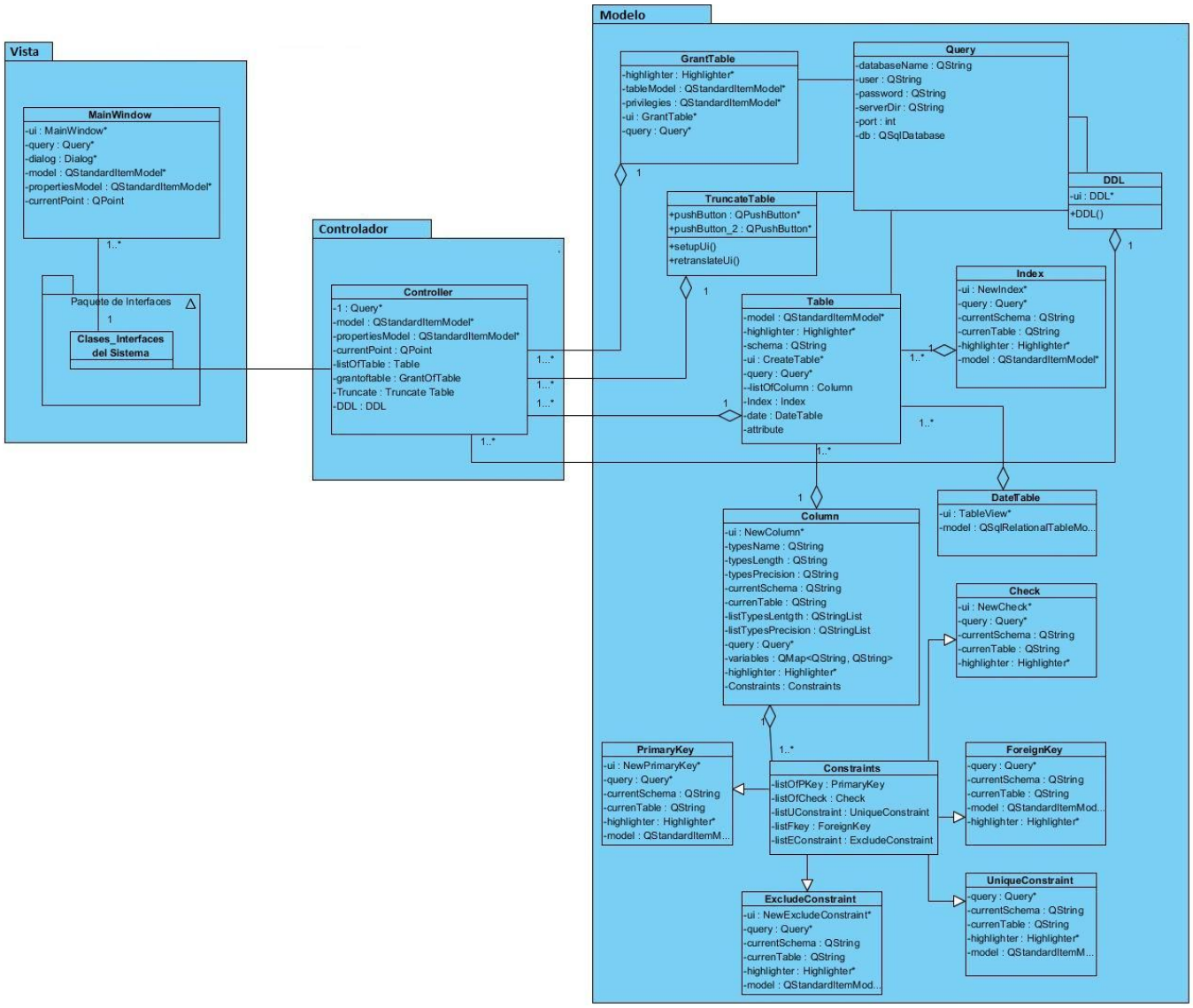


Figura 6. Patrón Arquitectónico MVC utilizado en el plugin Gestor de Tablas.

2.9 Patrones de Diseño.

Los patrones de diseño son el soporte de las soluciones a problemas comunes en el desarrollo de un software. Se refieren al diseño de interacción o interfaces que permiten establecer una estructura común ante problemas semejantes.

Patrones *GRASP*

Los Patrones Generales de Software para Asignar Responsabilidades GRASP (*General Responsibility Assignment Software Pattern* por sus siglas en inglés), describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (30).

Para obtener un buen diseño se aplicaron los patrones Experto, Creador, Controlador, Alta Cohesión y Bajo Acoplamiento. A continuación se describe cómo estos patrones solucionan los problemas de diseño en aplicaciones orientadas a objeto realizando una breve descripción de cada uno de ellos.

- **Experto:** asigna responsabilidad a las clases expertas con la información necesaria para llevar a cabo una tarea determinada.
- **Creador:** crea una nueva instancia en la clase que tiene la información necesaria para realizar la creación del objeto, almacena o maneja varias instancias de la clase y contiene o agrega la clase.
- **Controlador:** representa el sistema global, dispositivo o subsistema para el manejo de los eventos del sistema.
- **Alta Cohesión:** asigna responsabilidades de manera que la información que almacena una clase sea coherente y esté relacionada con la clase.
- **Bajo Acoplamiento:** diseñado con el objetivo de tener las clases lo menos ligadas entre sí. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, para potenciar la reutilización, y disminuir la dependencia entre las clases.

En las imágenes que se muestran a continuación se evidencia el uso de los patrones *GRASP* utilizados durante el diseño del *plugin* Gestor de Tablas, donde en una clase puede evidenciarse el uso de más de uno de ellos.

El patrón controlador se evidencia en la clase *Controller* pues se encarga de manejar los eventos del *plugin* y obtiene la información necesaria para llevarlo a cabo. Ver **Figura 7.**

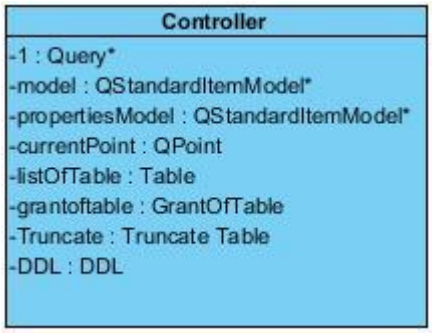


Figura 7. Ejemplo del patrón Controlador en el *plugin* Gestor de Tablas.

El uso de los patrones creador y bajo acoplamiento se evidencian en la clase *Controller* y *Table*, la clase *Controller* crea objetos de la clase *Table* para realizar sus responsabilidades, y esta no depende de ninguna otra clase para realizar su función, pues cuenta con todos los atributos necesarios para ello. Ver **Figura 8**.

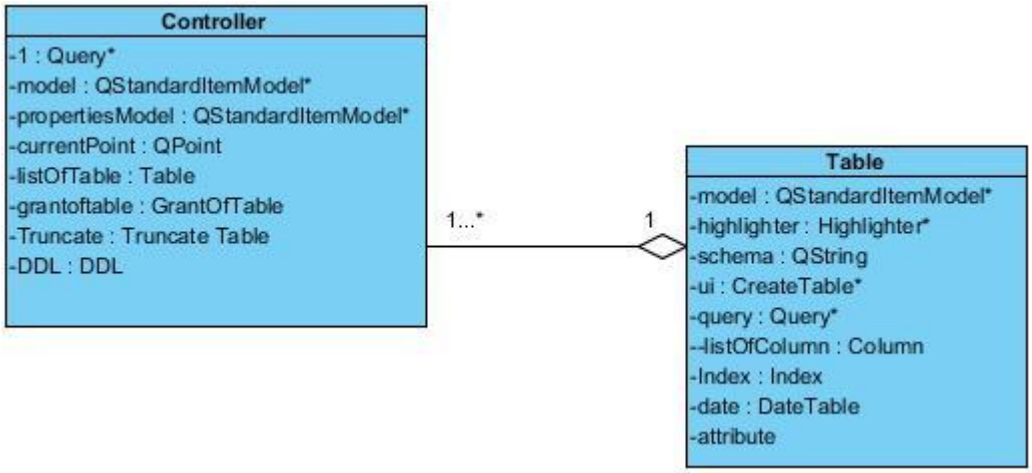


Figura 8. Ejemplo de los patrones Creador y Bajo Acoplamiento en el *plugin* Gestor de Tablas.

En la clase *GrantTable* se evidencia el uso de los patrones experto y alta cohesión. Su responsabilidad radica en ejecutar consultas a la BD, solamente ella es la encargada de otorgar permisos sobre las tablas y esta clase necesita de *Query* para la ejecución de sus tareas, pues para otorgar permisos se necesitan ambas clases. Ver **Figura 9.**

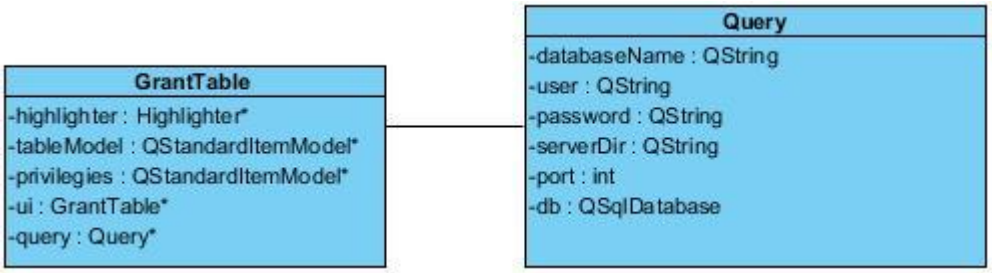


Figura 9. Ejemplo de los patrones Experto y Alta Cohesión en el *plugin* Gestor de Tablas.

Patrones *GoF*

Los patrones *GoF (Gang Of Four* por sus siglas en inglés*)* deben su nombre al grupo compuesto por sus creadores: *Erich Gamma, Richard Helm, Ralph Jhonson y John Vlisides* quienes en su publicación *"Design Patterns"* describen 23 patrones de diseño de gran utilidad y aplicables en el diseño de un sistema.

Estos patrones ofrecen las siguientes ventajas (31):

- Proporcionan elementos reusables en el diseño de sistemas de software.
- Efectividad comprobada en la resolución de problemas similares.
- Formalizan un vocabulario común entre diseñadores.
- Facilitan el aprendizaje de las nuevas generaciones de diseñadores y desarrolladores utilizando conocimiento ya existente.
- Estandarizan el diseño.

A continuación se nombran los patrones *GOF* utilizados en el diseño del *plugin* Gestor de Tablas:

***Brigde*:** encargada de desacoplar una abstracción de su implementación de modo que ambas puedan cambiar independientemente, esto queda reflejado en la integración del *plugin* con la herramienta HABD, de esta manera se podrán hacer cambios estructurales en cualquiera de los sistemas sin afectaciones.

***Observer*:** define la dependencia entre objetos. Cuando un objeto cambia de estado se actualizan automáticamente todos los objetos que dependen de él. El *framework* utilizado para la realización del *plugin* contiene por defecto este patrón y se evidencia en los mecanismos *Signals* y *Slot* entre las clases visuales pues al producirse una variación en un objeto este emite una señal que es capturada por una función encargada de hacer los cambios necesarios.

Conclusiones del capítulo

En el capítulo se realizó el modelo de dominio con el objetivo de comprender la estructura del negocio. Se diseñaron 24 TI agrupadas en nueve HU y fueron elaboradas 22 tarjetas CRC para facilitar la relación entre las clases de la aplicación y las funciones que deben cumplir. Como complemento a estas se realizó el diagrama de clases, que permitió modelar la arquitectura del *plugin* a través del patrón arquitectónico MVC. Para minimizar el riesgo de generar un mal diseño y con el propósito de obtener un *plugin* con calidad se utilizaron los patrones de diseño *GRASP* (experto, creador, controlador, bajo acoplamiento y alta cohesión) y *GoF* (*bridge*, *observer*).

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA DEL *PLUGIN*.

Introducción

El contenido de este capítulo aborda la implementación y validación del sistema propuesto, donde se dará cumplimiento a los requisitos definidos a través de las TI confeccionadas en el capítulo anterior utilizando estándares de codificación para facilitar la comprensión del código. Se especifica el método y técnica de prueba a realizar para verificar el correcto funcionamiento del *plugin*. Además se elabora un resumen de las no conformidades detectadas al aplicar las pruebas, las cuales fueron resueltas durante las tres iteraciones realizadas.

3.1 Implementación del sistema

La etapa de implementación de un software consiste en desarrollar las funcionalidades del sistema. Para ello el equipo de desarrollo se centra en las TI elaboradas a partir de las HU descritas durante la planificación del *plugin.* Para lograr el objetivo principal de este ciclo se debe tener en cuenta la arquitectura y diseño, así como la forma de organizar el código, pues serán las bases para el éxito del producto.

3.1.1 Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Algunas de las ventajas que presenta el uso de estos son:

- Legibilidad del código para que otros desarrolladores puedan comprender el sistema.
- Facilidades de mantenimiento y actualización del sistema, con código claro y bien documentado.

A continuación se describe el estándar de codificación utilizado en la implementación del *plugin* bajo el lenguaje C++.

Identación

La unidad de identado es de cuatro espacios.

La **Figura 10** muestra un ejemplo de la identación en el código del *plugin* Gestor de Tablas.

```
public:
    explicit CreateTable(QWidget *parent = 0);
    ~CreateTable();
    void Inicialize(Query*query);
    void setSchema(QString schema);
private:
    void UpdateSql();
    Ui::CreateTable *ui;
    Query*query;
    QStandardItemModel*model;
    Highlighter*highlighter;
```

Figura 10. Ejemplo del uso de identación en el *plugin*.

Comentarios

Es conveniente dejar información que pueda ser leída para entender qué se hizo en un fragmento de código. Los comentarios son escritos correctamente y claros en una sola línea.

La **Figura 11** muestra un ejemplo de la utilización de los comentarios en el código del *plugin* Gestor de Tablas.

```
//verificar si el schema existe
for(int i=0;i<schemas.length();i++)
{
    bool exist=false;
    for(int j=0;j<root->rowCount();j++)
    {
        if(root->child(j,0)->text()== schemas[i])
        {
          exist=true;
          break;
        }
    }
    if(!exist)
       schemasToAppend.append(schemas[i]);
}
```

Figura 11. Ejemplo de uso de comentarios en el *plugin*.

Declaración de variables y funciones

Cada variable o función es declarada en una línea, comienza con letra minúscula y la palabra relevante por la que esté compuesta es escrita con letra mayúscula.

La **Figura 12** muestra un ejemplo de declaración de variable en el código desarrollado.

```
private:
    QString databaseName,user,password,serverDir;
    int port;
    QSqlDatabase db;
```

Figura 12. Ejemplo de declaración de variables.

Sentencias

Las sentencias describen acciones algorítmicas que puede ejecutar un programa. Son elementos básicos en los que se divide el código según el lenguaje de programación que se utilice. Cada sentencia simple termina en punto y coma. En las sentencias compuestas la llave que inicia comienza en la línea siguiente al igual que la llave que termina y guardan la misma identación. La sentencia *for* se rige por la siguiente estructura:

for(……..)
{
……
……
}

La **Figura 13** muestra un ejemplo de la sentencia *for* donde se utiliza la estructura definida.

```
for(int i=0;i<schemas.length();i++)
{
    bool exist=false;
    for(int j=0;j<root->rowCount();j++)
    {
        if(root->child(j,0)->text()== schemas[i])
        {
          exist=true;
          break;
        }
    }
    if(!exist)
       schemasToAppend.append(schemas[i]);
}
```

Figura 13. Ejemplo de sentencia *for*.

3.1.2 Interfaces de la aplicación

El *plugin* Gestor de Tablas cuenta con una interfaz principal y otras que se derivan de ella de manera que solo se muestren las necesarias de acuerdo a la función que el usuario desee realizar. A continuación se muestran algunas de las interfaces por las que está compuesto.

La **Figura 14** muestra la interfaz principal del *plugin* Gestor de Tablas que contiene en la parte superior la barra de herramientas donde el usuario tiene acceso rápido a algunas de sus funcionalidades. En la parte izquierda se visualiza el árbol de objetos, donde se muestra la BD a la cual se está conectada y sus respectivos esquemas. Estos esquemas a su vez contienen las tablas, columnas, índices y restricciones a los que se les podrá realizar modificaciones. En la parte derecha se podrán visualizar los componentes que el usuario desee gestionar y se mostrarán las interfaces para realizar alguna operación sobre las tablas. En este caso se visualizan las propiedades de la tabla cargada y sus respectivos valores.

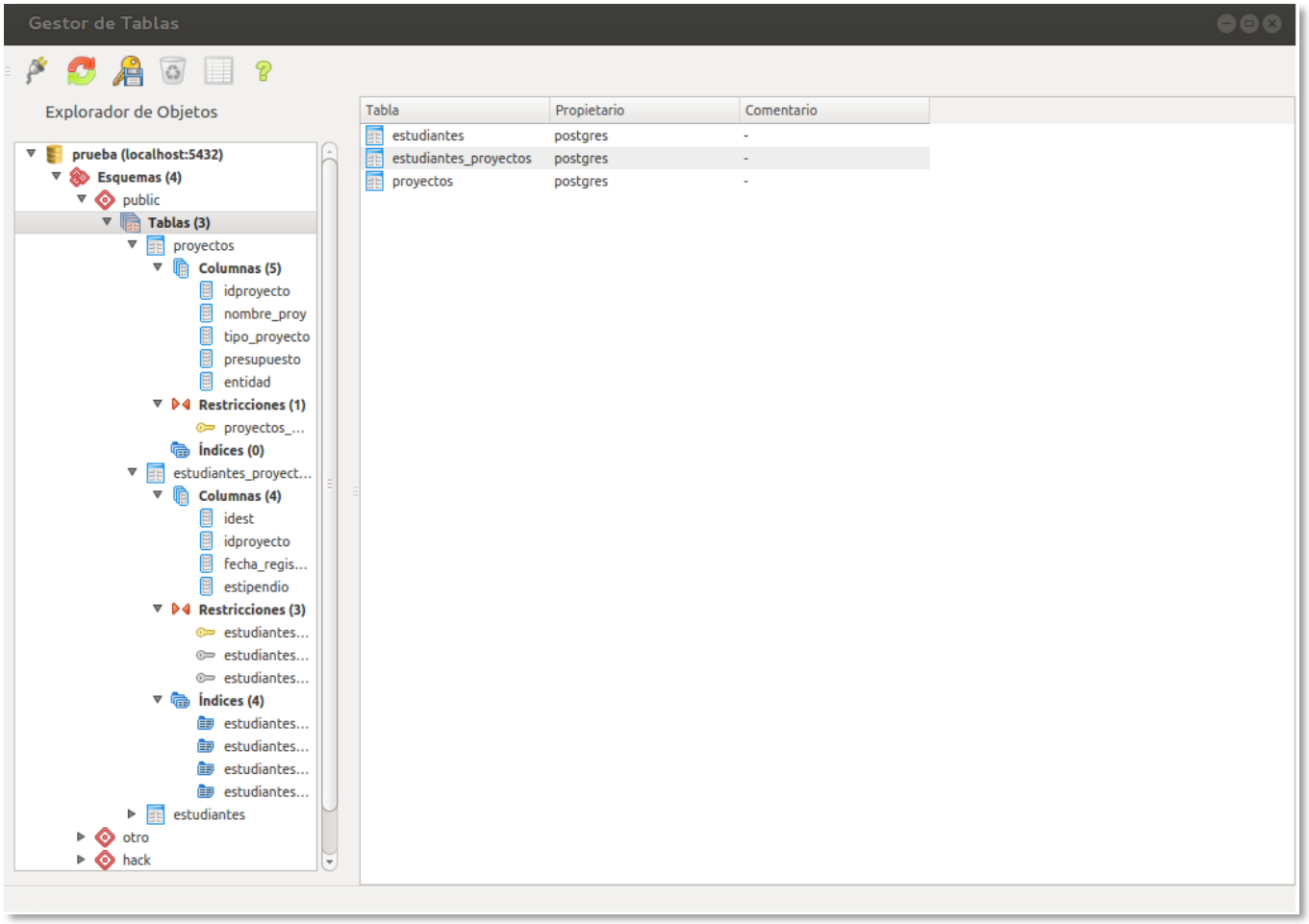


Figura 14. Interfaz principal del *plugin* Gestor de Tablas.

La **Figura 15** muestra la interfaz visual “Nueva Tabla” que contiene varios componentes donde el usuario podrá especificar las propiedades de la tabla.

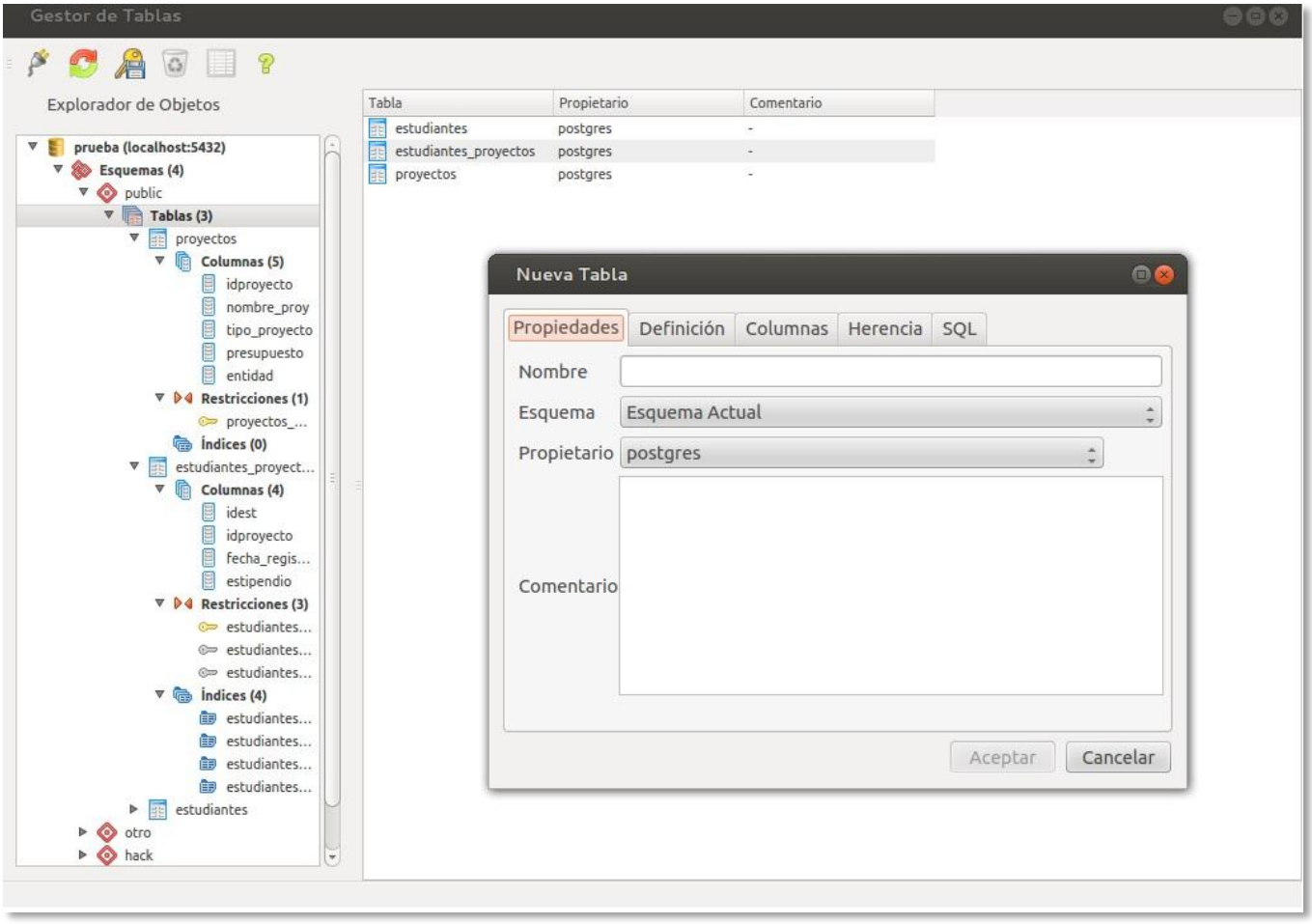


Figura 15. Interfaz visual "Nueva Tabla".

En la **Figura 16** se muestra la interfaz "Nueva Columna", en la cual el usuario agrega nuevas columnas a una tabla existente. La interfaz está compuesta por las características que puede tener una columna, entre ellas se encuentran el nombre de columna y el tipo de dato. Como elemento adicional permite filtrar la búsqueda del tipo de datos mientras se escribe.

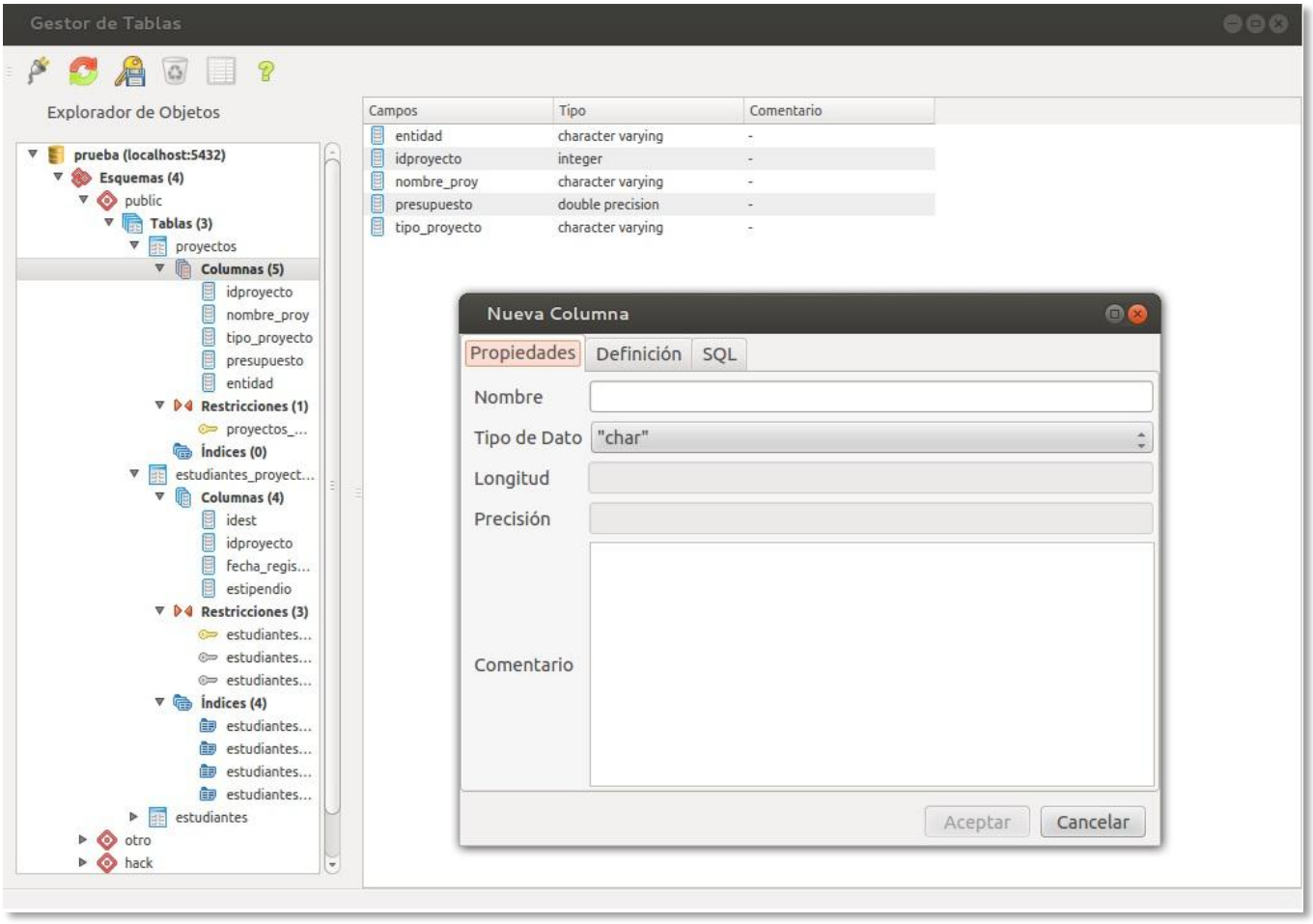


Figura 16. Interfaz visual "Nueva Columna".

A continuación en la **Figura 17** se muestra la interfaz visual "Nueva Llave Foránea", donde el usuario crea una restricción de este tipo sobre las columnas de la tabla. La aplicación cuenta con varias interfaces para administrar las restricciones puesto que existen cinco tipos de estas.

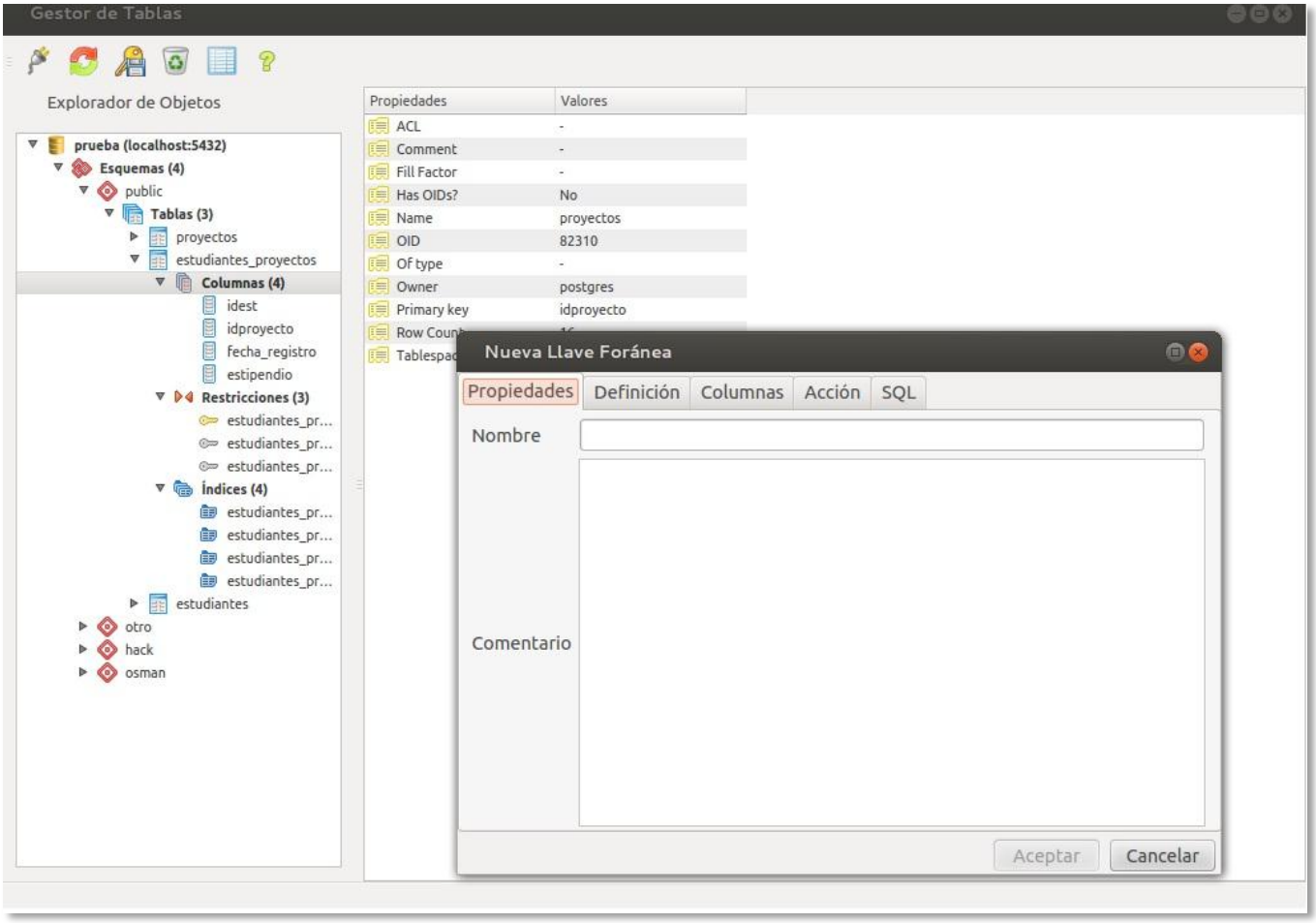


Figura 17. Interfaz visual "Nueva Llave Foránea".

En la **Figura 18** se muestra la interfaz "Conceder Permisos". Esta interfaz cuenta con tres pestañas en la que se especifica la tabla a las que se le desea conceder algún tipo de permiso, el tipo de privilegio que se desea otorgar y el usuario que podrá ejecutarlos.

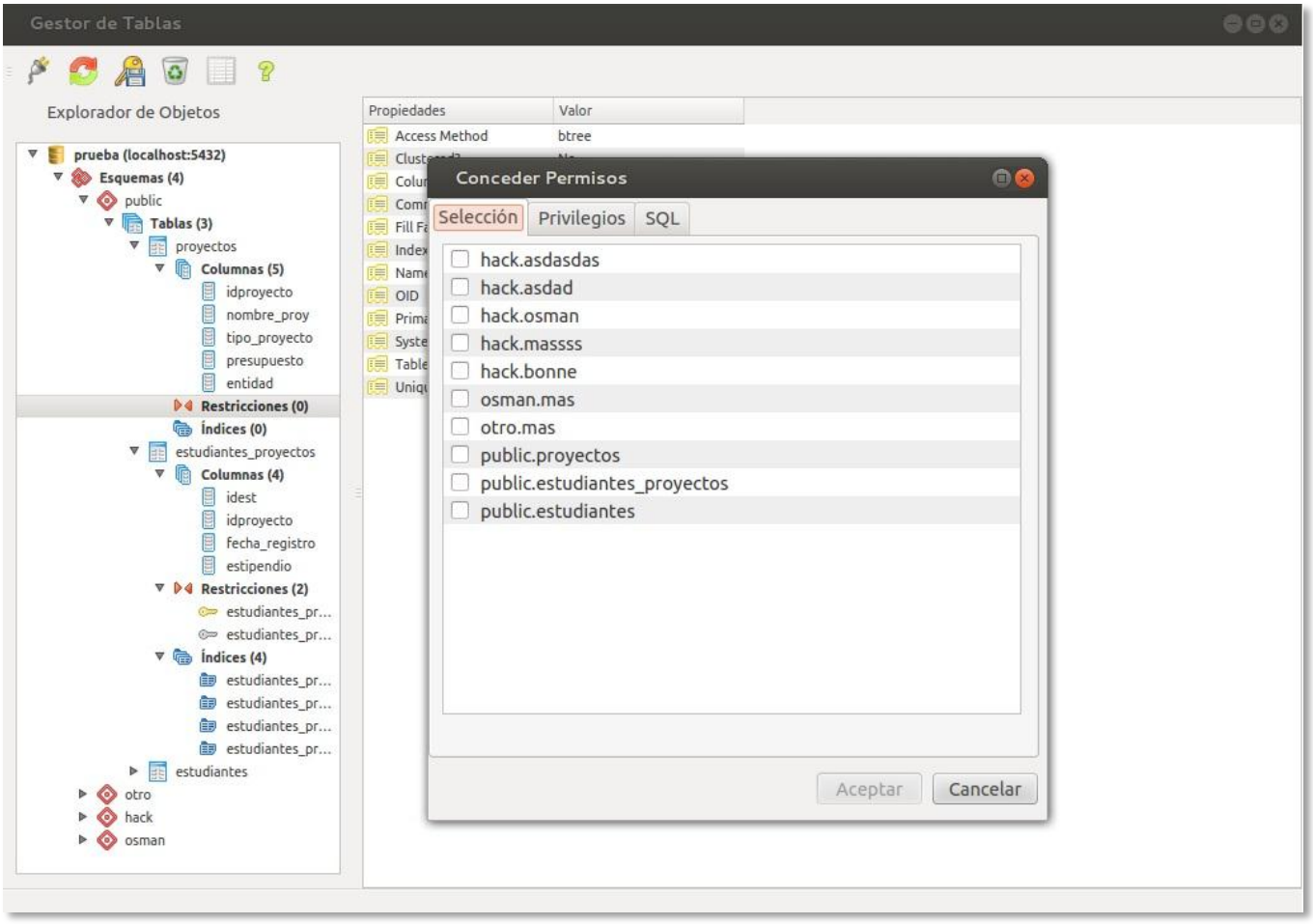


Figura 18. Interfaz visual "Conceder Permisos".

3.2 Pruebas del sistema

Las pruebas de software son un elemento crítico para la garantía de la calidad del software. Estas forman parte indispensable para evaluar y determinar la calidad de un producto. El objetivo de las pruebas es detectar los defectos en los requisitos, diseño, construcción y documentación del sistema, de manera que se eviten errores una vez que la aplicación sea entregada al usuario final. (24)

3.2.1 Estrategia de prueba

Según ***Pressman***[2] una estrategia de prueba del software: integra los métodos de diseño de casos de pruebas (CP) del software en una serie bien planeada de pasos que desembocan en la eficaz construcción del mismo. Cualquier estrategia de prueba debe incorporar la planeación, el diseño de CP, ejecución, recolección y evaluación de los datos resultantes (32).

XP como metodología ágil enfocada en las necesidades del cliente, propone para un buen funcionamiento del producto, la realización de pruebas funcionales, para ello se determina el método y técnica adecuada de acuerdo a los requisitos de software que se desean probar.

Método de Caja Negra

También conocido como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar el CP. El componente se ve como una "Caja Negra" cuyo comportamiento sólo puede ser determinado al estudiar sus entradas y las salidas obtenidas a partir de ellas (33). Ver **Figura 19.**

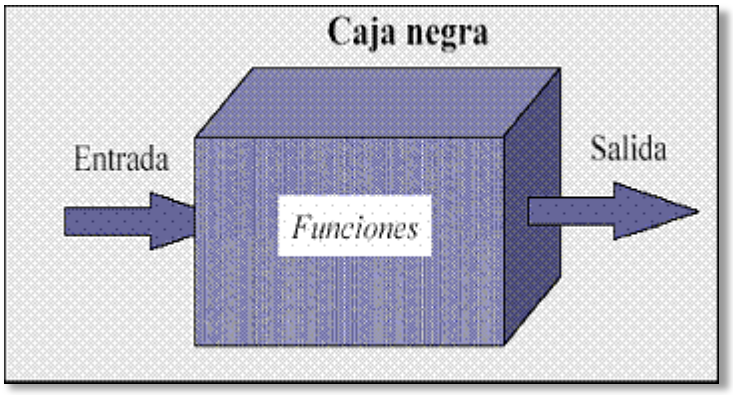


Figura 19. Método de caja negra.

Para confeccionar los CP este método determina varias técnicas entre las que se encuentra la Técnica de partición de equivalencia. Esta técnica divide la entrada de un programa en variables con juegos de datos de entrada y salida. De este modo se puede asumir que una prueba realizada con un valor representativo de cada variable es equivalente a una prueba realizada con cualquier otro valor de dicha variable.

[2] Diseñador y desarrollador de productos para la formación de ingeniería de software y mejora de procesos.

Para el proceso de validación del *plugin* Gestor de Tablas se emplean pruebas funcionales utilizando el método caja negra con la técnica de partición de equivalencia. Dicha selección se basa en las características de ellos y los resultados que son capaces de obtenerse aplicándolos.

3.2.2 Casos de Prueba basados en HU

Los CP especifican los requisitos de una aplicación, por lo que cada uno debe estar cubierto por un CP. Son un conjunto de condiciones y variables bajo los cuales se determina si el requisito de una aplicación es parcial o completamente satisfactorio. Cada CP está compuesto por varios pasos o escenarios a ejecutar, dependiendo de la complejidad del caso. Son ejecutados por un probador y arrojan un resultado esperado.

Para la validación del plugin Gestor de Tablas se realizaron nueve CP (Gestionar tablas, Gestionar columnas de tablas, Gestionar datos de tablas, Administrar índices de tablas, Administrar restricciones, Conceder permisos sobre tablas, Salvar DDL, Explorador visual de objetos, Truncar tabla), correspondientes a las HU del mismo nombre. Los CP Truncar tabla y Explorador visual de objetos no contienen variables, en consecuencia de esto se comprobó que el sistema ejecuta la funcionalidad correctamente en ambos CP y muestra las vistas correspondientes en caso de la HU Explorador visual de objeto.

A continuación en la **Tabla 8** se muestran la sección del CP “Conceder permisos” sobre tablas, realizado para comprobar que esta funcionalidad responde correctamente a la HU del mismo nombre. Para observar otros CP consulte el expediente de proyecto.

Tabla 8. Sección del CP “Conceder permisos”.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1. “Conceder Permisos”	SC 1.2. “Modificar Permisos”	Permite modificar permisos sobre las tablas en la BD.

La **Tabla 9** muestra las variables que el usuario ingresa para otorgar privilegios sobre una tabla. Para visualizar otras variables consulte el expediente de proyecto.

Tabla 9. Variables del CP Conceder permisos sobre tablas.

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	Selección	*Checkbox*	No	Este campo brinda la opción de seleccionar la tabla a la que se le quiere otorgar permiso. El usuario solo puede seleccionar una tabla.
2	Rol/Usuario	Lista de selección	No	Este campo brinda la opción de seleccionar el usuario que podrá ejecutar los permisos otorgados. Al seleccionar un usuario se debe presionar el botón Adicionar y se muestra en el área de texto.
3	Privilegios	*Checkbox*	No	Este campo brinda la opción de seleccionar el tipo de privilegios que se desea otorgar. Por defecto están marcados los que ya tiene otorgado el usuario seleccionado.
4	*SQL*	Área de texto	No	Este campo muestra el *sql* generado según las propiedades que el usuario seleccione.

La **Tabla 10** muestra la matriz de datos para el CP “Conceder Permisos”

Tabla 10. Matrix de datos del CP “Conceder Permisos”.

Escenario	Variables (enumeradas según descripción de las variables)				Respuesta del Sistema	Flujo central
	1	2	3	4		
EC1.1.“Conceder Permisos” sobre las tablas con datos correctos.	*public_* proyectos	*postgres*	*ALL*	NA	El sistema activa el botón Aceptar si los datos fueron introducidos correctamente.	1. Clic en el botón “Conceder Permisos” de la barra de herramientas principal. 2. Seleccionar la tabla para modificar los permisos. 3. Seleccionar el tipo de permiso, y los usuarios que tendrán acceso. 4. Presionar el botón Adicionar para asignarles los permisos al rol elegido. 5. Presionar el botón Aceptar.
EC1.2.“Conceder Permisos” sobre tablas con datos incorrectos.	No se selecciona	*postgres*	*ALL*	NA	El sistema no activa el botón Aceptar.	1. Clic en el botón Conceder permisos de la barra de herramientas principal. 2. No seleccionar el objeto.
	public_ proyectos	No se selecciona	*ALL*	NA		

3.2.3 No conformidades

Se realizaron tres iteraciones a las funcionalidades del *plugin* a través de los CP diseñados y se detectaron 22 **no conformidades**, definiendo como **significativas** errores funcionales y como **no significativas** errores ortográficos y entrada de datos incorrectos.

A continuación la **Tabla 11** muestra un registro de las **No conformidades** encontradas en la HU Gestionar Tablas.

Tabla 11. No conformidades de la HU Gestionar Tablas.

Elemento	No	No Conformidad	Aspecto correspondiente	Etapa de detección	Clasificación	Estado de la NC	Respuesta del Equipo Desarrollador
Aplicación	1	Al editar los componentes se generaba una nueva tabla con los cambios efectuados.	Interfaz Editar Tablas.	Prueba	S	15/5/2013 PD 16/5/2013 RA	Se corrigió el error encontrado. Ya modifica los cambios en la tabla seleccionada.
	2	No permite adicionar columnas en la misma interfaz de la tabla.	Interfaz Nueva Tabla	Prueba	S	15/05/2013 PD 17/05/2013 RA	Se corrigió el error encontrado, el usuario puede agregar columnas en la pestaña Columnas de la interfaz Crear tablas.
	3	No permite asignar la tabla a un esquema específico.	Interfaz Nueva			15/05/2013 PD	Se corrigió el error encontrado, el usuario puede seleccionar en que

			Tabla	Prueba	S	17/05/2013 RA	esquema desea crear la tabla.
	4	El botón Cancelar no funciona.	Interfaz Nueva Tabla	Prueba	NS	15/05/2013 PD16/05/2013 RA	Se corrigió el error encontrado.
	5	Componentes de la tabla sin traducir al idioma español en la interfaz.	Interfaz Nueva Tabla	Prueba	NS	15/05/2013 PD 16/05/2013 RA	Se corrigió el error encontrado. Se tradujo al español el nombre de todos los componentes de la tabla.

Este mismo procedimiento fue realizado para las ocho HU restantes, la **Tabla 12** muestra un resumen elaborado a partir de los resultados obtenidos en cada HU.

Tabla 12. No conformidades encontradas en el *plugin* Gestor de Tablas.

HU	Total de no conformidades	# de NC Significativas	# de NC No Significativas
HU 1: Truncar Tabla	2	0	2
HU 2: Conceder permisos	3	2	1
HU 3: Salvar *DDL*	2	2	0
HU 4: Explorador Visual de Objetos	0	0	0
HU 5: Gestionar Tablas	5	3	2
HU 6: Gestionar Columnas	4	2	2
HU 7: Administrar índice	1	0	1
HU 8: Gestionar datos	0	0	0
HU 9: Administrar restricciones	5	0	5
Total	22	9	13

A partir de las **no conformidades** encontradas en un total de 33 pruebas funcionales realizadas al *plugin*, se confeccionó el siguiente gráfico que representa un resumen de los resultados de las pruebas realizadas. En el mismo se muestra como disminuyen las **no conformidades** a medida que se avanzaba en las iteraciones, hasta finalmente quedar libre de errores como se muestra en la **Figura 20.**

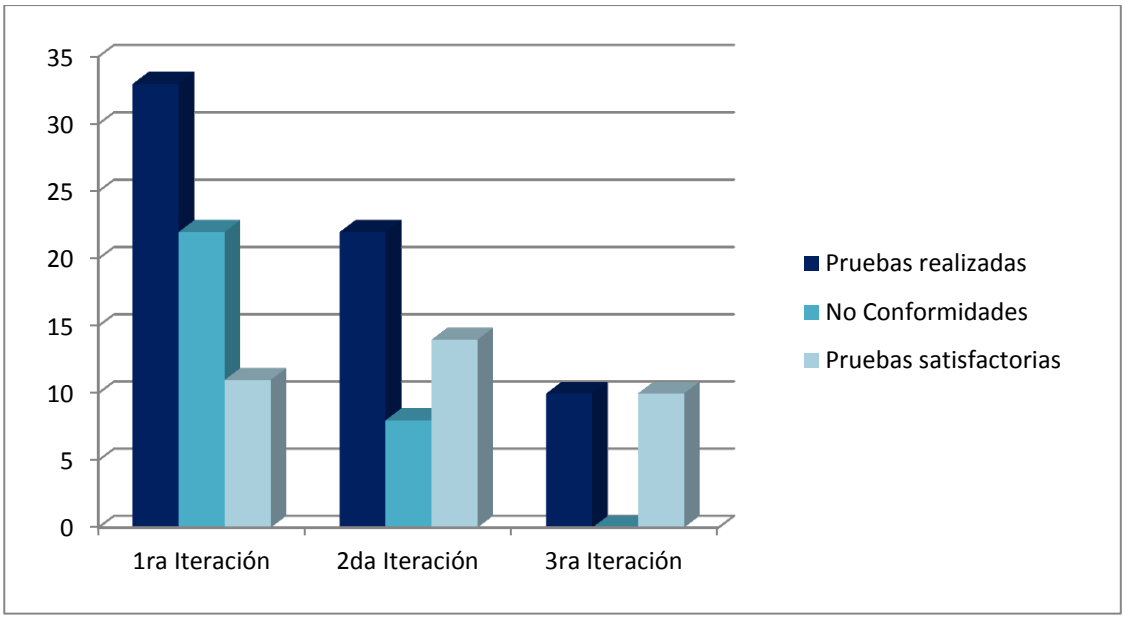


Figura 20. Resultados Obtenidos al aplicar las pruebas.

Conclusiones del capítulo

En este capítulo se definió el estándar de codificación con el objetivo de garantizar la uniformidad del código en el desarrollo del *plugin.* Se implementaron todos los requisitos y se evaluaron todas las funcionalidades del *plugin* utilizando el método de caja negra y la técnica de partición de equivalencia. Se diseñaron nueve CP con sus correspondientes escenarios que permitieron realizar pruebas a los 24 requisitos funcionales del *plugin*. Se encontraron 22 no conformidades las cuales fueron resueltas en tres iteraciones.

CONCLUSIONES

Teniendo en cuenta los objetivos trazados en la investigación y en aras de dar cumplimiento al problema propuesto, se arribó a las siguientes conclusiones:

- Se especificaron las herramientas a utilizar, definiendo como lenguaje de modelado *UML,* utilizando como herramienta *CASE Visual Paradigm* 8.0 y como lenguaje de programación C++, apoyado por el *framework* QT 4.5 y el *IDE* de desarrollo *QtCreator* 2.7*.*
- Se confeccionaron nueve HU a partir de las cuales se elaboraron 24 TI, y se representaron a través de 22 tarjetas CRC.
- Se utilizaron los patrones *GRASP (*experto, creador, controlador, bajo acoplamiento y alta cohesión*)* y *GoF (*bridge, observer*),* lo cual permitió reutilizar y estandarizar el diseño.
- Se implementaron las funcionalidades identificadas logrando la integración del *plugin* con la herramienta HABD, permitiendo aumentar las funcionalidades de dicha herramienta y facilitar la gestión de tablas en ella.
- Se realizaron pruebas funcionales a las nueve HU implementadas a través del método de caja negra y la técnica de partición de equivalencia, lo que posibilitó verificar el correcto funcionamiento del *plugin* y corregir las 22 no conformidades detectadas.

RECOMENDACIONES

Después de lograr los objetivos planteados que se trazaron al principio del presente trabajo se plantea la siguiente recomendación:

- Adicionar en la interfaz “Nueva Tabla” pestañas que permitan al usuario añadir más componentes en una misma interfaz.

REFERENCIAS BIBLIOGRÁFICAS

1. **Marqués, Mercedes.** *Apuntes de Ficheros y Bases de Datos.* Castellón de la Plana, España : Ingeniería Técnica en Informática de Gestión de la Universidad Jaume, 2009.

2. **Martel Jordan, Ernestina A. y Hernández Morera, Pablo.** *Sistema de Gestión de Asignaturas en Entorno Web.* Universidad de Las Palmas de Gran Canaria : s.n., 1997.

3. **DEITEL, Harvey M. y DEITEL, Paul J.** *Cómo programar en C/C++ y Java.* s.l. : Pearson Educación, 2004.

4. **Zambrano Ramírez , Raquel.** *Sistemas Gestores de Bases de Datos.* Granada : DEP.LegaL: GR 2922/2007, 2008. ISSN 1988-6047.

5. **Gil, Fidel, Abrigo, Javier y Do Rosario, Javier.** *Sistemas de Gestión de Base de Datos SGBD/DBMS.* Valencia : Universidad de Carabobo, Facultad experimental de Ciencias y Tecnología, Departamento de Computación, Unidad Académica de Base de Datos, 2005.

6. **Gil, Fidel , Albrigo, Javier y Do Rosario, Javier.** *SISTEMAS DE GESTIÓN DE BASE DE DATOS SGBD / DBMS.* Valencia : s.n., 2005.

7. **NOVELLA LATORRE, JAVIER.** *Sistema de gestión de base de datos PostgreSQL.* 2012.

8. Grupo de usuarios PostgreSQL de Argentina. [En línea] 2012. [Citado el: 2013 de 02 de 2.] http://www.arpug.com.ar/trac/wiki/TutorialPostgreSql.

9. http://es.kioskea.net. [En línea] 2008. [Citado el: 02 de 02 de 2013.] http://es.kioskea.net/contents/bdd/bddintro.php3.

10. Guía de la Comunidad para las herramientas GUI de PostgreSQL. [En línea] 18 de 05 de 2012. [Citado el: 18 de 04 de 2013.] http://wiki.postgresql.org/wiki/Gu%C3%ADa_de_la_Comunidad_para_las_herramientas_GUI_de_PostgreSQL.

11. software.com.ar. [En línea] 2007. [Citado el: 2013 de 01 de 23.] http://www.software.com.ar/dezign-for-databases.html..

12. Sqlmanager.net. [En línea] [Citado el: 14 de 01 de 2013.] http://www.sqlmanager.net/products/postgresql/manager.

13. mmc.geofisica.unam.mx. [En línea] 2001. [Citado el: 23 de 01 de 2013.] http://mmc.geofisica.unam.mx/LuCAS/Tutoriales/NOTAS-CURSO-BBDD/notas-curso-BD/node199.html.

14. Guía Documentada para Ubuntu. [En línea] 2012. [Citado el: 05 de 12 de 2012.] http://www.guia-ubuntu.org/.

15. **López Clemente, Alexis.** *Propuesta De Diseño De Una Base De Datos Para La Reducción Del Fondo Salarial En Una Universidad Mediante La Optimización Del Personal En Secretaría.* s.l. : Observatorio de la Economía Latinoamericana, 2012. 169.

16. msdn.microsoft.com. [En línea] 2013. [Citado el: 10 de 04 de 2013.] http://msdn.microsoft.com/es-es/library/ms189862%28v=sql.105%29.aspx.

17. **Sobrado, Ariel y Marrero, Luciano.** *Herramienta de Software para Dispersión de Archivos.* s.l. : Tesis Doctoral. Facultad de Informática., 2012.

18. **Canós, José H., Letelier, Patricio y Penadés, M. Carmen.** *Metodologías Ágiles en el Desarrollo de Software.* Valencia : Universidad Politécnica de Valencia, Camino de Vera, 2010. 46022.

19. es.kioskea.net. [En línea] 2008. [Citado el: 18 de 02 de 2013.] http://es.kioskea.net/contents/langages/langages.php3.

20. cprogramming.com. [En línea] 2011. [Citado el: 19 de 02 de 2013.] www.cprogramming.com.

21. **Lafosse, J.** *El framework de desarrollo de aplicaciones Java EE.* Barcelona : s.n., 2010.

22. **Martínez, Muñoz y León, Alberto.** *Implementación de un editor gráfico de circuitos eléctricos con Qt.* 2011.

23. *softpedia.es.* [En línea] 2013. [Citado el: 18 de 04 de 2013.] http://www.softpedia.es/programa-Qt-Creator-167814.html.

24. **Pérez Lovelle, Sonia.** *Herramienta para la verificación de sistemas basados en componentes y conectores genéricos.* España : Departamento de Lenguajes y Sistemas Informáticos,Universidad Politécnica de Catalunya, Barcelona, España.

25. **Marañon Rodríguez, Hermes Alexy y Duharte Montero, Pedro David.** *Planificación y Diseño del Portal para la Comunidad Técnica Cubana de PostgreSQL.* La Habana : s.n., 2010.

26. **Morejon Roque, Danichel y González Peraza, Adrián.** *Desarrollo de un bloque para Moodle 1.9.x que permita exportar cursos a un formato de libro electrónico interactivo.* La Habana : s.n., 2012.

27. genbetadev.com. [En línea] 28 de 02 de 2012. [Citado el: 04 de 05 de 2013.] http://www.genbetadev.com/metodologias-de-programacion/historias-de-usuario-una-forma-natural-de-analisis-funcional.

28. **Manuel Calero Solís.** Programación-extrema. [En línea] 2013. [Citado el: 16 de 05 de 2013.] http://programacion-extrema.wikispaces.com/7.+Artefactos.

29. mastermagazine.info. [En línea] 2012. [Citado el: 18 de 05 de 2013.] http://www.mastermagazine.info/termino/3916.php.

30. practicasdesoftware.com.ar. [En línea] 2011. [Citado el: 18 de 05 de 2013.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.

31. **Gamma, Erich, y otros, y otros.** *Design Patterns. Elements of Reusable Object-Oriented Software*|.

32. **Pressman, Roger S.** *Ingeniería de Software. Un enfoque práctico*|. s.l. : MC Graw Gill. 5ta Edición.

BIBLIOGRAFÍA

1. **Beck, Kent.** Embracing Change with Extreme Programming.
2. **Calidad y Software.com.** [En línea] 2009. [Citado el: 20 de 05 de 2013.] http://www.calidadysoftware.com/testing/pruebas_funcionales.phpv.
3. **Canós, José H., Letelier, Patricio y Penadés, M. Carmen.** *Metodologías Ágiles en el Desarrollo de Software.* Valencia : Universidad Politécnica de Valencia, Camino de Vera, 2010. 46022.
4. **Carrillo Pérez, Isaías, Pérez González, Rodrigo y Rodríguez Martín, Aureliano David.** *Metodología de Desarrollo del Software.* 2008.
5. **Coronado, Salvador Pozo.** *Curso de C++.*
6. **Cprogramming.com.** [En línea] 2011. [Citado el: 19 de 02 de 2013.] www.cprogramming.com.
7. **Cybertesis.upc.edu.p.** [En línea] 2008. [Citado el: 9 de 04 de 2013.] http://cybertesis.upc.edu.pe/upc/2008/vasquez_dc/html/TH.8.html .
8. **DBTools Software.** [En línea] 2001-2013. [Citado el: 2013 de 01 de 23.] http://www.dbtools.com.br/EN/index.php.
9. **DEITEL, Harvey M. y DEITEL, Paul J.** *Cómo programar en C/C++ y Java.* s.l. : Pearson Educación, 2004.
10. **Es.kioskea.net.** [En línea] 2008. [Citado el: 02 de 02 de 2013.] http://es.kioskea.net/contents/bdd/bddintro.php3.
11. **Es.kioskea.net.** [En línea] 2008. [Citado el: 18 de 02 de 2013.] http://es.kioskea.net/contents/langages/langages.php3.
12. **Gamma, Erich, y otros, y otros.** *Design Patterns. Elements of Reusable Object-Oriented Software*|.
13. **García de Jalón, Javier, y otros, y otros.** *Aprenda C++.* Universidad de Navarra : Escuela Superior de Ingenieros Industriales, 1998.
14. **Genbetadev.com.** [En línea] 28 de 02 de 2012. [Citado el: 04 de 05 de 2013.] http://www.genbetadev.com/metodologias-de-programacion/historias-de-usuario-una-forma-natural-de-analisis-funcional.
15. **Gil, Fidel , Albrigo, Javier y Do Rosario, Javier.** *SISTEMAS DE GESTIÓN DE BASE DE DATOS SGBD / DBMS.* Valencia : s.n., 2005.

16. **Gil, Fidel, Abrigo, Javier y Do Rosario, Javier.** *Sistemas de Gestión de Base de Datos SGBD/DBMS.* Valencia : Universidad de Carabobo, Facultad experimental de Ciencias y Tecnología, Departamento de Computación, Unidad Académica de Base de Datos, 2005.
17. **Grupo de usuarios PostgreSQL de Argentina**. [En línea] 2012. [Citado el: 2013 de 02 de 2.] http://www.arpug.com.ar/trac/wiki/TutorialPostgreSql.
18. **Guía de la Comunidad para las herramientas GUI de PostgreSQL.** [En línea] 18 de 05 de 2012. [Citado el: 18 de 04 de 2013.] http://wiki.postgresql.org/wiki/Gu%C3%ADa_de_la_Comunidad_para_las_herramientas_GUI_de_PostgreSQL.
19. **Guía Documentada para Ubuntu.** [En línea] 2012. [Citado el: 05 de 12 de 2012.] http://www.guia-ubuntu.org/.
20. **Juristo, Natalia, M. Moreno, Ana y Vegas, Sira.** *Técnicas de evaluación de software.* 2006.
21. **Lafosse, J.** *El framework de desarrollo de aplicaciones Java EE.* Barcelona : s.n., 2010.
22. **Larman, Craig y Hall, Prentice.** *UML y Patrones.* 2003.
23. **López Clemente, Alexis.** *Propuesta De Diseño De Una Base De Datos Para La Reducción Del Fondo Salarial En Una Universidad Mediante La Optimización Del Personal En Secretaría.* s.l. : Observatorio de la Economía Latinoamericana, 2012. 169.
24. **Manuel Calero Solís.** Programación-extrema. [En línea] 2013. [Citado el: 16 de 05 de 2013.] http://programacion-extrema.wikispaces.com/7.+Artefactos.
25. **Marañon Rodríguez, Hermes Alexy y Duharte Montero, Pedro David.** *Planificación y Diseño del Portal para la Comunidad Técnica Cubana de PostgreSQL.* La Habana : s.n., 2010.
26. **Marqués, Mercedes.** *Apuntes de Ficheros y Bases de Datos.* Castellón de la Plana, España : Ingeniería Técnica en Informática de Gestión de la Universidad Jaume, 2009.
27. **Martel Jordan, Ernestina A. y Hernández Morera, Pablo.** *Sistema de Gestión de Asignaturas en Entorno Web.* Universidad de Las Palmas de Gran Canaria : s.n., 1997.
28. **Martínez, Juan J.** *Programación C++.* 2003.
29. **Martínez, Muñoz y León, Alberto.** *Implementación de un editor gráfico de circuitos eléctricos con Qt.* 2011.
30. **Mastermagazine.info.** [En línea] 2012. [Citado el: 18 de 05 de 2013.] http://www.mastermagazine.info/termino/3916.php.

31. **Mmc.geofisica.unam.mx.** [En línea] 2001. [Citado el: 23 de 01 de 2013.] http://mmc.geofisica.unam.mx/LuCAS/Tutoriales/NOTAS-CURSO-BBDD/notas-curso-BD/node199.html.
32. **Molina, Jesús García.** *Patrones de Diseño.* Universidad de Murcia : Departamento de Informática y Sistemas, 2010.
33. **Molkenting, Daniel.** The Art of building Qt Aplications. Munich : s.n., 2007.
34. **Morejon Roque, Danichel y González Peraza, Adrián.** *Desarrollo de un bloque para Moodle 1.9.x que permita exportar cursos a un formato de libro electrónico interactivo.* La Habana : s.n., 2012.
35. **Msdn.microsoft.com.** [En línea] 2013. [Citado el: 10 de 04 de 2013.] http://msdn.microsoft.com/es-es/library/ms189862%28v=sql.105%29.aspx.
36. **NOVELLA LATORRE, JAVIER.** *Sistema de gestión de base de datos PostgreSQL.* 2012.
37. **Pérez Lovelle, Sonia.** *Herramienta para la verificación de sistemas basados en componentes y conectores genéricos.* España : Departamento de Lenguajes y Sistemas Informáticos,Universidad Politécnica de Catalunya, Barcelona, España.
38. **Practicasdesoftware.com.ar.** [En línea] 2011. [Citado el: 18 de 05 de 2013.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.
39. **Pressman, Roger S.** *Ingeniería de Software. Un enfoque práctico*|. s.l. : MC Graw Gill. 5ta Edición.
40. **Sánchez, Jorge.** *Principios sobre Bases de Datos Relacionales.* 2004.
41. **Sobrado, Ariel y Marrero, Luciano.** *Herramienta de Software para Dispersión de Archivos.* s.l. : Tesis Doctoral. Facultad de Informática., 2012.
42. ***Softpedia.es.*** [En línea] 2013. [Citado el: 18 de 04 de 2013.] http://www.softpedia.es/programa-Qt-Creator-167814.html.
43. **Sqlmanager.net.** [En línea] [Citado el: 14 de 01 de 2013.] http://www.sqlmanager.net/products/postgresql/manager.
44. **Zambrano Ramírez , Raquel.** *Sistemas Gestores de Bases de Datos.* Granada : DEP.LegaL: GR 2922/2007, 2008. ISSN 1988-6047.

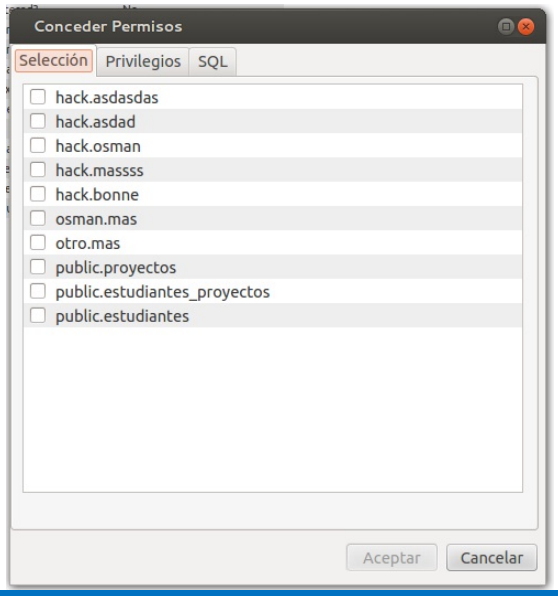
ANEXOS

Anexo 1: HU “Truncar tabla”

Historia de Usuario	
Número: 1	**Nombre de la Historia de Usuario:** Truncar tabla
Cantidad de modificaciones a la Historia de Usuario: 0	
Usuario: Yudelis Matos Martínez	**Iteración asignada:** 1ra
Prioridad en negocio: Alta	**Puntos estimados:** 0.6 semanas
Riesgo en desarrollo: Alta	**Puntos reales:** 0.6 semanas
Descripción: Borra el contenido de una tabla de manera rápida y elimina todo los registros sin ejecutar ninguna transacción.	
Observaciones: El usuario debe seleccionar el nombre de la tablas a la cual desea eliminar sus datos, sino se mostrará un mensaje de error.	

HU “Conceder Permisos sobre tablas”

Historia de Usuario	
Número: 2	**Nombre de la Historia de Usuario:** Conceder permisos sobre tablas
Cantidad de modificaciones a la Historia de Usuario: 0	
Usuario: Yudelis Matos Martínez	**Iteración asignada:** 2da
Prioridad en negocio: Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana

Descripción: Deniega o concede permisos a los usuarios para realizar acciones sobre las tablas de una BD.
Observaciones: El usuario puede otorgar o denegar privilegios en las tablas de insertar, eliminar, entre otros. Por defecto están seleccionados los que el usuario ya tiene otorgados.
Prototipo de Interfaz: 

Anexo 2: TI "Eliminar datos en una tabla utilizando menos recursos del sistema."

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Eliminar datos en una tabla utilizando menos recursos del sistema.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0.6 semanas
Fecha Inicio: 10/01/2013	**Fecha Fin:** 13/01/2013

Programador Responsable: Osman de Armas Sánchez
Descripción: Elimina todas las filas de una tabla sin registrar las eliminaciones individuales, utilizando menos recursos de registros de transacciones y de sistema. Al ejecutar esta acción se eliminarán todos los datos de una tabla; no permite el borrado selectivo. Esta funcionalidad vacía la tabla y conserva la estructura de la misma.

TI "Otorgar privilegios."

	Tarea de Ingeniería
Número Tarea: 2	**Número Historia de Usuario:** 2
Nombre Tarea: Otorgar privilegios	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 1 semana
Fecha Inicio: 15/01/2013	**Fecha Fin:** 20/01/2013
Programador Responsable: Osman de Armas Sánchez	
Descripción: Puede dar los siguientes derechos o privilegios a los usuarios de bases de datos o roles (grupos de usuarios) autenticadas en el sistema: *SELECT* (seleccionar datos de una tabla), *INSERT* (insertar nuevos registros en una tabla), UPDATE (modificar campos de datos en una tabla), *DELETE* (eliminar registros de una tabla), *ALL* (todos los derechos sobre una tabla), entre otros.	

Anexo 3: Tarjeta CRC "GrantTable"

Tarjeta CRC	
Clase: GrantTable	
Responsabilidades	**Colaboraciones**
Conceder *ALL, INSERT, DELETE, UPDATE, SELECT*	Controller Query

GLOSARIO DE TÉRMINOS

Datos: expresiones generales que describen características de las entidades sobre las que operan los algoritmos, en BD representan información perteneciente a un mismo contexto.

***DDL*:** lenguaje para definir y describir los objetos de la BD, su estructura, relaciones y restricciones.
Identación: espacios que se dan al inicio de una línea de código.

Implementación: poner en funcionamiento, aplicar métodos y medidas para la realización de un programa o componente software.

Procedurales: relativo al procedimiento; procesal.

QT: Nombre de la compañía que lo creó: *Quasar Technologies*.

SQL: tipo de lenguaje vinculado con la gestión de bases de datos de carácter relacional que brinda la posibilidad de realizar consultas con el objetivo de recuperar información de las bases de datos de manera sencilla.

Validación: operación de asegurar que los requerimientos funcionales y no funcionales están correcta y completamente implementados en el producto final.