Universidad de las Ciencias Informáticas

Facultad 6

Título: "Versión 2.0 del Módulo Diseñador de Consultas para el Generador Dinámico de Reportes v2.0."

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autores:

Jisel Aguilar Gómez

Rolando Enrique Fuentes Orozco

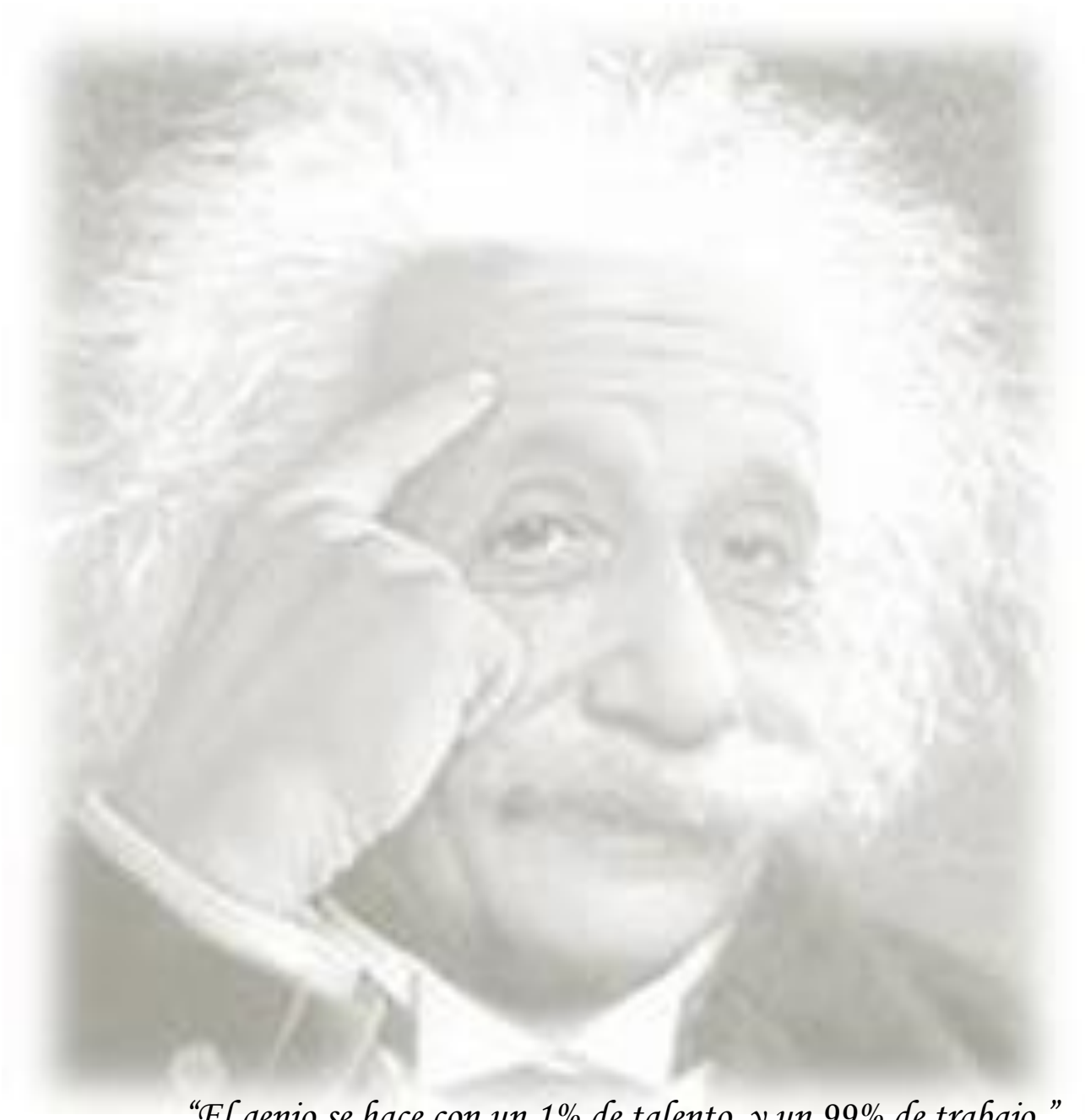
Tutores:

Ing. Adisley Reyes Crespo

Ing. Fernando Henríquez Amaya

La Habana, junio de 2015

"Año 57 de la Revolución"



"El genio se hace con un 1% de talento, y un 99% de trabajo."
Albert Einstein

DECLARACIÓN DE AUTORÍA

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

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

Jisel Aguilar Gómez

_____

Firma del autor

Rolando Enrique Fuentes Orozco

_____

Firma del autor

Adisley Reyes Crespo

_____

Firma del tutor

Fernando Henríquez Amaya

_____

Firma del tutor

DATOS DE CONTACTO

Tutor:

Ing. Adisley Reyes Crespo: Ingeniera en Ciencias Informáticas, es profesora asistente. Cuenta con 8 años de experiencia. Se graduó en el año 2007 en la Universidad de las Ciencias Informáticas.

Correo electrónico: areyesc@uci.cu.

Tutor:

Ing. Fernando Henríquez Amaya: Ingeniero en Ciencias Informáticas, graduado en el año 2013 en la Universidad de las Ciencias Informáticas. Cuenta con 1 año de experiencia trabajando en el centro DATEC de la facultad 6 de la Universidad de las Ciencias Informáticas.

Correo electrónico: fhenriquez@uci.cu.

AGRADECIMIENTOS

De Rolando:

En momentos como este miramos atrás y nos parece que comenzamos justo ayer, nos preguntamos como el tiempo pasó tan rápido. La era de la universidad se terminó, pero nos quedan los recuerdos, los amigos y los conocimientos. Aún recuerdo mis inicios en la programación, fue la primera vez que me enamoré, cuando mi primer programa dijo "Hola mundo". Hoy que he alcanzado esta gran meta de convertirme en ingeniero y quisiera agradecer a todas las personas que en gran medida contribuyeron a ello.

A mis padres por hacer tanto por mí y por estar siempre dispuestos a lo que sea necesario con tal de ayudarme, gracias por su amor, su cariño, su comprensión y su confianza, ustedes son lo más grande que tengo en la vida, espero que se sientan orgullosos de mí.

A mi padre Félix, gracias por el apoyo, por la amistad, por la preocupación, por la confianza, por los consejos y sobre todo por quererme como un hijo, puede estar seguro de que yo también lo quiero como un padre. Muchas gracias.

A mi abuela Juana, mi prima Karenia (tata), mi tía Sorelis y mi tío Francisco, gracias por creer siempre en mí, por darme ánimos, por ayudarme y por el amor que me brindan, son parte fundamental de mi éxito.

A mi compañera de tesis, más que compañera amiga, si hubiese que hacerlo de nuevo, también te volvería a elegir, gracias por guardar todas mis flores, por compartir tan buenos momentos, gracias por las observaciones, por la opiniones, por la confianza, gracias por tu paciencia, por soportarme y cuidarme. Sobre todo gracias por tu amistad, que vale oro.

A mis tutores, gracias por la ayuda y el apoyo, por sacar lo mejor de nosotros, por soportar nuestras discordias cuando algo no encajaba, gracias por guiarnos y por defendernos.

A los profesores Yuned, Alberto Garnache, Asdrubal, Liesner, Vania, Irina y Yuraisy por las recomendaciones, por la buena intención de ayudar, por la paciencia, por la sinceridad y sobre todo por hacernos mejorar de la manera más constructiva posible.

A mi amigo personal y repositorio público de conocimiento Luis Miguel Lemus, gracias por escuchar, por compartir, por la amistad y por ayudar siempre que te necesito. Estaré aquí para lo que necesites también.

A mis amigos José Luis y Alina, Lisandra Morgado, los muchachos del 125, mi amiga Agnes, Yoel, la gente del 106 201, la gente del dominó, que siempre estuvieron ahí cuando hizo falta.

A mis amigos y hermanitos Ale y Guillermo y mi novia Anaibis, personas súper importantes que me han convertido en alguien mejor y que me han ayudado mucho, sin tener que pedirlo, muchas gracias por todo, a todos los quiero.

De Jisel:

No importa cuánto desfallezca, cuantas veces haya caído en el intento, hoy después de muchos años de estudio estoy aquí compartiendo el resultado de tanta añoranza. Pasamos mucho tiempo esperando este momento, dibujándolo en hojas blancas, imaginando cada detalle y a unos días de que llegue estoy

segura que va a ser impredecible, porque los sueños siempre son inesperados y la vocación solo tiene meta en momentos como este. Son innumerables las personas a las que tengo que agradecerles por ayudarme a llegar.

A mi Mamá, mi Papá, mi hermano, Mairene, Alejandro por ser las personas más importantes en mi vida, por su amor, dedicación y confianza que siempre han depositado en mí, sin ustedes este sueño no hubiese sido posible.

Gorda gracias por estar en los buenos y malos momentos de mi vida, por ser más que una amiga mi hermana, que a pesar de estar lejos hemos seguido unidas y sé que así será.

A mi mejor amiga Lisi, que siempre sin duda estuvo ahí, no tengo como agradecerte, a pesar de los malos momentos y las fuertes pruebas que la vida nos puso en el camino regresamos juntas para hacer cumplir el sueño tanto de nosotras como de nuestra familia.

A mi dúo de tesis Rolando, si hubiese que hacer este trabajo de nuevo elegiría volver a ser tu compañera, pues a pesar de todos los obstáculos nunca dudamos y nos apoyamos incondicionalmente, por ayudarme a hacer realidad mi sueño, aguantarme todas mis malcriadeces y peleas, por estar siempre presente cuando más lo necesitaba y porque "sin ti no hubiera sido posible esto".

A todas las personas maravillosas que he conocido durante estos cinco cursos los que están hoy aquí y los que no con los cuales he pasado momentos inolvidables en especial a mis amigos Héctor, a mi bichi, Alfredo, Coco, Tahimi, laura, aylin, anabel, yisel, Eric, pollo, Fito, Graciela, Yosbel, Arian, Fuki, Kirenia, mi dudi bello, Viltre por ser un ejemplo para mí, por brindarme siempre tu apoyo incondicional en todos los momentos difíciles por los que pase sin dudarlo ni un instante, a yune por ser más que una tutora para mí por dedicarme todo el tiempo y ayudarme cada vez que la necesité sin tener un no como respuesta, a mi chuchita Irina por aconsejarme y ayudarme aunque sea al final pero siempre lo haces nunca lo dudé, a Yuraysi porque sé que a pesar de protestar y de no estar adaptada a convivir con nadie me aceptó y me dio un lugarcito en su corazón ,mama tú también tienes uno en el mío, aunque sé que muchas veces no apoyabas mis decisiones sé que estabas al tanto de ellas y te preocupabas ,a mi pupi Karolay por confiar sus alegrías y tristezas en mí , aconsejarme y escucharme cuando más lo necesitaba y sobre todo por ser una de las personas más fuertes que conocí en la vida gracias a ti he aprendido a afrontar decisiones difíciles, a Amauris por preocuparse por mí y estar al tanto de mis problemas ,a la segunda madre que hemos tenido en la universidad alguien que todos los días por las mañanas pasa a ver cómo estamos y a preocuparse por si tenemos algún problema y de buscar una solución Migdalia. Gracias por todos los buenos momentos que pasamos juntos y los recuerdos que quedaron grabados para siempre.

A mis tutores Adisley (chuchi) y Fernando por compartir conmigo sus conocimientos y experiencias, por brindarme siempre sus consejos oportunos. Gracias por su grandísimo esfuerzo, por toda la preocupación y por todos los regaños cuando fueron necesarios.

A mi tribunal y a mi oponente por estar dispuestos siempre a ayudarme en todo lo que necesité y por los buenos consejos me dieron siempre.

DEDICATORIA

Jisel:

Dios por darme la oportunidad de compartir este sueño con ustedes: Mi familia.

A mi mamá que con esfuerzo y desvelo ha dedicado su vida para cultivarme de valores humanos y forjarme como una mujer de bien. Ella que ha sentado sus esperanzas en verme convertida en profesional y nunca ha dudado de mis capacidades y espíritu de lucha, mamá no te defraudé. Te amo.

A mi papá por enseñarme que la perseverancia y el esfuerzo son el camino para lograr los objetivos que nos propongamos en la vida, por su apoyo, confianza, por todo su amor, dedicación y entrega, por ser oportuno con sus palabras y acciones cuando lo he necesitado, por guiarme certeramente en la ardua tarea de formarme como profesional.

A mi hermano por ser un ejemplo para mí y apoyarme siempre en mis decisiones. Gracias a tus consejos he llegado a realizar una de mis grandes metas, sin ti hubiese sido imposible alcanzar esta experiencia tan maravillosa. Este trabajo simboliza mi gratitud por toda la responsabilidad e invaluable ayuda que siempre me has proporcionado. Sé que este triunfo es motivo de orgullo para ti, y es porque este triunfo también es tuyo, eres lo más grande que tengo en la vida, te adoro.

Rolando:

Dios por darme la oportunidad de cumplir esta gran meta y cuidar de las personas que quiero.
A mi mamá, que ha sabido ser educadora, forjadora de mis virtudes y crítica de mis defectos. Quien me crió como hombre de bien y ha sido madre y amiga al mismo tiempo. Por ayudarme y apoyarme siempre que sea necesario, por sus consejos que me guían y por el amor que me brinda. Esta es la respuesta a todo su esfuerzo y sacrificio.

A mi papá por ser ejemplo a seguir como padre y amigo, por los consejos, por su amor, por apoyarme y animarme a seguir luchando cuando todo parece difícil, por comprenderme y escucharme, por ayudarme. Eres quien me enseñó a levantarme con más fuerza si me caigo. Con este triunfo cumplo una gran meta, ser ingeniero como tú, quiero que te sientas orgulloso.

A mi padre Félix por estar conmigo siempre en pensamiento. Gracias por sus consejos, por quererme como un hijo, por su apoyo, agradezco a dios por haberlo puesto en mi camino y hacerme merecedor de su amistad y amor.

A mi prima Karenia (tata) por creer siempre en mí y sobre todo por ser la hermana que no tuve. A mi abuela Juana por cuidarme, por ayudarme y por estar siempre pendiente de mí. A mi tía Sorelis, a mi tío Francisco, a mi abuela Nidia que me mira desde el cielo. Espero se sientan orgullosos de mí.

RESUMEN

El Departamento de Desarrollo de Componentes perteneciente al Centro de Tecnologías de Gestión de Datos de la Universidad de las Ciencias Informáticas, desarrolla el sistema Generador Dinámico de Reportes, el cual es una aplicación web que permite a sus clientes consultar la información almacenada en las bases de datos de sus organizaciones y poder generar reportes de forma rápida e interactiva con la información que estos manejan. Presenta entre sus módulos el Diseñador de Consultas el cual funciona como un editor gráfico de consultas SQL, permitiendo crear, editar y ejecutar consultas del tipo antes mencionado, sin necesidad de conocer el gestor de base de datos ni el lenguaje de consulta estructurado. La presente investigación tiene como objetivo desarrollar una solución informática que permita al usuario trabajar con las entidades de la base de datos de forma visual y que además les permita crear sus propias consultas mediante la vista SQL del Diseñador de Consultas. Como resultado se obtiene la versión 2.0 del módulo Diseñador de Consultas, que permite al usuario crear consultas simples, con parámetros y consultas anidadas para los gestores de bases de datos PostgreSQL, MySQL y SQLite. La solución fue validada mediante pruebas de unidad, integración y validación, comprobando su correcto funcionamiento.

Palabras Claves: Consultas SQL, Diseñador de Consultas, Generador Dinámico de Reportes.

ABSTRACT

The Components Development Department of the Data Technologies Center of the Informatics Sciences University, is developing the project Dynamic Reports Generator, which is a web application that enables its users to consult the information stored in the databases of their organizations and to generate reports quickly and interactively with the information they handle. The Query designer is one of its modules, and this one works as a graphical SQL queries editor, allowing creating, editing and executing queries of the types mentioned above without the need to know the database management system nor the structured query language. This investigation aims to develop an informatics solution which enables users to visually handle the database entities and to create their own queries using the Query designer SQL view. As result, version 2.0 of the Query designer module is obtained, which allows creating simple, parameterized and nested queries for the PostgreSQL, MySQL and SQLite data base management systems. The solution was validated through unit, integration and validation tests to check its proper performance.

Keywords: SQL queries, queries designer, Dynamic Reports Generator.

INDICE

ÍNDICE DETABLAS

ÍNDICE DEFIGURAS

INTRODUCCIÓN

Preservar la información que se maneja en las diferentes esferas del sector empresarial cobra vital importancia para su funcionamiento y desarrollo, pues la organización y accesibilidad de la misma agiliza la toma de decisiones en cuanto a los procesos sustantivos que en ellas se llevan a cabo. Un elemento que ha tenido un papel fundamental en el logro de este objetivo ha sido el desarrollo actual de las Tecnologías de la Información y las Comunicaciones (TICS), y dentro de ellas los Sistemas Gestores de Bases de Datos (SGBD), los cuales permiten almacenar y posteriormente acceder a los datos de las empresas de forma rápida y segura.

Una de las vías para mostrar la información de las bases de datos es mediante los reportes. En el ámbito de la informática *"son informes que organizan y exhiben la información contenida en una base de datos"* (Rodríguez, 2013). Su función es aplicar un formato determinado a los datos para mostrarlos por medio de un diseño intuitivo y que sea fácil de interpretar por los usuarios. Estos tienen diversos niveles de complejidad a la hora de visualizarlos ya que pueden ser desde listas hasta gráficos. Existen diferentes tecnologías que permiten generar reportes de forma dinámica y así lograr una mayor efectividad a la hora de hacer uso de ellos. Entre los que se encuentran los generadores de reportes, que pueden generar informes resumidos y detallados en tiempo real desde diferentes fuentes de datos.

Los sistemas generadores de reportes son herramientas complementarias de los sistemas de información. Utilizan una especie de lenguaje transparente para el usuario, por medio del cual este realiza consultas a la base de datos y obtiene información de ella en forma de reporte. (Silva Hernández, 2003). Dentro de los generadores de reportes un elemento importante son los diseñadores de consultas (*QueryBuilder*) pues estos le facilitan al usuario consultar y recuperar información de una base de datos sin tener conocimientos avanzados del SGBD y del lenguaje de consultas que el mismo emplea para almacenar la información.

Los diseñadores de consultas han alcanzado un desarrollo significativo que va más allá de brindar facilidades a los usuarios en el diseño de consultas dinámicas. Los mismos poseen una interfaz gráfica que facilita que se incluyan uniones de consultas y sub-consultas permitiendo guardarlas y editarlas. Es por ello que la elaboración de estas herramientas ha tenido un auge en los últimos tiempos, debido al

avance tecnológico y a las necesidades de los clientes, entre las que se encuentran MySQL-workbench, DbForgeQueryBuilderfor SQL Server, RazorSQL y el ExtJS Visual SQL QueryBuilder.

En la UCI, específicamente en el Centro DATEC perteneciente a la Facultad 6 se encuentra el Departamento de Desarrollo de Componentes (DDC). En dicho departamento se desarrolla el Generador Dinámico de Reportes (GDR); el cual permite obtener diferentes reportes a partir de los datos obtenidos mediante los sistemas de gestión de la información, con el objetivo de apoyar la toma de decisiones, realizar estudios investigativos o consultar información propia del negocio en el cual esté siendo utilizado.

El GDR cuenta con el módulo Diseñador de Consultas que permite la elaboración y manejo de consultas SQL desde la vista de diseño y la vista SQL. Ambas están interconectadas de forma tal que las modificaciones realizadas en el diseño, son actualizadas automáticamente en el código y las modificaciones realizadas en el código son representadas en el diseño.

En la versión actual, el diseñador de consultas sólo permite diseñar consultas simples, además el parser del diseñador sólo parsea estructuras SQL básicas para el gestor de bases de datos PostgreSQL; lo que trae consigo que los usuarios con dominio de los gestores MySQL y SQLite se encuentren imposibilitados de usar dicha herramienta para la creación de consultas; no permite además usar consultas anidadas, ni definir expresiones que se puedan sustituir por datos (en lo adelante parámetros). Todo lo anterior genera la necesidad de permitir al usuario diseñar consultas que le brinden resultados más detallados durante la creación de los reportes con mayor complejidad.

A partir de las irregularidades antes descritas, se plantea como **problema de la investigación**: La versión actual del Diseñador de Consultas del Generador Dinámico de Reportes sólo permite parsear consultas simples para el gestor de bases de datos PostgreSQL.

Defendiéndose como **objeto de estudio**: Desarrollo de sistemas para la generación dinámica de reportes, enmarcado en el **campo de acción**: Diseño de consultas SQL en el Generador Dinámico de Reportes v2.0.

Para darle solución al problema planteado se define como **objetivo general**: Desarrollar la versión 2.0 del Diseñador de Consultas para el Generador Dinámico de Reportes v2.0 que permita a los usuarios el diseño de consultas simples, anidadas y con parámetros para los gestores PostgreSQL, MySQL y SQLite.

Para guiar la investigación se definen las siguientes **preguntas de la investigación**:

- ¿Cuáles son las bases teóricas que fundamentan el diseño de consultas SQL para el GDR?
- ¿Cuáles son las características y capacidades que debe tener la versión 2.0 del módulo diseñador de consultas para que permita diseñar consultas simples, anidadas y con parámetros para los lenguajes PostgreSQL, MySQL y SQLite?
- ¿Cómo obtener una guía legible y comprensible para la generación de códigos y la posterior comprobación y soporte del diseñador de consultas?
- ¿Cómo traducir el diseño obtenido para la versión 2.0 del módulo diseñador de consultas en un producto con capacidad operacional?
- ¿Cómo validar el correcto funcionamiento de la versión 2.0 del módulo diseñador de consultas en función del diseño de consultas desde la vista SQL?

Para dar cumplimento al objetivo general se definen como **tareas de la investigación**:

- Análisis de los elementos conceptuales implicados en el desarrollo de la versión 2.0 del Módulo Diseñador de Consultas para elaborar el marco teórico de la investigación.
- Caracterización de la metodología y herramientas a utilizar para guiar el desarrollo de la versión 2.0 del Módulo Diseñador de Consultas para el Generador Dinámico de Reportes v2.0.
- Identificación de las necesidades funcionales y tecnológicas del Módulo Diseñador de Consultas para su correcto funcionamiento.
- Definición de los principios arquitectónicos y de diseño del Módulo Diseñador de Consultas para definir su estructura global.
- Implementación de los principios arquitectónicos y de diseño definidos para satisfacer las necesidades funcionales y tecnológicas de la versión 2.0 del Módulo Diseñador de Consultas.
- Realización de las pruebas de software para comprobar el correcto funcionamiento de la versión 2.0 del Módulo Diseñador de Consultas.

Para el desarrollo de la investigación se utilizaron diferentes métodos científicos agrupados en métodos teóricos y empíricos, los cuales tienen su sustento en la concepción materialista dialéctica y permiten una mejor recopilación de la información relacionada con la realización de estudios asociados a los diseñadores de consultas.

- **Métodos Teóricos**

Analítico-Sintético: Se utiliza este método para analizar los conceptos teóricos existentes sobre los diseñadores de consultas SQL y extraer las características que debe cumplir la versión del módulo a desarrollar para el Generador Dinámico de Reportes v2.0.

Modelación: Se utiliza para a través de los modelos de análisis, diseño e implementación representar la realidad objetiva en que se enmarca el sistema y para modelar la solución del software.

- **Métodos Empíricos**

Análisis Estático: Se emplea para la revisión del código fuente de la versión 1.8 del módulo Diseñador de Consultas del Generador Dinámico de Reportes v 2.0, con el objetivo de entender la estructura de la aplicación y la interrelación entre las clases que la componen.

Entrevista estructurada: Se realiza a especialistas del Departamento de Desarrollo de Componentes con el objetivo de recopilar información referente a la realización de GDR 1.8, y así obtener información relevante que permita fundamentar el problema científico, utilizando la técnica de Guía de Entrevista (Ver anexos 1).

- **Técnicas de recopilación de información**

Tormenta de ideas: Se emplea con el objetivo de identificar las características y capacidades con las que debe cumplir la versión 2.0 del Módulo Diseñador de Consultas para el Generador Dinámico de Reportes v2.0. Teniendo en cuenta los criterios del jefe de proyecto, los tutores de la investigación, desarrolladores de la versión 1.8 del módulo y los autores.

El presente trabajo de diploma está estructurado en 3 capítulos:

Capítulo 1: Fundamentación teórica del diseñador de consulta: En el presente capítulo se abordan los elementos principales que justifican y sirven de base a la solución del problema planteado, analizando los principales conceptos relacionados con el objeto de estudio y el campo de acción. Se realiza el análisis de alguna de las soluciones existentes a nivel mundial para el diseño de consultas y se describen las herramientas y metodología a utilizar durante el desarrollo de la solución propuesta.

Capítulo 2: Análisis y diseño del diseñador de consulta: En el presente capítulo se describe el proceso llevado a cabo durante el análisis y diseño. Se definen los requisitos funcionales y no funcionales del software. Se muestra la estructura del sistema a desarrollar a través de una representación visual de las

clases conceptuales en el modelo de dominio. Se expone una representación gráfica de los procesos de la solución que se propone mediante el diagrama de casos de uso del sistema. Se muestran los diagramas de clases del diseño, por medio de los cuales se representa la estructura del sistema.

Capítulo 3: Implementación y prueba del diseñador de consulta: En el presente capítulo se describe el modelo de implementación a partir del diseño realizado por cada uno de los casos de uso identificados. Además, se muestran fragmentos del código y las vistas principales de la aplicación. Se describen las pruebas a realizar, con el objetivo de comprobar la correcta implementación de cada una de las funcionalidades definidas, verificando que los resultados de las pruebas sean los esperados y el correcto funcionamiento del sistema.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA DEL DISEÑADOR DE CONSULTA.

En el presente capítulo se abordan los elementos principales que justifican y sirven de base a la solución del problema planteado, analizando los principales conceptos relacionados con el objeto de estudio y el campo de acción. Se realiza el análisis de algunas de las soluciones existentes a nivel mundial para el diseño de consultas y se describen las herramientas y metodología a utilizar durante el desarrollo de la solución propuesta.

1.1. Conceptos básicos relacionados con el dominio del problema.

A continuación se relacionan los principales conceptos o temáticas que están asociados al desarrollo de la investigación.

Base de Datos.

Es un conjunto de datos interrelacionados entre sí, almacenados con carácter más o menos permanente en la computadora. O sea, que una base de datos puede considerarse una colección de datos variables en el tiempo. (J.Date, 2001)

Un aspecto importante para las bases de datos es el manejo de las consultas por lo importante que resultan en el proceso de gestión de los datos. Una consulta es el método para acceder a los datos que existen en las mismas, permitiendo modificarlos, borrarlos, mostrarlos y agregarlos. (Mato Garcia, 2012) Para la obtención apropiada de la información almacenada son usados los sistemas Generadores de Reportes, los cuales hacen uso de consultas para conformar los reportes que darán conocimiento al usuario sobre el estado de la información.

Generadores de Reportes.

Los generadores de reportes son herramientas de apoyo a los sistemas de información, que permiten a los usuarios acceder de forma simple y rápida a los datos guardados, mostrándolos en forma de reportes. Además permiten realizar consultas a las bases de datos obteniendo así la información con un mejor dominio en el diseño y la visualización de los datos obtenidos. (Iliana Amabely Silva Hernández, 2013)

Generador Dinámico de Reportes. (GDR)

El GDR es una herramienta web que permite a sus clientes consultar los gestores de bases de datos de sus organizaciones y poder generar reportes con la información que estos manejan. Consta de 6 módulos: Diseñador de Modelos, Diseñador de Reportes, Diseñador de Consultas, Visor de Reportes,

Administrador de Reportes y Seguridad. El módulo Diseñador de Consultas del GDR funciona como un editor gráfico de consultas SQL permitiendo crear, editar y ejecutar consultas simples del tipo antes mencionado y consta con dos vistas básicas: la vista de diseño y la vista SQL. (VariosAutores, 2012)

Diseñador de Consultas.

El diseñador de consultas funciona como un editor gráfico de consultas SQL, permitiendo crear, editar y ejecutar consultas del tipo antes mencionado, sin necesidad de conocer el gestor de base de datos ni el lenguaje de consulta estructurado (SQL) . (GAZQUEZ MARTÍNEZ, 2011.)

Sus principales funcionalidades son:

- Diseñar consulta visualmente: Mediante la opción de arrastrar y soltar, permite colocar libremente dentro del área de diseño las tablas y/o vistas que conformarán la consulta deseada. Para seleccionar los campos que se quieren consultar, basta con marcarlos dentro de la entidad a la que pertenecen.
- Modificar consulta visual: Carga una consulta previamente diseñada, la cual puede ser modificada.
- Incluir funciones de agregación: Adiciona funciones de agregación a campos de la consulta (suma, cantidad, máximo, mínimo y promedio).
- Incluir condiciones a la consulta: Permite realizar comparaciones entre campos o campos con valores fijos.

Los operadores de comparación que soporta son: Igual a, mayor que, menor que, mayor-igual a, menor-igual a y distinto a.

- Establecer criterios de ordenamiento: Permite ordenar los datos de salida de la consulta seleccionando el o los campos por los cuales se desea ordenar.
- Establecer criterios de agrupamiento: Permite agrupar los datos de salida de la consulta seleccionando el o los campos por los cuales se desea agrupar.
- Relacionar tablas: Relaciona tablas mediante campos. Relación de: Unión interna (INNER JOIN), unión por la izquierda (LEFT JOIN), unión por la derecha (RIGHT JOIN) y unión completa (FULL JOIN).
- Ejecutar consulta diseñada: Ejecuta la consulta diseñada mostrando los resultados de la misma. Además muestra la sentencia SQL de dicha consulta.
- Limitar la consulta: Especifica la cantidad de datos que se obtendrán en la consulta diseñada.

Uno de los procesos que se debe ejecutar en un diseñador de consultas, para garantizar que las consultas generadas sean correctas es la fase de análisis léxico.

Análisis Léxico.

El analizador léxico es la primera fase de un compilador. Su función principal es leer los caracteres de entrada y producir como salida una secuencia de componente léxicos (*tokens*) que el parser luego utiliza en el proceso de análisis sintáctico. (Martínez., 2011). Entre las principales características de un analizador léxico se encuentran:

- Analiza caracteres.
- Produce componentes léxicos (*tokens*).
- Filtra comentarios.
- Filtra separadores múltiples (espacios, tabuladores y saltos de línea).
- Lleva el contador de líneas y columnas del texto fuente.
- Genera errores en caso de que la entrada no corresponda a ninguna categoría léxica.

Categoría léxica.

Tipo de símbolo elemental del lenguaje fuente, (identificadores, palabras numéricas, operadores, clave, constantes, cadenas literales y signos de puntuación, como paréntesis, coma y punto y coma).

Componente léxico (token):

Elemento perteneciente a una categoría léxica, es el terminal asociado a un patrón. Cada token se convierte en un número que es un código identificativo de cada patrón. En algunos casos, cada número tiene asociado un puntero a la tabla de símbolos. Se utiliza la palabra terminal desde el punto de vista de la gramática utilizada por el analizador sintáctico. Son los elementos más básicos sobre los cuales se desarrolla toda traducción de un programa, surgen en la primera fase, llamada análisis léxico, sin embargo se siguen utilizando en las siguientes fases (análisis sintáctico y análisis semántico) antes de perderse en la fase de síntesis.

Atributos de un componente:

Información del componente necesaria en etapas posteriores del análisis (valor de la constante o lexema, nombre de una variable, entrada en la tabla de símbolos, línea, tipo).

Lexema:

Un lexema es una secuencia de caracteres en el programa fuente que coincide con el patrón de un token y se identifica por el analizador léxico como una instancia de ese token.

Tabla de símbolos.

La tabla almacena la información que en cada momento se necesita sobre las variables del programa, información tal como: nombre, tipo, dirección de localización y tamaño. La gestión de la tabla de símbolos es muy importante, ya que consume gran parte del tiempo de compilación; de ahí que su eficiencia sea crítica. (Martínez., 2011)

La tabla de símbolos puede iniciarse con cierta información útil, tal como:

- Constantes: PI y E.
- Funciones de librería: EXP y LOG.
- Palabras reservadas: Esto facilita el trabajo al analizador léxico, que tras reconocer un identificador lo busca en la tabla de símbolos, y si es una palabra reservada devuelve un token asociado.

Luego de completada la fase de análisis léxico se genera una lista de tokens que es utilizada durante la fase de análisis sintáctico.

Análisis Sintáctico.

El análisis sintáctico es el proceso en el cual se examina una secuencia de tokens para determinar si el orden de esa secuencia es correcto de acuerdo a ciertas convenciones estructurales (reglas) de la definición sintáctica del lenguaje.

Comprueba que las sentencias que componen el código fuente son correctas en el lenguaje SQL, creando una representación interna que corresponde a la sentencia analizada. De esta manera se garantiza que sólo serán procesadas las sentencias que pertenezcan al lenguaje fuente. Durante el análisis sintáctico, así como en las demás etapas, se van mostrando los errores que se encuentran. Este verifica los componentes léxicos que fueron enviados por el análisis léxico, lo interpreta y genera un árbol de sintaxis abstracta, para ser enviado al análisis semántico. (Martínez., 2011)

Características del análisis sintáctico:

- Lee componentes léxicos (*tokens*).
- Comprueba que el orden de estos corresponde a la sintaxis predeterminada.
- Genera errores en caso de que el flujo de tokens no responda a la sintaxis.
- Genera árboles de análisis sintáctico.
- Integra el análisis semántico como un conjunto de rutinas a ejecutar durante la comprobación de la sintaxis.

Analizador Sintáctico (o parser).

El analizador sintáctico (o *parser*) se encuentra entre los componentes de un compilador que transforma su entrada en un árbol de sintaxis abstracta. (María del Mar Aguilera Sierra y Sergio Gálvez Rojas, 2014). Los analizadores sintácticos o parsers reconocen estructuras sintácticas a partir de secuencias de tokens.

Coincidiendo con los autores, a partir de los elementos estudiados el analizador sintáctico está compuesto por la clase parser la cual hace uso de la tabla de símbolos, el módulo gestor de errores y la lista de tokens obteniéndose como resultado la ejecución del análisis sintáctico.

Un elemento importante para hacer el parser es conocer las particularidades de los lenguajes que se tratan. A continuación se describen las diferencias sintácticas que poseen las consultas SQL pertenecientes a los lenguajes PostgreSQL, MySQL y SQLite que se tuvieron en cuenta a la hora de construir los *parser* que componen el módulo propuesto como solución.

Tabla 1: Tabla Comparativa sobre las diferencias sintácticas que poseen las consultas SQL.

Tabla comparativa			
Estructura	**Código de ejemplo**		
	PostgreSQL	**SQLite**	**MySQL**
Estructura sintáctica de una declaración de campo y tabla.	SELECT "usuario"."nombre" FROM "usuario";	SELECT `usuario`.`nombre` FROM `usuario`;	SELECT `usuario`.`nombre` FROM `usuario`;
Declaración de alias para los campos seleccionados.	SELECT "usuario"."nombre" As "Nombre" FROM "usuario";	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario`; SELECT `usuario`.`nombre` As `Nombre` FROM 'usuario';	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario`;
Declaración de una cadena de texto dentro de una condición	SELECT "usuario"."nombre" As "Nombre" FROM "usuario" WHERE "usuario"."apellido" = 'López';	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`apellido` = 'López'; SELECT `usuario`.`nombre`	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`apellido` = 'López'; SELECT `usuario`.`nombre` As

		As `Nombre` FROM `usuario` WHERE `usuario`.`apellido` = "López";	`Nombre` FROM `usuario` WHERE `usuario`.`apellido` = "López";
Uso del operador < dentro de una condición.	SELECT "usuario"."nombre" As "Nombre" FROM "usuario" WHERE "usuario"."id" < 5;	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`id` < 5; SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`id` !< 5;	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`id` < 5;
Uso del operador > dentro de una condición.	SELECT "usuario"."nombre" As "Nombre" FROM "usuario" WHERE "usuario"."id" > 5;	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`id` > 5; SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`id` !> 5;	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`id` > 5;
Estructura sintáctica de la declaración de una sentencia LIMIT seguido del OFFSET.	SELECT "usuario"."nombre" As "Nombre" FROM "usuario" WHERE "usuario"."id" > 5 LIMIT 50 offset 0;	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`id` > 5 LIMIT 50 offset 0;	SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`id` > 5 LIMIT 50 offset 0; SELECT `usuario`.`nombre` As `Nombre` FROM `usuario` WHERE `usuario`.`id` > 5 LIMIT 50, 0;

1.2. Análisis de soluciones existentes.

Los generadores de reportes utilizan los diseñadores de consultas como una herramienta que facilita al usuario, a través de una interfaz sugerente, la posibilidad de diseñar sus propias consultas. Además

tienen la ventaja de poder ser utilizados por aquellos usuarios que no tienen amplios conocimientos de un lenguaje de consultoría de datos. A continuación se realizará un estudio detallado enfatizando en la manera en que cada una de estas soluciones manejan el diseño de consultas, tomando como base aquellos aspectos que permitirán sustentar los conocimientos para el desarrollo del Módulo Diseñador de Consultas para GDR.

- **MySQL-workbench.**

MySQLWorkbench es una herramienta visual que proporciona el modelado de datos, desarrollo SQL y herramientas completas de administración de la configuración del servidor, además está disponible en Windows, Linux y Mac OSX. (MySQL, 2014)

- **DbForge Query Builder for SQL Server.**

DbForgeQueryBuilderfor SQL Server es una herramienta para la creación rápida de consultas SQL y gestión de datos ampliada. Permite crear consultas complejas a las bases de datos de SQL Server a través de una interfaz visual intuitiva y sin escribir código manual. El principal componente de QueryBuilder es un diagrama visual donde se puede "dibujar" la consulta, añadir tablas, crear nuevas consultas y agregar las condiciones de edición a dichas consultas.

- **RazorSQL.**

RazorSQL es una herramienta de consulta SQL, un navegador de base de datos, editor de SQL y herramienta de administración de base de datos. Compatible con casi treinta sistemas de bases de datos, como DB2, MySQL, PostgreSQL, Oracle, Informix o SQLite (por citar algunos), se ejecuta tanto en Windows como en Mac y Linux. (Richardson Software, 2013)

- **ExtJS Visual SQL QueryBuilder.**

El propósito de esta herramienta es crear aplicaciones sobre la marcha, por lo que es una aplicación Web sin necesidad de escribir una sola línea de código. No permite que el usuario avanzado pueda introducir código SQL. Se basa en la clase generada a partir de las tablas de la base de datos y también permite crear interfaces para los CRUD (*Create, Read, Update, Delete*) de las tablas de la base de datos.

Los diseñadores de consultas estudiados anteriormente presentan desventajas para su empleo debido a que en distintos casos:

- Tienen un alto coste adquisitivo.
- No son basados en tecnologías Web.

- No son multiplataforma.
- En su mayoría están orientados a un gestor de bases de datos específico.
- No brindan la posibilidad al usuario de editar el código SQL que se genera.

A partir de lo cual se llega a la conclusión que se hace pertinente mantener el diseñador de consultas perteneciente al GDR, añadiéndole nuevas características que mejoren el diseño de las consultas. Tomando en cuenta las potencialidades observadas en los diseñadores anteriores, el diseñador de consultas en su nueva versión:

- Permitirá diseñar consultas que contengan consultas anidadas.
- Posibilitará añadir parámetros dentro de las condiciones de las consultas para que puedan ser usados posteriormente por el GDR en la generación de reportes.
- Facilitará al usuario el diseño de consultas con un nivel de complejidad más alto que en su versión anterior.
- Brindará la posibilidad de parsear las consultas diseñadas para los gestores de bases de datos MySQL y SQLite, manteniendo PostgreSQL.

1.3. Ambiente de desarrollo.

Las herramientas constituyen un conjunto de programas utilizados por los diseñadores y los programadores para desarrollar sistemas. Las mismas deben ser seleccionadas cautelosamente, ya que pueden suponer el fracaso del desarrollo del sistema o pueden aumentar su complejidad. La adecuada selección de las herramientas permite aplicar conocimientos y habilidades con el objetivo de conseguir una solución que permita resolver un problema determinado y lograr satisfacer las necesidades del proyecto.

1.3.1. Metodología de Desarrollo de Software.

La metodología de software se basa en una combinación de los modelos de procesos genéricos para obtener como beneficio un software que solucione un problema. La metodología define con precisión los artefactos, roles y actividades, junto con prácticas, técnicas recomendadas y guías de adaptación de la metodología del proyecto. Sin embargo, la complejidad del proceso de creación de software es dependiente de la naturaleza del proyecto mismo, por lo que la selección de la metodología estará acorde al nivel de aporte del proyecto, ya sea pequeño, mediano o de gran nivel. (José Esteva, 2009) Para el desarrollo del diseñador de consultas se utilizará la metodología definida por el grupo de arquitectura del GDR que se muestra a continuación.

OpenUP.

OpenUP[1] es un proceso ágil y unificado, que contiene el conjunto mínimo de prácticas que ayudan a los equipos a ser más eficaces en el desarrollo de software. Se caracteriza por estar centrado en la arquitectura y guiado por los casos de uso. El ciclo de vida de un proyecto según esta metodología se divide en 4 fases fundamentales: Concepción, Elaboración, Construcción y Transición. (OpenUP, 2002)

Para la realización de la solución se decide utilizar la metodología OpenUP pues es una metodología ágil apropiada para proyectos pequeños, permite disminuir las probabilidades de fracaso e incrementar las probabilidades de éxito. Se acoge a las necesidades del equipo de desarrollo. Posibilita detectar errores tempranos a través de un ciclo iterativo. Evita la elaboración de documentación, diagramas e iteraciones innecesarios ya que es un proceso mínimo, completo y extensible. Tiene un enfoque centrado al cliente y con iteraciones cortas.

1.3.2. Lenguaje de modelado.

El modelado de sistemas de software es una técnica que ayuda al ingeniero de software a visualizar el sistema a construir.Adquiere mayor importancia cuanto mayor es la complejidad del sistema ya que permite estudiar en detalle sus partes elementales. De ahí que los modelos pueden utilizarse para la comunicación con el cliente, grupo de desarrollo y otros factores que intervienen en el proceso de desarrollo (Booch, 2007). La metodología Open Up emplea para la especificación, visualización, construcción y documentación de los artefactos que genera durante el proceso de desarrollo del software el Lenguaje Unificado de Modelado.

Lenguaje Unificado de Modelado. (UML) v2.0

Se utiliza el Lenguaje Unificado de Modelado (UML por sus siglas en inglés) para el modelado de sistemas de software el cual es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. Es un estándar para representar y modelar la información con la que se trabaja en las fases de análisis y, especialmente, de diseño. Permite representar el conocimiento acerca de los sistemas que se pretenden construir y las decisiones tomadas durante su desarrollo. (Booch, 2007)

En el desarrollo del presente trabajo investigativo se hace uso de este lenguaje pues permite realizar una correcta verificación y validación del modelo realizado. Otra de sus ventajas es que debido a que es un lenguaje visual es fácil de entender por cualquier persona del equipo de desarrollo y permite que los modelos creados puedan servir como documentación del ciclo de vida del sistema, con vista a facilitar el desarrollo de futuras versiones o mejoras. Se encarga de modelar, construir y documentar muchos de

[1](*Open Unified Process*).

los artefactos que necesita OpenUP como metodología seleccionada, entre ellos se encuentran los diagramas: de clases del diseño, de casos de uso del sistema, de componente y despliegue. Además permite documentar los artefactos que se generan durante el proceso de desarrollo de software.

1.3.3. Herramienta CASE de modelado.

Las herramientas de Ingeniería de Software Asistida por Computadora (CASE por sus siglas en inglés), fueron desarrolladas con el fin de automatizar los aspectos clave de todo el proceso de desarrollo de un sistema. Se puede definir a las herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. (Menéndez Alonso, 2009)

Visual Paradigm for UML v8.0.

Visual Paradigm para UML es una herramienta CASE desarrollada para cubrir todo el ciclo del desarrollo de software, permitiendo la captura de requisitos, análisis, diseño e implementación. Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Facilita la generación de reportes en formatos: PDF, MS Word, HTML. También está disponible para integrarse en los principales entornos de desarrollo (IDEs por sus siglas en inglés). (VisualParadigm, 2014)

Se utiliza Visual Paradigm debido a que es una herramienta libre, multiplataforma y presenta una interfaz de usuario intuitiva y sencilla que permite realizar los diagramas y artefactos que se generan durante el desarrollo del software. Es fácil de instalar y actualizar. Además ofrece un entorno amigable para el usuario, sugiere nuevos posibles componentes a utilizar, por lo que ya no es necesario localizarlos en la barra y así se crea fácilmente cualquier tipo de diagrama. Incluye gran variedad de estereotipos para la creación de diagramas de fácil entendimiento, los que organiza automáticamente. Además es la herramienta de modelado definida y utilizada para el desarrollo del GDR.2.0.

1.3.4. Lenguajes de Programación.

Un lenguaje de programación es un idioma artificial diseñado para expresar cómputos que pueden ser llevados a cabo por las computadoras. Puede usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión o como modo de comunicación humana. Permiten especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una gran cantidad de opciones posibles. (kioskea, 2014)

PHP 5.3.

PHP es un lenguaje de código abierto, adecuado para el desarrollo web y que puede ser incrustado en HTML. Es un lenguaje de secuencia de comandos diseñado específicamente para permitir a los programadores crear aplicaciones Web con distintas prestaciones. Una de las características más importantes de PHP es que permite el soporte para una gran cantidad de datos. Es simple para principiantes, pero a su vez, ofrece muchas características avanzadas para los programadores profesionales. (ColectivoAutores, 2011)

Se seleccionó para el desarrollo de la solución el lenguaje de programación PHP en su versión 5.3pues no solo es rápido al ser ejecutado sino que no genera retrasos en la máquina, por esto no requiere grandes recursos del sistema. Dispone de una amplia gama de librerías, y permite la posibilidad de agregarle extensiones. Esto le permite su aplicación en múltiples áreas, tales como encriptado, gráficos y XML. Además se ajusta a las necesidades de implementación del proyecto.

JavaScript.

JavaScript es un lenguaje de programación utilizado en el mundo del desarrollo de páginas web dinámicas, por ser muy versátil y potente, tanto para la realización de pequeñas tareas como para la gestión de complejas aplicaciones.

JavaScript es un lenguaje de programación interpretado, multiplataforma, orientado a eventos con manejo de objetos, cuyo código se incluye directamente en el mismo documento, usado para el desarrollo de aplicaciones cliente-servidor en páginas HTML, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. (Pérez, 2009)

Se decide utilizar JavaScript para la creación de interfaces de usuarios más complejas e interactivas haciendo uso de las bibliotecas de ExtJS. Puede combinarse con otras herramientas de desarrollo web, como las hojas de estilo CSS. Además es un lenguaje fácil de aprender.

1.3.5. Framework.

Un *framework* es una estructura de soporte definida, compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación, en la cual otro proyecto de software puede ser organizado y desarrollado. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. Facilita el desarrollo de software y evita los detalles de bajo nivel, permitiendo concentrar más esfuerzo y tiempo en identificar los requerimientos del mismo. (Alegsa, 2010)

Symfony 2.0.

Symfony2 ha sido ideado para aprovechar todas las nuevas características de PHP 5.3. Permite utilizar la arquitectura MVC (*Model-View-Controller*).Fue diseñado para optimizar el desarrollo de aplicaciones Web, proporcionando herramientas para agilizar aplicaciones complejas y guiando al desarrollador a acostumbrarse al orden y buenas prácticas dentro del proyecto. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web (Potencier, 2010). Es compatible con la mayoría de los gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. (Potencier, 2010)

Se utiliza Symfony como framework pues implementa el patrón arquitectónico Modelo Vista Controlador, está desarrollado completamente con PHP 5.Presenta un código fácil de leer que incluye comentarios y que posibilita que el mantenimiento sea una tarea sencilla. Es fácil de instalar y configurar en la mayoría de las plataformas, además sigue las mejores prácticas de los patrones de diseño para la web. Symfony permite estructurar el sistema en módulos independientes y se enfoca en agilizar el proceso de construcción al mismo tiempo que reduce el trabajo del implementador. Además se selecciona este framework debido a que cumple con las políticas definidas para el proyecto en el documento de arquitectura, haciendo uso de las ventajas que ofrecen las tecnologías libres.

ExtJS 3.4.

ExtJS puede desarrollar aplicaciones web multiplataforma con facilidad. Se basa en el diseño de componentes, de forma tal que se puedan ampliar fácilmente los componentes predeterminados para satisfacer las necesidades del propio usuario y las extensiones se pueden encapsular en solo esos componentes. Permitiendo a los desarrolladores implementar aplicaciones grandes y complejas sin necesidad de consultar el código restante. Proporciona una enciclopédica-colección de widgets de interfaz de usuario con un tema de partida elegante. (ExtJS, 2011)

Se utiliza ExtJS pues brinda una mejor compatibilidad entre los navegadores en que se puede ejecutar la aplicación. Además facilita crear aplicaciones complejas utilizando componentes que ya se encuentran predefinidos. Una de las ventajas de utilizar ExtJS es la comunicación asíncrona en las aplicaciones donde el motor de *render* que se encarga de representar el contenido en el navegador puede comunicarse con el servidor sin necesidad de estar sujeta a un clic o una acción del usuario, dándole la libertad de cargar información sin que el cliente se percate del proceso. Así como la relación entre Cliente-Servidor balanceado es decir la carga de procesamiento se distribuye, permitiendo que el servidor, al tener menor carga, pueda manejar más clientes al mismo tiempo.

1.3.6. IDE de desarrollo.

Un entorno de desarrollo o IDE (acrónimo en inglés de Integrated Development Environment), es un programa informático compuesto por un conjunto de herramientas integradas para el desarrollo de software. Puede dedicarse a un solo lenguaje de programación o bien puede utilizarse para varios. Las herramientas se ejecutan a través de una interfaz de usuario y constan de un editor de código, un compilador, un depurador y un intérprete. (antonionl, 2013)

NetBeans 8.0.

NetBeans IDE es una aplicación de código abierto ("*open source*") diseñada para el desarrollo de aplicaciones portables entre las distintas plataformas, haciendo uso de la tecnología Java. Ofrece diferentes vistas de los datos, lo que le permite profundizar en ellos de forma rápida y sencilla. Dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web, control de versiones, colaboración entre varias personas y resaltados de sintaxis. (Wielenga, 2014)

Se utiliza NetBeans 8.0 ya que es multiplataforma, libre y gratuito, permite desarrollar aplicaciones en PHP y brinda completamiento de código. Es fácil de instalar y se puede ejecutar tanto en Windows, Linux, Mac. Además ofrece una amplia documentación, recursos de capacitación y tiene soporte para utilizar Symfony.

1.3.7. Sistema Gestor de Base de Datos (SGBD).

Un sistema de gestión de bases de datos (SGBD) es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos, además de proporcionar herramientas para añadir, borrar, modificar y analizar los datos. Tiene como objetivo facilitar al usuario las herramientas que le posibiliten administrar la información, sin tener en cuenta cómo se van guardando éstas en el disco duro. Estos sistemas manejan de forma clara la información de modo que su uso sea transparente, disminuyendo el tiempo de desarrollo y aumentando la calidad del sistema.

PostgreSQL9.1.

Para el desarrollo del presente trabajo se seleccionó como sistema gestor de base de datos PostgreSQL 9.1 por ser un potente sistema de gestión de base de datos objeto-relacional multiplataforma .Es adaptable a las necesidades del cliente y posee una variada documentación en diferentes idiomas debido fundamentalmente a sus numerosas comunidades de desarrollo. Presenta entre sus características significativas la atomicidad, consistencia, aislamiento y durabilidad de las operaciones tratadas. Soporta casi toda la sintaxis SQL, incluyendo subconsultas, transacciones, tipos y funciones

definidas. Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Además se selecciona este gestor debido a que cumple con las políticas definidas para el proyecto en el documento de arquitectura, haciendo uso de las ventajas que ofrecen las tecnologías libres

PGAdmin III versión 1.18.1herramienta de administración de la base de datos.

PGAdmin III es una aplicación de diseño y manejo de bases de datos para su uso con PostgreSQL. Es un software libre publicado bajo la licencia PostgreSQL. Se puede utilizar para gestionar PostgreSQL 7.3 hasta versiones superiores y funciona sobre casi todas las plataformas. Fue diseñado para responder a las necesidades de los usuarios, desde la escritura de consultas simples SQL hasta la elaboración de bases de datos complejas. La interfaz gráfica es compatible con todas las características de PostgreSQL y facilita su administración. Incluye un editor de la sintaxis SQL y un editor de código del lado del servidor. La conexión del servidor se puede realizar mediante TCP/IP o *Unix Domain Sockets* (en plataformas *nix), y puede ser cifrado mediante SSL para la seguridad. No se requieren controladores adicionales para comunicarse con la base de datos del servidor. (Mariano Reingart, 2013)

Una característica interesante de PGAdmin III es que cada vez que se realiza alguna modificación en un objeto escribe la(s) sentencia(s) SQL correspondiente(s), Lo que hace que sea una herramienta útil y didáctica. También incorpora funcionalidades para realizar consultas, examinar su ejecución (como el comando *explain*) y trabajar con los datos.

1.3.8. Servidor Web.

Un servidor Web es un programa que permite acceder a páginas Web contenidas en un ordenador, así como guardar los registros de acceso al sitio (logs) o páginas web contenidas en él. (Servidor de Aplicaciones, 2015).

Apache 2.4.

Es un servidor Web potente, flexible y disponible para distintas plataformas y entornos. Es una tecnología gratuita y de código abierto, lo que proporciona transparencia en todo el proceso de instalación. Es configurable y de diseño modular, permitiendo a los administradores de sistemas ajustar Apache para ser más rápido según las necesidades y la naturaleza de las peticiones que tengan que atender. Estos módulos pueden ser seleccionados en tiempo de ejecución con lo que añade una mayor flexibilidad. Y presenta un rendimiento superior al de los servidores orientados a eventos.

La versión que será utilizada en la solución del problema de la investigación es Apache 2.4 este servidor Web tiene una fácil integración con el lenguaje de programación PHP. Posibilita un menor uso de memoria y permite un soporte detallado del uso de caché. Además permite personalizar la respuesta ante los posibles errores que puedan ocurrir en el servidor.

1.4. Conclusiones

A partir del análisis de las soluciones existentes para el diseño de consultas se constató la necesidad de desarrollar la nueva versión del Módulo Diseñador de Consultas del GDR para lograr un sistema más completo, acorde con los estándares internacionales para este tipo de aplicación. Identificándose los principales elementos a tener en cuenta para el diseño e implementación de la solución propuesta.

A su vez después de analizar las políticas y normativas arquitectónicas del Departamento de Desarrollo de Componentes y las particularidades de la propuesta de solución se decide utilizar como metodología de desarrollo OpenUP, visual Paradigm 8.0como herramienta de modelado y como lenguaje de modelado UML 2.0. Durante la implementación de la solución se hará uso del IDE de desarrollo NetBeans en su versión 8.0, utilizando como lenguajes de programación PHP 5.3 del lado del servidor y del lado del cliente JavaScript, soportados por los marcos de trabajos Symfony 2.0 y ExtJS 3.4 respectivamente. Además se usará como gestor de base de datos PostgreSQL 9.1, para la administración de la base de datos PGAdmin III y como servidor web Apache 2.4.

CAPÍTULO II: ANÁLISIS Y DISEÑO DEL SISTEMA DEL DISEÑADOR DE CONSULTA.

En el presente capítulo se describe el proceso llevado a cabo durante el análisis y diseño. Se definen los requisitos funcionales y no funcionales del software. Se muestra la estructura del sistema a desarrollar a través de una representación visual de las clases conceptuales en el modelo de dominio. Se expone una representación gráfica de los procesos de la solución que se propone mediante el diagrama de casos de uso del sistema. Se muestran los diagramas de clases del diseño, por medio de los cuales se representa la estructura del sistema.

2.1. Modelo de Dominio.

El modelo de dominio es un artefacto de la fase de elaboración, se representa en UML con un diagrama de clases en el que se pueden mostrar conceptos u objetos del dominio del problema, asociaciones entre las clases conceptuales o atributos de las mismas. No contiene conceptos propios de un sistema de software sino de la propia realidad física. (Pressman, 2010)También se le denomina modelo conceptual, ya que ayuda a comprender los conceptos claves de un negocio o un dominio del problema. Es utilizado por el analista como un medio para comprender el negocio en el cual va a ser utilizado el sistema. (Ver figura 1).

2.1.1. Descripción de las clases del modelo del dominio:

Usuario: En este caso se comporta como el diseñador de consulta que interactúa con el sistema Generador Dinámico de Reportes específicamente con el módulo Diseñador de Consulta.

GDR: Entidad que representa al sistema Generador Dinámico de Reportes.

Diseñador de consulta: Representa uno de los módulos que contiene el sistema GDR, el mismo funciona como un editor gráfico de consultas SQL permitiendo crear, editar y ejecutar dichas consultas.

Vista de diseño: Vista perteneciente al módulo Diseñador de Consulta, que permite mostrar y diseñar de forma gráfica una consulta simple.

Vista SQL: Vista perteneciente al módulo Diseñador de Consulta, que permite mostrar y diseñar el código fuente de una consulta simple.

Consulta SQL: Representa al código SQL contenido en la vista SQL que será verificado por el intérprete.

UCQueryBuilder: Es la clase controladora principal del módulo, contiene las principales operaciones que se realizan del lado del cliente y es la encargada de actualizar el código de la consulta en la vista SQL así como actualizar la visa de diseño una vez que el código ha sido modificado.

Parser: Representa al parser encargado de verificar la estructura y sintaxis del código SQL contenido en la vista SQL.

ErrorReporter: Representa a la clase encargada de llevar el control de los errores durante el proceso de verificación de la consulta y notificarlos en caso de ser detectados.

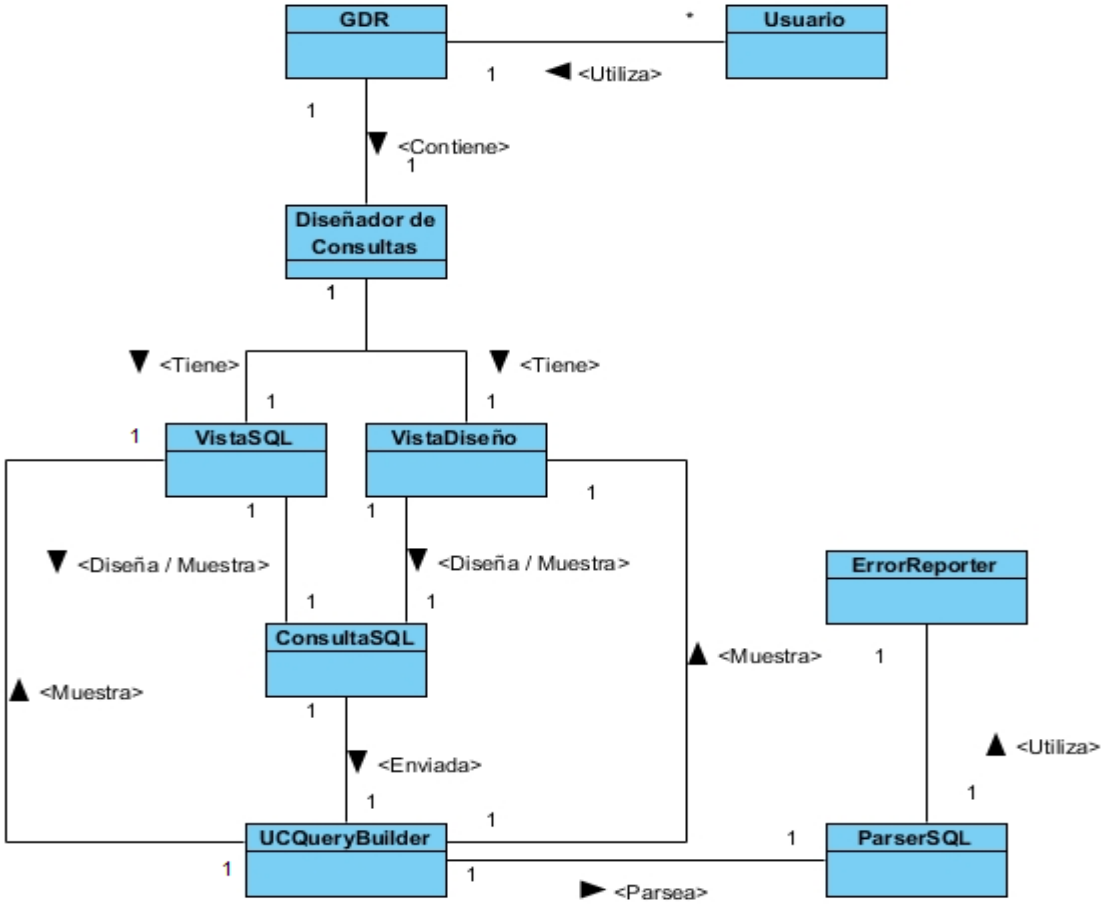


Fig.1 Modelo del Dominio.

2.2. Requisitos del sistema.

En la ingeniería del software, los requisitos se utilizan como datos de entrada en la etapa de diseño del producto y establecen qué debe hacer el sistema, pero no cómo hacerlo .Los requisitos de un sistema son los servicios proporcionados por el sistema y sus restricciones operativas. Estos requisitos reflejan las necesidades de los clientes, en la resolución de problemas tales como el control de un dispositivo o la gestión de la información. El proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones se denomina ingeniería de requisitos. Los requisitos de software se clasifican en funcionales y no funcionales. (Somerville, 2005)

2.2.1. Requisitos funcionales.

Los requisitos funcionales definen las funciones, capacidades o condiciones que el sistema debe ser capaz de realizar. Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. Básicamente establecen los comportamientos del sistema (Somerville, 2005). A continuación se describen los requisitos funcionales identificados para los módulos a implementar:

RF1: Crear consulta desde la vista SQL.
Descripción: El sistema permitirá crear la consulta en la vista SQL.
Entrada: Líneas de código SQL que conforman la consulta.
Salida: Consulta SQL creada en la vista SQL.

RF2: Crear consulta desde la vista diseño.
Descripción: El sistema permitirá crear la consulta en la vista de diseño.
Entrada: XML de la consulta.
Salida: Consulta creada en la vista de diseño.

RF3: Modificar consulta desde la vista de SQL.
Descripción: El sistema permitirá modificar desde la vista SQL la consulta que se está diseñando.
Entrada: Cambios en la estructura de la consulta SQL que se desea modificar.
Salida: Consulta SQL modificada en la vista correspondiente.

RF4: Modificar consulta desde la vista de diseño.
Descripción: El sistema permitirá modificar desde la vista de diseño la consulta que se está diseñando.
Entrada: Cambios en la estructura de la consulta SQL que se está diseñando.
Salida: Consulta SQL modificada en la vista de diseño.

RF5: Guardar la consulta desde la vista SQL.
Descripción: El sistema permitirá guardar la consulta que se está diseñando en la vista SQL.
Entrada: El nombre de la consulta.
Salida: Consulta guardada.

RF6: Guardar la consulta desde la vista de diseño.
Descripción: El sistema permitirá guardar la consulta que se está diseñando en la vista de diseño.
Entrada: El nombre de la consulta.
Salida: Consulta guardada.

RF7: Actualizar la vista de diseño.
Descripción: El sistema verificará que la sentencia de la consulta haya sido modificada y validará que los datos de la misma sean correctos, posteriormente actualizará en la vista de diseño la consulta modificada.
Entrada: Consulta SQL en formato XML.
Salida: Diseño actualizado en la vista correspondiente.

RF8: Actualizar la vista SQL.
Descripción: El sistema captura el XML correspondiente a la consulta en diseño, lo transforma a código SQL actualizando dicho código contenido en la vista SQL.
Entrada: Consulta SQL en formato XML.
Salida: Diseño actualizado en la Vista SQL.

RF9: Realizar análisis léxico de la consulta SQL.
Descripción: El sistema verifica que todos los caracteres presentes en la consulta SQL estén correctamente escritos según la composición o la estructura del lenguaje.
Entrada: Consulta SQL.
Salida: Una variable que contiene la información que permite determinar si en el análisis se encontraron o no errores y además una lista de tokens que será utilizada en el análisis sintáctico.

RF10: Realizar análisis sintáctico de la consulta SQL.
Descripción: El sistema verifica que los tokens presentes en la consulta SQL estén escritos de forma tal que se corresponda con alguna de las sentencias válidas de acuerdo a la gramática definida para el lenguaje.
Entrada: Lista de tokens generada en el análisis léxico.
Salida: Una variable que contiene la información que permite determinar si en el análisis se encontraron o no errores, además del registro de todos los campos y tablas que componen la consulta que permitirá posteriormente construir el XML.

RF11: Construir el XML de la consulta SQL.
Descripción: A partir del resultado de realizar el análisis sintáctico el sistema construye el XML que será utilizado para actualizar la vista.
Entrada: Registro de todos los campos y todas las tablas que componen la consulta.
Salida: XML de la consulta.

RF12: Ejecutar la consulta SQL.
Descripción: El sistema ejecuta la consulta escrita en el gestor correspondiente.
Entrada: El código de la consulta escrita en la vista SQL.
Salida: Una tabla con el resultado de la ejecución.

RF13: Renombrar la consulta.
Descripción: El sistema permite cambiar el nombre de la consulta seleccionada.
Entrada: Nombre de la consulta.
Salida: Consulta renombrada.

RF14: Eliminar la consulta.
Descripción: El sistema permite eliminar la consulta seleccionada.
Entrada: Nombre de la consulta.
Salida: Consulta eliminada.

RF15: Listar las consultas.
Descripción: El sistema muestra un listado de las consultas existentes en la base de datos.
Entrada: Id de modelo.
Salida: Listado de las consultas existentes en la base de datos.

RF16: Arrastrar y soltar las tablas al área de trabajo.
Descripción: El sistema muestra la tabla en el área de trabajo con los campos seleccionados.
Entrada: Nombre de la tabla.
Salida: Tabla mostrada en el área de trabajo.

RF17: Actualizar el explorador de consultas.
Descripción: El sistema muestra en el explorador de consultas, las estructuras y los campos que componen la consulta en diseño.
Entrada: XML de la consulta en diseño.
Salida: Explorador de consultas actualizado.

RF18: Abrir consulta.
Descripción: El sistema abre la consulta seleccionada y la muestra en el área de trabajo.
Entrada: Nombre de la consulta.
Salida: Consulta representada en el área de trabajo.

2.2.2. Requisitos no funcionales.

Los requisitos no funcionales surgen con las necesidades del usuario y definen las restricciones y propiedades del sistema. Estos requisitos como su nombre lo sugiere, son aquellos requisitos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de este como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De manera general estos tipos de requisitos se encuentran estrechamente vinculados a los funcionales, atendiendo a que una vez que se conoce lo que el sistema debe hacer, es preciso determinar las cualidades que debe cumplir y cuán rápido debe ser.(Somerville, 2005) Los requisitos no funcionales que debe cumplir el sistema para garantizar su correcto funcionamiento son los siguientes:

RNF 1. Usabilidad.

- El sistema podrá ser utilizado por cualquier persona que posea conocimientos básicos en el manejo de la computadora.
- El usuario debe poder diseñar una consulta SQL de manera ágil y sencilla, sin ser un experto en conocimientos de base de datos.

RNF 2. Interfaz.

- Las interfaces de usuario serán diseñadas a modo de Aplicaciones Enriquecidas de Internet(RIA por sus siglas en inglés), lo que permite a los usuarios contar con aplicaciones web con una experiencia de usuario similar a la de las aplicaciones de escritorio.
- El sistema deberá contar de manera general, con una interfaz sencilla, que sea intuitiva y de fácil comprensión garantizando la correcta interacción con el sistema.
- El sistema debe tener indicadores que permitan al usuario conocer las acciones que debe realizar, por ejemplo botones con íconos sugerentes y alternativa textual.
- El sistema debe permitir al usuario transitar de una tarea a otra sin necesidad de obligarlo a realizar acciones no deseadas, facilitándole la realización de las mismas con no más de tres clic.

RNF 3. Restricciones de Diseño e Implementación.

- El sistema deberá ser implementado en el lenguaje de programación PHP versión 5.3.
- Se utilizará el framework de desarrollo Symfony en su versión 2.0 y la librería de JavaScript Ext-JS versión 3.4.
- Se empleará la herramienta de desarrollo NetBeans 8.0.

RNF 4. Software.

El servidor donde se instalará la aplicación debe cumplir con los siguientes requisitos:

- Sistema Operativo: GNU/Linux preferentemente Ubuntu GNU/Linux 14.04, Debian 4.0 GNU/Linux.
- Paquetes:
  - ✓ **Comunes**: apache2, php5, libapache2-modphp5, php5-cli, php5-xsl, php5-gd.
  - ✓ **PostgreSQL:** php5-pgsql.
  - ✓ **SQLite:** php5-sqlite.
  - ✓ **MySQL:** php5-MySQL.

El servidor donde se instalará la base de datos debe cumplir con los siguientes requisitos:

- Sistema Operativo: GNU/Linux preferentemente Ubuntu GNU/Linux 14.04, Debian4.
- PostgreSQL versión 8.4.
- PGAdmin III o algún administrador para PostgreSQL
- PHPMyAdmin o algún administrador de MySQL.

La PC utilizada para acceder a la aplicación debe cumplir con los siguientes requisitos:

- Sistema Operativo: GNU/Linux preferentemente Ubuntu GNU/Linux 14.04, Debian 4 GNU/Linux, Microsoft Windows XP.
- Navegador Web Mozilla Firefox preferentemente en su versión 35.0.

RNF 5. Hardware.

El servidor donde se instalará la aplicación debe cumplir con los siguientes requisitos mínimos:

- Procesador Intel Pentium 4 a 1.7 GHz o AMD similar.
- 512 MB de RAM.
- 40 GB de espacio en disco duro.

El servidor donde se instalará la base de datos debe cumplir con los siguientes requisitos mínimos:

- Procesador Intel Pentium 4 a 1.7 GHz o AMD similar.
- 512 MB de RAM.
- 40 GB de espacio en disco duro.

La PC utilizada para acceder a la aplicación debe cumplir con los siguientes requisitos mínimos:

- Procesador Intel Pentium 4 a 1.7 GHz, o AMD similar.
- 256 MB RAM.
- 20 GB de espacio en disco duro.

2.3. Modelo de casos de uso del sistema.

Un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software (Presman, 2010). Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. En ocasiones, se utilizan usuarios sin experiencia junto a los analistas para el desarrollo de casos de uso.

Un caso de uso es una secuencia de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico.

2.3.1. Diagrama de casos de uso del sistema.

Los diagramas de casos de uso son el punto de partida en todo proyecto de desarrollo de software basado en UML y constituyen una representación gráfica de los procesos y su interacción con los actores. Sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/o otros sistemas. Constituye una entrada para el análisis, el diseño y las pruebas. (Mastermagazine, 2012)

En la figura 2 se muestra el diagrama de casos de uso del sistema propuesto para el desarrollo del sistema. En el que se incluyen todos los casos de uso necesarios para satisfacer los requisitos funcionales especificados.

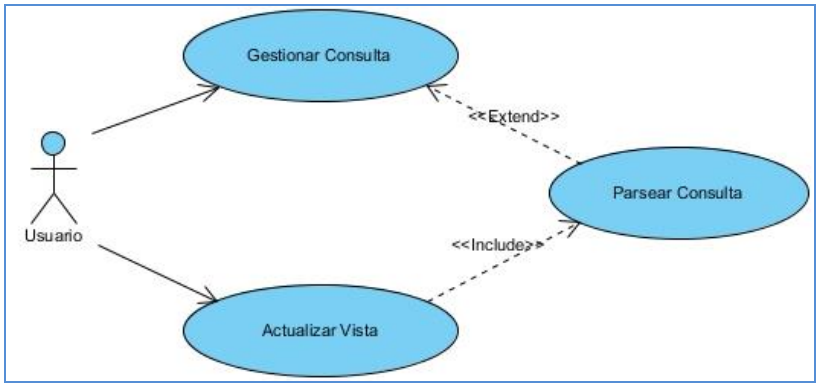


Fig.2 Diagrama de Casos de Uso del Módulo Diseñador de Consultas.

Patrones de Caso de Uso.

Entre los diferentes patrones de caso de uso se encuentra el **CRUD Completo**, que propone formar un caso de uso a partir de los requisitos funcionales relacionados con las acciones de crear, actualizar,

guardar, ejecutar, renombrar, abrir, eliminar y modificar una consulta SQL. Haciendo uso de este se conforma el caso de uso "Gestionar Consulta". Ver figura 2.

Inclusión concreta: En términos muy simples, cuando relacionamos dos casos de uso con un "include", estamos diciendo que el primero (el caso de uso base) incluye al segundo (el caso de uso incluido). Es decir, el segundo es parte esencial del primero por lo que es necesario que ocurra el caso incluido para satisfacer el objetivo del caso de uso base. Haciendo uso de este patrón se propone el caso de uso "Actualizar Vista" como caso base y el "Parsear Consulta" como el caso incluido. Ver figura 2.

Extensión concreta: En ciertos escenarios el caso de uso base no podría cumplir su objetivo si no se ejecutara la extensión. Pero una de las diferencias básicas es que en el caso del "*extend*" hay situaciones en que el caso de uso de extensión no es indispensable que ocurra, y cuando lo hace ofrece un valor extra (extiende) al objetivo original del caso de uso base. Haciendo uso de este patrón se propone el caso de uso "Gestionar Consultas" como caso base y el caso de uso "Parsear Consulta" como el caso extendido. Ver figura 2.

En la tabla 2 se muestran los casos de uso identificados, así como los requisitos funcionales que cada uno de estos agrupan.

Tabla 2: Relación entre los casos de uso y los requisitos funcionales.

Casos de Uso	Referencia Requisitos Funcionales
Gestionar consulta	RF1, RF2, RF3, RF4, RF5, RF6, RF12, RF13, RF14, RF15, RF16,RF18
Parsear consulta	RF9, RF10, RF11
Actualizar vista	RF7, RF8, RF17

2.3.2. Descripción de los casos de uso:

Gestionar consulta SQL: Permite crear, guardar, ejecutar, renombrar, abrir, eliminar, listar consultas, arrastrar las tablas y modificar una consulta.
Parsear consulta: Permite verificar que la consulta esté correctamente escrita en correspondencia con el lenguaje SQL del gestor que se esté utilizando.
Actualizar vista: Permite actualizar la vista a partir de la consulta descrita en la vista SQL.

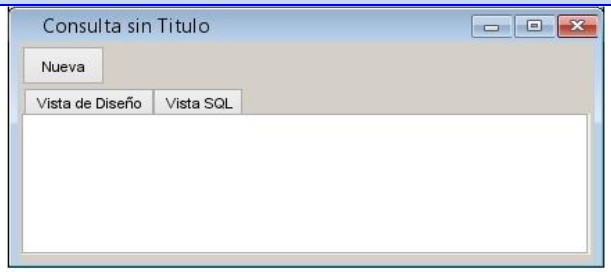
CU: Gestionar Consulta SQL.

Tabla 3: Descripción del caso de uso Gestionar consulta SQL.

Nombre	**Gestionar consulta SQL.**
Objetivo	Este caso de uso tiene como objetivo crear, salvar, ejecutar, renombrar, abrir, eliminar, listar consultas, arrastrar las tablas y modificar una consulta en la vista SQL.
Actores	Usuario
Resumen	El CU se inicia cuando el usuario selecciona el modelo a diseñar, el Diseñador de Consultas brinda la posibilidad de crear, actualizar, guardar, ejecutar, renombrar, abrir, eliminar o modificar la consulta SQL desde la vista deseada y termina luego de actualizarse la consulta en la vista de diseño.
Complejidad	Alta
Prioridad	Crítico
Precondiciones	Que el usuario esté autenticado en el sistema y que haya accedido al módulo Diseñador de Consultas.
Poscondiciones	Según la acción que realice el usuario: ➢ Se crea una nueva consulta. ➢ Se modifica una consulta existente. ➢ Se salvan los cambios en una consulta. ➢ Se ejecuta una consulta. ➢ Se renombra una consulta. ➢ Se elimina una consulta. ➢ Se abre una consulta. ➢ Se listan consultas. ➢ Se arrastran las tablas al área de trabajo.
Flujo de eventos	
Flujo básico < Gestionar consulta SQL>	

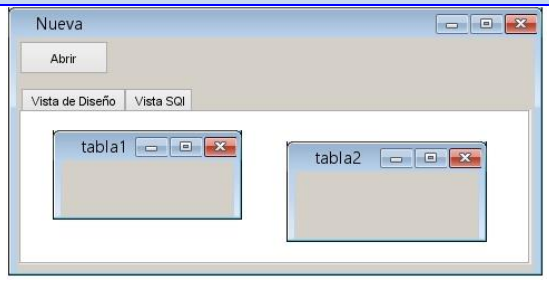
	Actor	**Sistema**
1.	El usuario accede al módulo Diseñador de Consultas y selecciona cuál acción desea realizar: a) Crear consulta: b) Modificar consulta: c) Salvar la consulta: d) Ejecutar la consulta:	El sistema, en dependencia de la acción solicitada por el usuario, muestra la interfaz correspondiente: **a.** Crear consulta: **Ver sección "Crear consulta".** **b.** Modificar consulta: **Ver sección "Modificar consulta".**

	e) Renombrar la consulta: f) Eliminar la consulta: g) Abrir la consulta. h) Listar consultas. i) Arrastrar las tablas al área de trabajo.	**c.** Salvar la consulta: **Ver sección "Salvar la consulta".** **d.** Ejecutar la consulta: **Ver sección "Ejecutar la consulta".** **e.** Renombrar la consulta: **Ver sección "Renombrar la consulta".** **f.** Eliminar la consulta: **Ver sección "Eliminar la consulta".** **g.** Abrir consulta: **Ver sección "Abrir la consulta".** **h.** Listar consultas: **Ver sección "Listar consultas".** **i.** Arrastrar las tablas al área de trabajo: **Ver sección "Arrastrar las tablas al área de trabajo".** **Terminando así el caso de uso.**
Sección a: "Crear nueva consulta"		
Flujo básico < Crear nueva consulta >		
	Actor	**Sistema**
		1. El sistema muestra una interfaz con dos vistas la vista SQL y la vista de diseño para que el usuario seleccione donde desea diseñar la consulta.
	2. El usuario selecciona la vista en la que desea crear la consulta.	
	3. Si el usuario selecciona la vista SQL.	4. El sistema muestra un área de trabajo para que el usuario introduzca la sintaxis de la consulta SQL. El sistema brinda la opción de guardar la consulta **Ver sección: "Salvar la consulta"**. Terminando así el caso de uso.

	5. Si el usuario selecciona la vista de diseño.	**6.** El sistema muestra un área de trabajo para que el usuario introduzca las tablas y las relaciones de manera visual. El sistema brinda la opción de guardar la consulta **Ver sección: "Guardar la consulta"**. Terminando así el caso de uso.
Prototipo de interfaz		
		
Sección b: "Modificar consulta"		
Flujo básico <Modificar consulta>		
	Actor	**Sistema**
	1. El usuario solicita las consultas que han sido creadas mediante la opción ´´Abrir´´.	2. El sistema muestra una interfaz con todos los modelos existentes en la base de datos mediante un árbol para que el usuario seleccione la que desea modificar.
	3. El usuario selecciona el modelo del que desea obtener la consulta.	3. El sistema carga las consultas pertenecientes al modelo seleccionado y las muestra en el árbol.
	4. El usuario selecciona del listado de consultas la que desea modificar y hace clic en la opción ´´Aceptar´´.	5. El sistema carga la consulta y la muestra en el área de trabajo de la vista de diseño.
Prototipo de interfaz		

	5 El usuario modifica los datos que desea actualizar en cualquiera de las vistas: vista de diseño o vista SQL.	6. El sistema muestra los cambios realizados en la consulta.
	7. El usuario selecciona la opción guardar: **Ver sección3 ´´Salvar la consulta´´.**	**8.** El sistema salva los cambios realizados en la consulta. Terminando así el caso de uso. El sistema valida el nombre introducido y si existe error "Ver flujo alterno error al salvar", sino guarda los datos modificados en la consulta. Terminando así el caso de uso.

Prototipo de interfaz



Flujos alternos

Nº Evento <Cancelar Modificación>

	Actor	Sistema
	8.1 Si el usuario no desea guardar los cambios, selecciona el botón "Nueva".	El sistema muestra un mensaje ´´El diseño ha sido modificado. ¿Qué desea hacer?, brindándole al usuario las opciones "Guardar", "Descartar cambios"

		y "Cancelar".
	El usuario selecciona la opción "Descartar cambios".	El sistema cancela la modificación y reinicia el área de trabajo para que el usuario pueda diseñar o modificar otra consulta. Terminando así el caso de uso.
Sección c: "Salvar consulta"		
Flujo básico <Salvar consulta >		
Precondiciones	Que exista una nueva consulta en cualquiera de las dos vistas.	
	Actor	**Sistema**
	1. El usuario selecciona la opción ´´Salvar´´.	2. El sistema muestra una nueva ventana que contiene un formulario para introducir el nombre de la consulta.
	3. El usuario introduce el nombre de la consulta y selecciona la opción ´´Salvar´´.	4. El sistema captura la consulta creada y la guarda, luego muestra un mensaje de confirmación "La operación se realizó con éxito" si el proceso se realizó correctamente. En caso de ocurrir algún error ver Flujo Alterno **"Errores al salvar la consulta."** Terminando así el caso de uso.
Prototipo de interfaz		
		
Flujos alternos		
Nº Evento < Errores al salvar la consulta >		
Actor		**Sistema**

	El sistema muestra un mensaje "Error al procesar la consulta" informando que han ocurrido errores. Terminando así el caso de uso.

Prototipo de interfaz

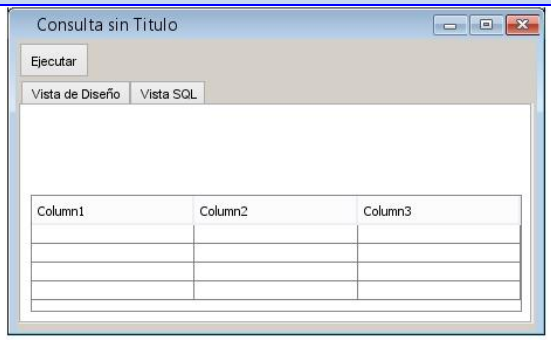
Sección d: "Ejecutar consulta"

Flujo básico <Ejecutar consulta>

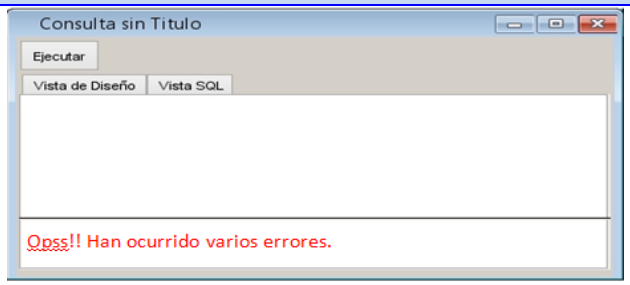
Precondiciones	Que exista una consulta en cualquiera de las dos vistas.

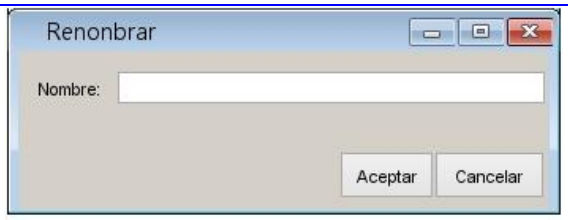
	Actor	**Sistema**
	1 El usuario selecciona la opción ejecutar.	2 El sistema captura la consulta creada y la envía al gestor de bases de datos para ser ejecutada, luego cambia a la vista SQL y muestra los resultados. En caso de ocurrir algún error ver Flujo Alterno **"Errores al ejecutar la consulta.".** Terminando así el caso de uso.

Prototipo de interfaz



Flujos alternos

Nº Evento < Errores al ejecutar la consulta >		
	Actor	**Sistema**
		El sistema muestra un mensaje ´´La consulta es incorrecta´´ y debajo muestra los errores de la misma. Terminado así el caso de uso.
Prototipo de interfaz		
		
Sección e: "Renombrar consulta"		
Flujo básico <Renombrar consulta>		
Precondiciones	Que la consulta esté guardada en la base de datos.	
	Actor	**Sistema**
	1 El usuario hace clic derecho encima de la consulta y selecciona la opción "Renombrar".	2 El sistema muestra una nueva ventana que contiene un formulario para introducir el nuevo nombre de la consulta.
	3 El usuario introduce el nombre y presiona el botón Aceptar.	4 El sistema comprueba que el nombre no esté repetido .En caso que esté repetido envía un mensaje ¨ Ya existe una consulta con ese nombre ¨.En caso contrario renombra la consulta mostrando un mensaje ¨La operación se realizó con éxito¨´. Terminando así el caso de uso.
Prototipo de interfaz		



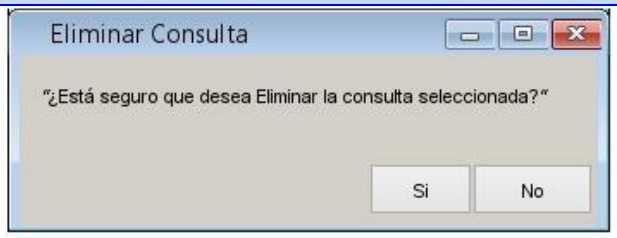
Sección f: "Eliminar consulta"

Flujo básico <Eliminar consulta>

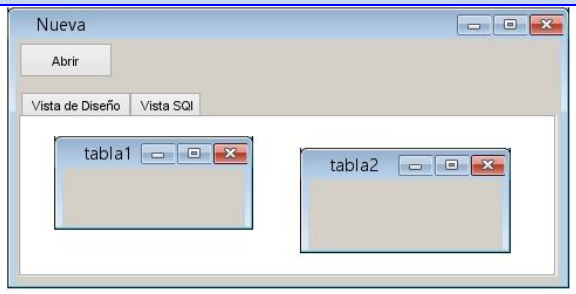
Precondiciones	Que la consulta esté guardada en la base de datos.

	Actor	**Sistema**
	1 El usuario selecciona la consulta que desea eliminar del listado de consultas y presiona el botón "Eliminar consulta".	2 El sistema muestra el mensaje de advertencia "¿Está seguro que desea Eliminar la consulta seleccionada", permitiendo: • **Si** donde se procederá a la eliminación de la consulta seleccionada. Terminando así el caso de uso. • **No.** (Ver **Flujo Alterno: "Cancelar operación").**

Prototipo de interfaz



	3 El usuario presiona el botón "Si".	4 El sistema elimina la consulta del sistema, mostrando el mensaje de información "Se eliminó correctamente la consulta seleccionada". En caso de ocurrir algún error ver Flujo Alterno "**Errores al eliminar la consulta**". Terminando así el caso de uso.
Flujos alternos		
Nº Evento <Cancelar operación>		
	Actor	**Sistema**
	El usuario presiona el botón "Cancelar".	El sistema regresa a la pantalla anterior. Terminando de esta forma el caso de uso.
Flujos alternos		
Nº Evento < Errores al eliminar la consulta >		
	Actor	**Sistema**
		El sistema muestra el mensaje ´´Error al ejecutar la acción´´. Terminado así el caso de uso.
Prototipo de interfaz		
		
Sección g: "Abrir consulta"		
Flujo básico <Abrir consulta>		
Precondiciones	Que exista al menos una consulta guardada en la base de datos.	
	Actor	**Sistema**
	1. El usuario selecciona la opción "Abrir".	2. El sistema muestra todas las consultas existentes en la base de datos mediante un árbol para que el

		usuario seleccione la que desea modificar.
	3. El usuario selecciona el modelo del que desea obtener la consulta.	4. El sistema carga las consultas pertenecientes al modelo seleccionado y las muestra en el árbol.
	5. El usuario selecciona la consulta deseada y hace clic en el botón ´´Abrir´´.	6. El sistema carga la consulta seleccionada y la muestra en el área de trabajo.
Prototipo de interfaz		
		
Sección h: "Listar consultas"		
Flujo básico < Listar consultas>		
Precondiciones	Que exista al menos una consulta guardada en la base de datos.	
	Actor	**Sistema**
	1. El usuario despliega en el explorador de modelos el modelo sobre el que desea trabajar.	2. El sistema muestra las consultas y los esquemas que componen el modelo.
	3. El usuario despliega el nodo "Consultas".	4. El sistema carga las consultas guardadas pertenecientes al modelo desplegado y las muestra en el árbol.
Prototipo de interfaz		

Sección i: "Arrastrar tablas"

Flujo básico < Arrastrar tablas>

Precondiciones	Que exista al menos una consulta guardada en la base de datos.

	Actor	**Sistema**
	1. El usuario selecciona y despliega en el explorador de modelos el modelo sobre el que desea trabajar.	2. El sistema muestra las consultas y los esquemas que componen el modelo.
	3. El usuario despliega el nodo "Esquemas".	4. El sistema carga los esquemas pertenecientes al modelo seleccionado.
	5. El usuario despliega el esquema sobe el que desea trabajar.	6. El sistema muestra las tablas pertenecientes al esquema seleccionado.
	7. El usuario hace clic sobe la tabla que desea, la arrastra hacia el área de trabajo y la suelta.	8. El sistema representa la tabla en el área de trabajo. Terminando así el caso de uso.

Relaciones	**CU Incluidos**	Actualizar Vista
	CU Extendidos	Parsear Consulta
Requisitos no funcionales		

2.4. Elementos arquitectónicos de software.

Una arquitectura de software, también denominada arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema. La arquitectura de software establece los fundamentos para que analistas, diseñadores y programadores trabajen en una línea común que permita alcanzar los objetivos del sistema. Se selecciona y diseña con base en objetivos y restricciones. Los objetivos son aquellos prefijados para el sistema, pero no solamente los de tipo funcional, también otros objetivos como la mantenibilidad, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para la implementación del sistema. (Reynoso, 2009)

En la definición de la arquitectura del módulo se utiliza el siguiente estilo y patrón arquitectónico.

2.4.1. Estilo arquitectónico utilizado.

Cliente-Servidor.

El estilo de arquitectura cliente-servidor es un modelo para el desarrollo de sistemas de información en el que las operaciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos. Se denomina cliente al proceso que inicia el diálogo o solicita los recursos y servidor al proceso que responde a las solicitudes. En este modelo las aplicaciones se dividen de forma que el servidor contiene la parte que debe ser compartida por varios usuarios, y en el cliente permanece sólo lo particular de cada usuario. (Manuel, 2004)

En el modelo cliente servidor, el cliente envía un mensaje solicitando un determinado servicio a un servidor (hace una petición), y este envía uno o varios mensajes con la respuesta (provee el servicio) (Ver Figura 3).

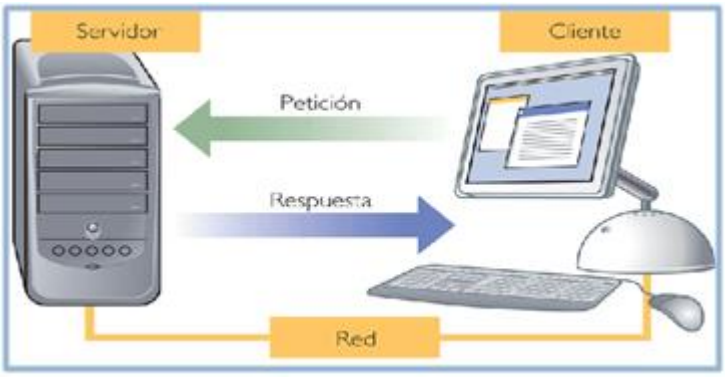


Fig.3 Arquitectura cliente-servidor. **(payolis and nirvana john, 2010)**

Una vez realizado el análisis al estilo arquitectónico cliente – servidor se decidió utilizar el mismo teniendo en cuenta que GDR estará ubicado en un servidor central, el cual se encargará de dar respuesta a las peticiones realizadas por los clientes y el objetivo del presente trabajo de investigación es desarrollar un módulo que forma parte de este sistema. Como es el caso del módulo Diseñador de Consultas el cual adopta este estilo por las ventajas mencionadas.

2.4.2. Patrón arquitectónico utilizado

Modelo Vista Controlador (MVC).

La selección de patrón arquitectónico MVC se basa en la utilización del marco de trabajo Symfony2 para el desarrollo de la solución, a continuación se explica de manera breve en qué consiste dicho patrón.

Modelo Vista Controlador (MVC): es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de Base de Datos y la lógica de negocio; y el controlador es el responsable de recibir los eventos de entrada desde la vista. (Leon Welicki, 2009)

- **Modelo**: Este es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos. Es el responsable de acceder a la capa de almacenamiento de datos.
- **Vista**: Este transforma el modelo en una interfaz visual que permite al usuario interactuar con ella.
- **Controlador**: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. (Larman, 1999)

El MVC el controlador es el encargado de redirigir un procesamiento determinado para cada petición que reciba. Por tanto, el controlador debe poseer algún modo de conocer la correspondencia entre peticiones y posibles respuestas; deberá tener un mapa de peticiones y respuestas. El modelo representa la aplicación que responde una petición: los datos y las reglas de negocio. Por último, la vista define la forma de mostrar la información al usuario. En el caso del estilo arquitectónico MVC, el procesamiento se lleva a cabo entre sus tres componentes. El controlador recibe una petición y decide quién la lleva a cabo en la capa del modelo. Una vez que el modelo termina las operaciones pertinentes, devuelve el control de ejecución al controlador, y éste envía los resultados a la capa de la vista para que muestre los resultados al usuario como se representa en la figura 4. Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos, en el MVC corresponde al modelo.

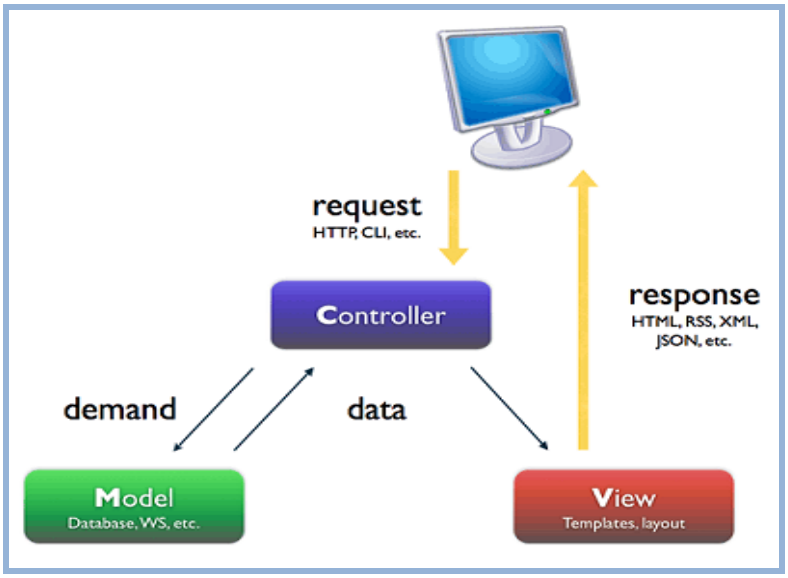


Fig.4 Representación del Patrón Arquitectónico Modelo Vista Controlador. **(Zaman Riaz, 2014)**

2.5. Patrones de diseño utilizados en la solución.

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces, para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haberse comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. (Prieto, 2009)

Los patrones de diseño se encuentran agrupados en dos grandes grupos fundamentalmente, estos son:

- Patrones GRASP (*General Responsibility Assignment Software Patterns*).
- Patrones GoF (*Gang of four*).

2.5.1. Patrones GRASP.

Los patrones GRASP son usados para asignar responsabilidades dentro de la misma solución, de forma tal, que cada objeto conozca el rol que le corresponde. (Larman, 1999)

Experto: Experto en información es el principio básico de asignación de responsabilidades. Indica que la creación de un objeto debe recaer sobre la clase que conoce toda la información necesaria para crearlo. En la figura 5 se muestra como la clase PostgresModuleParser es la encargada de crear los objetos de las clases Parser y Reporter.

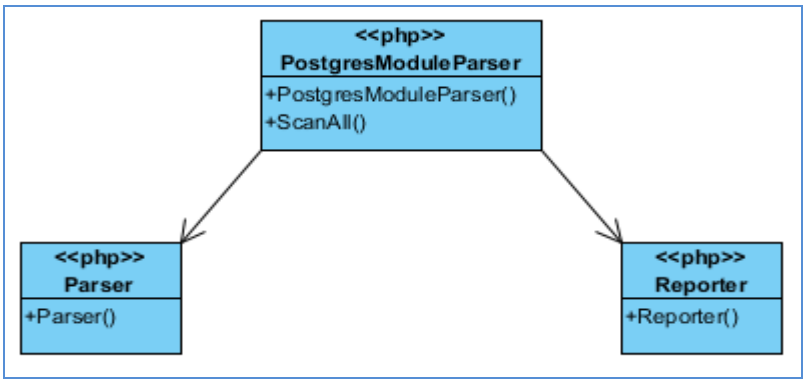


Fig.5 Fragmento del diagrama de clases del diseño donde se evidencia la utilización del patrón GRASP: Experto.

Creador: Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos. Propone asignar a una clase la responsabilidad de crear los objetos de la otra en los casos de contener, agregar, registrar o utilizar a esta (Hern, 2002).El propósito fundamental de este patrón es encontrar un creador que permita conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. En la figura 6 se muestra como la claseParser es la encargada de crear los objetos que representan los componentes de una consulta con el objetivo de poder usar estos más adelante en la creación del XML.

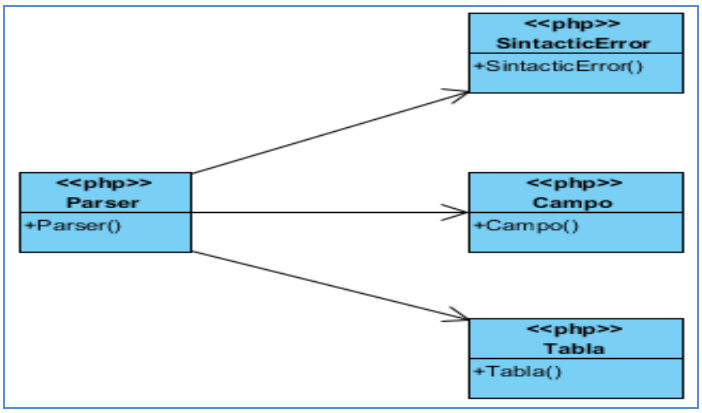


Fig.6 Fragmento del diagrama de clases del diseño donde se evidencia la utilización del patrón GRASP: Creador.

Controlador: Asigna la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase. Este patrón establece el uso de una clase controladora, que funciona como intermediaria entre cada una de las clases interfaces y el algoritmo que la implementa, de forma tal que es la que recibe los datos del usuario y manda a ejecutar las acciones correspondientes. En la figura 7 se muestra como la clase QueryController es la encargada de capturar la información proveniente de la vista, dicha clase identifica el gestor al que pertenece el código de la consulta y manda a ejecutarse el parser del lenguaje correspondiente. Además recibe el XML resultante y lo envía a la vista.

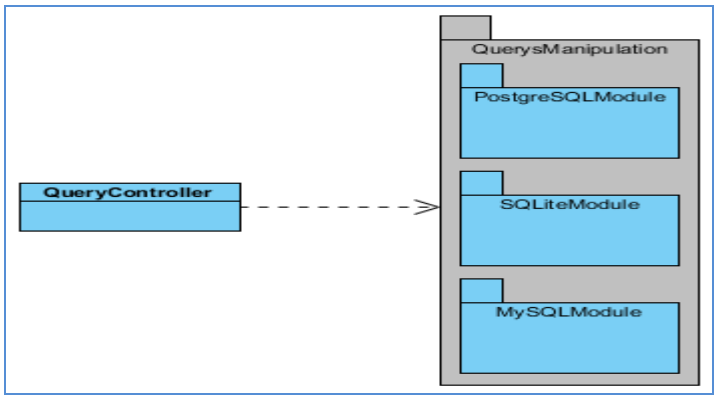


Fig.7 Fragmento del diagrama de clases del diseño donde se evidencia la utilización del patrón GRASP: Controlador.

Alta Cohesión Es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión garantiza que las clases con responsabilidades estrechamente relacionadas no realicen un trabajo excesivo. Además de asignar las responsabilidades de forma tal que se garantice la existencia de la alta cohesión entre las clases. En el sistema se muestra como la clase PostgresModuleParser es la que contiene toda la información necesaria para realizar el análisis sintáctico para posteriormente generar el XML de la consulta, por tanto es la encargada de repartir el trabajo entre las clases que componen el módulo.

Bajo acoplamiento Este patrón guía la asignación de responsabilidades de forma tal que las clases se comuniquen con el menor número de clases que sea posible, de manera que de producirse una modificación en algunas de ellas se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización, y disminuyendo la dependencia entre las mismas. En el sistema se muestra cómo la clase Lexer es la que garantiza la realización del análisis léxico relacionándose únicamente con las clases que son indispensables para realizar este proceso, ejemplo de estas clases son *lexical error, tokenKind ,keyWordList ,simbolTable y tokens*.

Polimorfismo

"*Siempre que se tenga que llevar a cabo una responsabilidad que dependa del tipo, se tiene que hacer uso del polimorfismo, cuando las alternativas o comportamientos relacionados varían según el tipo (clase), asigne la responsabilidad para el comportamiento-utilizando operaciones polimórficas a los tipos para los que varía el comportamiento. Asigna el mismo nombre a servicios en diferentes objetos*". (Larman, 1999)El uso de este patrón se aprecia en la clase **Condición** de la cual heredan las demás condiciones como se muestra en la figura 8 las cuales reimplementan sus métodos y agregan métodos propios.

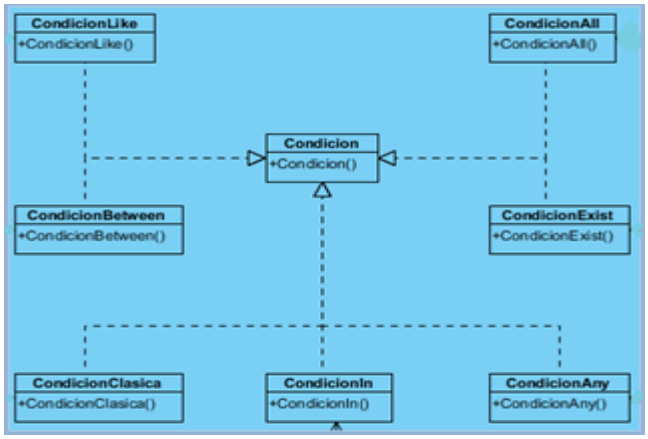


Fig.8 Fragmento del diagrama de clases del diseño donde se evidencia la utilización del patrón GRASP: Polimorfismo.

2.5.2. Patrones GoF.

Los patrones GoF de la banda de los cuatro están considerados como el fundamento de todos los patrones. Ellos los clasifican en tres grupos: creacional, estructural y de comportamiento. (Desing Patterns. E. Gamma, 1995.)

- **Patrones de creación**: Manifiestan cómo crear objetos cuando se necesitan decisiones; estas se realizan dinámicamente conociendo qué clases instanciar.
- **Patrones estructurales**: Determina la organización de diferentes tipos de objetos para trabajar con estos.
- **Patrones de comportamiento**: Son utilizados para organizar y administrar comportamientos, es decir, las relaciones entre los objetos del sistema.

***Facade* (Fachada):** Simplifica el acceso a un conjunto de clases proporcionando una única clase que todos utilizan para comunicarse con dicho conjunto de clases. Se evidencia en la clase QueryController la cual se comporta como controladora. Todas las peticiones vienen a ella, reparte la responsabilidad, obtiene los resultados y devuelve la respuesta en formato Json. Ver figura 9.

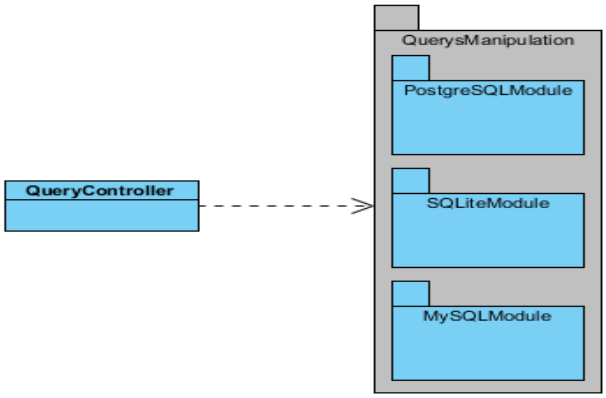


Fig.9 Fragmento del diagrama de clases del diseño donde se evidencia la utilización del patrón GOF: Facade (Fachada).

***Factory Method* (Método de fabricación):** Separa la clase que crea los objetos, de la jerarquía de objetos a instanciar. Además facilita la escalabilidad del sistema. Se evidencia en la clase Campo la cual tiene en su implementación un método que se redefine así mismo por tanto puede ser llamado desde otra clase de varias formas utilizando el mismo nombre del método pero pasándole diferentes parámetros. Ver figura 10.

```
/**
 * Constructor de la clase Campo
 * @param type $Nombre
 * @param type $Tabla
 * @param type $Alias
 * @param type $Operation
 */
public function __construct($Nombre, $Tabla = null, $Alias = null, $Operation = null, $type = null) {
    $this->Nombre = $Nombre;
    $this->Tabla = $Tabla;
    $this->Alias = $Alias;
    $this->Operation = $Operation;
    $this->type = $type;
}
```

Fig.10 Fragmento de código donde se evidencia la utilización del patrón GOF: Factory Method (Método de Fabricación).

***Composite* (Objeto compuesto):** Compone objetos en jerarquías todo-parte y permite tratar objetos simples y compuestos de manera uniforme. Además permite tratamiento uniforme de objetos simples y complejos así como composiciones recursivas. Se evidencia en la clase Operador Join o Condición Clásica porque un objeto de esta clase en su composición comprende objetos de otras clases que lo conforman. Ver figura 11.

```php
/**
 * Constructor de la clase CondicionClasica
 * @param type $campoIzq
 * @param type $Op
 * @param type $campoDer
 * @param type $condNot
 * @param type $opNeg
 */
function __construct($campoIzq, $Op, $campoDer, $condNot = null, $opNeg = null) {
    parent::__construct($condNot, $opNeg);
    $this->campoIzq = $campoIzq;
    $this->Op = $Op;
    $this->campoDer = $campoDer;
}
```

Fig.11 Fragmento de código donde se evidencia la utilización del patrón GOF: Composite (Objeto Compuesto).

Adapter: El uso de este patrón permite convertir la interfaz de una clase en otra interfaz esperada por los clientes (Prieto, 2009), se evidencia en la clase UCQueryBuilder, encargada de crear la interfaz que se ocupa de agregar las condiciones a la consulta en la vista de diseño. Ver figura 12.

```
node = entity.getFieldEntityOperation(field);
windowOfWhereCondition = new gdr2.query_designer.view.query.WindowOfCondition(node, UCQueryBuilderScope
windowOfWhereCondition.getPanelOfCondition().getComboLogicOperation().setValue(logicOperator);
windowOfWhereCondition.getPanelOfCondition().getComboFromWhere().setValue(conditionOperator);
windowOfWhereCondition.getPanelOfCondition().getTextAreaFieldOfEntity().setValue(fieldName);
if (isEntityValue == "true") {
    windowOfWhereCondition.getPanelOfCondition().getRadioDatabaseField().setValue(true);
    windowOfWhereCondition.getPanelOfCondition().getRadioArmValue().setValue(false);
    windowOfWhereCondition.getPanelOfCondition().getComboFieldsFromTable().setValue(rightField);
    windowOfWhereCondition.getPanelOfCondition().getTextAreaDescription().hide();
    windowOfWhereCondition.getPanelOfCondition().getComboFieldsFromTable().show();

    isEntityValue = true;
}
else {
    windowOfWhereCondition.getPanelOfCondition().getTextAreaDescription().setValue(rightField);
    isEntityValue = false;
}
```

Fig.12 Fragmento de código donde se evidencia la utilización del patrón GOF: Adapter (Adaptador).

***Iterator*:** Proporciona una forma de acceder a los elementos de una colección de objetos de manera secuencial sin revelar su representación interna (Prieto, 2009). Se evidencia en la clase Lexer.php en la cual se recorre la cadena de caracteres con la finalidad de conformar los tokens. Ver figura 13.

```
private function consume() {
    $this->currentChar = $this->source->nextChar();
}

private function skipAll() {
    while ($this->currentChar == ' ' || $this->currentChar == '\t' || $this->currentChar == '\n'
        $this->consume();
    }
}
```

Fig.13 Fragmento de código donde se evidencia la utilización del patrón GOF: Iterator (Iterador).

2.6. Modelo de Diseño.

Un Modelo de Diseño es básicamente un modelo físico que proporciona la estructura y la forma que tendrá el sistema que se está construyendo (Presman, 2010). En la elaboración y confección de un software es importante realizar previamente un correcto diseño para facilitar la implementación del mismo, ya que las bases siempre estarán fomentadas en gran medida en la eficacia de lo que se ha diseñado anteriormente.

2.6.1. Diagrama de clases del diseño.

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación (Larman, 1999). El Diagrama de clase es el diagrama principal de análisis y diseño para un sistema. En él, se especifica la estructura de clases del sistema, con relaciones y estructuras de herencia. Muestra los atributos y métodos de cada clase, representando de forma sencilla la colaboración y las responsabilidades de cada una de ellas, entorno al sistema que conforman. Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal. Durante el diseño, se usa el mismo diagrama, y se modifica para satisfacer los detalles de las implementaciones.

2.6.2. Descripción de las clases del modelo de diseño.

UCQueryBuilder: Es la clase controladora principal del módulo, contiene las principales operaciones que se realizan del lado del cliente, entre estas operaciones se encuentra VerifyQuery(), implementada como la acción que se encarga de obtener la consulta SQL modificada y de verificar si está correctamente estructurada, otra función perteneciente a esta clase es UpdateDesign() encargada de actualizar el diseño una vez que se haya comprobado que la sintaxis de la consulta SQL este correctamente estructurada.

QueryController: Es la clase perteneciente al controlador que se encarga de delegar las responsabilidades a cada una de las clases controladoras que intervienen en el proceso de gestionar la consulta.

Modelo: Es la clase del modelo que contiene toda la información sobre las tablas pertenecientes a la base de datos en cuestión.

A continuación en la figura 14, se presenta el diagrama de clases del diseño para el caso de uso **Gestionar consulta**.

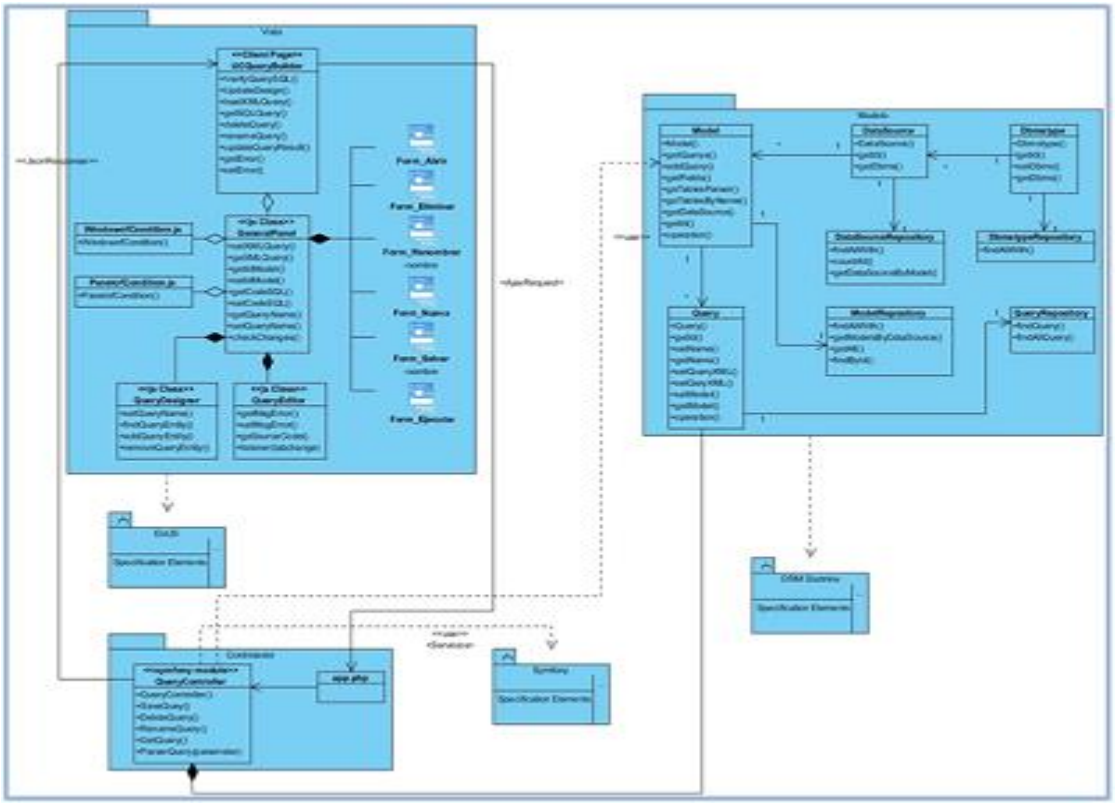


Fig.14 Diagrama de clases del diseño para el caso de uso Gestionar consulta.

Diagrama de Despliegue.

Los diagramas de despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria. (Marca Huallpara, 2009) Ver figura 15.

PC Cliente: Representa una computadora desde la cual el usuario podrá acceder a la aplicación.

Servidor Web: Representa el servidor donde estará situada la aplicación de GDR.

Servidor de Bases de Datos: Representa el servidor donde estarán los SGBD que darán respuesta a las peticiones hechas por la aplicación.

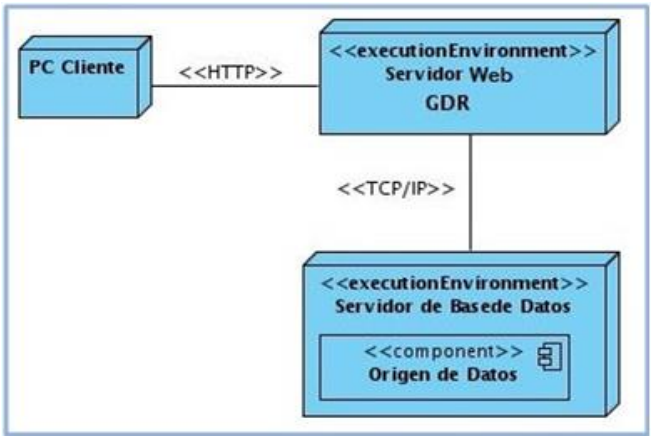


Fig.15 Modelo de Despliegue.

El diagrama de despliegue realizado representa tres nodos principales. El nodo PC_Cliente que requiere de un navegador que soporte Java Script, el nodo Servidor de Bases de Datos desde donde se cargará el Origen de Datos y el nodo Servidor de Aplicaciones, en el cual debe estar instalado el servidor web.
El nodo PC Cliente estará conectado al nodo Servidor de Aplicación por el protocolo de transferencia de hipertexto (HTTP por sus siglas en inglés) como protocolo de comunicación. La conexión entre el Servidor de Aplicaciones y el Servidor de Base de Datos se realizará mediante el protocolo de comunicación TCP/IP.

2.7. Conclusiones.

La confección del diagrama de clases del dominio permitió visualizar la relación existente entre las clases u objetos significativos del diseñador de consultas en el dominio del problema. A partir del levantamiento de requisitos se obtuvieron 5 categorías de requisitos no funcionales y 18 requisitos funcionales, estos últimos fueron agrupados en tres casos de uso, garantizando que el desarrollo de la aplicación cumpla con las necesidades de los clientes. El diseño de los diagramas de clases brindó la posibilidad de mostrar la estructura de clases del diseñador, así como las relaciones entre las mismas. La utilización de los estilos y patrones arquitectónicos seleccionados proporcionaron un correcto diseño del sistema. La realización del diagrama de despliegue permitió comprender la distribución física de la aplicación propuesta como solución.

CAPÍTULO III: IMPLEMENTACIÓN Y PRUEBA DEL SISTEMA DEL DISEÑADOR DE CONSULTA.

En el presente capítulo se describe el modelo de implementación a partir del diseño realizado porcada uno de los casos de uso identificados. Además, se muestran fragmentos del código y las vistas principales de la aplicación. Se describen las pruebas a realizar, con el objetivo de comprobar la correcta implementación de cada una de las funcionalidades definidas, verificando que los resultados de las pruebas sean los esperados y el correcto funcionamiento del sistema.

3.1. Modelo de Implementación.

"El modelo de implementación describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados y cómo dependen los componentes unos de otros". (Somerville, 2005) Esta descripción es de gran utilidad a la hora de implementar el sistema, facilita la organización del trabajo y lo hace más entendible a los desarrolladores. Este modelo está conformado por el diagrama de componentes.

3.1.1. Diagrama de componentes.

En los diagramas de componentes se muestran los elementos de diseño de un sistema de software, el mismo permite visualizar con más facilidad la estructura general del sistema y el comportamiento del servicio que estos componentes proporcionan y utilizan a través de las interfaces. Representa la separación de un sistema de software en componentes físicos y muestra las dependencias entre estos. Cada subsistema corresponde a un paquete físico y cada componente a un módulo, fichero o biblioteca existente en la memoria de almacenado. (Somerville, 2005).

El objetivo de estos diagramas es mostrar la estructura de alto nivel del modelo de implementación, especificando los subsistemas de implementación y sus dependencias a la hora de importar código y organizar los subsistemas en capas. A continuación se muestra un ejemplo del diagrama de componentes correspondiente al caso de uso **Gestionar consultas** del sistema. Ver figura 16.

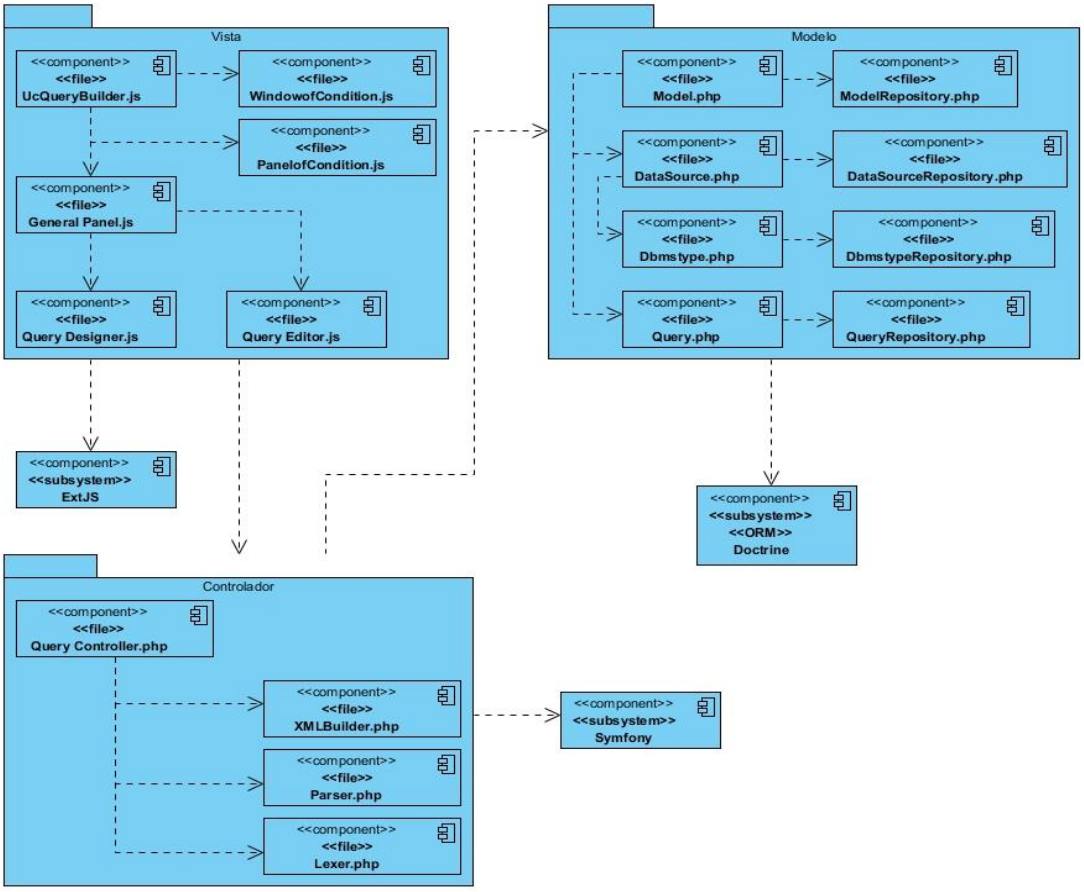


Fig.16 Diagrama de componentes del CU Gestionar consultas.

Seguidamente se describe el diagrama de componente del CU "Gestionar consulta" el cual cuenta con 3 paquetes de implementación básicos. Estos paquetes son:

- El Paquete de Clases Vista, agrupa los componentes que se encargan de manejar la visualización de la información que reciben del controlador. En este paquete se encuentra la clase **UCQueryBuilder,** que contiene las principales operaciones que se realizan del lado del cliente.
- El Paquete de Clases Controlador, que contiene las clases que manipulan los eventos del usuario, estas reciben la notificación de la acción solicitada por el mismo, gestionan el evento de entrada, accediendo posteriormente al modelo y otorgando finalmente a los objetos de la vista la tarea de mostrar la interfaz apropiada. En este paquete se encuentra la clase controladora **QueryController** encargada de recibir la petición de comprobar si la consulta está correctamente estructurada apoyándose en las clases **Lexer**, **Parser** y **XMLBuilder** para dar respuesta a esta petición.

- El Paquete de Clases Modelo, que agrupa las clases que permiten acceder a la capa de almacenamiento de datos o acceso a datos y responden las instrucciones del controlador. En este paquete se encuentra la clase **Model** que permite obtener los datos del modelo así como las tablas y los campos por tablas.

3.2. Código Fuente.

Para obtener una versión funcional de la aplicación se deben implementar los componentes que se han definido, como resultado se obtienen archivos que contienen el código fuente del módulo. Es un conjunto de instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento. Estas instrucciones son escritas en un lenguaje de programación que consiste en un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

3.2.1. Estándares de codificación.

La ISO10 define un estándar como: *''Acuerdos documentados que contienen especificaciones técnicas u otros criterios precisos para ser usados consistentemente como reglas, guías o definiciones de características para asegurar que los materiales, productos, procesos y servicios cumplan con su propósito".*

Estilo de codificación utilizado:

- Los bloques de código siempre deben estar encerrados por llaves.
- El nombre de métodos y variables, debe permitir que con solo leerlo se conozca el propósito de la misma.
- Todos los métodos y nombres de clases se escriben en estructura lowerCamelCase, este estilo define la primera letra de cada palabra en mayúscula excepto la primera palabra que la pone en minúscula completa.
- Colocar espacios en blanco entre operadores lógicos-aritméticos y sus operandos.
- Todas las variables y atributos se escriben en estructura lowerCamelCase.
- Todas las funciones y clases estarán comentadas para explicar el flujo del código y el propósito de las funciones o variables.
- Todas las etiquetas PHP deben ser completas (<?php ?>)... no reducidas (<? ?>).
- Se usará una identación para los siguientes casos (Tamaño = 4 (espacios) para):
  - ✓ Declaraciones dentro de las clases.
  - ✓ Enunciado dentro de métodos y funciones.

- ✓ Enunciados dentro de bloques de comandos.

Ejemplo de código fuente.

A continuación la figura17 muestra el fragmento de código del método: "Parse" el cual permite parsear la consulta con el objetivo de detectar los errores sintácticos existentes y capturar la información necesaria para construir el XML. En esta imagen se evidencian los estándares de codificación mencionados anteriormente.

```
/**
 * Parsea la consulta
 */
public function parse() {
    $this->query();
}

/**
 * Parseo completo de la consulta.
 */
private function query() {
    $this->select();
    $this->from();
    $this->where();
    $this->groupBy();
    $this->having();
    $this->orderBy();
    $this->limit();
    if ($this->lookaheadKind() == tokenKind::tk_SemiColon) {
        $this->match(tokenKind::tk_SemiColon);
        $this->match(tokenKind::tk_EOT);
    }
}
```

Fig.17 Fragmento de código del método "Parse".

3.3. Pruebas del software.

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación.

3.3.1. Niveles de Prueba.

Los niveles de prueba son diferentes ángulos de verificar y validar un producto de software. Las pruebas se aplican durante todo el ciclo de desarrollo del software para diferentes objetivos y en distintos niveles de trabajo. Según Pressman, estos niveles se pueden ver como una espiral, como se ilustra en la figura 18.

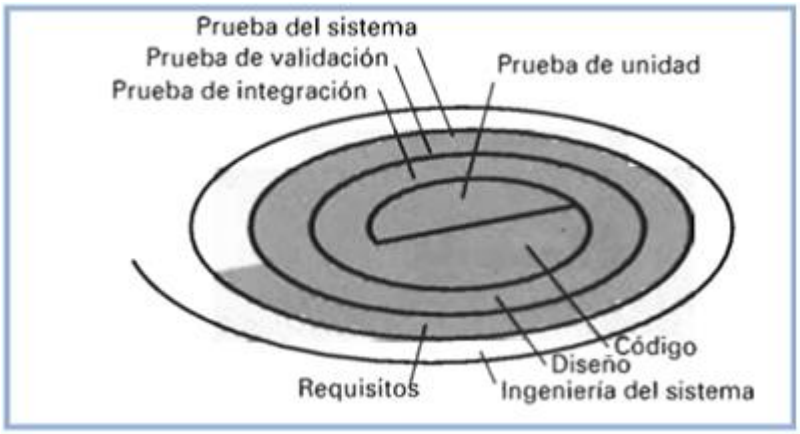


Fig. 18 Niveles de Pruebas. **(Pressman, 2002)**

Para comprobar que el módulo Diseñador de Consultas cumple con los requisitos previamente identificados en la fase de elaboración se aplicarán las pruebas a nivel de unidad, integración y validación.

Nivel de Unidad

"Las pruebas de unidad o pruebas unitarias como también se le conocen centran el proceso de verificación en la menor unidad del diseño del software: el componente de software o módulo. Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada. Se prueban las condiciones límite para asegurar que el módulo funciona correctamente en los límites establecidos".(Pressman, 2002)

Nivel de Integración

Según Pressman, las pruebas de integración se llevan a cabo durante la construcción del sistema, involucran a un número creciente de módulos y terminan probando el sistema como conjunto. Las pruebas de integración tienen dos objetivos principales, descubrir errores asociados con las interfaces de los módulos y ensamblar sistemáticamente los módulos individuales para formar subsistemas y al final un sistema completo. (Pressman, 2002)

Nivel de Validación

"Tras la culminación de la prueba de integración, el software está completamente ensamblado como un paquete, se han encontrado y corregido los errores de interfaz y puede comenzar una serie final de pruebas del software: la prueba de validación. La validación puede definirse de muchas formas, pero una simple definición es que la validación se consigue cuando el software funciona de acuerdo con las expectativas razonables del cliente". (Pressman, 2002)

3.3.2. Tipos de pruebas

A partir de los niveles de prueba anteriormente definidos, se seleccionan los siguientes tipos de pruebas para comprobar el correcto funcionamiento del módulo Diseñador de Consultas:

Pruebas funcionales: Prueban una funcionalidad completa, donde pueden estar implicadas una o varias clases y la propia interfaz de usuario.

Pruebas de aceptación de tipo alfa: El cliente en el lugar de desarrollo verificará que el sistema cumple con los requerimientos especificados al inicio de la fase de elaboración, en presencia del desarrollador, quien en caso de existir errores registrará los mismos y les dará solución, para que la prueba sea ejecutada nuevamente, repitiendo el proceso hasta lograr que el producto quede libre de errores y el cliente obtenga el producto que solicitó.

Pruebas de integración de tipo ascendente: La integración incremental se aplica de forma que el programa se construye y prueba en pequeños incrementos, lo cual permite aislar y corregir los errores que surjan. La integración incremental ascendente comienza con la construcción y prueba de módulos atómicos. Los componentes se integran desde los niveles más bajos hacia arriba, de forma que los módulos de bajo nivel se combinen en grupos que realicen una función del software y se prueba el grupo (Pressman, 2002).

3.3.3. Método de prueba

Existen métodos de prueba independientemente de la técnica que se utilice o el nivel en que se enmarquen estas técnicas. Estos métodos proporcionan distintos criterios para generar casos de prueba que provoquen fallos en los programas. En el proceso de pruebas en cuestión se hace uso del método de prueba de Caja Negra.

- **Método de Caja Negra:**

El método de caja negra, también denominado prueba de comportamiento, se centra en los requisitos funcionales del software, para definir los escenarios y las secciones a probar, el diseñador de prueba elabora un caso de prueba para cada requisito funcional, permitiéndole al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa, como se muestra en la figura 19.

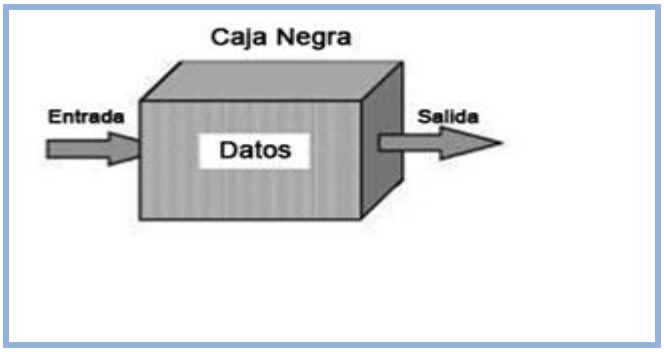


Fig. 19 Método Caja Negra. Fuente: Elaboración propia.

Para la aplicación del método de caja negra se utiliza la técnica **de Partición de Equivalencia** que permite examinar los valores válidos e inválidos de las entradas existentes en el software, descubre de forma inmediata una clase de errores que, de otro modo requerirían la ejecución de muchos casos antes de detectar el error genérico.

3.4. Diseño de caso de prueba

Los casos de pruebas se realizan con el objetivo de determinar que una funcionalidad ha sido implementada satisfaciendo las necesidades del cliente. Para cada caso de uso debe estar asociado un caso de prueba que recoja la especificación del mismo, dividido en secciones y escenarios, detallando las funcionalidades descritas en él y describiendo cada variable que recoge el caso de uso en cuestión.
A continuación se muestra la matriz de datos del caso de prueba perteneciente al caso de uso Actualizar consulta, donde se evaluó y probó la validez de cada uno de los datos introducidos en el sistema, específicamente en la sección que se estuvo probando. A continuación se muestra la tabla 4 donde se describe el escenario Actualizar Vista.

Matriz de Datos

Escenario Actualizar Vista

Tabla 4: Caso de prueba del CU Actualizar Vista.

Escenario	Descripción	Variables	Respuesta del sistema	Flujo central
EC 1.1 Actualizar vista con consulta correcta.	*En este escenario se actualiza la vista a partir de una consulta escrita correctamente.*	*SELECT ("fix"."address") AS "address", ("fix"."name")*	*Se actualiza correctamente la vista, se representan*	*1. Módulo Diseñador de Consultas/ Vista SQL. 2. Se introduce los datos de la consulta*

		AS "name" FROM "public"."fix" "fix" LIMIT 50 offset 0;	correctamente las tablas y se cargan todos los datos de la consulta.	correctamente en la vista SQL. 3. Se cambia a la vista de diseño.
Actualizar vista con consulta incorrecta.	En este escenario se actualiza la vista a partir de una consulta escrita incorrectamente.	SELECT ("fix"."address") AS "address", ("fix"."na FROM "public"."fix" "fix" LIMIT 50 offset 0;	El sistema devuelve al usuario a la vista SQL y muestra los errores presentes en la consulta.	1. Módulo Diseñador de Consultas/ Vista SQL. 2. Se introduce los datos de una consulta incorrecta en la vista SQL. 3. Se cambia a la vista de diseño.

3.5. Resultados de las pruebas

Método Caja negra

Después de realizar las pruebas funcionales mediante el método de caja negra, utilizando los casos de prueba asociados a cada caso de uso, se comprobó el correcto funcionamiento del módulo Diseñador de Consultas. Durante las tres iteraciones realizadas, se detectaron un total de siete no conformidades, quedando todas resueltas, como se muestra sobre la figura 20 y la tabla 5.

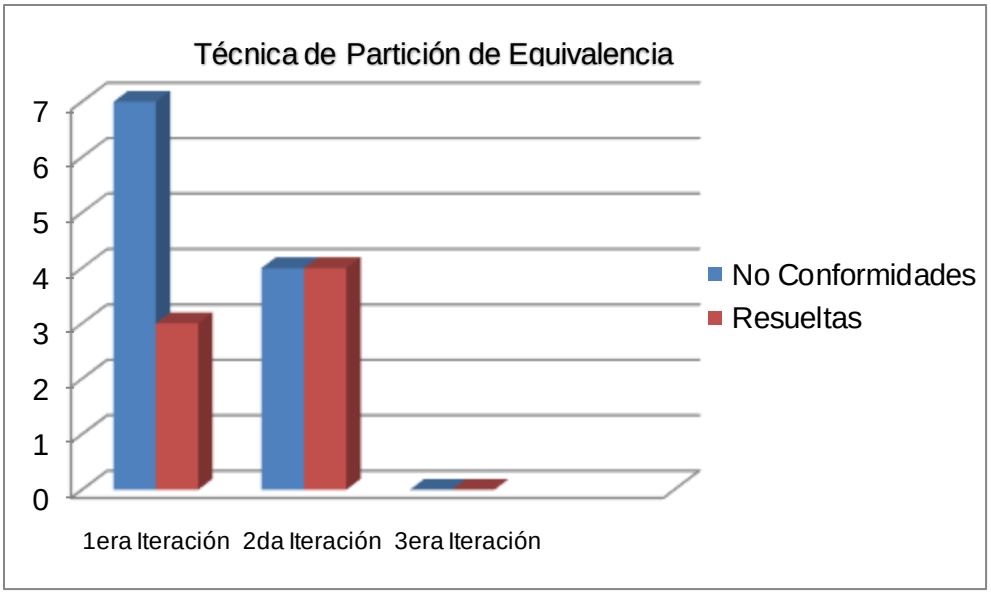


Fig. 20 Gráfico que representa la relación de No Conformidades (NC) Detectadas y NC Corregidas para la técnica: Partición de Equivalencia.

Luego de realizar las pruebas de caja negra se integró el parser SQL al módulo Diseñador de Consultas y se pudo comprobar que está adecuadamente integrado y que funciona correctamente. Cuando se modifican los datos en la vista SQL de la consulta que se encuentra en diseño y se cambia de vista, se actualiza la vista del diseño de la consulta modificada.

Tabla 5: Descripción de las iteraciones realizadas durante las pruebas.

No. Iteración	Cant. NC	Tipo Error	Impacto
1	***7***	2 Ortografía	Bajo
		1 Interfaz	Medio
		4 Validación	Alto
2	***4***	4 Validación	Alto
3	***0***	**-**	**-**

Integración Ascendente

En la aplicación el tipo de prueba integración incremental ascendente estuvo reflejada durante el desarrollo de cada uno de los módulos que fueron (PostgreSQL, MySQL y SQLite) probados por separados a medida que se iban realizando o modificando cada una de sus funcionalidades. Luego

fueron añadidos al Diseñador de Consultas del GDR, se integraron a la clase controladora, después de modificada la controladora se realizó la actualización de las clases de la vista con el objetivo de garantizar la integración completa. Para realizar el proceso de integración se siguieron los siguientes pasos:

Pasos para realizar la integración:

- Se añadieron los tres módulos al paquete de librerías de GDR dentro del paquete QuerysManipulation.
- Se agrega al módulo Postgres_Module la clase PostgresModuleParser.php para garantizar que dicho módulo pueda ser conectado con la controladora QueryController.php.
- Se agrega al módulo Mysql_Module la clase MysqlModuleParser.php para garantizar que dicho módulo pueda ser conectado con la controladora QueryController.php.
- Se agrega al módulo Sqlite_Module la clase SqliteModuleParser.php para garantizar que dicho módulo pueda ser conectado con la controladora QueryController.php.
- Se modifica la clase controladora para que reparta las responsabilidades del parseo a los módulos de PostgreSQL, MySQL y SQLite que estarán vinculados a ella.
- Se le agrega a los tres módulos la clase XMLFactory.php encargada de generar el XML.
- Se modifica en la clase UCQueryBuilder.js los métodos loadFieldsFromTable, callBackLoadQuery y showSelectedFieldOfEntity con el objetivo de poder pintar las tablas en el área de diseño a partir de la nueva estructura del XML.
- Se crea en la clase QueryEntity.js el método getFieldEntityOperation() para poder ubicar en el árbol de exploración de consultas (*QueryExplorer*) los campos seleccionados de cada tabla así como los campos que poseen funciones de agregación a partir del XML creado.
- Se modifica la clase PanelOfCondition.js para añadir a la vista de diseño un panel donde el usuario pueda insertar una subconsulta de forma manual o utilizar como subconsulta el SQL de una consulta existente, cargándola desde la base de datos.
- Se modifica la clase WindowofCondition.js para obtener los valores de los campos pertenecientes a los nuevos paneles añadidos, los cuales contienen el valor de la subconsulta seleccionada o escrita y una vez obtenido el valor agregar la condición al explorador de consultas (*QueryExplorer*).

Una vez terminado el parser se integra al módulo Diseñador de Consultas hasta que funciona como un todo. Se prueban las funcionalidades del módulo para asegurar que los datos que se introducen en la consulta modificada correspondan con los necesarios para el correcto funcionamiento del mismo. Se comprueba que queden validados todos los datos que se introducen en la vista SQL para modificar la

consulta que se encuentra en diseño. Además se modifican las clases correspondientes de la vista, para que pueda ser posible representar las nuevas estructuras SQL soportadas por el parser.

Aceptación

Las pruebas de aceptación se realizan con el objetivo de validar la solución a través de un encuentro con el Jefe de proyecto del GDR, el cual comprueba que el sistema cumple con el funcionamiento esperado y da muestra de su conformidad en el documento oficial Carta de Aceptación del cliente donde quedan plasmados los resultados.(Ver anexo 2)

3.6. Conclusionesparciales del capítulo

La creación del diagrama de componentes sirvió para representar una vista estática de la aplicación, mostrando la organización y dependencia que existe entre los componentes físicos que se necesitan para ejecutar la misma. Los estándares de codificación y los estilos de programación utilizados fueron descritos para hacer más fácil el entendimiento del código y para permitir un mejor mantenimiento a la aplicación en un futuro. Las pruebas pertenecientes a los niveles de unidad, integración y de validación permitieron comprobar que se cumplieron los requisitos del sistema.

CONCLUSIONES.

Una vez culminada la investigación se puede afirmar que se le dio cumplimiento a los objetivos planteados, arribando a las siguientes conclusiones:

- ✓ El estudio de los principales conceptos relacionados a los diseñadores de consultaspermitió sentar las bases para el desarrollo de la versión 2.0 del módulo Diseñador de Consultas para el Generador Dinámico de Reportes v2.0.
- ✓ A partir de la realización del análisis y diseño del módulo Diseñador de Consultas v2.0 se obtuvo como resultado los artefactos necesarios para guiar el desarrollo del mismo.
- ✓ La implementación del módulo Diseñador de Consultas v2.0 dio cumplimiento a los 18 requisitos funcionales identificados en el análisis y diseño.
- ✓ El diseño y ejecución de las pruebas a nivel de unidad, integración y de validación permitió comprobar el correcto funcionamiento del módulo Diseñador de Consultas v2.0 y el cumplimiento de las funcionalidades incorporadas.
- ✓ Como resultado se obtuvo el módulo Diseñador de Consultas v2.0 que permite a los usuarios el diseño de consultas dinámicas, extiende su uso a los lenguajes MySQL y SQLitey disminuyendo el rango de erroresdurante el proceso.

RECOMENDACIONES.

Dados los resultados de la investigación, en relación con el problema que aborda, se sugiere:

- ✓ Recomendamos luego de parsear la consulta realizar el análisis semántico delegado hasta ahora en el sistema gestor de bases de datos, lo cual permita que el usuario conozca el grado acierto que tiene la consulta sin tener que recurrir a la ejecución de la misma para saberlo.
- ✓ Incluir en la vista de diseño la posibilidad de representar visualmente las restantes estructuras extendiendo así su uso y funcionalidad:
  - Estructura DISTINCT.
  - Estructura DISTINCT ON.
  - Estructura BETWEEN.
  - Estructura EXIST.
  - Operaciones aritméticas con números y funciones de agregación.

REFERENCIAS BIBLIOGRAFICAS.

1. **Alegsa, Leandro. 2010.** DICCIONARIO DE INFORMÁTICA Y TECNOLOGÍA. [Online] julio 2, 2010. [Cited: octubre 16, 2014.] http://www.alegsa.com.ar/Dic/framework.php#sthash.vlolXZ1j.dpuf.
2. **antonionl. 2013.** Entorno de Desarrollo Integrado o IDE. [Online] enero 25, 2013. [Cited: diciembre 15, 2014.] https://antonionll.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-o-ide/.
3. **Booch, Rumbaugh y Jacobson. 2007.***El Lenguaje Unificado de Modelado.* 2007. Vol. Cap 1 y 2.
4. **ColectivoAutores. 2011.** lenguajephp. [Online] Mayo 2, 2011. [Cited: diciembre 12, 2014.] http://lenguajephpjc.blogspot.com/2011/05/caracteristicas-del-lenguaje-php.html.
5. **Desing Patterns. E. Gamma, R. Helm, R. 1995..***Johnson, and J. Vlissides. Design Patterns.* 1995.
6. **ExtJS. 2011.** ExtJS[En línea]. [Online] 2011. [Cited: septiembre 22, 2013.] http://www.sencha.com/products/sencha-cmd/#overview.
7. **GAZQUEZ MARTÍNEZ, Orelvi and SERRANO GARCÍA, Yadrian. 2011..***Diseño e Implementación del módulo Diseñador de Consultas del Generador Dinámico de Reportes. .* La Habana: Universidad de las Ciencias Informáticas. : s.n., 2011.
8. **Hern, Marcello Visconti. 2002.***Fundamentos de Ingeniera de Software.* 2002.
9. ***I, Universidad Jaime. Ingeniería Informática.*** *Procesadores de lenguajes. Estructura de los compiladores e interprétes. .* Castellón de la Plana (España) : s.n. s.n., 2009..
10. **Iliana Amabely Silva Hernández. 2013.** Generador Automático de Reportes Dinámicos. [Online] 2013. http://www.cs.cinvestav.mx/tesisgraduados/2003/resumenIlianaAma.html.
11. **J.Date, Chris. 2001.***Introducción a los Sistemas de Bases de Datos.* México : PEARSON DUCACIÓN, 2001. 968-444-419-2.
12. **JavaScript. 2011 .** Programación en JavaScript. [Online] octubre 2 , 2011 . [Cited: noviembre 25, 2014.] http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_JavaScript/Introducci%C3%B3n.
13. **José Esteva, Lianet. 2009.***Introducción a OpenUP/Basic.* [Online] Teleinformatics Technology Group, 2009. [Cited: febrero 2, 2013.] http://epf.eclipse.org/wikis/openupsp/openup_basic/customcategories/introduction_to_openup_basic,_BTJ_YMXwEduywMSzPTUUwA.html.

14. **kioskea. 2014.** kioskea. [Online] 2014. [Cited: septiembre 28, 2014.] http://es.kioskea.net/contents/304-lenguajes-de-programacion.
15. **Larman, Craig. 1999.***UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : Prentice Hall, Inc, 1999. pp. 11. 40,193 - 221. ISBN: 970-17-0261-1.
16. **Leon Welicki. 2009.** Modelo Vista Controlador . [Online] 2009. [Cited: enero 28, 2015.] http://msdn.microsoft.com/en-us/library/bb972251(es-es).aspx.
17. **Manuel, Bertha Mariel y José. 2004.** arquitectura cliente servidor.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf. [Online] 2004. [Cited: marzo 20, 2014.]
18. **Marca Huallpara, Hugo Michael,Quisbert Limachi,Nancy Susana. 2009.***ANALISIS Y DISEÑO DE SISTEMAS II Diagrama de Despliegue.* 2009.
19. **María del Mar Aguilera Sierra y Sergio Gálvez Rojas. 2014.** Traductores, Compiladores e Intérpretes. [Online] 2014. [Cited: noviembre 9, 2014.] http://www.lcc.uma.es/~galvez/ftp/tci/tictema5.pdf.
20. **Mariano Reingart. 2013.** PgAdmin. [Online] 2013. [Cited: octubre 21, 2014.] http://www.postgresql.org.ar/trac/wiki/PgAdmin?format=pdfarticle.
21. **Martínez., Liesner Acevedo. 2011.***Técnicas de Compilación: Manual Práctico para estudiantes de Informática.* La Habana : s.n., 2011.
22. **Mastermagazine. 2012.** Definición de Casos de uso. [Online] 2012. [Cited: marzo 8, 2014.] http://www.mastermagazine.info/termino/4184.php.
23. **Mato Garcia, Rosa Maria. 2012.***Surgimiento histórico de las bases de datos integradas. 2. 2.* Habana : Editorial Pueblo y educacion, 2012. García, Lic Rosa María Mato.Diseño de Base de Datos. . [En línea] [Citado el: 17 de Octubre de 2012.]Junio 2009.
24. **Menéndez Alonso, Evelyn. 2009.** Herramientas CASE para el proceso de desarrollo de Software. [Online] Diciembre 26, 2009. [Cited: noviembre 8, 2014.] http://www.monografias.com/trabajos73/herramientas-case-proceso-desarrollo-software/herramientas-case-proceso-desarrollo-software.shtml.
25. **OpenUP, Introduction to. 2002.** Introduction to OpenUP (Open Unified Process). [En línea]] (2002). [Online] 2002. [Cited: enero 14, 2013.]
26. **Payolis and nirvana john. 2010.** payolis and nirvana john. [Online] julio 1, 2010. [Cited: abril 20, 2015.] http://wwwpayojohn.blogspot.com.
27. **Pérez, Javier Eguíluz. 2009.***Introduccion a javascript.* 2009.
28. **Potencier, Fabien. 2010.***Symfony la Guía Definitiva.* 2010.

29. **Presman, Roger S. 2010.***Ingenieria de Software Un Enfoque Práctico.* s.l. : McGraw-Hill, 2010. ISBN 978-0-07-337Z.
30. **Pressman, Roger S. 2002.***Ingeniería de Software, un enfoque práctico.* s.l. : s.l. : McGraw-Hill Companies, 2002. ISBN: 8448132149., 2002.
31. **Prieto, Félix. 2009.** Patrones de diseño. [Online] 2009. [Cited: marzo 28, 20015.] http://www.infor.uva.es/~felix/datos/priii/tr_patrones-2x4.pdf..
32. **Reynoso, Carlos Billy. 2009.***Introducción a la arquitectura de software.* 2009.
33. **Richardson Software. 2013.** razorsql. [Online] 2013. [Cited: octubre 25, 2014.] http://razorsql.softonic.com/mac.
34. **Somerville, Ian. 2005.***Ingeniería del software 7 edición.* Madrid : Pearson Educación, 2005. ISBN/84-7829-074-5.
35. **SQLServer . 2013.** devart. [Online] 2013. http://www.devart.com/dbforge/sql/querybuilder/.
36. **TENORIO, R. R. 2010.***Las pruebas de software y su importancia en las organizaciones .* 2010.
37. **VariosAutores. 2012.** Generador Dinámico de Reportes versión 1.8.0. [Online] 2012. [Cited: octubre 15, 2014.] http://publicaciones.uci.cu/index.php/SC/article/view/889.
38. **VisualParadigm. 2014.** Visual Paradigm for UML - Software design tools for agile software development. [Online] 2014. [Cited: Enero 22, 2014.] [http://www.visual-paradigm.com/product/vpuml/]..
39. **Wielenga, Geertjan. 2014.** The top 5 features of NetBeans IDE 8. [Online] enero 16, 2014. [Cited: marzo 23, 2014.] http://jaxenter.com/the-top-5-features-of-netbeans-ide-8-107412.html.
40. **Young, Ralph R. 2004.***The Requirements Engineering Handbook.* s.l. : ARTECH HOUSE, INC., 2004. pp. 1, 51.
41. **Zaman Riaz. 2014.** MVC. [Online] agosto 14, 2014. [Cited: abril 21, 2015.] http://code.tutsplus.com/tutorials/from-beginner-to-advanced-in-opencart-understanding-mvc--cms-21627.

ANEXOS.

Anexo 1. Preguntas realizadas al cliente como parte de la entrevista no estructurada realizada a los especialistas del Departamento de Desarrollo de Componentes con el objetivo de recopilar información referente a la realización de GDR v1.8.

1. ¿Cómo está compuesto el Generador Dinámico de Reportes v1.8?
2. ¿Cuáles son las clases que componen la vista del GDR que pertenecen al módulo Diseñador de Consultas?
3. Estructura que debe tener el XML que seusapara actualizar la vista del GDR que pertenecen al módulo Diseñador de Consultas.
4. Necesidades de mejoras del parser además de las consultas anidadas ylos parámetros.

Anexo 2. Carta de Aceptación.

UCI

CARTA DE ACEPTACIÓN

En cumplimiento con la fase de desarrollo y en función de la ejecución del proyecto Generador Dinámico de Reportes (GDR v2.0) para el Departamento de Desarrollo de Componentes del Centro de Tecnología de Gestión de Datos de la facultad 6, DATEC, se hace entrega de los productos que se relacionan a continuación:

- Versión 1.1 del Módulo Diseñador de Consultas para el Generador Dinámico de Reportes v2.0. (código fuente).

Entrega	Recibe	Recibe
Nombre y apellidos:	Nombre y apellidos:	Nombre y apellidos:
José Aguilar Gómez Rolando Enrique Fuentes Orozco	Glennis Tamayo Morales	Yoander Iñiguez Bermúdez
Cargo: Tesistas	Cargo: Jefe de Departamento Desarrollo de Componentes	Cargo: Jefe de proyecto GDR v2.0
Firma:	Firma:	Firma:

Fecha: 10/06/2013

UCI Decanato Facultad 6