

Universidad de las Ciencias Informáticas

Facultad 6



Título: Plugin “Editor de funciones y disparadores” para la herramienta de administración de bases de datos HADB.

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

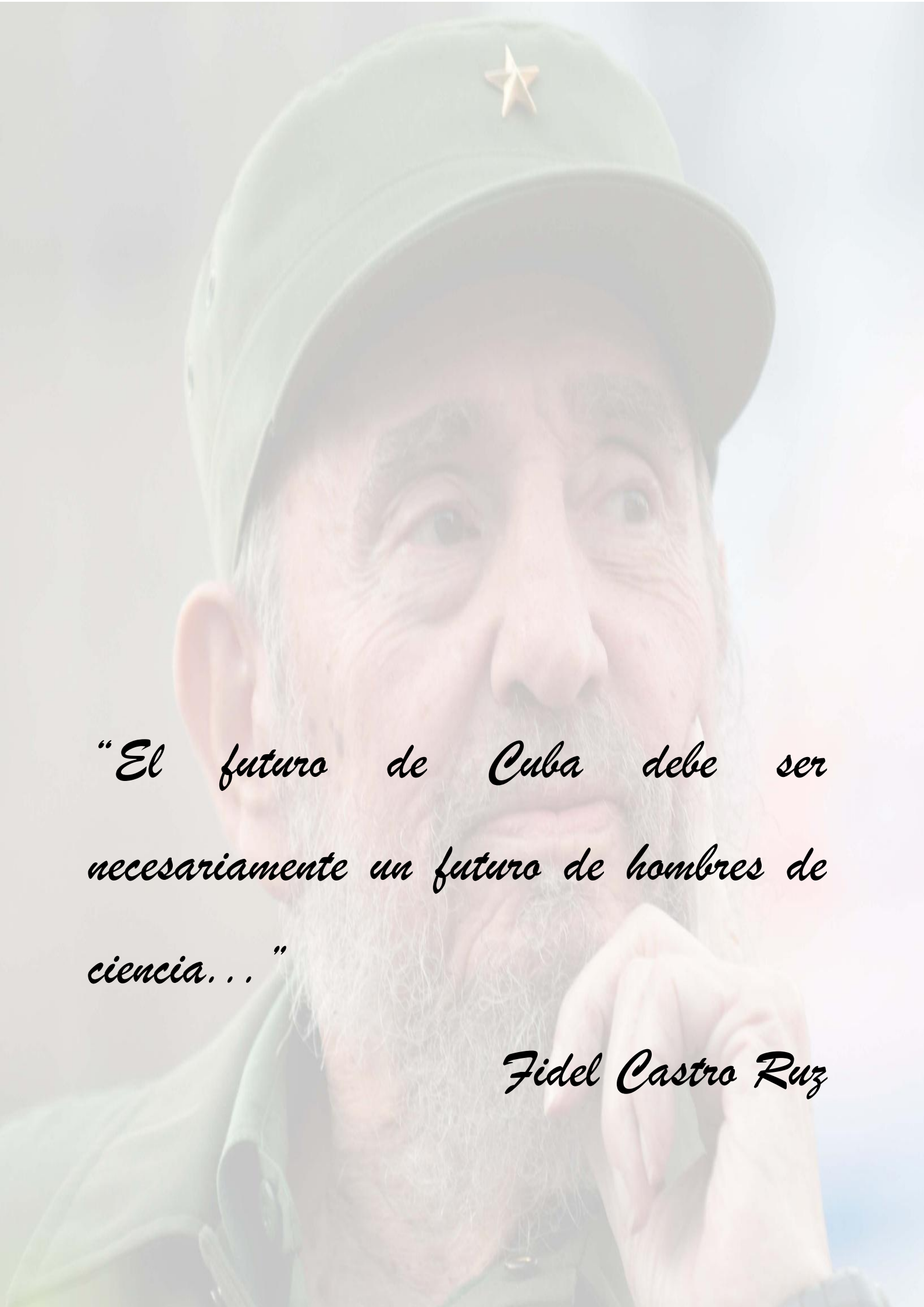
Autores:

Tahimé Vázquez Montero
Daniel Alejandro Linares Brooks

Tutores:

Ing. Aislein Blanco González
Ing. Yunior Bauta Pentón

La Habana, Cuba
“Año 54 de la Revolución”
Junio 2012



“El futuro de Cuba debe ser necesariamente un futuro de hombres de ciencia...”

Fidel Castro Ruz

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo de diploma y se reconoce a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma la presente a los ____ días del mes de _____ del año _____.

Tahimé Vázquez Montero

Daniel Alejandro Linares Brooks

Ing. Yunior Bauta Pentón

Ing. Aislein Blanco González

Tutor

Ing. Yunior Bauta Pentón
Universidad de las Ciencias Informáticas, Habana, Cuba
e-mail: ypenton@uci.cu

Tutora

Ing. Aislein Blanco González
Universidad de las Ciencias Informáticas, Habana, Cuba
e-mail: ablanco@uci.cu

Agradecimientos

Resulta difícil describir con palabras todo lo que siento por esas personas a las que les debo hasta la última gota de mi sangre. Nunca podré agradecerles como se merecen porque no existen palabras para describir todo lo que siento. Gracias por el esfuerzo tan grande que han hecho para que no me falta de nada, gracias por convertirme en lo que soy hoy y discúlpeme si alguna vez hice cosas de niña mal agradecida.

Mama, ahora al intentar expresar mi amor por ti, mi voz se desaparece. Lo importante es que tú sepas que existe y que continuará vivo aunque estemos lejos. Siempre me ha resultado difícil decirte “Te quiero”, hoy en uno de los días más importantes de mi vida quiero que sepas, que no te quiero. “Te adoro”.

Papi, gracias por creer en mí, supiste desde niña que iba ser ingeniera. Nunca se me va a olvidar esta conversación -papi yo quiero ir a La Habana - irás cuando estudies, esfuérate para que estudies allá. Gracias por darte cuenta de la importancia de un “Te quiero”, esas palabras tuyas me dieron más fuerza para terminar mi carrera.

Gracias a mi familia, todos de una forma u otra me han apoyado. No puedo dejar de mencionar a mi hermano, el que más me malcría. Aunque no me quieras enseñar a manejar yo te quiero mucho, te agradezco esos buenos consejos que a su tiempo llegaron, gracias Maiquel por ser como eres.

Les doy las gracias a mis tutores Yúnior y Aislein por la paciencia y comprensión en el transcurso de la tesis. Gracias al departamento PostgreSQL, en especial, a Edgar, Marianela, Marcos y Glennis por su apoyo y preocupación.

Muchas son las personas que me ayudaron no solo en el proceso de la tesis, sino también a lo largo de la carrera, gracias por su amistad y ayuda a el profe Yunier René, Osmar, Carlos Ernesto, Raulito, Alexis o como le decimos todos JBich, José Mario, Emilia (Tahymi), Daylig, Ricardo, Yosmany, Manuel, Lincoln, Despaigne, Yeneysi, chuchita (Yadira), chuchito (Karel) y a los embarca play Richard, Yerandy y el Yoe.

En estos últimos meses de risas y llantos quiero agradecerles a Ibet y a Betty por estar siempre ahí, por escucharme hablar cosas sin sentido después de un día sin dormir, por hacerme reír cuando en realidad lo que tenía eran ganas de llorar, gracias a las dos por su amistad incondicional.

Tahimé Vázquez Montero

Dedicatoria

Dedico este trabajo a mis padres Tania Montero Borges y Jorge Luis Vázquez Rondón, esas personas que dieron todo por mí sin esperar nada a cambio, que siempre han confiado en mí y me han brindado su apoyo ilimitado. A ustedes que me dieron la vida, que me inculcaron la importancia de estudiar para ser lo que hoy he logrado.

Gracias por ser como son, los quiero.

Resumen

En la Universidad de las Ciencias Informáticas, específicamente en el Centro de Tecnologías de Gestión de Datos existe una línea de producción que lleva por nombre PostgreSQL. En la misma se propone el desarrollo de una herramienta de administración de base de datos con una arquitectura basada en plugins llamada HABD. Esta herramienta provee una interfaz amigable y su arquitectura permite incorporar nuevas funcionalidades de manera sencilla, evitándole al usuario la necesidad de utilizar varias aplicaciones para resolver un problema. Actualmente HABD se encuentra en espera de nuevos componentes para su completitud. El presente trabajo propone el desarrollo del Plugin “Editor de funciones y disparadores” que será incorporado a dicha herramienta. Luego del estudio de varios editores de diferentes herramientas de administración de bases de datos; el plugin logra una interfaz compacta, intuitiva y sencilla, permitiendo de esta forma la gestión, el filtrado por nombre de funciones y disparadores entre otras. La implementación de estas funcionalidades constituye una necesidad primaria para los usuarios ya que les permite definir operadores propios y almacenar procedimientos que podrán lanzarse cuando el usuario desee.

PALABRAS CLAVE

Disparadores, Editor de funciones y disparadores, Funciones, HABD, Plugin

ÍNDICE

| | |
|--|----|
| INTRODUCCIÓN..... | 1 |
| CAPÍTULO 1: FUNDAMENTO TEÓRICO..... | 5 |
| Introducción..... | 5 |
| 1.1 Base de datos y sistemas gestores de bases de datos..... | 5 |
| 1.2 Herramientas de administración de base de datos..... | 7 |
| 1.4 Editor de funciones..... | 9 |
| 1.5 Editor de disparadores..... | 12 |
| 1.6 Metodología de desarrollo de software. Extreme Programming..... | 16 |
| 1.7 Tecnologías y herramientas a utilizar..... | 17 |
| Conclusiones parciales..... | 21 |
| CAPÍTULO 2: CARACTERÍSTICAS DEL PLUGIN “EDITOR DE FUNCIONES Y DISPARADORES”..... | 22 |
| Introducción..... | 22 |
| 2.1 Identificación del problema..... | 22 |
| 2.2 Modelo de dominio..... | 22 |
| 2.3 Propuesta del Sistema..... | 23 |
| 2.4 Historias de Usuarios..... | 24 |
| 2.5 Lista de reserva del producto..... | 26 |
| 2.6 Tareas de la ingeniería..... | 29 |
| 2.7 Plan de iteraciones..... | 30 |
| 2.8 Diseño del sistema..... | 30 |

| | |
|--|----|
| 2.9 Patrón de arquitectura..... | 33 |
| 2.10 Patrones de diseño | 35 |
| 2.11 Estándares de codificación..... | 40 |
| 2.12 Interfaces de la aplicación..... | 45 |
| Conclusiones parciales | 47 |
| CAPÍTULO 3: VALIDACIÓN DEL PLUGIN “EDITOR DE FUNCIONES Y DISPARADORES” | 48 |
| 3.1 Enfoques de diseño de pruebas..... | 48 |
| 3.2 Pruebas de aceptación..... | 49 |
| 3.3 Presentación de resultados..... | 52 |
| Conclusiones parciales | 52 |
| CONCLUSIONES | 53 |
| RECOMENDACIONES | 54 |
| REFERENCIA BIBLIOGRÁFICA..... | 55 |
| BIBLIOGRAFÍA..... | 57 |

Índice de figuras

| | |
|--|--------------------------------------|
| Figura 1: Interfaz visual del editor de funciones del PgAdmin3 | 10 |
| Figura 2: Interfaz visual del Editor de funciones del EMS SQL Manager for PostgreSQL | 11 |
| Figura 3: Interfaz visual del editor de disparadores del PgAdmin3 | 13 |
| Figura 4: Editor de disparadores del SQL Manager for PostgreSQL | 14 |
| Figura 5: Modelo de dominio | 23 |
| Figura 6: Mapa conceptual. Propuesta del sistema | 24 |
| Figura 7: Diagrama de clases del sistema..... | ¡Error! Marcador no definido. |
| Figura 8: Patrón MVC | 34 |
| Figura 9: Ejemplo del patrón Experto | 35 |
| Figura 10: Ejemplo del patrón Creador | 36 |
| Figura 11: Ejemplo del patrón Alta cohesión | 37 |
| Figura 12: Ejemplo del patrón Bajo acoplamiento | 38 |
| Figura 13: Patrón Singleton | 40 |
| Figura 14: Interfaz visual del plugin "Editor de funciones y disparadores" para la herramienta de administración HABD. Funciones..... | 46 |
| Figura 15: Interfaz visual del plugin "Editor de funciones y disparadores" para la herramienta de administración HABD. Disparadores..... | 47 |
| Figura 16: Resultados de la aplicación de los casos de pruebas. | 52 |

Índice de tablas

| | |
|--|----|
| Tabla 1: Características seleccionadas para el desarrollo del plugin "Editor de funciones y disparadores"..... | 15 |
| Tabla 2: Ejemplo de Historia de Usuario | 25 |
| Tabla 3: Lista de reserva del producto | 26 |
| Tabla 4: Ejemplo de Tareas de Ingeniería..... | 30 |
| Tabla 5: Tarjeta CRC. Funciones..... | 32 |
| Tabla 6: Tarjeta CRC. Disparadores | 32 |
| Tabla 7: Tarjeta CRC. Argumentos | 32 |
| Tabla 8: Ejemplo del caso de pruebas aplicado a la Historia de Usuario "Exportar función" | 50 |
| Tabla 9: Ejemplo de la descripción de las variables, correspondiente a la sección "Exportar función" | 51 |

Introducción

En la década de los sesenta se introducen las primeras computadoras para la automatización de las empresas, convirtiendo la gestión de los datos en un tema de considerable repercusión. Nació así, la primera generación de productos de bases de datos en red, donde su uso se desarrolló a partir de la necesidad existente de almacenar grandes cantidades de datos producidas por las nuevas empresas responsables de la generación de cuantioso cúmulo de información, para su posterior consulta.

Las bases de datos constituyen una de las herramientas de almacenamiento ampliamente difundidas en la actual sociedad, utilizadas ante todo como fuentes secundarias en cuanto a recuperación y acumulación de información en todos los campos a nivel científico, social, económico, político y cultural. (1)

Surgió la necesidad de contar con un Sistema Gestor de Base de Datos (SGBD) para controlar, administrar y acceder, tanto a los usuarios como a los datos, suministrando de igual manera a los usuarios no informáticos como a los analistas, programadores o al administrador, los medios necesarios para recuperar y manipular la información almacenada, manteniendo su integridad, confidencialidad y seguridad.

Se pueden encontrar SGBD libres y propietarios: los libres son aquellos que se obtienen sin costo alguno, por ejemplo PostgreSQL y MySQL; y los propietarios son por los que se tiene de pagar según el tipo de licencia, ejemplo Microsoft SQL server, Oracle y MySQL; este último posee Licencia Dual, en dependencia del uso. (2)

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional de propósito general, multiusuario y de código abierto, liberado bajo la licencia Berkeley Software Distribution (BSD). Soporta gran parte del estándar Structure Query Language (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 optimista en 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 agregadas, ventanas, métodos de indexado y lenguajes procedurales. Es reconocido como un gestor de bases de datos de código abierto de gran potencia (3). Cuenta

con una amplia y sólida comunidad de desarrollo independiente ya establecida. Libera versiones mayores anualmente y cada tres meses (aproximadamente), versiones menores para arreglar los errores (bugs) de las mayores (4).

PostgreSQL incluye dentro de sus liberaciones la consola psql, mediante la cual se pueden hacer todas las operaciones básicas del gestor. El trabajo con la misma requiere de conocimientos avanzados por lo que se han creado varias herramientas con interfaces gráficas para administrar fácilmente los servidores, muchas de ellas propietarias, entre las cuales se pueden mencionar PGManager, Lightning Admin for PostgreSQL, EMS Manager for Postgresql, Navicat entre otras. También existen herramientas libres para la administración del gestor, de ellas la más conocida por su gran facilidad de uso es PgAdmin3.

En el 2011 en la Universidad de las Ciencias Informáticas (UCI) se realizó una investigación que propone, el desarrollo de una herramienta de administración de bases de datos para PostgreSQL llamada HABD, con una arquitectura basada en plugins. Los usuarios que interactúen con ella pueden agregar tantas funcionalidades como necesiten. Dicha herramienta ya está implementada y en espera de nuevos componentes para su completitud.

Al usuario que interactúe con la herramienta se le dificulta definir operadores propios y almacenar procedimientos, que estos a su vez permiten incorporar lógica del negocio dentro de la base de datos. Además, existen situaciones donde el usuario necesita realizar recuperaciones de datos o evitar la pérdida de información a manos del usuario.

Por lo antes planteado se define el siguiente **problema de la investigación**: ¿Cómo realizar la programación de rutinas del lado del servidor desde la herramienta de administración de bases de datos HABD? Se plantea como **objeto de estudio**: Herramientas de administración de bases de datos PostgreSQL. Enmarcados en el **campo de acción**: Editores de funciones y disparadores de las herramientas de administración para PostgreSQL.

El **objetivo general** de la investigación es: Desarrollar un plugin para la herramienta de administración de bases de datos HABD para editar funciones y disparadores del lado del servidor.

Para dar solución al problema antes mencionado se trazan los siguientes **objetivos específicos**:

1. Caracterizar las principales funcionalidades de las herramientas de administración de

bases de datos en PostgreSQL, haciendo énfasis en los editores de funciones y disparadores de las mismas.

2. Diseñar el Plugin Editor de funciones y disparadores.
3. Implementar el Plugin Editor de funciones y disparadores.
4. Realizar pruebas al Plugin Editor de funciones y disparadores.

Para dar cumplimiento a estos objetivos se definen las siguientes **tareas de la investigación**:

1. Revisión bibliográfica sobre los diferentes SGBD y sus herramientas de administración.
2. Revisión bibliográfica sobre la implementación de funciones y disparadores dentro del gestor de base de datos PostgreSQL.
3. Selección de las características y funcionalidades que tendrá el “Editor de funciones y disparadores”.
4. Elaboración de los artefactos “Historias de Usuarios”.
5. Elaboración del artefacto “Lista de Reserva del Producto”.
6. Elaboración del artefacto “Modelo de Diseño”.
7. Elaboración del artefacto “Tarea de Ingeniería”.
8. Elaboración del artefacto “Plan de Release”.
9. Implementación del Plugin Editor de funciones y disparadores.
10. Diseño de los casos de pruebas a aplicar en el Plugin Editor de funciones y disparadores.
11. Aplicación de los casos de pruebas para validar que el Plugin Editor de funciones y disparadores responda a las necesidades del usuario final.

La presente investigación está compuesta por 3 capítulos en los cuales se reflejará todo lo referente al desarrollo del “Editor de funciones y disparadores” para la herramienta HABD. A continuación se muestra una breve descripción de cada capítulo.

Capítulo 1: Fundamento Teórico. En el presente capítulo se realiza un análisis de las herramientas de administración de bases de datos y las características de los editores de

funciones y disparadores. Contiene las definiciones generales de las tecnologías y herramientas a utilizar en el desarrollo de este trabajo de diploma.

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”. Se describen las características principales que posee el sistema. Se realiza el modelo de dominio, se confeccionan las Historias de Usuarios que serán implementadas y las Tareas de la Ingeniería asociadas a cada Historias de Usuario. Para el diseño del sistema se elabora el diagrama de clases y las tarjetas CRC. De esta forma se tendrá un entendimiento más amplio de la propuesta del plugin.

Capítulo 3: Validación del Plugin “Editor de funciones y disparadores”. Es necesario comprobar de alguna forma que el plugin cumple con los requisitos especificados en las Historias de Usuarios. Para comprobar su buen funcionamiento se escoge dentro de las pruebas que define XP, la prueba que permita lo antes planteado. El resultado de las pruebas realizadas será mostrado mediante una tabla para un mejor entendimiento.

Capítulo 1: Fundamento teórico

Introducción

En el presente capítulo se realiza un análisis de las herramientas de administración de bases de datos y las características de los editores de funciones y disparadores. Contiene las definiciones generales de las tecnologías y herramientas a utilizar en el desarrollo de este trabajo de diploma.

1.1 Base de datos y sistemas gestores de bases de datos

Las bases de datos constituyen un conjunto de datos almacenados y organizados lógicamente. Tienen la capacidad de acumular una gran cantidad de información, teniendo como prioridad garantizar la persistencia de los mismos por un período de tiempo. Para definir, crear y mantener una base de datos, se crean los Sistemas Gestores de Base de Datos (SGBD). Existen varios SGBD como MySQL, Oracle, SQL Server y PostgreSQL, entre otros; estos ayudan a realizar las siguientes acciones dirigidas al trabajo con los datos: definición, mantenimiento e integridad en la base de datos, manipulación, control de la seguridad y privacidad.

Existen distintos objetivos que deben cumplir como:

- ✓ **Abstracción de la información.** Los SGBD ahorran a los usuarios detalles acerca del almacenamiento físico de los datos. No es relevante si una base de datos ocupa uno o cientos de archivos, este hecho se hace transparente al usuario. Así, se definen varios niveles de abstracción.
- ✓ **Independencia.** La independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.
- ✓ **Consistencia.** En aquellos casos en los que no se ha logrado eliminar la redundancia, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea. Por otra parte, la base de datos representa una realidad determinada que tiene determinadas condiciones, por ejemplo que los menores de edad no pueden tener licencia de conducir. El sistema no debería aceptar datos de un conductor menor de edad.

- ✓ **Seguridad.** La información almacenada en una base de datos puede llegar a tener un gran valor. Los SGBD deben garantizar que esta información se encuentra segura de permisos a usuarios y grupos de usuarios, que permiten otorgar diversas categorías de permisos.
- ✓ **Manejo de transacciones.** Una transacción es un programa que se ejecuta como una sola operación. Esto quiere decir que luego de una ejecución en la que se produce una falla es el mismo que se obtendría si el programa no se hubiera ejecutado. Los SGBD proveen mecanismos para programar las modificaciones de los datos de una forma mucho más simple que si no se dispusiera de ellos.
- ✓ **Tiempo de respuesta.** Lógicamente, es deseable minimizar el tiempo que el SGBD demora en proporcionar la información solicitada y en almacenar los cambios realizados.
(5)

➤ **PostgreSQL**

En la UCI se contribuye a la independencia tecnológica que quiere alcanzar Cuba en cuanto a la utilización de software libre. Una de las líneas de producción que conforman al Centro de Tecnologías de Gestión de Datos (DATEC) es PostgreSQL. Esta línea tiene como objetivo desarrollar varias herramientas de administración para el gestor PostgreSQL. Actualmente se han implementado algunas herramientas como CRUD-PG, Naire, Enmascarador de datos y HABD. La presente investigación apoya la independencia tecnológica al contribuir con la herramienta HABD.

"PostgreSQL 9.1 provee algunas de las más avanzadas características empresariales de cualquier base de datos de código abierto, y es soportado por una entusiasta e innovativa comunidad con probados casos de éxito. PostgreSQL está muy bien preparada para construir y correr aplicaciones en la nube." palabras de Charles Fan, Sr. VP R&D, VMware. (6)

PostgreSQL, engloba características que lo hacen considerablemente potente y robusto. Esta se caracteriza por su estabilidad, potencia, robustez, facilidad de administración e implementación de estándares. Para este gestor no significa un problema las grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez en el sistema. Entre otras características se

evidencian las copias de seguridad en caliente, juegos de caracteres internacionales, acceso encriptado vía SSL, multiplataforma e integridad referencial.

1.2 Herramientas de administración de base de datos

La administración de base de datos se realiza con un sistema llamado Sistema de Administración de Bases de Datos, por sus siglas en inglés, Data Base Management System (DBMS). El DBMS es un conjunto de servicios que permite un fácil acceso a los datos y a la información por parte de múltiples usuarios, así como la manipulación de los datos encontrados en la base de datos, dígame insertar, eliminar o editar.

Algunas de las funcionalidades de un BDMS son: la definición de la base de datos, cómo la información va a ser almacenada y organizada, creación de la base de datos, almacenamiento, recuperación y actualización de datos; consultas y reportes, cambiar los contenidos de la base de datos, programación de aplicaciones para el desarrollo de software, control de la integridad y monitoreo del comportamiento de la base de datos. (7)

Existen diversas herramientas de administración para diferentes tipos de gestores. Las herramientas DreamCoder for Oracle y Oracle Enterprise Manager son utilizadas para el gestor Oracle, mientras que para MySQL se utilizan otras como MySQL Woekbench y phpMyAdmin. Herramientas para PostgreSQL son muchas, en la actualidad las más utilizadas por sus características son, el PgAdmin3 y EMS SQL Manager 2007 for PostgreSQL.

Es necesario realizar un estudio de las herramientas de administración para PostgreSQL, con el fin de analizar las características visuales que presentan los editores de funciones y disparadores e indagar en la forma intuitiva en que se obtienen y tratan los datos para la gestión de funciones y disparadores. De esta forma el equipo de desarrollo obtiene una visión de las funcionalidades que puede o no tener el plugin referente a esta investigación.

De acuerdo al carácter comercial, se pueden agrupar en herramientas de:

- ✓ **Código abierto / Software Libre:** PgAdmin III, PGAccess, phpPgAdmin, OpenOffice.org, Herramientas gráficas para la Red Hat Database, Administrador de RHDB y Explain Visual, Xpg: cliente de Java para PostgreSQL, Mergeant, Tora (herramienta de Oracle con soporte para PostgreSQL), KNoda, PGInhaler, SquirrelL, AnySQL Maestro, PostgreSQL PHP, Generator y WaveMaker Ajax GUI Design Tool.

- ✓ **Comerciales o privativas:** Lightning Admin for PostgreSQL, Borland Kylix, DBOne, DBTools Manager, PgManager, Rekall, Data Architect, SyBase Power Designer, Microsoft Access, DeZign for Databases, PGExplorer, Case Studio 2, pgEdit, RazorSQL, MicroOLAP Database Designer, Aqua Data Studio, EMS SQL Management for PostgreSQL, Navicat, SQL Maestro Group products for PostgreSQL, DB Data Diffective PostgreSQL Edition, DB Schema Diffective PostgreSQL Edition, DB MultiRun PostgreSQL Edition, SQLPro y SQL Image Viewer. (8)

Teniendo en cuenta el grado de aceptación que tienen en la actualidad las herramientas en disímiles comunidades de desarrollo, el grado de facilidad de uso por parte de los usuarios, el número de funcionalidades que incorporan en el manejo de PostgreSQL, plataformas de Sistema Operativo que soportan, el grado de mantenimiento y el desarrollo que presentan en el proyecto, se escogen las herramientas EMS SQL Managment for PostgreSQL y PgAdmin III para el análisis de sus características esenciales.

El EMS SQL Management for PostgreSQL en estos momentos es una de las herramientas comerciales o privativas disponible y las características de su editor de funciones y disparadores cumplen con el objetivo de la investigación.

La herramienta PgAdmin3 es una de las más usadas, tiene buena aceptación por parte de los usuarios y al ser software libre se puede analizar el comportamiento de su editor de funciones y disparadores.

➤ **PgAdmin3**

PgAdmin3 es la herramienta de administración de código abierto por excelencia para las BD PostgreSQL. Incluye una interfaz gráfica de administración, una herramienta para el trabajo con SQL, un editor de código de procedimientos y funciones, entre otras. Está diseñado para darle respuesta a las necesidades de la mayoría de los usuarios, desde la escritura de consultas simples en SQL hasta el desarrollo de BD complejas. Permite la construcción gráfica de una determinada consulta, la inclusión del framework pgScript para el desarrollo de scripts usados en la ejecución de consultas, buscador de objetos FTS, menú para habilitar/deshabilitar reglas, la posibilidad de borrar o reasignar un rol a una determinada BD. (9)

➤ **EMS SQL Management for PostgreSQL**

EMS SQL Manager for PostgreSQL es una herramienta de alto rendimiento para la administración del servidor de bases de datos PostgreSQL. Se ofrece una amplia gama de potentes herramientas para los usuarios experimentados tales como diseñador de bases de datos Visual, Visual Query Builder y un editor BLOB de gran alcance para satisfacer todas sus necesidades. EMS SQL Manager for PostgreSQL tiene un nuevo estado de la interfaz gráfica de usuario, muy clara en su uso, que incluso un novato no debe confundirse con ella. (10)

Estas herramientas tienen en común que permiten la creación de base de datos y tablas, además permiten gestionar funciones y disparadores. La investigación estará centrada en estas dos últimas funcionalidades.

1.4 Editor de funciones

Las funciones son bloques de código que se ejecutan en el servidor, estos pueden ser escritos en varios lenguajes, desde las operaciones básicas de la programación hasta las complejidades de la programación orientada a objetos o la programación funcional. Las funciones pueden recibir parámetros, especificándoles el tipo de dato. (11) Una función está compuesta por tres partes:

Encabezado: define el nombre de la función y el tipo de retorno.

Cuerpo: cadena de texto.

Especificación: especifica el lenguaje utilizado.

Algunos gestores de bases de datos relacionales pueden incorporar múltiples lenguajes de programación a una base de datos en particular. Esto permite almacenar procedimientos en la base de datos que podrán lanzarse cuando convenga, además de definir operadores propios.

La gestión de funciones se puede realizar mediante la consola o de forma visual. A través de la consola es necesario tener conocimientos avanzados del lenguaje, por lo que se decide analizar las formas visuales de los editores de las herramientas PgAdmin3 y EMS SQL Manager for PostgreSQL. Se analizan con el objetivo de incorporar funcionalidades al plugin referente a esta investigación, la cual permitirá la gestión de funciones de forma visual.

Editor de funciones del PgAdmin3

El Editor de funciones de la herramienta PgAdmin3 para la creación y edición de funciones presenta varias secciones como:

- ✓ Propiedades: recoge los datos generales (nombre, propietario, tipo devuelto, lenguaje y comentarios) de la función al ser creada.
- ✓ Opciones: especifica los aspectos referentes a cómo se puede ejecutar la función.
- ✓ Parámetros: recoge el listado de los argumentos con los que va a trabajar la función.
- ✓ Definición: declara el cuerpo de la función, es decir, lo que realizará la subrutina.
- ✓ Variables: permite utilizar variables propias de PostgreSQL.
- ✓ Privilegios: define los permisos que tendrá el usuario para ejecutar la función.
- ✓ SQL: obtiene el código asociado a la creación de la función.

La siguiente imagen muestra como el editor de funciones en la herramienta PgAdmin3 brinda una serie de facilidades a los usuarios tales como: gestionar una función a través de una interfaz gráfica de usuario sin hacer uso de la consola psql.

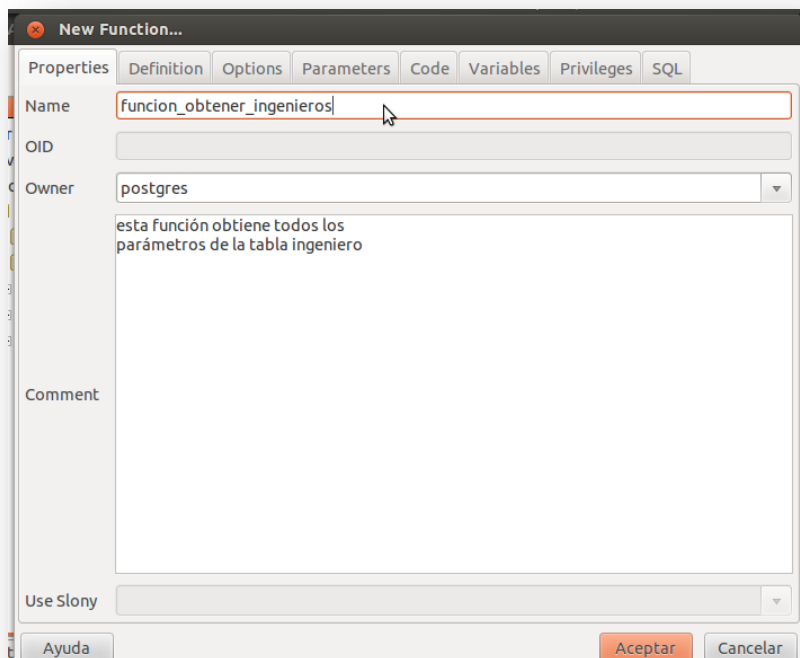


Figura 1: Interfaz visual del editor de funciones del PgAdmin3

Editor de funciones del EMS SQL Manager for PostgreSQL

En la herramienta EMS SQL Manager for PostgreSQL el editor de funciones presenta varias secciones para crear una función:

- ✓ Función: recoge los datos (nombre, tipo devuelto, lenguaje, argumentos y definición o cuerpo de la función) necesarios para crear una función.
- ✓ Dependencias: muestra si esa función creada depende de algún otro objeto de PostgreSQL.
- ✓ Descripción: describe la acción que realizará la función.
- ✓ DDL: muestra el código asociado a la función creada.
- ✓ Permisos: establece los privilegios por grupos y usuarios.
- ✓ Resultados: muestra el resultado asociado a la función.

Esta herramienta tiene algunas características ventajosas. Algunas de estas son, que en una sección o pestaña se recoge toda la información necesaria para la creación de la función, tiene además varias vistas para presentar los resultados, lo que permite un mejor manejo de los mismos en caso de gran cantidad de datos obtenidos. La siguiente imagen muestra lo antes mencionado:

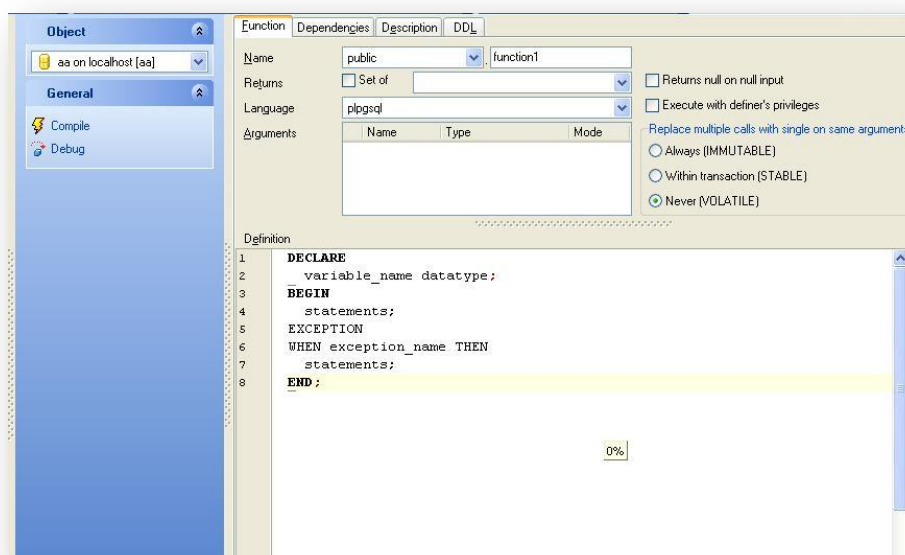


Figura 2: Interfaz visual del Editor de funciones del EMS SQL Manager for PostgreSQL

1.5 Editor de disparadores

Un disparador es un código o programa que es almacenado en la base de datos y se ejecuta en respuesta a un evento específico. Este evento está asociado con una tabla, una vista o un esquema dentro de una base de datos. Es un mecanismo con el cual puede automatizar tareas, de forma que estas sean ejecutadas antes o después de producido un determinado evento sobre un objeto.

En PostgreSQL el mecanismo de disparadores está constituido por dos partes, las funciones disparadoras y el disparador en sí. Las funciones disparadoras tienen como características: no pueden recibir argumentos y deben retornar el tipo de dato especial denominado **trigger**. El disparador es quien se encarga de asociar una función determinada a la tabla y el evento deseado. El editor de disparadores permite generar disparadores recogiendo los datos necesarios para su creación, así como editarlos mostrando todos sus datos disponibles y editables. A continuación se analizan las formas visuales de los editores de las herramientas: PgAdmin3 y SQL Manager for PostgreSQL.

Editor de disparadores del PgAdmin3

En la herramienta PgAdmin3 para crear un disparador se debe crear una función disparadora (función que devuelve un tipo de dato llamado trigger). Otra vía para su creación sería utilizar alguna función de este tipo ya creada con anterioridad. El editor de disparadores posee las siguientes secciones:

- ✓ Propiedades: recogen datos (nombre, argumentos, comentario), se selecciona la función disparadora existente, se especifica el evento (pueden ser: INSERT, DELETE, UPDATE y TRUNCATE) y se especifica además el tipo (BEFORE o AFTER).
- ✓ SQL: muestra el código asociado al disparador creado.

En la siguiente imagen se muestran las características de forma visual para una mejor comprensión de sus propiedades.

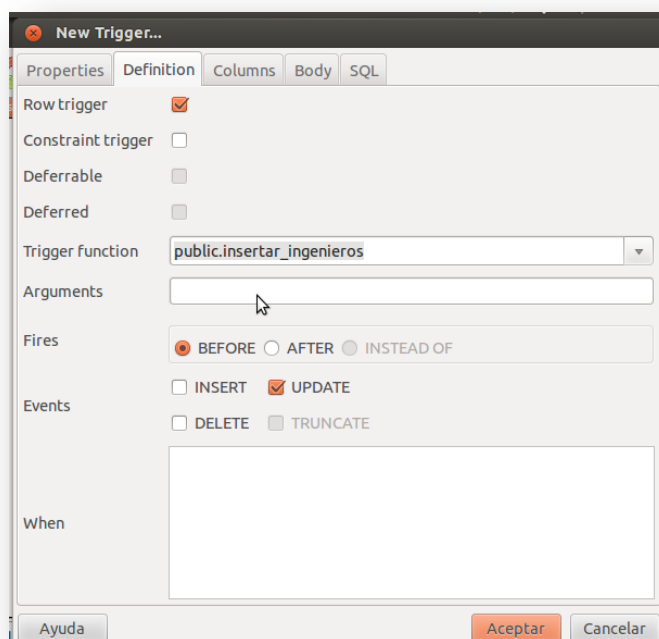


Figura 3: Interfaz visual del editor de disparadores del PgAdmin3

En esta herramienta los disparadores están asociados a tablas en específico, lo que no permite que ese disparador sea utilizado en otra tabla. Para los usuarios del PgAdmin3 el editor de disparadores les permite ejecutar subrutinas después o antes de un evento, automatizando así las tareas.

Editor de disparadores del EMS SQL Manager for PostgreSQL

En la herramienta EMS SQL Manager for PostgreSQL el editor de disparadores contiene secciones tales como:

- ✓ Trigger: recoge los datos (nombre, tipo: before o after), nombre de la función, lenguaje necesarios para crear un disparador. Además, especifica el evento (INSERT, UPDATE, DELETE y TRUNCATE), define si se va a realizar el disparador para cada fila o para la declaración completa. En la misma sección hay dos etiquetas, Función y Argumentos. En la etiqueta Función se selecciona o se crea una función disparadora y en la etiqueta Argumentos se especifican los argumentos necesarios para que la función disparadora se ejecute.

- ✓ Dependencias: muestra los objetos a los que el disparador afecta y los objetos que hacen uso de los mismos.
- ✓ Descripción: comentario que te indica la acción que realizara el disparador.
- ✓ DDL: muestra el código asociado a la creación del disparador.

En la Figura 4 se muestra como están recogidos en una sola sección todos los elementos necesarios para crear un disparador.

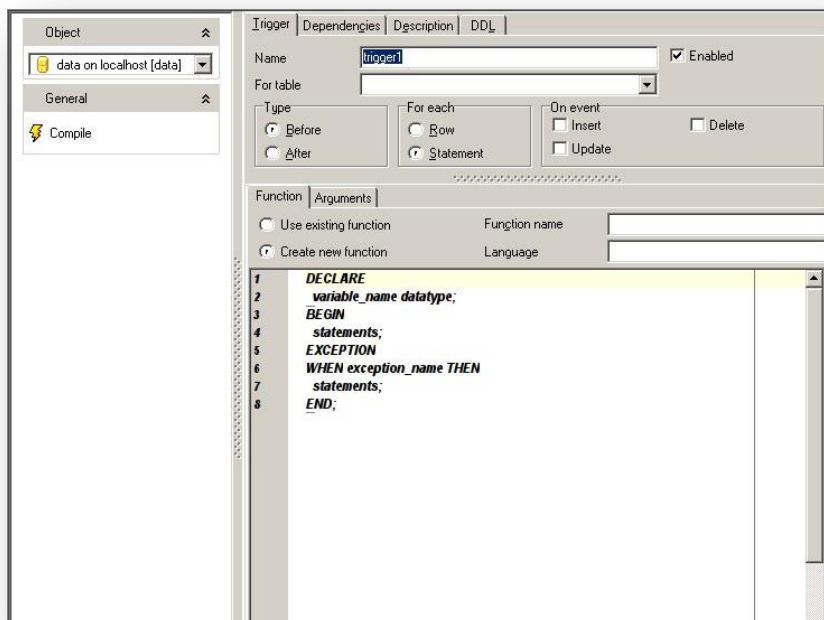


Figura 4: Editor de disparadores del SQL Manager for PostgreSQL

En EMS SQL Manager for PostgreSQL las funciones disparadoras se crean en la misma interfaz visual de las funciones, con la diferencia que hay que especificar que el tipo que devuelva sea un trigger.

Luego del estudio de los editores de las herramientas PgAdmin3 y EMS SQL Manager for PostgreSQL, el equipo de desarrollo detecta varias características, las cuales serán incluidas en el desarrollo del plugin. A continuación se muestra una tabla que especifica las características escogidas de las herramientas de administración.

Tabla 1: Características seleccionadas para el desarrollo del Plugin "Editor de funciones y disparadores"

| Editores de las herramientas | Características comunes | Características específicas |
|---------------------------------------|---|---|
| PgAdmin3 | <ul style="list-style-type: none"> - Nombrar la función. -Gestionar argumentos de la función. -Compilar el código de la función. | <ul style="list-style-type: none"> -Visualizar el identificador de objeto (OID) del disparador. |
| EMS SQL Manager for PostgreSQL | <ul style="list-style-type: none"> - Completamiento de código en la definición de la función. -Nombrar el disparador. -Definir el tipo de disparador. -Definir sobre que tabla actuará el disparador. | <ul style="list-style-type: none"> -Salvar el código de la función. -Seleccionar la tabla sobre la que actuará el disparador. -Ejecutar función. -Completamiento en los campos de componentes <i>combobox</i> para entrar valores. - Editar todos los campos de una función o de un disparador, sin crearlos nuevamente. |

A partir de los conocimientos adquiridos en el estudio y desarrollo de la investigación, se le agregan al plugin nuevas características. Las nuevas funcionalidades agregadas permiten lo siguiente:

- 1- Filtrar una función o un disparador por su nombre: busca una función en listado de funciones de forma rápida.
- 2- Validar campos de las interfaces "Nueva función" y "Nuevo disparador", del lado de la aplicación para liberar al gestor PostgreSQL de validaciones innecesarias.
- 3- Incorporar una ayuda de uso sencillo para el usuario, apoyada en imágenes del propio editor que incluye versión en formato pdf.

1.6 Metodología de desarrollo de software. Extreme Programming

Una metodología de desarrollo es un conjunto de pasos y procedimientos que deben seguirse para el desarrollo de un software. No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto.

Históricamente las metodologías tradicionales han intentado abarcar la mayor cantidad de situaciones posibles, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos cambiantes, tal es el caso del proyecto PostgreSQL. Las metodologías ágiles ofrecen una solución casi a la medida para los proyectos que tienen estas características.

La siguiente imagen (obtenida en (12)), compara las distintas metodologías ágiles con el objetivo de conocer el grado de agilidad que estas presentan.

| | CMM | ASD | Crystal | DSDM | FDD | LD | Scrum | XP |
|----------------------------------|-----|-----|---------|------|-----|-----|-------|-----|
| Sistema como algo cambiante | 1 | 5 | 4 | 3 | 3 | 4 | 5 | 5 |
| Colaboración | 2 | 5 | 5 | 4 | 4 | 4 | 5 | 5 |
| Características Metodología (CM) | | | | | | | | |
| - Resultados | 2 | 5 | 5 | 4 | 4 | 4 | 5 | 5 |
| - Simplicidad | 1 | 4 | 4 | 3 | 5 | 3 | 5 | 5 |
| - Adaptabilidad | 2 | 5 | 5 | 3 | 3 | 4 | 4 | 3 |
| - Excelencia técnica | 4 | 3 | 3 | 4 | 4 | 4 | 3 | 4 |
| - Prácticas de colaboración | 2 | 5 | 5 | 4 | 3 | 3 | 4 | 5 |
| Media CM | 2.2 | 4.4 | 4.4 | 3.6 | 3.8 | 3.6 | 4.2 | 4.4 |
| Media Total | 1.7 | 4.8 | 4.5 | 3.6 | 3.6 | 3.9 | 4.7 | 4.8 |

Figura 5: Ranking de agilidad

Las metodologías con mayor grado de agilidad son Adaptive Software Development (ASD) y Extreme Programming (XP). Analizando las características y puntuaciones de la tabla, se puede decir que XP tiene mayor simplicidad y excelencia técnica que la metodología ASD.

La metodología XP fue desarrollada por Kent Beck, él mismo dijo:

“Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar.” (13)

Esta metodología se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. XP se plantea como una metodología para emplear en proyectos de riesgos, se dice además que debido a sus características aumenta la productividad. (14)

La metodología de desarrollo XP promueve algunos valores fundamentales como: comunicación, coraje, simplicidad y continuo seguimiento (*feedback*). Proporciona además algunas ventajas como: se pueden obtener productos fáciles de usar, con rapidez, fiables y robustos contra los fallos. Las continuas versiones que se ofrecen responden a las necesidades del usuario con mayor exactitud, reduciendo así el número de errores. En esta metodología la propiedad del código es colectiva, cualquier persona del grupo de trabajo puede desarrollar, diseñar, simplificar, así como realizar cualquier necesidad del proyecto.¹

1.7 Tecnologías y herramientas a utilizar

No cabe duda que en el entorno competitivo actual, el empleo de herramientas informáticas aumenta la productividad y competitividad de las empresas que la utilizan. En este sentido, la incorporación de nuevas tecnologías a los procesos de producción o prestación constituye un desafío, pero a esta altura de los acontecimientos, también es una necesidad. En el caso de las empresas de software y servicios informáticos, el uso de estas tecnologías está inscripto en su propia naturaleza, dado que es una condición para el desenvolvimiento de sus actividades diarias.

¹ Se puede encontrar más información en el documento de la arquitectura del Proyecto PostgreSQL del centro DATEC.

Para el desarrollo de plugin “Editor funciones y disparadores” es necesario realizar un análisis de las herramientas y tecnologías a utilizar, por lo que es imprescindible el estudio de las mismas.

Lenguaje de modelado UML

Lenguaje Unificado de Modelado (LUM o UML), por sus siglas en inglés, Unified Modeling Language. Permite visualizar, especificar, construir y documentar un software. Se aplica en el desarrollo de software para dar soporte a una metodología de desarrollo del mismo, sin especificar que metodología usar. Como lenguaje, es usado para la comunicación y un mejor entendimiento entre el diseñador y el programador. (15) UML consiste en fragmentar los proyectos en diagramas que van a representar diferentes vistas del proyecto.

Herramienta para el modelado.

En la actualidad muchas empresas utilizan las herramientas Computer-Aided Software Engineering (CASE) con el objetivo de automatizar los aspectos claves del proceso de desarrollo de un sistema. Algunas de estas herramientas son muy costosas y requieren además de costos de entrenamiento personal.

✓ *Visual Paradigm para UML 8.0*

Visual Paradigm es una herramienta CASE multiplataforma de modelado UML, la cual soporta el ciclo de vida completo del desarrollo de software. Permite dibujar todos los tipos de diagramas de clases, genera código desde diagramas, código inverso e informes en formato PDF, Word o HTML. Esta herramienta presenta licencia gratuita y comercial, además de ser fácil de instalar y actualizar.

La herramienta ofrece un entorno de creación de diagramas para UML, hace uso de un lenguaje estándar para facilitar la comunicación entre el equipo de desarrollo, el modelo y el código; (16) permanece sincronizado en todo el ciclo de desarrollo. Cuenta con múltiples versiones para cada necesidad y disponibilidad de integrarse, en los principales Entorno de Desarrollo Integrado por sus siglas en inglés Integrated Development Environment (IDE).

Lenguaje C++

Un lenguaje de programación no es más que un conjunto de instrucciones, operadores y reglas de sintaxis que permite crear programas capaces de comunicarse con los dispositivos hardware y software existentes.

Uno de estos es C++, este es un lenguaje imperativo orientado a objetos derivado de C, que nació para agregarle nuevas características de las que carecía. Constituye además, un lenguaje procedural orientado a algoritmo. Algunas de las características más importantes de este lenguaje son:

- ✓ Con el correcto y debido uso de los punteros, C++ se convierte en un lenguaje potente, seguro y rápido.
- ✓ El consumo de memoria de los programas hechos en este lenguaje es moderado si se compara con otros.
- ✓ Uso extensivo de llamadas a funciones.
- ✓ Comandos breves (poco tecleo).
- ✓ Lenguaje estructurado.
- ✓ Programación de bajo nivel (nivel bit).
- ✓ El uso de constructores de alto nivel.
- ✓ El poder manejar actividades de bajo-nivel.
- ✓ Genera programas eficientes.
- ✓ La posibilidad de poder ser compilado en una variedad de computadoras, con pocos cambios (portabilidad).

Framework de desarrollo. Qt 4.0

Un framework es un conjunto de bibliotecas orientadas a la reutilización a gran escala de componentes de software para el desarrollo rápido de aplicaciones. La utilización de los frameworks tiene ventajas como: el desarrollo rápido de aplicaciones, la reutilización de componentes de software y programación de componentes que siguen una política de diseño uniforme.

El framework Qt es una biblioteca libre, de código abierto y multiplataforma. Permite desarrollar interfaces gráficas y programas sin interfaz gráfica. Utiliza como lenguaje de programación C++ de forma nativa, pero ofrece soporte para otros lenguajes como Python mediante PyQt, Java mediante QtJambi, o C# mediante Qyoto. (17) La Interfaz de Programación de Aplicaciones (API) de esta librería permite acceder a bases de datos mediante SQL. Además desarrolla la reutilización de código, haciendo más fácil el trabajo del programador. Qt está distribuido bajo los términos de Lesser General Public License (LGPL), en español Licencia Pública General Menor de GNU.

Entorno de desarrollo integrado. Qt Creator 2.0.1

Un entorno de desarrollo integrado o Integrated Development Environment (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica.

El entorno de desarrollo integrado, QT Creator es un IDE multiplataforma para desarrollar aplicaciones en C++ de manera sencilla y rápida. Está basado en la librería Qt y cuenta con las siguientes características: editor avanzado en C++, diseñador de formularios (GUI) integrado, herramientas para la administración, construcción de proyectos y soporta los lenguajes: C#/.NET Languages, Python, PyQt y PySide, Ada, Pascal, Perl, PHP y Ruby. Posee un depurador visual, una ayuda sensible al contexto integrada, resaltado y auto-completado de código. (17) Este IDE está distribuido bajo tres tipos de licencias: Qt Commercial Developer License, Qt GNU LGPL v. 2.1 y Qt GNU GPL v. 3.0.

Controlador de versiones. Subversion 6.0

Un sistema de control de versiones es una tecnología para seguir y controlar los cambios realizados en ficheros del proyecto, como el código de fuente y la documentación. Da paso a la comunicación entre los desarrolladores, un manejo de los lanzamientos, la administración de los fallos, la estabilidad entre el código y los esfuerzos de desarrollo experimental y autorización en los cambios de los desarrolladores. (18)

Subversion es un sistema controlador de versiones, comparte información y constituye un repositorio en forma de árbol con una jerarquía de directorios y archivos. Un repositorio es un

depósito en un sitio centralizado donde se almacena y mantiene información digital. Subversion guarda cada cambio que se realice en el repositorio. Recuerda cambios realizados a cada archivo así como cambios en el árbol de directorios: archivos y directorios nuevos, borrados, modificados o cambiados de lugar. Generalmente un cliente lee la versión más actual. Subversion brinda la posibilidad de leer estados anteriores del sistema de archivos, además de permitir conocer los cambios realizados, cuándo se realizaron y quién los realizó.

Conclusiones parciales

Luego de estudiar las herramientas PgAdmin3 y SQL Manager for PostgreSQL se escogen algunas características de ambas herramientas y se agregan otras escogidas por el equipo de desarrollo según los conocimientos adquiridos en el transcurso de la investigación. De esta forma el “Editor de funciones y disparadores” para la herramienta de administración HADB quedará confeccionado lo más funcional e intuitivo posible. Para el desarrollo se define como lenguaje de modelado UML y el lenguaje de desarrollo C++. Además se utiliza el framework QT, el IDE QT Creator, como herramienta de modelado Visual Paradigm para UML y como controlador de versiones Subversion.

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”

Introducción

Se describen las características principales que posee el sistema. Se realiza el modelo de dominio, se confeccionan las Historias de Usuarios que serán implementadas y las Tareas de la Ingeniería asociadas a cada Historias de Usuario. Para el diseño del sistema se elabora el diagrama de clases y las tarjetas CRC. De esta forma se tendrá un entendimiento más amplio de la propuesta del plugin.

2.1 Identificación del problema

En el centro DATEC, específicamente en el proyecto PostgreSQL, se desarrolló una herramienta de administración llamada HABD, con una arquitectura basada en plugins. La herramienta carece de la funcionalidad capaz de gestionar funciones y disparadores.

El resultado de esta investigación será un plugin que permita gestionar funciones y disparadores. Estas funcionalidades son básicas para la administración de una base de datos. Estas permiten la programación de rutinas del lado del servidor, es decir, la definición de operadores propios, incorporar lógica del negocio dentro de la base de datos y realizar recuperaciones de datos o evitar la pérdida de información a manos del usuario.

2.2 Modelo de dominio

El modelo de dominio es una representación visual estática del entorno real del proyecto, se realiza con el objetivo de lograr un mejor entendimiento para la realización del sistema. Este ayuda a comprender los conceptos con los que trabajan los usuarios. En el modelo se muestra como el usuario interactúa con el Front-End configurando las opciones de apariencias que no son más que la forma en que se muestra la herramienta; al Front-End se le incorpora un plugin que es el encargado de gestionar funciones y disparadores.

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”

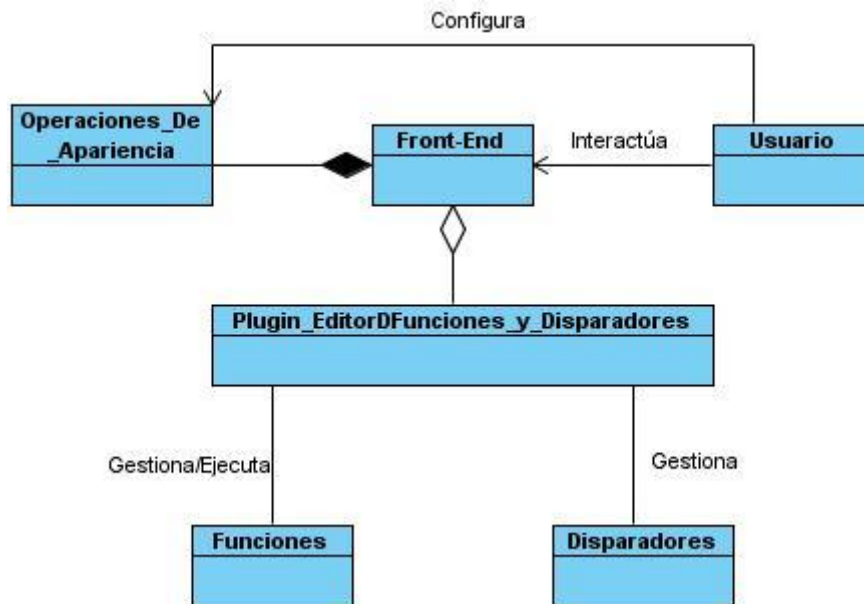


Figura 6: Modelo de dominio

Usuario: Persona que interactúa con el Front-End.

Front-End: Herramienta a la cual se le incorporan los plugins para su funcionamiento como herramienta de administración de base de datos.

Operaciones_De_Apariencia: Opciones que le permiten al usuario realizar cambios en la apariencia del Front-End.

Plugin_EditorDFunciones_y_Disparadores: Plugin o módulo que le permitirá al Front-End la gestión de funciones y disparadores.

Funciones: Módulo que permite crear, editar, eliminar, mostrar, filtrar y ejecutar funciones.

Disparadores: Módulo que permite crear, editar, eliminar, mostrar y filtrar disparadores.

2.3 Propuesta del Sistema

El editor de funciones brinda la posibilidad de crear una nueva función, listando posibles lenguajes a seleccionar para dicha función. Permite además el completamiento de código, facilitando el trabajo a los usuarios. Edita las funciones existentes cargando la información de las

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”

funciones hacia el editor. El Editor de disparadores permite crear un nuevo disparador listando las posibles funciones disparadoras a seleccionar, además edita el disparador existente.

La imagen que se muestra a continuación representa el mapa conceptual de la propuesta del sistema. Este es donde se muestra como el SGBD PostgreSQL está estrechamente relacionado con la herramienta de administración HABL, a la cual se le incorporarán varios plugins como el Editor de funciones y disparadores; este será el encargado de crear, editar, visualizar y eliminar tanto las funciones como a los disparadores.

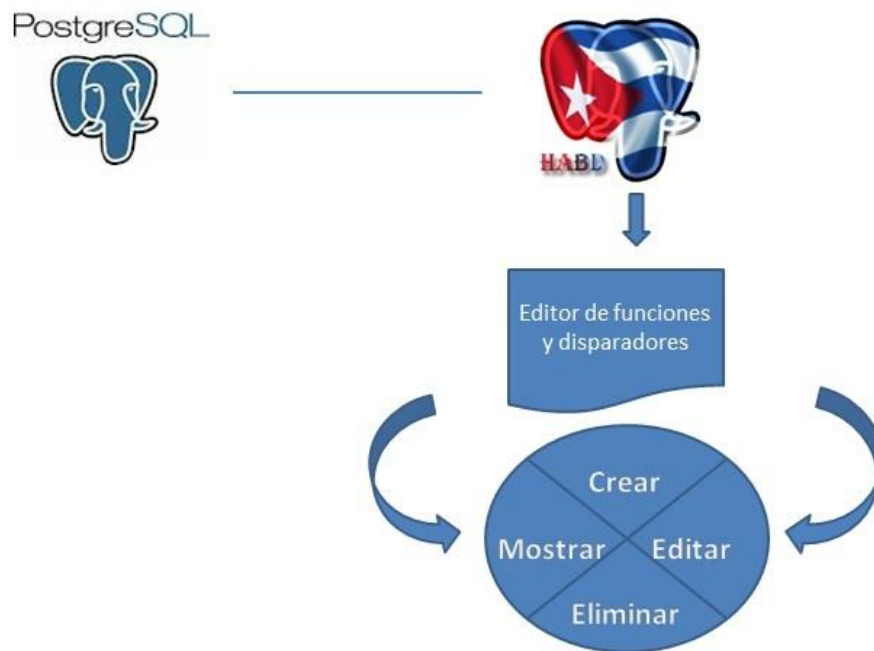


Figura 7: Mapa conceptual. Propuesta del sistema

2.4 Historias de Usuarios

Las Historias de Usuarios son la especificación de los requisitos funcionales, permitiendo la descripción e implementación. Unas de las características principales es que deben ser escritas por los clientes y responden rápidamente a los requisitos cambiantes. En la fase de prueba son fundamentales ya que se utilizan para verificar si el programa cumple con lo que el usuario desea.

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”

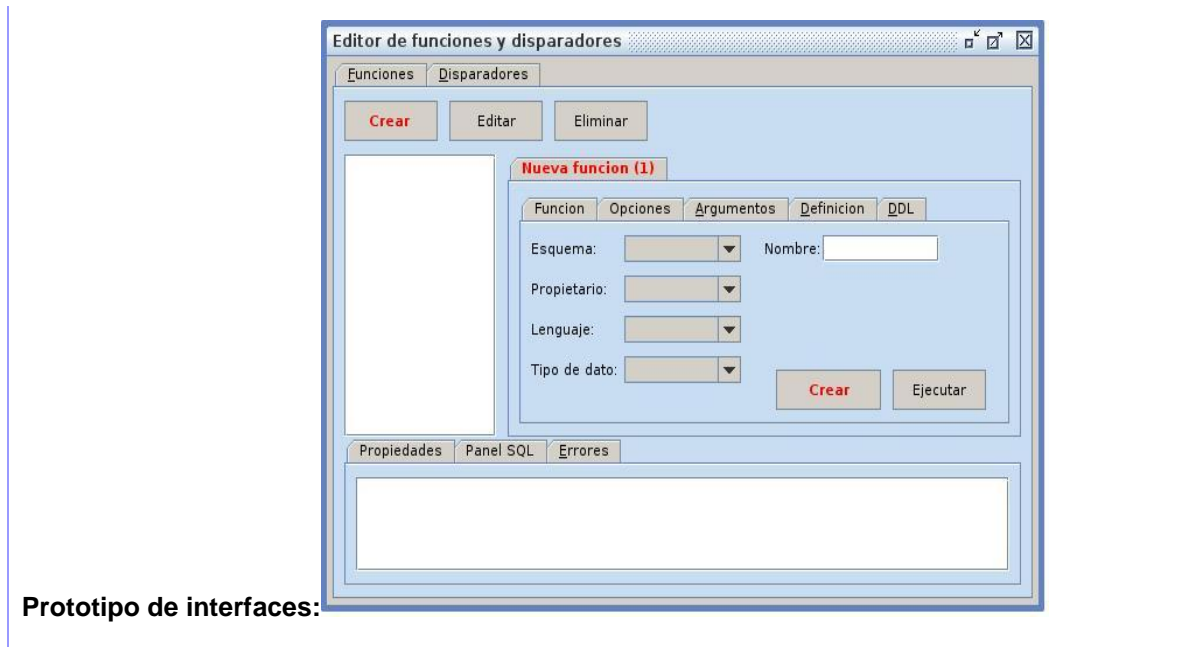
Para implementar una Historia de Usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia. El tiempo de desarrollo ideal para una Historia de Usuario es de 1 a 3 semanas. (19)

Se confeccionaron 17 Historias de Usuario, 3 con prioridad Muy alta, 7 Alta, 5 Media y 2 Baja. A continuación se muestra un ejemplo (Ver más en [Anexos](#)):

Tabla 2: Ejemplo de Historia de Usuario

| Historia de Usuario | |
|--|--|
| Número: 1 | Nombre de la Historia de Usuario: Crear función |
| Cantidad de modificaciones a la Historia de Usuario: Ninguna | |
| Usuario: Daniel Alejandro Linares Brooks | Iteración asignada: 1 |
| Prioridad en negocio: Muy Alta | Puntos estimados: 2 semanas |
| Riesgo en desarrollo: Alto | Puntos reales: 2 semanas |
| Descripción: Crea la función en la base de datos seleccionando el nombre de esquema, el nombre de la función, seleccionando el tipo de dato que devuelve y el lenguaje y escribiendo el cuerpo (campos obligatorios). También se puede crear añadiendo argumentos y seleccionando otras opciones (campos opcionales). | |
| Observaciones: Validar que el usuario entre los campos obligatorios. La función creada debe ser mostrada en la lista de funciones creadas. Mostrar mensaje de función creada. | |

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”



2.5 Lista de reserva del producto

La Lista de Reserva del Producto (LRP) es un artefacto donde se mencionan todas las funcionalidades que el sistema debe ser capaz de hacer; especificándoles a cada una, quién estima esa funcionalidad y la estimación en semana de la misma. Las funcionalidades se dividen según su prioridad, pueden ser muy altas, altas, medias y bajas; en las bajas se puntualizan además las propiedades o características requeridas por el sistema. A continuación se muestran las funcionalidades correspondientes a esta investigación.

Tabla 3: Lista de reserva del producto

| Ítem * | Descripción | Estimación | Estimado por |
|----------------------------|----------------------------------|-------------|--------------|
| Prioridad: Muy Alta | | | |
| 1 | Obtener lista de esquemas. | 0.4 semanas | Analista |
| 2 | Obtener lista de lenguajes. | 0.4 semanas | Analista |
| 3 | Obtener lista de propietarios. | 0.4 semanas | Analista |
| 4 | Obtener lista de tipos de datos. | 0.4 semanas | Analista |
| 5 | Validar campos de la función. | 0.4 semanas | Analista |

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”

| | | | |
|------------------------|---|-------------|----------|
| 6 | Actualizar tabla. | 1 semanas | Analista |
| 7 | Capturar argumentos. | 1 semanas | Analista |
| 8 | Obtener lista de tablas. | 2 semanas | Analista |
| Prioridad: Alta | | | |
| 9 | Obtener función a editar. | 0.5 semanas | Analista |
| 10 | Mostrar datos a editar de la función. | 0.5 semanas | Analista |
| 11 | Guardar cambios de la función editada. | 0.5 semanas | Analista |
| 12 | Obtener función a eliminar. | 0.5 semanas | Analista |
| 13 | Permitir confirmación para eliminar. | 0.5 semanas | Analista |
| 14 | Visualizar funciones. | 0.3 semanas | Analista |
| 15 | Actualizar lista de funciones al crearlas. | 0.3 semanas | Analista |
| 16 | Actualizar lista de funciones al eliminarlas. | 0.3 semanas | Analista |
| 17 | Crear argumentos. | 0.3 semanas | Analista |
| 18 | Editar argumentos. | 0.3 semanas | Analista |
| 19 | Eliminar argumentos. | 0.3 semanas | Analista |
| 20 | Mostrar argumentos. | 0.3 semanas | Analista |
| 21 | Validar argumentos. | 0.3 semanas | Analista |
| 22 | Concatenar argumentos. | 0.3 semanas | Analista |
| 23 | Obtener disparador a editar. | 0.5 semanas | Analista |
| 24 | Mostrar datos a editar del disparador | 0.5 semanas | Analista |
| 25 | Guardar cambios del disparador editado. | 0.5 semanas | Analista |
| 26 | Obtener disparador a eliminar. | 0.5 semanas | Analista |
| 27 | Permitir confirmación para eliminar el disparador. | 0.5 semanas | Analista |
| 28 | Visualizar disparadores. | 0.6 semanas | Analista |
| 29 | Actualizar lista de disparadores después de creadas. | 0.6 semanas | Analista |
| 30 | Actualizar lista de disparadores después de eliminadas. | 0.6 semanas | Analista |

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”

| Prioridad: Media | | | |
|-------------------------|--|--------------|----------|
| 31 | Mostrar propiedades de la función. | 0.5 semanas | Analista |
| 32 | Mostrar errores de la función. | 0.5 semanas | Analista |
| 33 | Mostrar DDL de la función. | 0.5 semanas | Analista |
| 34 | Abrir directorio de archivos. | 0.25semanas | Analista |
| 35 | Guardar fichero. | 0.25 semanas | Analista |
| 36 | Obtener palabras reservadas. | 0.3 semanas | Analista |
| 37 | Guardar palabras reservadas en un fichero. | 0.3 semanas | Analista |
| 38 | Mostrar palabras reservadas mientras se escribe. | 0.3 semanas | Analista |
| 39 | Mostrar propiedades del disparador. | 0.3 semanas | Analista |
| 40 | Mostrar errores del disparador. | 0.3 semanas | Analista |
| 41 | Mostrar DDL del disparador. | 0.3 semanas | Analista |
| 42 | Abrir directorios de archivo para el disparador. | 0.25 semanas | Analista |
| 43 | Guardar fichero en dirección de archivo para el disparador. | 0.25 semanas | Analista |
| Prioridad: Baja | | | |
| 44 | Buscar nombre de la función. | 0.5 semanas | Analista |
| 45 | Mostrar la función buscada. | 0.5 semanas | Analista |
| 46 | Buscar nombre del disparador. | 0.5 semanas | Analista |
| 47 | Mostrar el disparador buscado. | 0.5 semanas | Analista |
| | <p>Usabilidad.</p> <p>El plugin puede ser utilizados por todas aquellas personas que tengan un conocimiento básico sobre bases de datos.</p> <p>Software.</p> <p>Para utilizar el plugin deberá estar instalada la herramienta de administración HABD, el gestor</p> | | Analista |

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”

| | |
|---|----------|
| PostgreSQL a partir la versión 9.1.0. | |
| Hardware. El ordenador donde se utilice el plugin integrado a la herramienta HADB debe contar con 100 Mb de memoria RAM como mínimo, un microprocesador de 300MHz de frecuencia y un espacio en disco duro de al menos 60 Gb. | Analista |
| Apariencia o interfaz externa. El plugin poseerá una interfaz amigable e intuitiva de tal forma que el usuario pueda realizar las operaciones. | Analista |
| Restricciones del diseño y la implementación. Para el diseño e implementación de la aplicación se utiliza Visual Paradigm 3.4 para el modelado, lenguaje de programación C++ y como IDE de desarrollo Qt Creator. | Analista |

2.6 Tareas de la ingeniería

Las Tareas de Ingeniería son las funcionalidades y características que se deben cumplir. Las mismas pertenecen a una HU en específico. Tiene gran importancia ya que describe con profundidad lo que se realiza en la aplicación.

Se identificaron 47 Tareas de Ingeniería, a continuación se muestra un ejemplo. Para ver las demás dirigirse a los [Anexos](#).

Capítulo 2: Características del Plugin “Editor de funciones y disparadores”

Tabla 4: Ejemplo de Tareas de Ingeniería

| Tarea de Ingeniería | |
|---|--------------------------------------|
| Número Tarea: 2 | Número Historia de Usuario: 1 |
| Nombre Tarea: Obtener lista de lenguajes | |
| Tipo de Tarea : Desarrollo | Puntos Estimados: 0.4 semanas |
| Fecha Inicio: 16-11-2011 | Fecha Fin: 17-11-2011 |
| Programador Responsable: Daniel Alejandro Linares Brooks | |
| Descripción: Llena un combo box widget con los lenguajes de la base de datos | |

2.7 Plan de iteraciones

Cuando en un proyecto se trabaja bajo la metodología XP, este para que tenga éxito debe realizarse mediante iteraciones, llevándose a cabo la ejecución de las Historias de Usuarios correspondientes a cada iteración.

Iteración 1: En esta iteración se desarrollarán las Historias de Usuarios 1, 2, 3, 4, 5, 6, 7, 8, 9 y 10 correspondientes al desarrollo del Editor de funciones.

Iteración 2: En la 2da iteración se realizarán las Historias de Usuarios 11, 12, 13, 14, 15, 16 y 17 correspondientes al desarrollo del Editor de disparadores.

Se decide realizar las Historias de Usuarios correspondientes a la gestión de funciones en la 1era iteración, debido a que los disparadores usan funciones para su creación.

2.8 Diseño del sistema

El diseño del sistema en la metodología XP se realiza a través de las tarjetas CRC (Clase, Responsabilidad y Colaboración). Las tarjetas CRC permiten especificar la descripción detallada de cada una de las clases con sus colaboraciones. En este caso se utiliza además el diagrama de clases, para una mejor comunicación, en este se muestra la información de mayor importancia.

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

Diagrama de clases

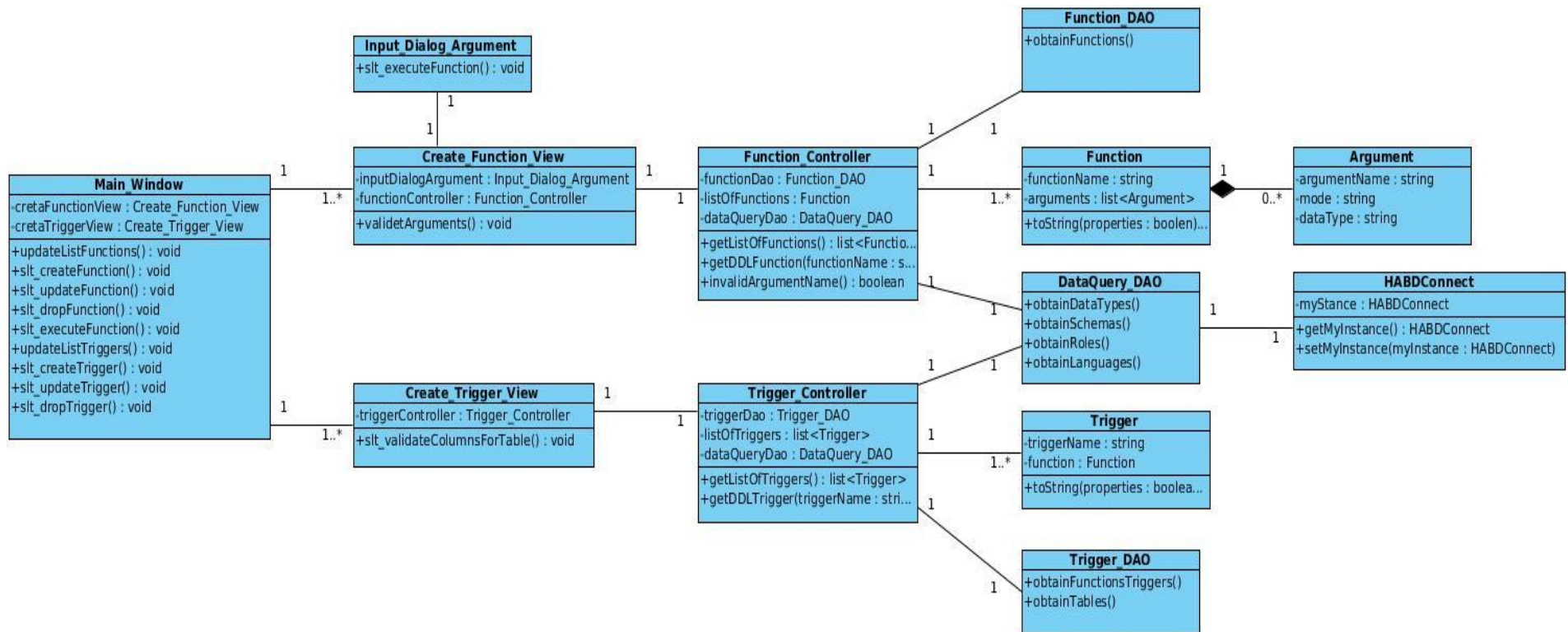


Figura 8: Diagrama de clases del sistema

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

Tarjetas CRC

Tabla 5: Tarjeta CRC. Funciones

| Tarjeta CRC | |
|--|----------------|
| Clase: Funciones | |
| Responsabilidades | Colaboraciones |
| -Crear función. -Editar función. -Eliminar función. -Mostrar propiedades de la función. -Filtrar nombre de la función. -Ejecutar función. | -Argumentos |

Tabla 6: Tarjeta CRC. Disparadores

| Tarjeta CRC | |
|---|----------------|
| Clase: Disparadores | |
| Responsabilidades | Colaboraciones |
| -Crear disparador. -Editar disparador. -Eliminar disparador. -Mostrar propiedades del disparador. -Filtrar nombre del disparador. | -Funciones |

Tabla 7: Tarjeta CRC. Argumentos

| Tarjeta CRC | |
|--|----------------|
| Clase: Argumentos | |
| Responsabilidades | Colaboraciones |
| -Crear disparador. -Editar disparador. -Eliminar disparador. | |

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

- | | |
|--------------------------------------|--|
| -Mostrar propiedades del disparador. | |
| -Filtrar nombre del disparador. | |

2.9 Patrón de arquitectura

El patrón Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que divide a una aplicación interactiva en tres componentes: Modelo, Vistas y Controladores.

Modelo: contiene los datos y la funcionalidad esencial. Encapsula el núcleo funcional y los datos involucrados. En cuanto a su arquitectura, exporta procedimientos que realizan procesamiento específico de la aplicación; provee funciones para que las vistas accedan a la información.

Vistas: despliegan la información al usuario y presentan esta información por pantalla. En cuanto a su arquitectura, tienen procedimientos de actualización para recibir nuevos datos.

Controlador: define la forma en la que debe reaccionar la interfaz del usuario, frente a la entrada de datos del usuario. En cuanto a su arquitectura, existe un controlador por cada vista y recibe las entradas de las vistas como eventos y los interpreta.

Este patrón permite que los sistemas sean flexibles y adaptables. Contiene múltiples vistas para el mismo modelo, vistas sincronizadas y vistas intercambiables. Con la utilización de este patrón es mucho más sencillo agregar múltiples representaciones de datos o información; facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de las otras capas; facilita además el mantenimiento en caso de errores y ofrece maneras sencillas para probar el correcto funcionamiento del sistema. (20)

Emplear esta arquitectura en la implementación del “Editor de funciones y disparadores” tiene como ventajas que ofrece a los desarrolladores una mayor flexibilidad para personalizar la presentación de las vistas a partir de la separación de la funcionalidad introducida por esta arquitectura, además proporciona una interfaz de modelo estándar, que permite utilizar una amplia gama de fuentes de datos para ser utilizada con las vistas de elementos existentes. A continuación se muestra como se evidencia este patrón en el plugin Editor de funciones y disparadores.

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

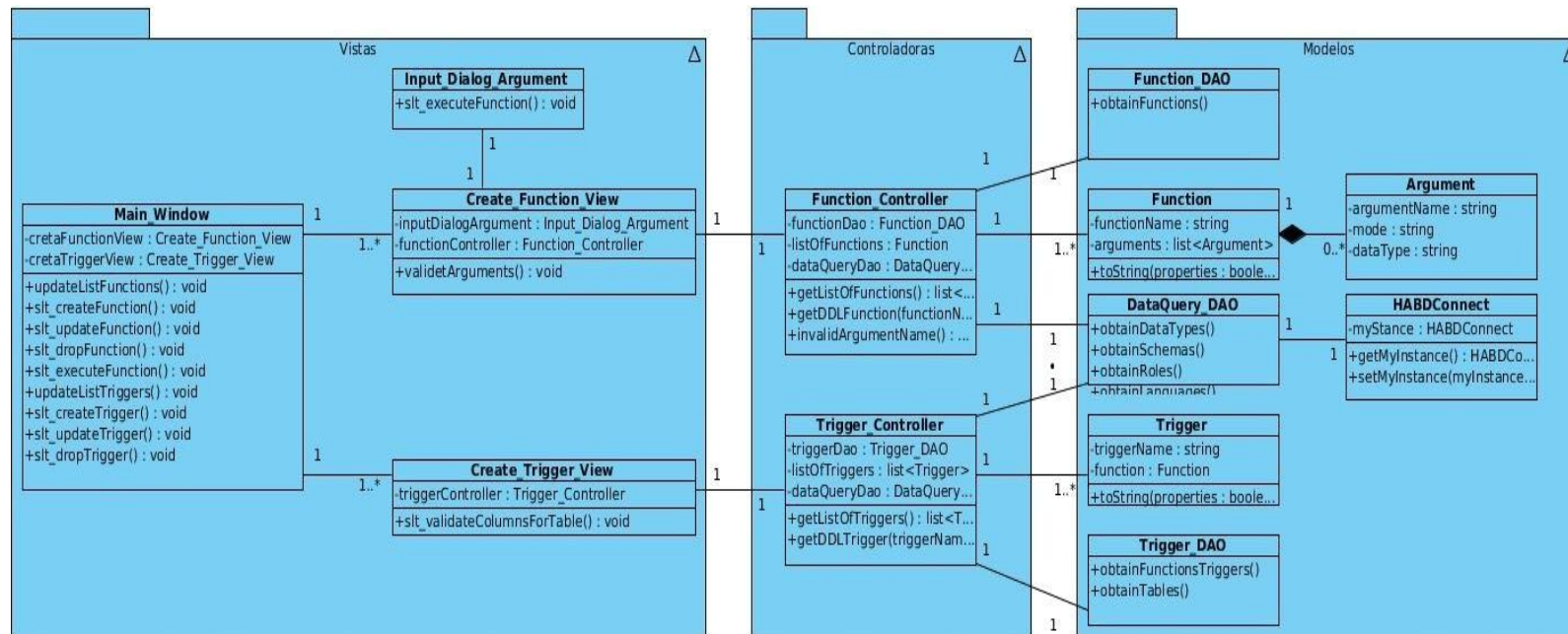


Figura 9: Patrón MVC

Las clases agrupadas en la capa **Vistas**, manejan eventos, capturan y visualizan los datos de interés para el usuario. Estos datos son manipulados, validados y controlados por la lógica del negocio, es decir, por las clases agrupadas en la capa **Controladoras**; tomando la información necesaria de las clases persistentes y de las de acceso a datos, agrupadas en la capa **Modelos**.

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

2.10 Patrones de diseño

Un patrón de diseño brinda una solución, ya probada y documentada, a problemas de desarrollo de software, son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. (21)

Existen varias familias de patrones de diseño una de ellas son los GRASP, en español, Patrones Generales de Software para Asignación de Responsabilidades, por sus siglas en inglés, General Responsibility Assignment Software Patterns. Estos son utilizados para la asignación de responsabilidades. Esta familia de patrones son una serie de buenas prácticas de aplicación recomendable en el diseño de software. Ejemplos de patrones son experto, creador, bajo acoplamiento, alta cohesión y controlador. Para el desarrollo de esta aplicación se utilizan los siguientes patrones de diseño pertenecientes a la familia GRASP:

Experto: Es donde la clase contiene toda la información necesaria para realizar la acción encomendada y la responsabilidad de realizar una operación es de la clase que tiene los datos involucrados. La siguiente imagen muestra un ejemplo donde se evidencia el patrón Experto:

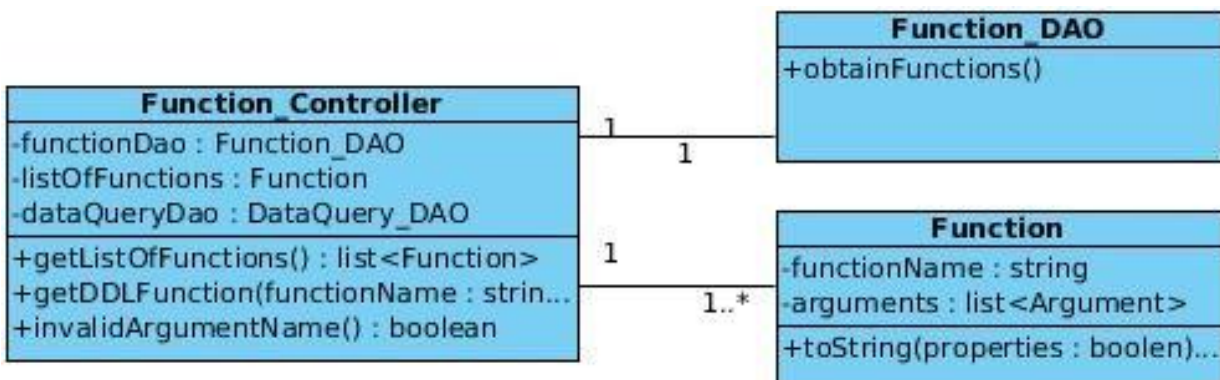


Figura 10: Ejemplo del patrón Experto

La clase **Function_Controller** aplica el principio básico de asignación de responsabilidades en diseño Orientado a Objetos; indicando que la responsabilidad de obtener el DDL y listar funciones recae sobre ella, ya que conoce toda la información necesaria para hacerlo.

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

Creador: Este patrón es el que crea, en la creación de objetos es el que guía la asignación de responsabilidades. Su objetivo se enmarca en encontrar un creador que debemos conectar con el objeto producido en cualquier evento. La siguiente figura muestra un ejemplo de donde se evidencia el patrón Creador:

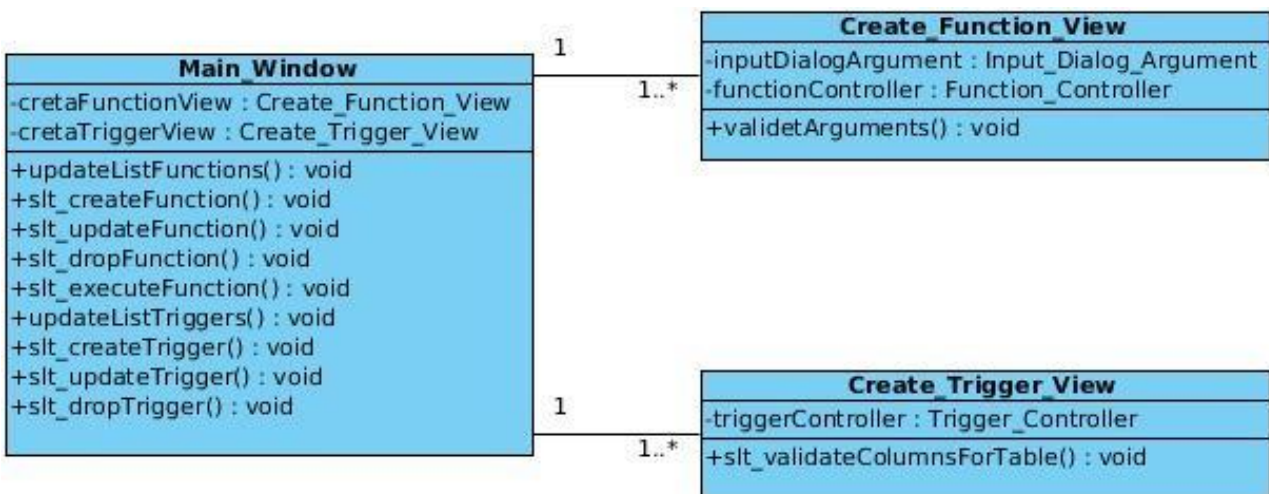


Figura 11: Ejemplo del patrón Creador

La clase **Main_Window** es la responsable de la creación de nuevos objetos de las clases **Create_Function_View** y **Create_Trigger_View**.

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

Alta cohesión: La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una clase con este patrón, compartirá la responsabilidad de una operación con otras clases. A continuación se muestra un ejemplo de este patrón:

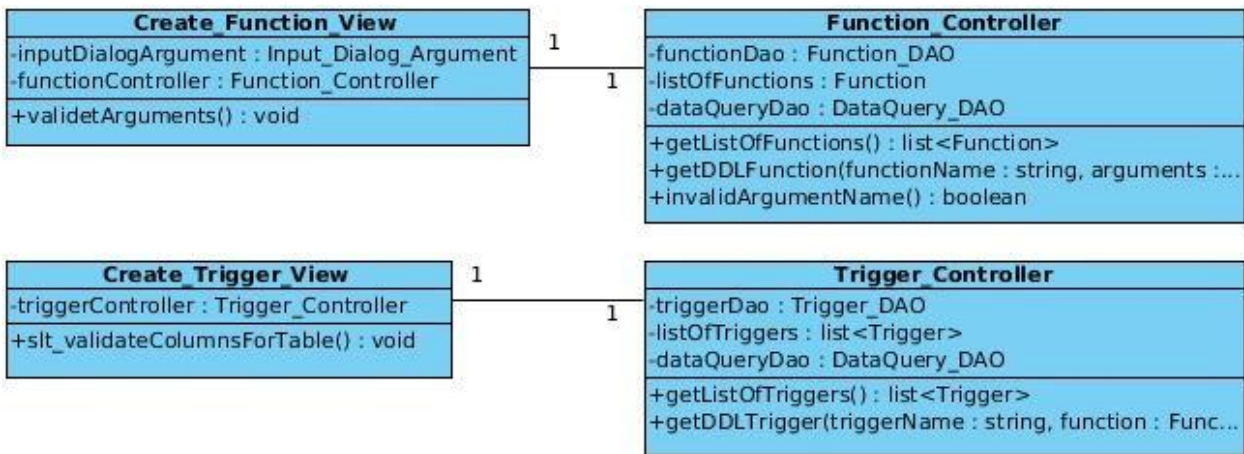


Figura 12: Ejemplo del patrón Alta cohesión

La clase **Create_Function_View** tiene alta cohesión ya que comparte la responsabilidad de validar los argumentos de una función con validaciones que realiza **Function_Controller**.

La clase **Create_Trigger_View** solo se hace responsable de manejar el evento crear un disparador dejando las responsabilidades de obtener el DDL a **Trigger_Controller**.

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

Bajo acoplamiento: Es un principio que asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas; facilitando la centralización de actividades. Otros de los beneficios es que no afectan por cambios de otros componentes, son fáciles de entender por separado y fáciles de reutilizar.

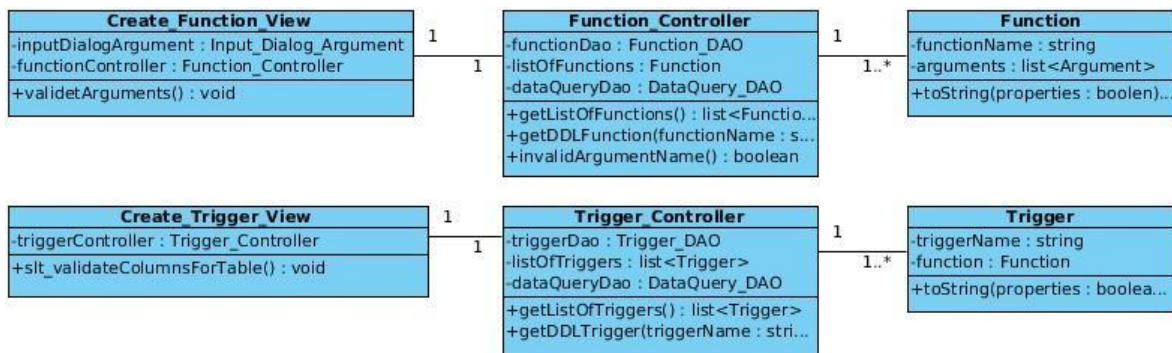


Figura 13: Ejemplo del patrón Bajo acoplamiento

La clase **Create_Function_View** al estar menos atada con **Function_Controller** y esta a su vez con **Function** se puede producir una modificación en **Create_Function_View** para cambiar la visualización de los datos de **Function** sin que esta se vea afectada ni afecte las responsabilidades de **Function_Controller** de obtener el DDL, potenciando la reutilización de código y disminuyendo la dependencia entre las clases. Los beneficios de este factor se evidencian en que no se afectan por cambios en **Create_Trigger_View** los datos persistentes en **Trigger**, siendo fáciles de entender estos cambios por separado y fáciles de reutilizar.

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

Patrones Gang of Four

Otra familia de patrones de diseño son los patrones Gang of Four (GOF). El mismo fue compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, quienes en el libro “Design Patterns” describen 23 patrones de diseño; estos patrones son comúnmente utilizados en problemas de diseño usando modelamiento UML. Los patrones pertenecientes a esta familia se agrupan en, patrones creacionales, estructurales y de comportamiento.

Dentro de los patrones creacionales, para el desarrollo del plugin se escoge el siguiente:

Instancia única (Singleton): El patrón Singleton asegura que exista una única instancia de una clase. A primera vista, se puede pensar que pueden utilizarse clases con miembros estáticos para el mismo fin, sin embargo, los resultados no son los mismos, ya que, en este caso la responsabilidad de tener una única instancia recae en el cliente de la clase.

Este patrón hace que la clase sea responsable de su única instancia, quitando así este problema a los clientes. El mismo puede tener un control estricto sobre cómo y cuándo acceden los clientes a la instancia. Permite el refinamiento de operaciones y la representación.

En esencia, la idea es que cada objeto en la aplicación use la misma instancia de Singleton, esto evita tener que pasar la instancia a todos los objetos que quieran utilizarla; este patrón hace a los objetos responsables de sí mismos. (22)

En el desarrollo de la herramienta de administración de bases de datos HADB es necesario que la aplicación establezca una conexión a la base de datos para operar con las mismas. Partiendo de la vulnerabilidad que surge, si al manipular la conexión como objeto, por parte de cada plugin, cada uno crea su propia instancia por error de alguno de los desarrolladores, creando objetos distintos de tipo conexión, cuando realmente debe ser uno para ser usados por todos los plugins, independientemente de las funciones que los mismos realicen. En la siguiente imagen se muestra donde se evidencia el patrón Singleton:

Capítulo 2: Características del plugin “Editor de funciones y disparadores”



Figura 14: Patrón Singleton

La clase **HABDConnect** asegura que sólo tiene una instancia, y proporciona un punto global de acceso a dicha instancia. De tal forma que siempre que se instancie se obtenga el mismo objeto en cualquier parte del código, evitando posibles errores que resultan de manipular, más de un objeto **HABDConnect** por contener los valores de la conexión que establece el **Fron-End** con el plugin.

2.11 Estándares de codificación

Para asegurar la calidad de un software uno de los instrumentos que se utilizan es la adopción de estándares de código. Los mismos tienen un sin número de ventajas como:

- ✓ Asegurar la legibilidad del código entre distintos programadores facilitando el traceo del mismo.
- ✓ Proveer una guía para el encargado de mantenimiento/actualización del sistema, con código claro y bien documentado.
- ✓ Facilitar la portabilidad entre plataformas y aplicaciones.

Es por ello que se definen para el presente plugin varios requisitos que a continuación serán descritos.

Identación

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

Longitud de la Línea

Evitar líneas con más de 80 caracteres. Cuando una sentencia sobrepase una línea simple del editor, esta debe ser separada en otras. La separación debe hacerse preferentemente luego de una coma o después de un operador. Realizar la ruptura después de un operador disminuye la

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

probabilidad de que al realizar un copiado del código se cometa un error por olvido de la inserción del punto y coma. La siguiente línea debe ser indentada con 5 espacios.

Comentarios

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

Declaración de variables

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

```
QMap<DependenceContainer*, bool> installedInstances; //plugins instalados
Qlist<DependenceContainer*> loadedInstancesLst; //plugins cargados
```

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

Ejemplo: `loadedInstancesLst`, `installedInstances`.

Declaración de funciones

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

```
Function ejemploPrincipal(c, d)

{

var e = c * d;

return inner(0, 1);

}
```

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

Identificadores

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

Sentencias

Sentencias Simples

Cada línea debe contener como máximo una sentencia. Debe poner un punto y coma “;” al final de cada sentencia simple. Tenga en cuenta que una sentencia de asignación puede resultar en la asignación de una función o de un objeto como literal y en todos los casos como sentencia de asignación debe estar finalizada con un punto y coma.

Sentencias Compuestas

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

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

Etiquetas

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

Sentencia return

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

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

Sentencia *if*

La sentencia *if* debe ser escrita de esta manera:

```
if (condition)
{
statements
}
```

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

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

Estructuras repetitivas

Las estructuras repetitivas deben ser escritas de esta manera:

```
for (initialization; condition; update)
{
statements
}
```

```
while (condition)
```

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

```
{
statements
}
```

foreach (valor1, valor2)

```
{
statements
}
```

Sentencia switch

La sentencia *switch* debe ser escrita de la siguiente forma:

```
switch (expression)
{
case expression:
statements
case expression:
statements
default:
statements
}
```

Espacios en blanco

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

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

Declaraciones de clases

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

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

Ejemplo incorrecto:

fichero.h

```
class A
{
}
```

```
class B
{
}
```

Ejemplo correcto:

ficheroA.h

```
class A
{
}
```

ficheroB.h

```
class B
{
}
```

2.12 Interfaces de la aplicación

La aplicación cuenta con dos interfaces fundamentales, una está asociada a la gestión de funciones y la otra a la gestión de disparadores. A continuación se muestra una breve descripción de las interfaces principales, apoyado en la imagen de dicha interfaz.

La figura 14 muestra la interfaz del Editor de funciones. Se evidencia como tiene una lista de funciones existentes, se muestran las propiedades y datos que recoge el sistema para la creación o edición de la función. Se evidencian además cuatro botones fundamentales, que son para crear, editar, eliminar y ejecutar función. Contiene una sección donde se filtrará la misma mediante el nombre. Cuenta con una ayuda donde se explica con claridad cómo funciona el editor.

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

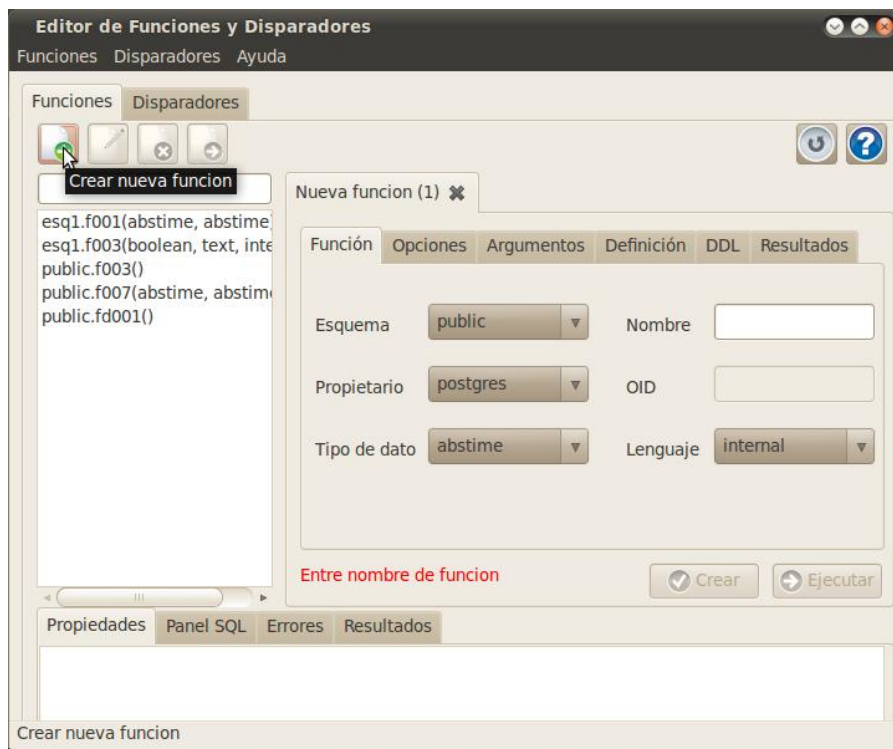


Figura 15: Interfaz visual del Plugin “Editor de funciones y disparadores” para la herramienta de administración HABD. Funciones

La figura 15 muestra la interfaz del Editor de disparadores. Se evidencia como tiene una lista de disparadores existentes, se muestran las propiedades y datos que recoge el sistema para la creación o edición del disparador. Se muestran tres botones fundamentales, que son para crear, editar y eliminar disparador, contiene una sección donde se filtrará el mismo mediante el nombre. Cuenta con una ayuda donde se explica con claridad cómo funciona el editor.

Capítulo 2: Características del plugin “Editor de funciones y disparadores”

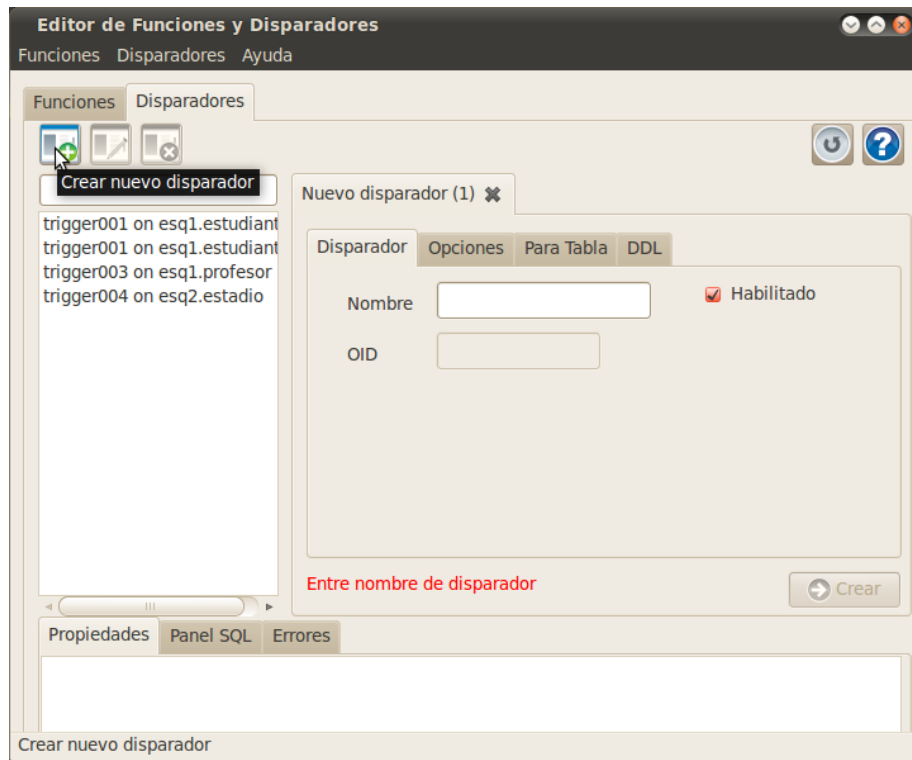


Figura 16: Interfaz visual del Plugin “Editor de funciones y disparadores” para la herramienta de administración HBD. Disparadores.

Conclusiones parciales

En este capítulo se realizó el diseño e implementación del plugin. Se identificaron 17 Historias de Usuario de ellas, tres con criticidad muy alta, siete alta, cinco medias y dos bajas. Para la elaboración del plugin se utilizó el patrón arquitectónico Modelo Vista Controlador y los patrones de diseño, Experto, Creador, Alta cohesión, Bajo acoplamiento y Singleton. Se aplica un estándar de código para un mejor entendimiento del mismo y se describen las interfaces de la aplicación.

Capítulo 3: Validación del plugin “Editor de funciones y disparadores”

Capítulo 3: Validación del Plugin “Editor de funciones y disparadores”

Introducción

Es necesario comprobar de alguna forma que el plugin cumple con los requisitos especificados en las Historias de Usuarios. Para comprobar su buen funcionamiento se escoge dentro de las pruebas que define XP, la prueba que permita lo antes planteado. El resultado de las pruebas realizadas será mostrado mediante una tabla para un mejor entendimiento.

3.1 Enfoques de diseño de pruebas

Las pruebas de software, son los procesos que permiten verificar y revelar la calidad de un producto. Son utilizadas para identificar posibles fallos de implementación, calidad, o usabilidad de un programa. Básicamente es una fase en el desarrollo de software consistente en probar las aplicaciones construidas. Las pruebas de software pueden probar la presencia de errores pero no la ausencia de ellos. En la metodología XP existen dos tipos de pruebas, las pruebas unitarias que son las que verifican el código y son diseñadas por los programadores; y las pruebas de aceptación que son las destinadas a evaluar si se consiguió la funcionalidad requerida por el cliente. (23)

Pruebas Unitarias

Las pruebas unitarias se utilizan para comprobar el funcionamiento de un fragmento de código. Se utilizan para asegurar el correcto funcionamiento de un módulo o método por separado. Estas facilitan al desarrollador cambiar el código para mejorar su estructura y permite asegurarse que los cambios no han introducido errores, por esto se dice que fomentan el cambio. Es necesario aclarar que las pruebas unitarias no sacarán a la luz todos los errores del código. Generalmente se utilizan para dar paso a las pruebas de integración y para que sean efectivas deben realizarse en conjunto con otras pruebas de software. Además que, aún después del más cuidadoso proceso de pruebas por parte del desarrollador, quedan una serie de errores que sólo aparecen cuando el cliente comienza a usarlo.

Capítulo 3: Validación del plugin “Editor de funciones y disparadores”

Pruebas de Aceptación

Las pruebas de aceptación se realizan con el objetivo de comprobar que un sistema cumple con el funcionamiento esperado, y permitir al cliente que determine su aceptación desde el punto de vista de su funcionalidad y rendimiento. Por esto las pruebas son definidas, ejecutadas y aprobadas por el cliente, aunque el equipo de desarrollo las prepare.

Estas pruebas son preparadas en base a las Historias de Usuario donde el cliente especifica el o los escenarios para comprobar que cumple con las especificaciones de la misma. El cliente es el rol fundamental en este tipo de prueba, ya que es el responsable de que los resultados de las pruebas sean correctos, en caso de que fallen algunas, es el encargado de indicar el orden de prioridad de resolución. Después de pasar todas las pruebas de aceptación, es cuando se puede decir que las historias de usuarios están terminadas.

Método seleccionado

Luego de un análisis de estos dos tipos de pruebas se decide realizar las pruebas de aceptación ya que son basadas en Historias de Usuarios. Con esta se comprueba si realmente la implementación de la Historia de Usuario es satisfactoria y si el sistema cumple con el funcionamiento esperado por el cliente.

3.2 Pruebas de aceptación

Se emplean dos técnicas para las pruebas de aceptación, la prueba alfa y la prueba beta. La prueba alfa se lleva a cabo por el cliente en el lugar de desarrollo; donde la aplicación se usa de forma natural con el desarrollador de espectador, es decir, se lleva a cabo en un entorno controlado. Se debe crear un ambiente con las mismas condiciones que se encontrará el sistema en las instalaciones del usuario. Luego se realizan las pruebas y se documentan los resultados. La prueba beta la realiza el usuario final en el lugar de trabajo de los primeros clientes. Esta se aplica en un entorno no controlado por el desarrollador. El cliente registra todos los errores encontrados y los informa. Como resultado de estos problemas durante la prueba beta, se realizan modificaciones, preparando así una versión de la aplicación. (24)

Capítulo 3: Validación del plugin “Editor de funciones y disparadores”

La técnica utilizada es la prueba alfa. La presencia del desarrollador en la realización de la pruebas, es necesaria. El objetivo del desarrollador en la aplicación de estas es recoger todos los problemas encontrados de forma entendible para él, no siendo así si los errores son elaborados por los clientes.

Para llevar a cabo la prueba, antes deben ser diseñados los casos de pruebas.

Casos de pruebas

Los casos de pruebas son un conjunto de condiciones o variables bajo las cuales se comprueba que la Historia de Usuario de la aplicación es parcial o completamente satisfactorio. El ejemplo mostrado corresponde a la Historia de Usuario, Exportar función. El caso de prueba realizado está compuesto por una sección, en este el nombre de la sección coincide con el nombre de la Historia de Usuario; una sección se compone por uno o varios escenarios, en dependencia de las posibles situaciones que se presenten. Por cada escenario se debe especificar la descripción, las variables usadas, la respuesta del sistema al entrar las variables especificadas y el flujo central para proceder a la acción que especifica el nombre del escenario.

Tabla 8: Ejemplo del caso de pruebas aplicado a la Historia de Usuario “Exportar función”

SC Exportar función

| Escenario | Descripción | Variable 1 | Variable 2 | Respuesta del sistema | Flujo central |
|--|--|------------------------------|-----------------------------------|---|--|
| EC 1.1 Exportar función correctamente | Permite exportar el DDL de la función, ya sea creándola o editándola | V(C:\Users\) | V(dd_funcion.txt) | El sistema guarda el DDL de la función en un archivo nombrado por el usuario en una dirección especificada. | 1-El usuario da clic en el botón Crear función o selecciona una función del listado y da clic en el botón Editar función. 2-Va a la pestaña DDL. 3- Da click en el botón Exportar. 4-Selecciona dirección donde se guardará el archivo. 5-Escribe nombre de archivo. 6-Da clic en botón Guardar. |
| EC 1.2 Exportar función con | No exporta el DDL de la función | I() I() V(C:\Users\) | I() V(dd_funcion.txt) I() | El sistema no guarda el DDL de la función. No | 1-El usuario da clic en el botón Crear función o selecciona una |

Capítulo 3: Validación del plugin “Editor de funciones y disparadores”

| | | | | | |
|--|--|---------------------------------------|---|---|--|
| campos vacíos | | | | cierra el cuadro de diálogo donde se especifica la dirección y el nombre archivo hasta entrarlos correctamente o cancelar la operación. | función del listado y da clic en el botón Editar función. 2-Va a la pestaña DDL. 3-Da clic en el botón Exportar. 4-Selecciona dirección donde se guardará el archivo. 5-Escribe nombre de archivo. 6-Da clic en botón Guardar. |
| EC Exportar función con campos no válidos | 1.3 No exporta el DDL de la función | V(C:\Users\) V(C//:g) V(C//:g) | I(*/*) V(dd_funcion.txt) I(*/*) | El sistema no guarda el DDL de la función. No cierra el cuadro de diálogo donde se especifica la dirección y el nombre archivo hasta entrarlos correctamente o cancelar la operación. | 1-El usuario da clic en el botón Crear función o selecciona una función del listado y da clic en el botón Editar función. 2-Va a la pestaña DDL. 3-Da clic en el botón Exportar. 4-Selecciona dirección donde se guardará el archivo. 5-Escribe nombre de archivo. 6-Da clic en botón Guardar. |

Por cada sección se especifican las variables utilizadas mediante una tabla, a la que se le llama, Descripción de variables. En la misma se enumeran, se le asignan un nombre sugerente al campo correspondiente a la variable, se especifica el componente donde se recogen los datos, además de, si puede o no tener valor nulo, y se proporciona una breve descripción donde se detallan los valores válidos que permite.

Tabla 9: Ejemplo de la descripción de las variables, correspondiente a la sección “Exportar función”

| No | Nombre de campo | Clasificación | Valor Nulo | Descripción |
|----|-------------------|---------------|------------|---|
| 1 | Dirección | Line edit | Si | Direcciones similares a C:\Users\, válidas al seleccionarlas en el navegador de archivos del sistema operativo. |
| 2 | Nombre de archivo | Line edit | Si | Alfanumérico. No admite "\<>?/". Único para cada archivo. Es obligatorio entrar un nombre correctamente. |

Capítulo 3: Validación del plugin “Editor de funciones y disparadores”

3.3 Presentación de resultados

Se realizaron 3 iteraciones para la corrección de las no conformidades encontradas en la 1ra iteración de la aplicación de los casos de pruebas. En la 1ra iteración se encontraron 7 no conformidades, 3 de ortografía, 1 de interfaz y 3 de validación. En la 2da iteración se encontraron 3 no conformidades, 1 de ortografía y 2 de validación. En la 3ra iteración se obtuvieron resultados satisfactorios, encontrándose 0 no conformidades.

La siguiente imagen muestra el comportamiento de la cantidad de no conformidades de ortografía, interfaz y validación en las 3 iteraciones.

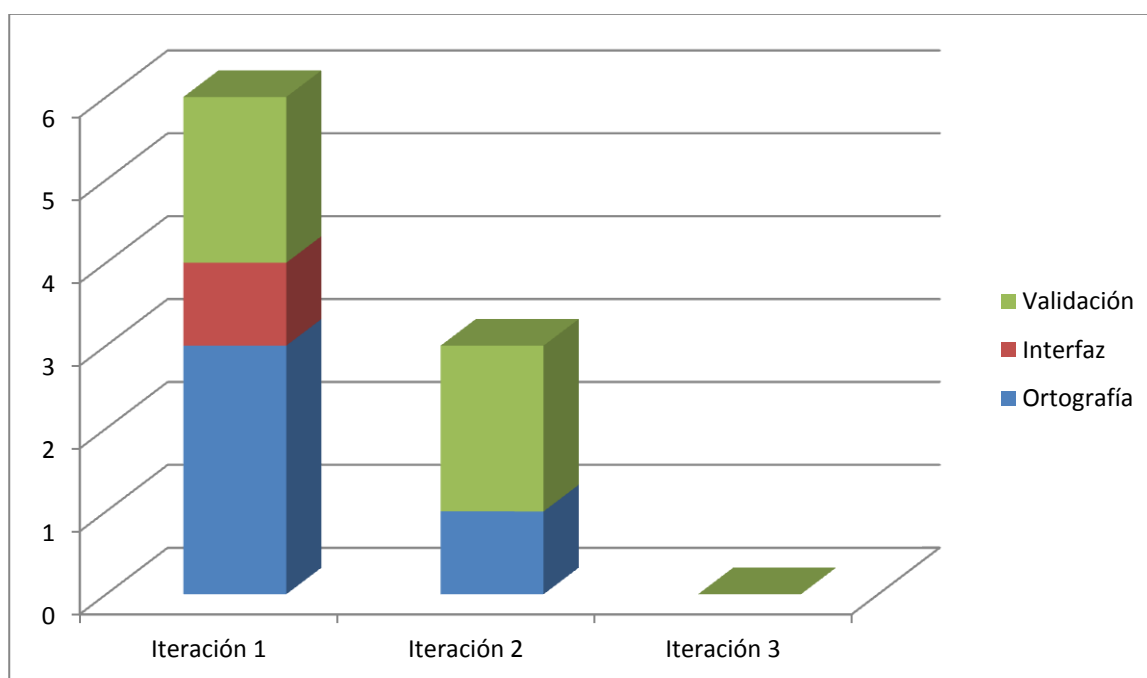


Figura 17: Resultados de la aplicación de los casos de pruebas.

Conclusiones parciales

La validación del sistema se realiza mediante las pruebas de aceptación utilizando la técnica alfa. Luego de aplicados los casos de pruebas diseñados, se encontraron 7 no conformidades. A partir de estas se realizaron 2 iteraciones para solucionar los errores, obteniendo resultados satisfactorios en la 3ra iteración.

Conclusiones

El desarrollo de la presente investigación obtuvo satisfactoriamente un plugin capaz de definir operadores propios y almacenar procedimientos que permiten incorporar lógica del negocio dentro de la base de datos a través de la programación de rutinas del lado del servidor para la herramienta HABD.

Para el plugin “Editor de funciones y disparadores” se implementaron todos los requerimientos, cumpliendo así con las Historias de Usuario definidas.

Con dicho resultado se pretende contribuir a la administración del gestor PostgreSQL en la Comunidad Técnica Cubana de PostgreSQL.

Se constata, en específico, un plugin que integrado a HABD permite a los usuarios que interactúen con él crear, editar, mostrar y eliminar funciones y disparadores. Además la búsqueda de una función o disparador de forma rápida, las validaciones de campos desde interfaz gráfica sin delegar esta responsabilidad al gestor PostgreSQL y la inclusión de una ayuda apoyada en imágenes permitiendo al usuario un fácil manejo de la aplicación son los valores agregados del resultado de esta investigación.

La investigación fue presentada en el evento Jornada Científica Estudiantil a nivel de facultad donde obtuvo mención.

Recomendaciones

Para el desarrollo de futuras versiones del Plugin “Editor de funciones y disparadores” se recomienda:

1. Incorporar al plugin un analizador sintáctico del lenguaje procedural plpgsql en aras de corregir errores de sintaxis sin necesidad de utilizar el propio de PostgreSQL.
2. Incorporar otros lenguajes procedurales como plperl, pljava, plpython y c para extender el uso del “Editor de funciones y disparadores”.
3. Incorporar al plugin la funcionalidad para mostrar las dependencias asociadas a las funciones y disparadores.

Referencia bibliográfica

1. **Mahiques, Sergio.** Editorial CEP. [En línea] 2012. <http://www.editorialcep.com/oposiciones-secundaria/muestra/PGA2012.pdf>.
2. **Ecured.** Ecured. [En línea] <http://www.ecured.cu/index.php/SGBD>.
3. **Lockhart, Thomas.** Grupo de usuarios PostgreSQL de Argentina. [En línea] 2009. <http://www.arpug.com.ar/trac/wiki/TutorialPostgreSql>.
4. PostgreSQL. [En línea] 2010. http://www.postgresql.org.es/sobre_postgresql.
5. **Ecured.** Ecured. [En línea] <http://www.ecured.cu/index.php/SGBD>.
6. **Group, The PostgreSQL Global Development.** PostgreSQL. [En línea] 12 de septiembre de 2011. <http://www.postgresql.org/about/press/presskit91/es/#features>.
7. Kioskea. [En línea] 2008. <http://es.kioskea.net/contents/bdd/bddintro.php3>.
8. Guía de la Comunidad para las herramientas GUI de PostgreSQL. [En línea] [Citado el: 23 de Noviembre de 2011.] postgres.org.com.
9. Guía de la Comunidad para las herramientas GUI de PostgreSQL. [En línea] [Citado el: 15 de Noviembre de 2011.] <http://www.pgadmin.org/>.
10. **Down.** Down. [En línea] 2009. <http://es.downv.com/download-EMS-SQL-Manager-for-PostgreSQL-10126856.htm>.
11. **Perez, Reisel Gonzalez.** [En línea] 2011. <http://www.monografias.com/trabajos-pdf2/sistema-gestion-base-datos-postgresql/sistema-gestion-base-datos-postgresql.pdf>.
12. **Highsmith, J.** *Agile Software Development Ecosystems*. s.l. : Addison-Wesley, 2002.
13. **Beck, Kent.** [En línea] <http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>.
14. **Escribano, Gerardo Fernández.** [En línea] 2002. <http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>.
15. **Maldonado, Daniel M.** El código. [En línea] 2008. <http://www.elcodigok.com.ar/category/uml/page/2/>.
16. **Sierra, Maria.** [En línea] <http://personales.unican.es/ruizfr/is1/doc/lab/01/is1-p01-trans.pdf>.
17. **Qt, Zona.** Zona Qt. [En línea] <http://www.zonaqt.com/book/export/html/282>.
18. [En línea] 2012. <http://qt.nokia.com>.
19. **Fogel, Karl.** [En línea] 2007. <http://producingoss.com/es/vc.html>.

20. **Castillo Oswaldo, Figueroa Daniel, Sevilla Hector.** TRIPOD. [En línea] <http://programacionextrema.tripod.com/index.htm>.
21. **López, Alejandro Rivera.** [En línea] 2008. http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/rivera_l_a/capitulo2.pdf.
22. **Tedeschi, Nicolás.** [En línea] <http://msdn.microsoft.com/es-es/library/bb972240.aspx>.
23. Apuntes de Programación. [En línea] <http://www.apuntes.delibertad.com/java/patron-singleton/>.
24. **J. J. Gutierrez, M. J. Escalona, M. Mejías, J. Torres.** [En línea] https://docs.google.com/viewer?a=v&q=cache:2WiXBhENRwUJ:www.lsi.us.es/~javierj/investigacion_ficheros/PSISEXTREMA.pdf+dise%C3%B1os+de+pruebas+en+xp&hl=es&gl=cu&pid=bl&srcid=ADGEESiEQ7w-TlqUUkw76IJ-1T5MmQU2CitFFph40AWJIRX9zBI_QFzrRXiVQIPE3Qkt4N5JbVFR8iYEEK.
25. **Presman, Roger S.** *Ingeniería de software un enfoque practico.*

Bibliografía

1. **Mahiques, Sergio.** Editorial CEP. [En línea] 2012. <http://www.editorialcep.com/oposiciones-secundaria/muestra/PGA2012.pdf>.
2. **Ecured.** Ecured. [En línea] <http://www.ecured.cu/index.php/SGBD>.
3. **Lockhart, Thomas.** Grupo de usuarios PostgreSQL de Argentina. [En línea] 2009. <http://www.arpug.com.ar/trac/wiki/TutorialPostgreSql>.
4. PostgreSQL. [En línea] 2010. http://www.postgresql.org/es/sobre_postgresql.
5. **Ecured.** Ecured. [En línea] <http://www.ecured.cu/index.php/SGBD>.
6. **Group, The PostgreSQL Global Development.** PostgreSQL. [En línea] 12 de septiembre de 2011. <http://www.postgresql.org/about/press/presskit91/es/#features>.
7. Kioskea. [En línea] 2008. <http://es.kioskea.net/contents/bdd/bddintro.php3>.
8. Guía de la Comunidad para las herramientas GUI de PostgreSQL. [En línea] [Citado el: 23 de Noviembre de 2011.] postgres.org.com.
9. Guía de la Comunidad para las herramientas GUI de PostgreSQL. [En línea] [Citado el: 15 de Noviembre de 2011.] <http://www.pgadmin.org/>.
10. **Down.** Down. [En línea] 2009. <http://es.downv.com/download-EMS-SQL-Manager-for-PostgreSQL-10126856.htm>.
11. **Perez, Reisel Gonzalez.** [En línea] 2011. <http://www.monografias.com/trabajos-pdf2/sistema-gestion-base-datos-postgresql/sistema-gestion-base-datos-postgresql.pdf>.
12. **Highsmith, J.** *Agile Software Development Ecosystems*. s.l. : Addison-Wesley, 2002.
13. **Beck, Kent.** [En línea] <http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>.
14. **Escribano, Gerardo Fernández.** [En línea] 2002. <http://www.info-ab.uclm.es/asignaturas/42551/trabajosAnteriores/Presentacion-XP.pdf>.
15. **Maldonado, Daniel M.** El código. [En línea] 2008. <http://www.elcodigok.com.ar/category/uml/page/2/>.
16. **Sierra, Maria.** [En línea] <http://personales.unican.es/ruizfr/is1/doc/lab/01/is1-p01-trans.pdf>.
17. [En línea] 2012. <http://qt.nokia.com>.
18. **Fogel, Karl.** [En línea] 2007. <http://producingoss.com/es/vc.html>.
19. **Castillo Oswaldo, Figueroa Daniel, Sevilla Hector.** TRIPOD. [En línea] <http://programacionextrema.tripod.com/index.htm>.

20. **López, Alejandro Rivera.** [En línea] 2008. http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/rivera_l_a/capitulo2.pdf.
21. **Tedeschi, Nicolás.** [En línea] <http://msdn.microsoft.com/es-es/library/bb972240.aspx>.
22. Apuntes de Programación. [En línea] <http://www.apuntes.delibertad.com/java/patron-singleton/>.
23. **J. J. Gutierrez, M. J. Escalona, M. Mejías, J. Torres.** [En línea] https://docs.google.com/viewer?a=v&q=cache:2WiXBhENRwUJ:www.lsi.us.es/~javierj/investigacion_ficheros/PSISEXTREMA.pdf+dise%C3%B1os+de+pruebas+en+xp&hl=es&gl=cu&pid=bl&srcid=ADGEESiEQ7w-TlqUUkw76IJ-1T5MmQU2CitFFph40AWJIRX9zBI_QFzrRXiVQIPE3Qkt4N5JbVFR8iYEEK.
24. **Presman, Roger S.** *Ingeniería de software un enfoque practico.*
25. [En línea] [Citado el: 27 de Noviembre de 2011.] <http://pesona.mmu.edu.my/~wruslan/SE2/Readings/detail/Reading-7.pdf>.
26. Grupo de usuarios PostgreSQL de Argentina. [En línea] 2010. <http://www.arpug.com.ar/trac/wiki/PgAdmin>.
27. Microsoft. [En línea] 2008. <http://www.microsoft.com/es-es/sqlserver/get-sql-server/how-to-buy.aspx>.
28. **Griffith, Arthur.** *KDE 2/QT Programing Bible.* s.l. : IDG Books Worldwide. ISBN: 0-7645-4682-1.
29. **Robles, M.C..E.A.** *Métricas para la Gestión de Proyectos de Software.* pág. 65.
30. **610-1990, IEEE.** Calidad de Software. [En línea] Enero de 2011. [Citado el: 17 de Diciembre de 2011.] http://www.ecured.cu/index.php/Calidad_de_Software.
31. *Especificación de Requisitos de Software según el estándar de IEEE 830 IE78.* **Agut, Ing. Raúl Monferrer.** Departamento de Informática. Universidad de Jaume : s.n., Ingeniería de Software. 5to Curso de Ingeniería Informática(2000-2001).
32. **Vázquez, Roberto Hugo.** Taller de Calidad de Sostware: Introducción a la Calidad de Software. [En línea] [Citado el: 15 de Diciembre de 2011.] <http://gridtics.frm.utn.edu.ar/docs/Introduccion%20a%20la%20Calidad%20de%20Software%20Vazquez.pdf>.
33. Oracle. [En línea] 2011. www.oracle.com/us/corporate/pricing/technology-price-list-070617.pdf.
34. PostgreSQL. [En línea] 2011. <http://www.postgresql.org.es/>.

35. SQLManager.net. [En línea] <http://sqlmanager.net/products/postgresql/manager>.
36. **Duerto, Ing. Adriana.** Scribd. [En línea] 25 de marzo de 2012. <http://es.scribd.com/doc/86667874/75/PgAdmin-III>.