FACULTAD 6

Constructor Gráfico de Consultas para AUDAT

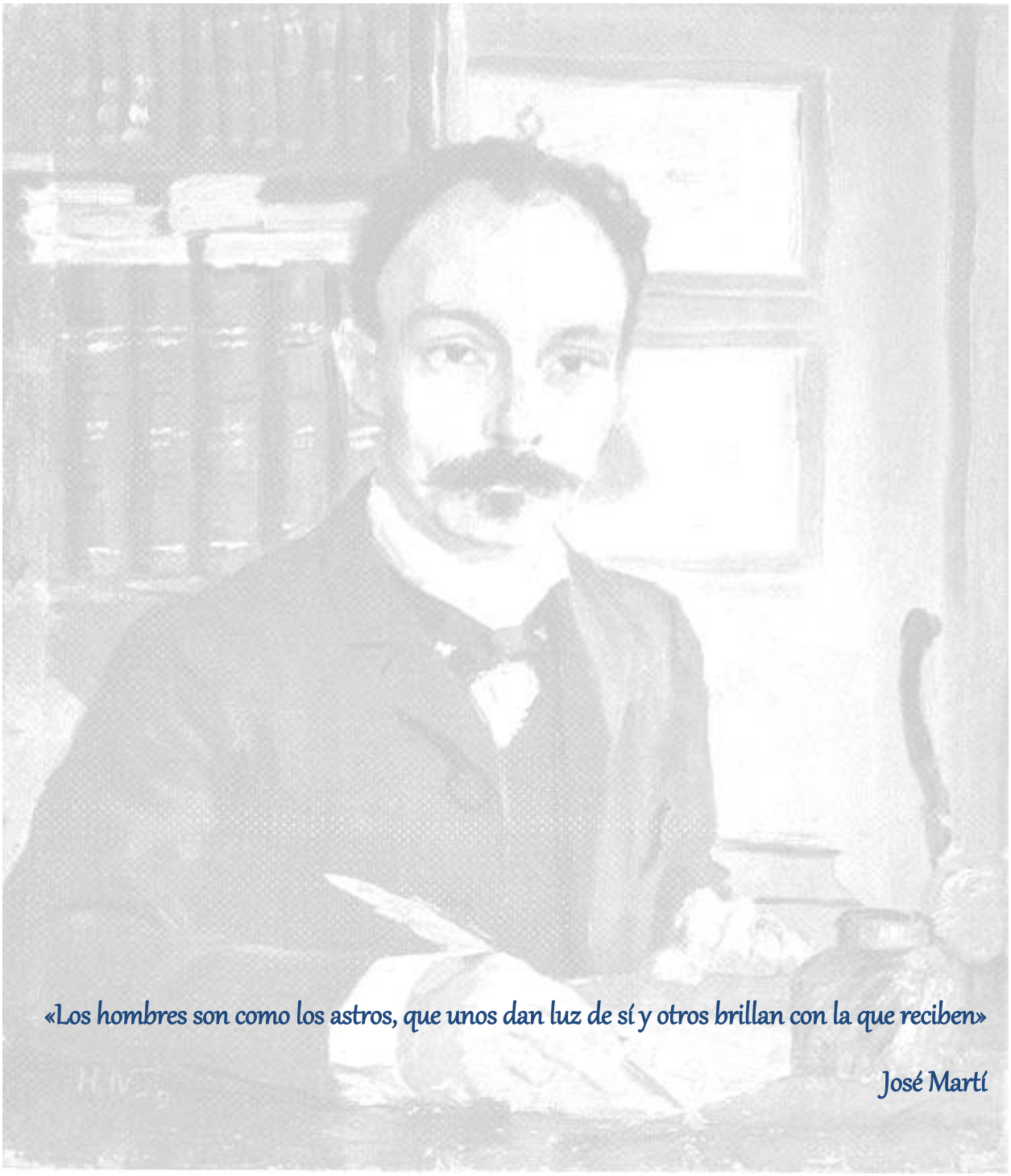
Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Liosdanys Rodríguez Pérez

Tutora: MsC. Yudisney Vazquez Ortíz

LA HABANA, JUNIO DE 2015

"AÑO 57 DE LA REVOLUCIÓN"



«Los hombres son como los astros, que unos dan luz de sí y otros brillan con la que reciben»

José Martí

Declaración de autoría

Declaro ser autor de la presente tesis y otorgo 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 _____.

Liosdanys Rodríguez Pérez	MsC. Yudisney Vazquez Ortíz
Autor	Tutora

Por depositar toda su confianza en mí;

por ser lo más importante que tengo en la vida y dedicar la suya a que yo sea una persona de bien;

porque cada mañana, cuando despierto, doy mil gracias por tenerlos cerca y poder contar con ellos;

por tomar mi mano y ayudarme a subir esta montaña para que pueda estar aquí, donde estoy hoy;

por eso y por todo lo que no puede ser descrito con palabras:

Dedico este trabajo a mis padres Dania Pérez Neira y Luis Rodríguez Acosta y a mi hermano Lisdan Rodríguez Pérez

Resumen

Las auditorías son una manera efectiva de poner en orden los recursos de la empresa para lograr un mejor desempeño y productividad. El actual desarrollo tecnológico y la importancia que han alcanzado las bases de datos, como soporte en los sistemas informáticos, han propiciado que la mayoría de las empresas almacenen su información a través de estas, siendo indispensable la creación de sistemas de auditoría a ellas. Las auditorías a bases de datos son las encargadas de controlar el acceso, la actualización, la integridad y calidad de los datos; permitiendo determinar el uso de los recursos y los flujos de información dentro de la organización auditada. El Centro de Tecnologías de Gestión de Datos desarrolló el Sistema de Auditoría de Datos (AUDAT) para la Contraloría General de la República, que le permite a sus especialistas la detección de incidencias en la manipulación de las bases de datos de los sistemas auditados. Aun cuando el sistema fue entregado a los clientes cumpliendo con sus requerimientos y perspectivas, su explotación reveló que el empleo del editor de ecuaciones, utilizado en la mayoría de las funcionalidades relevantes para la auditoría de los datos, limitaba el aprovechamiento de las potencialidades que el sistema ofrecía. En aras de resolver la situación existente surge la investigación, con el propósito de desarrollar un constructor gráfico de consultas SQL para AUDAT, que permita su empleo por los auditores en el proceso de auditorías de datos.

Palabras clave: AUDAT, auditorías a bases de datos, constructor gráfico de consultas SQL

Abstract

Audits are an effective way to put in order the resources of the company to achieve better performance and productivity. The current technological development and the importance reached by the databases as computer systems support, have led most companies to store their information in these ones, been indispensable the creation of audit systems in databases. Audits to databases are responsible for controlling the data access, updating and integrity, allowing detecting the resources use and information flows within the audited organization. In the Data Technologies Center it was developed the Data Audit System (AUDAT), which allows to specialists of the Comptroller General of the Republic of Cuba the detection of incidents in the manipulation of the databases of audited systems. Although the system was delivered to the clients and meets their requirements and perspectives, its use showed that equations editor employment, used in most relevant features for auditing data, limited the exploitation of the potentials that the system offered. To solve this problem this investigation came up, with the propose of developing a graphical SQL query builder for AUDAT, to facilitate auditors their job when auditing data.

Keywords: *AUDAT, audit databases, graphical SQL query builder*

Índice de contenidos

Índice de figuras

Índice de tablas

Introducción

En una época altamente competitiva, con tanta demanda de productos y servicios y en un entorno económico convulso, organismos, empresas y entidades luchan por ofrecer bienes de calidad, que están estrechamente relacionados con una buena salud económica y financiera.

En este ámbito juegan un papel fundamental las auditorías como mecanismo para evaluar la eficiencia con que, estas empresas y entidades, desarrollan sus procesos y emplean sus recursos humanos y materiales. (1)

Las auditorías, además, contribuyen con la toma de decisiones, ya que permiten la identificación de riesgos y vulnerabilidades, facilitando la prevención de errores y fraudes económicos. Surgen ante la necesidad de las empresas de validar su información económica y financiera y se desarrollan como parte de un servicio propio o de una empresa independiente. Se definen como un "examen que se realiza sobre los registros patrimoniales[1] de un individuo o empresa a fin de verificar su estado financiero", resultando ser una manera efectiva de poner en orden los recursos del auditado para lograr un mejor desempeño y productividad. (2)

Esta práctica suele ser ejecutada por auditores y data de finales del siglo XVIII, aunque la proliferación de su uso comenzó con la aparición de las grandes empresas; siendo la tarea del auditor la de analizar la exactitud y veracidad de los registros mostrados por una empresa a fin de corregir errores e irregularidades. Actualmente existen dos tipos de auditorías: (3)

- Externas o legales: Aquellas desarrolladas por una empresa sobre sus propios suministradores o subcontratistas o que realiza un cliente sobre ella.
- Internas: Es la propia empresa quien investiga sus sistemas, procedimientos y actividades para cerciorarse que son adecuados y que se cumplen; proporcionan información acerca del cumplimiento de sus políticas, la eficiencia de sus sistemas y si se precisan cambios.

La realización de auditorías en las empresas influyen en que estas ofrezcan productos y servicios de calidad; constituyendo una herramienta que contribuye al mejoramiento de la eficiencia.

[1] Proviene de patrimonio: Conjunto de bienes, derechos y obligaciones de una unidad económica o empresa.

El desarrollo tecnológico actual y la importancia que han alcanzado las bases de datos, como soporte de los sistemas informáticos, han propiciado que la mayoría de las empresas almacenen su información a través de estas; siendo indispensable la creación de sistemas de auditorías sobre bases de datos que faciliten este proceso sobre los datos almacenados.

Las auditorías a bases de datos son las encargadas de controlar el acceso, la actualización, la integridad y calidad de los datos; permitiendo detectar, de forma sistemática, el uso de los recursos y los flujos de información dentro de una organización, determinar qué información es crítica para el cumplimiento de su misión y objetivos e identificar necesidades, duplicidades, costos, valor y barreras que obstaculicen flujos de información eficientes. (4)

Actualmente existen varias herramientas que son utilizadas para las auditorías en bases de datos según sus especificaciones o el uso que se les dé, algunas de las más populares son *ManageEngine EventLog Analyzer, PowerBroker Databases, SecureSphere Database Security* e *InfoSphere Guardium*. (5)

En Cuba, el órgano encargado de la realización de auditorías a empresas y entidades de sus procesos, incluidas sus bases de datos, es la Contraloría General de la República (CGR), creada en el 2009 en el marco de la Asamblea Nacional del Poder Popular mediante la Ley No. 107 “De la Contraloría General de la República” como resultado de un proceso de fortalecimiento de la Entidad Fiscalizadora Superior. Este tiene las misiones de auxiliar a la Asamblea Nacional y al Consejo de Estado en la ejecución de la más alta fiscalización sobre los órganos del Estado y Gobierno; proponer la política integral en materia de preservación de las finanzas públicas y el control económico-administrativo, dirigiendo, ejecutando y comprobando su cumplimiento; dirigir metodológicamente y supervisar el sistema nacional de auditoría; ejecutar las acciones necesarias con el fin de velar por la correcta y transparente administración del patrimonio público y; prevenir y luchar contra la corrupción. (6)

Cuatro años después de su creación, la CGR solicitó a la Universidad de las Ciencias Informáticas (UCI), institución educativa también dedicada a la producción de *software* en Cuba, el desarrollo de una solución informática que facilitara a sus auditores el proceso de auditorías de bases de datos, realizado en empresas e instituciones de forma manual o con el empleo de una versión obsoleta del sistema propietario

Data Analysis Software (IDEA[2]). El Centro de Tecnologías de Gestión de Datos (DATEC), perteneciente a esta institución, desarrolló el Sistema de Auditoría de Datos (AUDAT), que facilita la detección de incidencias en la manipulación de las bases de datos de los sistemas auditados. AUDAT proporciona un aumento en la calidad del proceso auditor, optimizando el trabajo al minimizar la carga manual y, por ende, facilitando la toma de decisiones durante dicho proceso.

Este *software* tiene un alto impacto tanto económico como social, ya que al facilitar el proceso de realización de auditorías de datos a empresas estatales y no estatales que pueden ser auditadas por la Contraloría, contribuye a la lucha contra las ilegalidades que pudieran materializarse.

Aun cuando el sistema fue entregado a los clientes cumpliendo con sus requerimientos y perspectivas, su explotación por los auditores, tanto contables como informáticos, reveló que el empleo del editor de ecuaciones, utilizado en la mayoría de las funcionalidades relevantes para la auditoría de los datos, limitaba el aprovechamiento de las potencialidades que el sistema ofrece, acarreando una serie de problemas entre los que destacan que las consultas realizadas sobre los registros de las bases de datos importadas:

- Soportan solamente operadores matemáticos, impidiendo la realización de análisis que involucren otros operadores.
- Se realizan sobre tablas, por lo que para realizar acciones sobre el resultado de una operación previa no se puede emplear el editor de ecuaciones.
- Se pueden realizar solo sobre una tabla, impidiendo la realización de análisis más complejos que involucren varias tablas.

Producto de la situación existente surge el presente estudio, identificando como **problema de investigación** que las opciones de escritura de consultas que brinda el editor de ecuaciones de AUDAT limitan su empleo por los auditores, en los procesos de auditorías de datos llevados a cabo por la Contraloría General de la República.

[2] IDEA: *Software* de análisis y auditoría de bases de datos; permite leer, visualizar, analizar, manipular, obtener muestras y extraer datos desde archivos provenientes de múltiples orígenes de datos. Pertenece a la serie de *software* de IBM y su licencia excede los 1000 USD (57).

Definiéndose como **objeto de estudio** los constructores de consultas SQL y enmarcándose en **el campo de acción** los constructores gráficos de consultas SQL; ya que para dar solución al problema de la investigación se determinó como **objetivo general** desarrollar un constructor gráfico de consultas SQL para AUDAT que permita su empleo por los auditores, en los procesos de auditorías de datos llevados a cabo por la Contraloría General de la República.

Para darle cumplimiento al objetivo general, se desglosa en los siguientes **objetivos específicos**:

- Determinar si existe un constructor gráfico de consultas SQL que pueda ser integrado a AUDAT, así como las tecnologías necesarias para el desarrollo de la solución.
- Implementar la solución para que AUDAT cuente con un constructor gráfico de consultas SQL, que permita su empleo por los auditores.
- Demostrar mediante la realización de pruebas el correcto funcionamiento de la solución desarrollada.

Además de los objetivos específicos se plantea el problema a través de una serie de **preguntas de investigación**:

- ¿Cuáles son las características y principales funcionalidades de los constructores gráficos de consultas SQL existentes?
- ¿Cuál de los constructores gráficos de consultas SQL existentes pudiera considerarse la solución a integrar a AUDAT?
- ¿Cuáles son las tecnologías y herramientas necesarias para que AUDAT cuente con un constructor gráfico de consultas SQL?
- ¿Qué funcionalidades se deben implementar para que AUDAT cuente con un constructor gráfico de consultas SQL?
- ¿Qué pruebas son más factibles aplicar para la validación del constructor gráfico de consultas SQL con que contará AUDAT?

Para el logro de los objetivos propuestos y dar respuesta a las preguntas de investigación, se definen las siguientes **tareas de la investigación**:

- Caracterización de los constructores gráficos de consultas SQL existentes y su posible integración a AUDAT.

- Selección de la metodología, herramientas y tecnologías a utilizar para el desarrollo de la solución que permita a AUDAT contar con un constructor gráfico de consultas SQL.
- Diseño, según la metodología seleccionada, de la solución para que AUDAT cuente con constructor gráfico de consultas SQL.
- Implementación de la solución para que AUDAT cuente un constructor gráfico de consultas SQL.
- Definición de una estrategia de pruebas que garantice la calidad del constructor gráfico de consultas SQL para AUDAT.
- Ejecución de las pruebas para garantizar la calidad de la solución del constructor gráfico de consultas SQL para AUDAT.

Para el desarrollo de la investigación se utilizaron los métodos científicos[3] siguientes:

- Analítico-Sintético: Para analizar los conceptos teóricos existentes sobre los constructores gráficos de consultas SQL y determinar la solución a integrar a AUDAT.
- Histórico-Lógico: Ya que la investigación está basada en el estudio de los constructores gráficos de consultas SQL, sus antecedentes, las herramientas que realizan estos procesos y las tecnologías necesarias para implementar una solución para AUDAT.
- Modelación: A partir del cual se representa la realidad existente modelando la necesidad actual en un diagrama de clases del dominio, facilitando una mejor comprensión del objeto de estudio; además para la modelación mediante diagramas de clases de la aplicación de los patrones de diseño.

La investigación está estructurada de la siguiente forma:

- Capítulo 1 Fundamento teórico sobre los constructores gráficos de consultas: Aborda los conceptos fundamentales para la investigación, asociados principalmente con los constructores gráficos de consultas SQL existentes, sus características y funcionalidades, así como el estudio de las tecnologías para el desarrollo de la solución, seleccionándose las más indicadas para la misma.
- Capítulo 2 Características y diseño del constructor gráfico de consultas para AUDAT: Aborda los elementos técnicos relacionados con el diseño del constructor gráfico de consultas SQL. Se

[3] Tienen su sustento en la concepción materialista dialéctica y permiten una mejor recopilación de la información relacionada con la realización de estudios relacionados con los constructores gráficos de consultas.

describe el diagrama de clases del dominio, las historias de usuario, los requisitos a tener en cuenta para su desarrollo, agrupados en la lista reserva del producto, las tarjetas Clase-Responsabilidad-Colaboración, el estilo y patrón arquitectónico y los patrones de diseño utilizados.

- Capítulo 3 Implementación y pruebas del constructor gráfico de consultas para AUDAT: Aborda los elementos técnicos relacionados con la implementación del sistema; para darle solución a las historias de usuario definidas en el capítulo anterior se desarrollan las tareas de ingeniería, se define el estándar de codificación y se especifica la estrategia de pruebas a la que fue sometida la aplicación como parte de su validación.

1 Fundamento teórico sobre los constructores gráficos de consultas

En el capítulo se realiza un estudio de los constructores gráficos de consultas SQL existentes para determinar la solución idónea a integrar al Sistema de Auditoría de Datos, con el fin de facilitar su empleo por los auditores. Para ello, además, se analizan las herramientas y tecnologías necesarias a emplear para el desarrollo de la solución, determinándose finalmente aquellas a utilizar.

1.1. Editor de consultas

Editor, del latín *editor*, es aquel o aquello que permite editar. El verbo editar se refiere a publicar algo a través de algún medio o a corregir o adaptar una obra de acuerdo a ciertas reglas y normas. En el área de la informática, un editor es una aplicación que permite modificar, crear o almacenar archivos informáticos. (7)

Por su parte, el término consulta procede del latín *consulere* que puede traducirse como “pedir consejo”. La acción y efecto de consultar se conoce como consulta. El verbo permite referirse a examinar un asunto con una o más personas o buscar datos sobre algún asunto o materia en particular. Quien consulta espera obtener información de utilidad para satisfacer sus necesidades o cumplir sus objetivos. (8)

En la informática una consulta sobre bases de datos está compuesta por un conjunto de instrucciones en un lenguaje específico de manipulación de datos con el que se puede seleccionar, ordenar, filtrar, mostrar, modificar, añadir y borrar datos. (9)

Para el caso de las bases de datos de tipo relacional[4] uno de los lenguajes más empleados para la realización de consultas es SQL (por sus siglas en inglés *Structured Query Language*), lenguaje declarativo[5] de acceso a bases de datos que permite su manipulación y control. Está basado en el álgebra

[4] Una base de datos relacional cumple con el modelo relacional, modelo ampliamente usado en la actualidad, definido por Edgar Frank Codd en 1960, se trata de un modelo lógico que establece una estructura sobre los datos, donde estos son organizados y representados en formas de tablas y relaciones. (55)

[5] Especifican o declaran un conjunto de condiciones, proposiciones, restricciones o transformaciones que describen el problema, los cuales indican a la computadora qué es lo que se desea obtener o qué es lo que se está buscando. (56)

y el cálculo relacional para efectuar consultas con el fin de recuperar, de forma sencilla, información de las bases de datos y realizar cambios en ellas. (10)

Las consultas SQL están compuestas por comandos, cláusulas, operadores y funciones de agregado que se combinan y constituyen las instrucciones para crear, actualizar y manipular las bases de datos; suelen comenzar con una palabra clave de acción o comando que especifica la operación que se desea realizar. (11)

Teniendo en cuenta los conceptos analizados y el objetivo de la investigación, se considera que un editor de consultas es un programa que permite la definición de estas con el fin de manipular los datos de bases de datos mediante un lenguaje de consultas determinado.

1.2. Constructores gráficos de consultas SQL

Producto al amplio uso de las bases de datos de tipo relacional existen programas exclusivamente dedicados a tratar con bases de datos relacionales, los cuales se conocen como Sistemas de Gestión de Bases de Datos Relacionales (SGBDR). Entre los gestores más populares, de estos tipos, se encuentran *Oracle*, *SQL Server*, *MySQL, PostgreSQL, SQLite* y DB2, dando lugar a que exista una amplia proliferación de constructores gráficos de este tipo. (12)

Los constructores gráficos de consultas SQL son herramientas que facilitan la consulta de los datos existentes en una base de datos mediante su construcción, edición y ejecución de forma gráfica. Las tablas almacenadas se muestran en un área de diseño, brindando la posibilidad de seleccionar y relacionar los campos de los cuales se desea obtener información. Generalmente estos sistemas se desarrollan para satisfacer a un usuario con poco dominio del SQL. (13)

Los constructores gráficos de consultas permiten crear consultas simples o complejas sin necesidad de tener conocimientos avanzados de SQL, ya que a través del editor visual se seleccionan las tablas de las que se requiere la información y se especifican las condiciones que deben cumplir las tuplas resultantes, generándose la consulta con la posibilidad de que pueda ser modificada mediante código. Generalmente los constructores gráficos de consultas SQL están compuestos por cuatro paneles. (Ver figura 1) (14)

- Diseño: Muestra las tablas a consultar con sus atributos y las relaciones entre ellas.
- Criterios: Permite definir condiciones que deben cumplir las tuplas resultantes de la consulta, como qué columnas y filas se van a mostrar y cómo se ordenará el resultado.

- SQL: Muestra el código de la consulta que se irá conformando en el panel Diseño o el que puede ir escribiendo de forma manual el propio usuario.
- Resultados: Muestra el conjunto de resultados de la consulta creada y ejecutada.

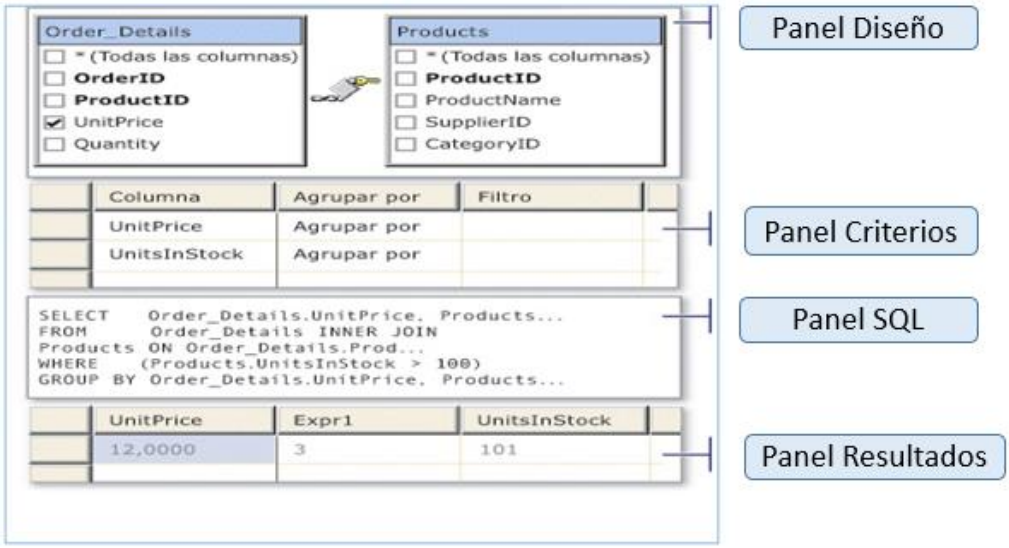


Fig. 1 Componentes de los constructores gráficos de consultas SQL

Entre las ventajas más importantes que propicia el empleo de los constructores gráficos de consultas SQL, se destacan que: (15)

- Los usuarios pueden seleccionar tablas específicas de forma sencilla.
- Proporcionan una representación visual de las tablas.
- Visualizan los resultados de las consultas.
- Permiten generar el código SQL a partir de lo creado visualmente.
- No es necesario tener conocimientos avanzados sobre SQL para crear consultas.

Por tanto, los constructores gráficos de consultas SQL tienen como fin permitir el acceso a los datos de las bases de datos, mediante una interfaz visual que facilite, de forma intuitiva, la especificación de opciones para la obtención de los resultados deseados.

Los epígrafes siguientes muestran las características principales con que cuentan los constructores gráficos de consultas SQL de las herramientas de administración de bases de datos más difundidas, con el fin de recopilar las características esenciales con que debe contar el Constructor Gráfico de Consultas para AUDAT.

1.2.1. Constructor gráfico de consultas del PgAdmin

PgAdmin es una herramienta de código abierto para la administración de bases de datos PostgreSQL, incluye una interfaz gráfica administrativa, un editor de código procedural y un agente de planificación SQL. Está diseñada para responder a las necesidades de la mayoría de los usuarios en el desarrollo de consultas simples y complejas, está disponible en más de una docena de idiomas y en varios sistemas operativos incluyendo Microsoft Windows, Linux, Mac OS y Solaris. (16)

Esta herramienta cuenta con una pestaña para el diseño de consultas de forma visual (ver figura 2) que permite la realización solamente de consultas de tipo SELECT con la posibilidad de emplear las cláusulas WHERE, GROUP BY y ORDER BY, el soporte de concatenaciones de tipo natural sin opción de elegir otros tipos y, la eliminación de tablas.

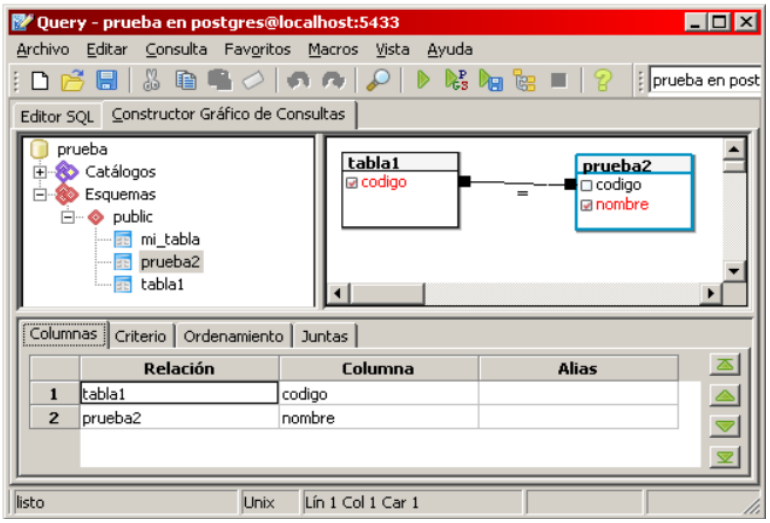


Fig. 2 Constructor gráfico de consultas de la herramienta PgAdmin

1.2.2. Constructor gráfico de consultas del Generador Dinámico de Reportes

El Generador Dinámico de Reportes (GDR) es una aplicación web, desarrollada en la UCI, para la gestión de la información, que permite la confección de reportes de manera dinámica y, que pueden contribuir a la toma de decisiones de las empresas e instituciones que lo utilicen. Posibilita diseñar reportes tabulares (con gráficos incluidos), tablas pivote y cruzadas desde los gestores de bases de datos *SQL Server*, *SQLite*, *Oracle*, *MySQL* y *PostgreSQL*. Dicha herramienta permite a los usuarios abstraerse de los conocimientos relacionados con los gestores de bases de datos y generar reportes en varios formatos con

gran variedad de opciones en su diseño, marcando una diferencia entre los reportes tradicionales y los dinámicos. (17)

Este sistema está compuesto por 6 módulos, entre los que se encuentra un diseñador de consultas que funciona como un constructor gráfico, permitiendo crear, editar y ejecutar consultas de tipo SELECT sin necesidad de conocer el gestor de bases de datos. Entre las principales funcionalidades que soporta, destacan que permite: (17)

- Diseñar consulta visualmente, mediante las opciones de arrastrar y soltar se puede colocar libremente dentro del área de diseño las tablas y/o vistas de las que se extraerán los datos requeridos; para seleccionar los campos a consultar basta con marcarlos dentro de la tabla a la que pertenecen.
- Modificar consulta visualmente, mediante la carga de la consulta previamente diseñada para su modificación.
- Incluir funciones de agregación, mediante la adición de funciones de agregación (suma, cantidad, máximo, mínimo y promedio) a campos de la consulta.
- Relacionar tablas, mediante el establecimiento de relaciones entre las tablas a través de sus campos y haciendo uso de las opciones de concatenación INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL JOIN.
- Incluir condiciones a la consulta, mediante la adición de criterios que deben cumplir los valores de los atributos pertenecientes al resultado de la consulta.
- Soportar operadores de comparación, mediante el uso de “igual a”, “mayor que”, “menor que”, “mayor-igual a”, “menor-igual a” y “distinto a”.
- Establecer criterios de agrupamiento, mediante la agrupación de los datos de la salida de la consulta seleccionando el o los campos por los cuales se desea agrupar.
- Establecer criterios de ordenamiento, mediante el establecimiento del orden de los datos de la salida de la consulta, seleccionando el o los campos por los cuales se desea ordenar.
- Ejecutar consulta diseñada, mediante la ejecución de la consulta, mostrando sus resultados y la sentencia SQL generada.
- Limitar el resultado de la consulta, mediante la especificación de la cantidad de registros a obtener.

1.2.3. Constructor gráfico de consultas del EMS SQL Manager para PostgreSQL

El *EMS SQL Manager* para *PostgreSQL* es una herramienta para la administración de bases de datos implementadas para este gestor. Es funcional para cualquier versión de *PostgreSQL* y ofrece una amplia gama de herramientas tales como un diseñador visual de bases de datos para crearlas y un constructor gráfico para realizar consultas complejas. Su interfaz gráfica resulta ser una de las más intuitivas y cómodas para los usuarios. (18)

El constructor gráfico de esta herramienta permite crear, de forma sencilla y simple, consultas SQL mediante una interfaz visual amigable al usuario, que le permite conectarse a la base de datos, seleccionar tablas y campos para una consulta y fijar un criterio de selección de las tuplas resultantes. Permite, además, trabajar con varias consultas de forma simultánea, visualizar los resultados de ejecución de las mismas en diferentes tipos de salidas y manipular los datos mostrados.

1.2.4. Constructor gráfico de consultas de HABD

HABD (Herramienta de Administración de Bases de Datos), es una herramienta de administración de bases de datos desarrollada en DATEC (en la UCI). Cuenta con un editor que permite a los usuarios la ejecución de consultas en el panel de código con opciones de autocompletamiento y de escritura de comentarios; permite visualizar los resultados o errores de las consultas ejecutadas guardadas en el historial; además, tiene un constructor gráfico de consultas donde se pueden crear tablas, sus relaciones, sus datos y generar el código automáticamente de toda la representación visual sin tener conocimientos previos de SQL. (19)

El constructor gráfico de consultas de esta herramienta permite la confección de consultas de manera visual, donde se escogen las tablas a utilizar para la realización de la consulta, permitiendo la selección de los atributos deseados. Permite, además, al dar clic derecho sobre la tabla que se desea gestionar, su edición o eliminación, posibilitando insertar datos, actualizarlos o eliminarlos y, la especificación en la confección de consultas SELECT de las cláusulas WHERE, JOIN y ORDER BY. (19)

El estudio de estas herramientas permitió la identificación de un grupo de funcionalidades que proveen, en mayor o menor medida, para cumplir con el objetivo de facilitar la generación de consultas de manera intuitiva. La tabla siguiente muestra una comparación entre las herramientas analizadas en el cumplimiento de dichas funcionalidades, con el fin de poder determinar la idoneidad de alguna para ser integrada a AUDAT.

Tabla 1 Funcionalidades principales de los constructores gráficos de consultas SQL

Funcionalidad	Constructor			
	PgAdmin	GDR	EMS	HABD
Definir consultas SELECT	Sí	Sí	Sí	Sí
Establecimiento de condiciones	Sí	Sí	Sí	Sí
Opciones de agrupamiento	Sí	Sí	Sí	Sí
Opciones de ordenamiento	Sí	Sí	Sí	Sí
Definir consultas INSERT	No	No	Sí	Sí
Definir consultas UPDATE	No	No	Sí	Sí
Definir consultas DELETE	No	No	Sí	Sí
Realizar concatenaciones	Sí	Sí	Sí	Sí
Elegir tipo de concatenación	No	Sí	Algunas	Algunas
Eliminar tablas	Sí	No	Sí	Sí
Soportar estructuras no relacionales	No	No	No	No
Permitir reusabilidad	No	Sí	No	Sí
Soportar diferentes gestores	No	Sí	No	No

Como muestra la tabla 1, los constructores gráficos analizados no pueden ser integrados a AUDAT porque solamente soportan operaciones sobre bases de datos relacionales que no incluyen a *SQLite*, que es el gestor con el que trabaja AUDAT. Sin embargo, pudiera valorarse la idea de integrar uno de los existentes y desarrollarle el soporte al mismo, pero:

- *PgAdmin*: Está desarrollado en C++ lo que dificulta su reutilización debido a que AUDAT está desarrollado en *Java*.
- GDR: Fue desarrollado en la Universidad, su código puede solicitarse así como asesoría para su entendimiento, pero está desarrollado sobre tecnologías web, por lo que no se acopla a la arquitectura definida para AUDAT, lo que haría más engorrosa su integración.
- EMS SQL *Manager for PostgreSQL*: Es una herramienta propietaria por lo que no se tiene acceso a su código.

- HABD: Fue desarrollado en la Universidad, su código puede solicitarse así como asesoría para su entendimiento, pero integrarlo a AUDAT y desarrollar el soporte a *SQLite* resultaría más trabajoso que desarrollarle un módulo propio a AUDAT.

Por tanto, se hace necesario desarrollar un constructor gráfico para AUDAT que permita de forma intuitiva la realización de consultas SQL. No obstante, el estudio realizado permitió la identificación de funcionalidades a incluir en la solución como:

- Crear consultas de tipo SELECT mediante la agregación visual de criterios, tales como condiciones y opciones de ordenamiento.
- Concatenar tablas haciendo usos de los operadores “=”, “>”, “>=”, “<” y “<=”.

1.3. Metodología de desarrollo de software

Las metodologías de desarrollo permiten guiar el proceso de desarrollo de un *software* mediante procedimientos, técnicas, herramientas y soporte documental. Hoy día existen una serie de metodologías que pueden dividirse en dos grandes grupos: (20)

- Las metodologías formales: Orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas y notaciones a utilizar.
- Las metodologías ágiles: Orientadas a la interacción con el cliente y el desarrollo incremental del *software*, mostrando versiones parcialmente funcionales al cliente en intervalos cortos de tiempo para ir probándolas y corrigiéndolas según se desee.

Para seleccionar el tipo de metodología a emplear para el desarrollo del Constructor Gráfico de Consultas se tomó como punto de partida el modelo propuesto por *Barry Boehm* y *Richard Turner* en su libro *Balancing Agility and Discipline*, que posibilita elegir uno de los dos tipos de metodologías de acuerdo a las características del equipo y el *software* a desarrollar. (21)

La figura 3 muestra el área resultante de otorgarle valores aproximados a las variables críticas definidas en el modelo por *Boehm* y *Turner* de acuerdo a las características del proyecto a desarrollar para la propuesta de solución, donde:

- Tamaño: El equipo de desarrollo para la implementación del Constructor Gráfico de Consultas está conformado por una persona.
- Criticidad: La criticidad del sistema pudiera considerarse baja teniendo en cuenta que la solución

no es indispensable para el funcionamiento correcto de AUDAT.

- Personal: El desarrollador no es experto, posee conocimientos, que pudieran considerarse medios, de las tecnologías a emplear para el desarrollo de la solución.
- Dinamismo: Es pequeño ya que se parte de una captura de requisitos poco variable.
- Cultura: Estrechamente relacionada con la variable Personal, mientras más capacitado y experto sea, más alto será el porciento de adaptación a cambios en el proceso de desarrollo de *software*.

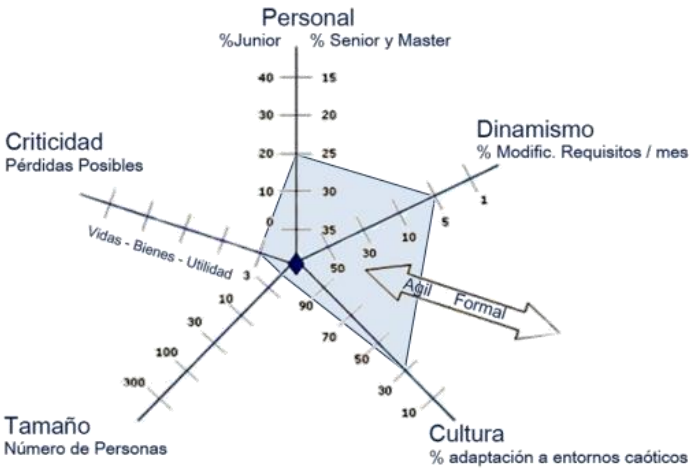


Fig. 3 Modelo de Boehm y Turner ajustado a las condiciones de desarrollo de la solución propuesta

Como se evidencia, el área de expansión en la figura se acerca al origen de coordenadas, y aun cuando de las cinco variables hay tres en un término medio, que pudiera decidirse por cualquiera de los dos tipos de metodologías, se decide emplear una de tipo ágil para el desarrollo de la propuesta de solución ya que pesa más el tamaño del equipo del proyecto unido al poco tiempo de desarrollo.

1.3.1. Extreme Programming (XP)

La metodología XP (Programación Extrema) es uno de los procesos ágiles de desarrollo de *software* más utilizados en la actualidad. La Programación Extrema se diferencia de las metodologías tradicionales principalmente en que se enfoca más en la adaptabilidad que en la previsibilidad. XP es capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto, siendo una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en ellos. (22)

XP promueve valores como simplicidad, comunicación y retroalimentación, está basada en diferentes ideas acerca de cómo enfrentar el desarrollo para obtener un *software* funcional, el cliente puede añadir nuevas funcionalidades o quitarlas aumentando su participación y, es mucho más fácil de aplicar y de aprender, por lo que equipos sin experiencia pueden incorporarla.

1.4. Herramientas y tecnologías necesarias para el desarrollo de la solución

Con el objetivo de desarrollar el Constructor Gráfico de Consultas con la calidad requerida, se realizó un estudio de las tecnologías y herramientas necesarias para su implementación, seleccionándose finalmente aquellas a utilizar.

1.4.1. Herramientas y tecnologías utilizadas por AUDAT

Al formar el Constructor parte de AUDAT, para su desarrollo se emplearán herramientas y tecnologías previamente utilizadas en la implementación de su versión 1.0, entre las que destacan el lenguaje de programación y el sistema de gestión de bases de datos empleados.

Lenguaje de programación Java

Un lenguaje de programación permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis, que le permiten al programador la comunicación con los dispositivos de *hardware* y *software* existentes; es aquella estructura que, con una cierta base sintáctica y semántica, imparte distintas instrucciones a un programa de computadora. (23)

Existen varios lenguajes de programación, siendo uno de los más empleados *Java*. *Java* es un lenguaje de programación interpretado y compilado, desarrollado por *Sun Microsystems*, que fue presentado en la segunda mitad del año 1995 y desde su génesis se ha convertido en un lenguaje de programación muy popular. Es un lenguaje de desarrollo de propósito general y, como tal, se emplea para realizar todo tipo de aplicaciones de escritorio. Es muy valorado porque los programas en *Java* se pueden ejecutar en diversas plataformas con sistemas operativos como *Windows*, *Mac OS*, *Linux* o *Solaris*. (24)

Este lenguaje posee una combinación de características que lo hacen una opción valorada, entre ellas destacan que es: (25)

- Sencillo, elimina la complejidad de otros lenguajes.

- Orientado a objetos, la filosofía de la programación orientada a objetos facilita la creación y mantenimiento de programas.
- Independiente de la arquitectura y portable, al compilar un programa en *Java* el código resultante es un tipo de código binario conocido como *Java Bytecode*[6], código interpretado por diferentes computadoras, por lo que un programa compilado puede ser utilizado por cualquier computadora.
- Robusto, simplifica la gestión de la memoria.
- Multitarea, se pueden ejecutar diferentes bloques de código al mismo tiempo.
- Dinámico, no es necesario cargar completamente el programa en memoria, sino que las clases compiladas pueden ser cargadas bajo demanda en tiempo de ejecución.
- Portable, los programas se compilan a un lenguaje intermedio denominado *Bytecode;* este código es interpretado por la máquina virtual de *Java* del entorno de ejecución (JRE), lográndose la portabilidad en distintas plataformas.
- Gratuito, liberado bajo la licencia GPL[7].

Para el desarrollo del Constructor Gráfico de Consultas para AUDAT se utiliza este lenguaje ya que, además de las características mencionadas previamente que lo hacen ser un lenguaje potente, es el lenguaje en el que está implementado AUDAT, siendo un factor determinante para garantizar la compatibilidad del Constructor con la aplicación.

Sistema de gestión de bases de datos SQLite 3.0

Un sistema de gestión de bases de datos (SGBD) es un *software* de propósito general que facilita la definición, construcción y manipulación de las bases de datos para distintas aplicaciones. (26)

El objetivo fundamental de un SGBD es proporcionar eficiencia y seguridad a la hora de almacenar y extraer datos de las bases de datos. Están diseñados para gestionar grandes bloques de datos, lo que implica tanto la definición de estructuras para el almacenamiento como de mecanismos para su gestión.

SQLite es un sistema de gestión de bases de datos relacional liberado bajo licencia GPL, contenida en una librería programada en C, que implementa un completo motor de bases de datos multiplataforma.

[6] Código intermedio entre el código fuente y el código de máquina, es una forma de salida utilizada por los programadores para reducir la dependencia con respecto al *hardware* y facilitar su interpretación.

[7] GPL es la Licencia Pública General de GNU o más conocida en inglés GNU *General Public License,* es la más usada en el mundo del *software* libre y garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el *software*.

Permite utilizar un amplio subconjunto del estándar SQL (implementa el SQL92) y se destaca por su versatilidad.

Combina el motor y la interfaz de la base de datos en una única biblioteca y almacena los datos en un único archivo de texto plano. Esto hace que cada usuario pueda crear tantas bases de datos como desee sin necesidad de la intervención de un administrador de bases de datos que gestione los espacios de trabajo, usuarios y permisos de acceso. Su potencia se basa fundamentalmente en la simplicidad, lo que hace que no sea una buena solución en entornos de tráfico elevado y/o alto acceso concurrente a los datos. Entre sus principales características destacan que: (27)

- Es un sistema completo de bases de datos que soporta múltiples tablas, índices, disparadores y vistas.
- No necesita un proceso separado funcionando como servidor ya que lee y escribe directamente sobre archivos que se encuentran en el disco duro.
- El formato de la base de datos es multiplataforma e indistintamente se puede utilizar el mismo archivo en sistemas de 32 y 64 *bits*.

Para la implementación de la solución se utiliza *SQLite* en su versión 3.0 porque, además de las características que lo hacen un gestor válido, AUDAT lo tiene como soporte, garantizándose con su empleo la compatibilidad del editor con la solución ya desarrollada y, evitándose la introducción de una nueva herramienta. Además de que *SQLite* no utiliza un servidor para gestionar la información, se puede trabajar de manera independiente y no se necesitan muchos recursos de *hardware* para su empleo, lo que puede contribuir con la forma en que trabajan los auditores de la Contraloría, que realizan los procesos de auditorías con una laptop en la entidad o empresa auditada.

1.4.2. Lenguaje de modelado

Para modelar el análisis y diseño se decidió utilizar el Lenguaje Unificado de Modelado (UML), el cual está soportado por la metodología elegida. UML, en su versión 2.0, es un lenguaje gráfico que permite: (28)

- Visualizar, especificar, construir y documentar un sistema de *software*.
- Ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema y aspectos concretos como

expresiones de lenguajes de programación, esquemas de bases de datos y componentes de *software* reutilizables.

- Está pensado para usarse durante todas las etapas del ciclo de vida del *software*.

Para la presente investigación se decide utilizar UML como lenguaje de modelado pues es uno de los modelos más utilizado y referenciado en la actualidad. (29)

Entre sus principales ventajas destacan que permite: (30)

- Modelar sistemas utilizando conceptos orientados a objetos.
- Hacer uso de un lenguaje de modelado utilizado tanto por humanos como por máquinas.
- Dar un mejor soporte a la planeación y control de proyectos.
- Ofrecer una alta reutilización.

1.4.3. Herramienta CASE

Se puede definir como herramienta CASE (*Computer Aided Software Engineering*, Ingeniería Asistida por Computadora) a una serie de tecnologías que ayudan al ingeniero de *software* a desarrollar y mantener la aplicación durante una o más fases de su ciclo de vida. (31)

Las herramientas CASE informatizan algunos de los procesos llevados a cabo en estas fases, influyendo en la productividad y calidad del producto final. Entre las más usadas se encuentran *Rational Rose* y *Visual Paradigm*.

Visual Paradigm Community Edition 12.1

Es una versión comunitaria del *Visual Paradigm*, siendo este un *software* de modelado UML que permite analizar, diseñar, codificar, probar y desplegar aplicaciones informáticas. Soporta diferentes tipos de diagramas UML, genera código fuente a partir de dichos diagramas y posibilita la elaboración de documentos. Entre sus características fundamentales destacan que soporta: (32)

- Compatibilidad con UML.
- Diagrama de flujo de datos.
- Editor de detalles de los casos de uso.

Visual Paradigm para UML es una plataforma de desarrollo visual, compatible con el ciclo de vida completo del desarrollo integral del *software*, desde el análisis y diseño hasta la realización de pruebas y su posterior despliegue.

Rational Rose 7.0

Es una herramienta para el modelado visual mediante UML de sistemas. Permite especificar, analizar y diseñar el sistema antes de codificarlo. Garantiza un desarrollo más rápido de aplicaciones ya que provee un entorno de modelado que permite agilizar este proceso. Algunas de sus características más relevantes son que: (33)

- Chequea la sintaxis UML.
- Genera documentación automáticamente.
- Soporta ingeniería inversa (crea modelos a partir de código).
- Está disponible en múltiples plataformas.

Fundamentación de la herramienta CASE seleccionada

Ambas herramientas proveen funcionalidades necesarias para el desarrollo del sistema ya que soportan totalmente UML y, además, facilitan la gestión de los requerimientos de *software*. Sin embargo *Rational Rose* tiene la desventaja de ser una herramienta propietaria, lo que anula los principios de soberanía tecnológica por los que aboga el país, siendo la opción a utilizar en el desarrollo del sistema la versión comunitaria de *Visual Paradigm*, que puede ser utilizada libre de costos en proyectos sin carácter comercial.

1.4.4. Entorno de desarrollo integrado

Un entorno de desarrollo integrado, conocido como IDE (del inglés *Integrated Development Environment*), es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. En él se encuentran como mínimo un editor, un compilador, un intérprete y un depurador de uno o varios lenguajes de programación. Un IDE ofrece un marco de trabajo para la mayoría de los lenguajes de programación tales como C++, *Python*, *Java*, C#, *Delphi*, *Visual Basic*, PHP, etcétera; brinda la posibilidad de poder integrarse con un sistema de control de versiones y facilita la construcción de interfaces gráficas de usuarios. (34)

Netbeans 7.4

Netbeans es un entorno de desarrollo de código abierto y gratuito sin restricciones de uso. Abarca un amplio rango de tecnologías de desarrollo tanto para escritorio como aplicaciones web o dispositivos móviles. Da soporte a varios lenguajes de programación como *Java*, PHP, *Groovy*, C/C++, HTML5, entre otros. Además puede instalarse en varios sistemas operativos como *Windows*, *Linux* y *Mac OS.* (35)

Este IDE está desarrollado principalmente para el lenguaje de programación *Java*, ha alcanzado un gran éxito y cuenta con una amplia base de usuarios y una comunidad en constante crecimiento. Es una eficaz herramienta tanto para desarrolladores profesionales como para los que acaban de iniciarse en él.

Entre sus características más relevantes destacan que: (36)

- Es extensible, se le puede agregar fácilmente *plugins*[8].
- Permite el acceso a bases de datos, desde el propio IDE se pueden establecer conexiones a distintos sistemas de gestión de bases de datos.
- Permite la optimización de código, ayuda a la optimización de las aplicaciones e intenta hacer que se ejecuten más rápido y con el mínimo uso de memoria.
- Cuenta con herramientas para el depurado de errores.
- Permite la navegación por el código y el proyecto, contando con vistas del proyecto especializadas, vistas de estructura de archivos y salto rápido entre archivos, clases y métodos.

Eclipse Luna 4.4.1

Eclipse es una herramienta de desarrollo multiplataforma, diseñada para ser extendida a través de *plugins*, que cuenta con un *widget* para *Java* llamado *Standard Widget Toolkit* (SWT) que facilita la navegación por la interfaz visual. Es un proyecto de código abierto y gratuito, con una gran base de usuarios y documentación disponible. Permite el desarrollo de *software* utilizando otros lenguajes de programación como C/C++ y *Python*. Fue liberado bajo la licencia *Eclipse Public License* (Licencia Pública de Eclipse), donde los receptores de esta herramienta pueden utilizar, modificar, copiar y distribuir el trabajo sin costo alguno.

[8] Es una aplicación que se relaciona con otra para aportarle una funcionalidad nueva y generalmente muy específica.

Este IDE provee herramientas para administrar áreas de trabajo, construir, lanzar y depurar aplicaciones, donde sus características fundamentales son: (37)

- IDE que permite el uso de *plugins*.
- Incluye un editor de textos con opciones para el resaltado de sintaxis.
- Compilación en tiempo de edición (para detectar y corregir errores).
- Refactorización del código.

Fundamentación del IDE seleccionado

Netbeans y *Eclipse* son dos IDE similares para el trabajo con *Java*, y aunque ambos son opciones válidas, se decide utilizar *Netbeans* por ser la herramienta dominada por el equipo de desarrollo.

Conclusiones del capítulo

En el capítulo se realizó un estudio bibliográfico de los constructores gráficos de consultas SQL de aquellos clientes de administración para *PostgreSQL* con mayor aceptación y los desarrollados en la Universidad, analizándose sus características y funcionalidades principales y evidenciándose que no pueden ser integrados a AUDAT por no cumplir con los requerimientos necesarios, de ahí la necesidad de la implementación de un nuevo constructor gráfico de consultas SQL para AUDAT. Para ello, se determinó utilizar como metodología de desarrollo de *software* XP, como lenguaje de modelado UML 2.0 y herramienta de modelado *Visual Paradigm Community Edition* 12.1; para la implementación de la solución fueron seleccionados como lenguaje de programación *Java*, como gestor de bases de datos *SQLite* 3.0 y como IDE de desarrollo *Netbeans* 7.4.

2 Características y diseño del Constructor Gráfico de Consultas para AUDAT

El diseño de un producto es fundamental para el desarrollo exitoso del mismo, durante esta fase se definen, revisan y documentan sus requisitos y los detalles procedimentales para su implementación. En este capítulo se diseña el Constructor Gráfico de Consultas para AUDAT; para lo que se confeccionan un grupo de artefactos estrechamente relacionados con la metodología de desarrollo XP (adoptada para el desarrollo de la solución) y otros que, aunque no forman parte de los elementos que la metodología propone, se elaboran para facilitar el entendimiento y desarrollo del proyecto. Los artefactos desarrollados incluyen el diagrama de clases del dominio, las historias de usuario, la lista de reserva del producto, las tarjetas Clase-Responsabilidad-Colaboración y el plan de iteraciones, elementos que guían el desarrollo de la solución.

2.1. Modelo de dominio

El modelo de dominio es una representación visual de las clases conceptuales u objetos del mundo real en un área de interés y, es el mecanismo fundamental para comprender el problema y establecer conceptos comunes. (38)

Entre las ventajas que propicia el uso del modelo de dominio destacan que permite: (39)

- Describir y limitar el alcance del dominio del problema.
- Verificar y validar la comprensión del dominio del problema entre las diversas partes interesadas.
- Definir un vocabulario común.
- Servir como herramienta de comunicación.
- Añadir precisión y enfoque para la discusión entre clientes y desarrolladores.

El diagrama de clases del dominio tomado como punto de partida para el diseño del sistema se muestra en la figura siguiente.

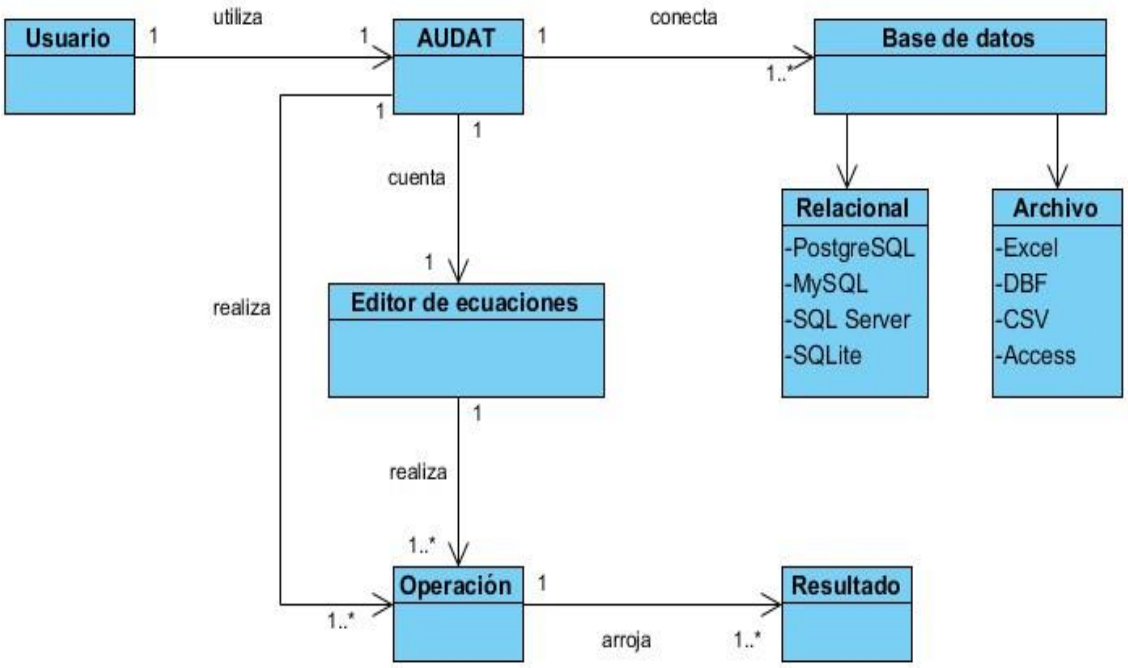


Fig. 4 Diagrama de clases del dominio

La tabla 2 muestra los conceptos asociados al diagrama de clases del dominio definido.

Tabla 2 Descripción de los conceptos del diagrama de clases del dominio

No	Concepto	Descripción
1	Usuario	Persona que interactúa con AUDAT
2	AUDAT	Sistema de Auditoría de Datos
3	Base de datos	Base de datos importada por AUDAT
4	Base de datos relacional	Tipo de base de datos importada por AUDAT que implementa el modelo relacional
5	Archivo	Tipo de fichero importado por AUDAT
6	Editor de ecuaciones	Componente que mediante la selección de funciones permite la ejecución de operaciones sobre las tablas de las bases de datos importadas por AUDAT
7	Operación	Acción realizada sobre las tablas de la base de datos
8	Resultado	Elemento resultante obtenido de ejecutar operaciones sobre las tablas de la base de datos

2.2. Descripción del sistema propuesto

Para solventar las deficiencias derivadas del empleo del editor de ecuaciones de AUDAT se propone la implementación de un constructor gráfico de consultas SQL, que permitirá a los auditores de la Contraloría General de la República de Cuba la realización de consultas sobre las bases de datos importadas por el sistema durante los procesos de auditorías. Las consultas podrán ser realizadas a través del constructor gráfico de manera visual, permitiendo la realización de aquellas de tipo SELECT mediante la especificación de criterios como condiciones, opciones de ordenamiento y, la relación de tablas mediante concatenaciones.

2.2.1. Historias de usuario

Las historias de usuario son descripciones cortas y escritas en el lenguaje del usuario (sin terminología técnica), que proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo tardará su implementación. (40)

Son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos y sin requerir de mucho tiempo para administrarlos, permitiendo responder rápidamente a los requerimientos cambiantes. Constan de 3 o 4 líneas escritas por el cliente sin profundizar en detalles técnicos y sin describir posibles algoritmos para su implementación, entre sus características fundamentales destacan que: (41)

- Son usadas para estimar los tiempos de desarrollo del *software* que describen.
- Son usadas en la fase de pruebas para verificar si el *software* cumple con lo que especifican.
- Para implementarlas se requiere de encuentros entre clientes y desarrolladores para especificar y definir lo que tienen que hacer.
- Tienen un tiempo de desarrollo ideal de entre 1 y 3 semanas.

Para definir las historias de usuario se utiliza una plantilla que contiene todos los datos necesarios para desarrollar la funcionalidad descrita. En las tablas siguientes se detallan las 13 historias de usuario definidas para el desarrollo de la propuesta de la solución de la investigación.

Tabla 3 Historia de usuario Mostrar en el área de diseño la tabla de la base de datos importada en AUDAT

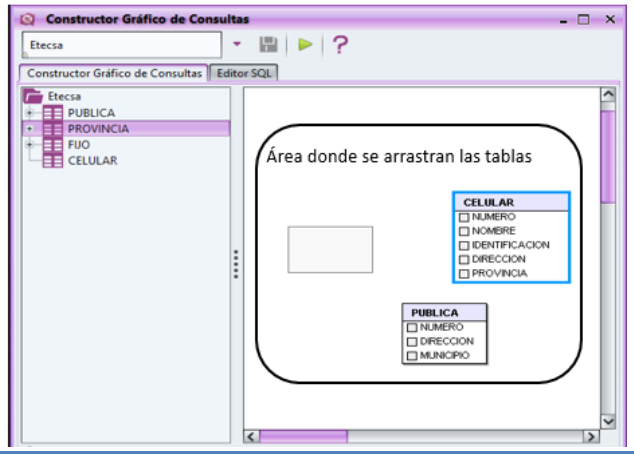
Historia de usuario	
Número: 1	**Nombre:** Mostrar en el área de diseño la tabla de la base de datos importada en AUDAT y seleccionada para la consulta
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario seleccionar la tabla de la que desea emplear u obtener datos en la consulta, así como arrastrarla al área de diseño del constructor gráfico	
Prototipo de interfaz: 	

Tabla 4 Historia de usuario Mostrar los atributos que formarán parte del procesamiento o resultado de la consulta

Historia de usuario	
Número: 2	**Nombre:** Mostrar los atributos de la tabla que formarán parte del procesamiento o resultado de la consulta
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario seleccionar, haciendo clic sobre el *combo box* asociado a la tabla arrastrada previamente al área de diseño, cuáles de los atributos de esta van a ser utilizados durante el procesamiento de la	

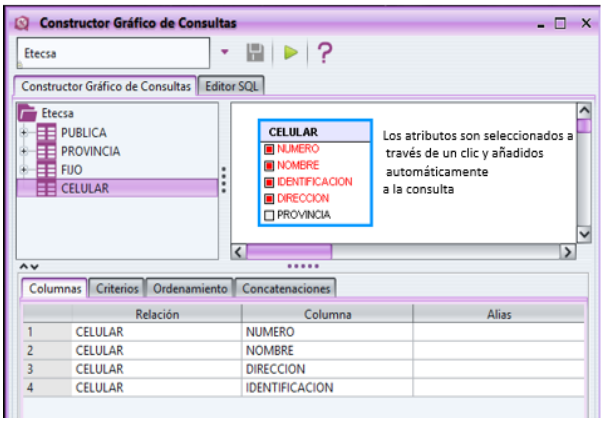
consulta o formarán parte del resultado de su ejecución
Prototipo de interfaz: 

Tabla 5 Historia de usuario Gestionar alias de una columna seleccionada

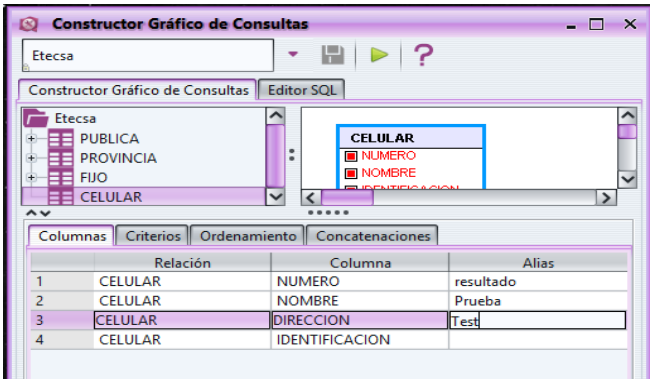
Historia de usuario	
Número: 3	**Nombre:** Gestionar alias de una columna seleccionada
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario agregar, modificar o eliminar el nombre de una columna, especificándole un alias que puede contribuir a facilitar el entendimiento por los usuarios del resultado de la consulta	
Prototipo de interfaz: 	

Tabla 6 Historia de usuario Gestionar criterio de selección de las tuplas resultantes de la consulta

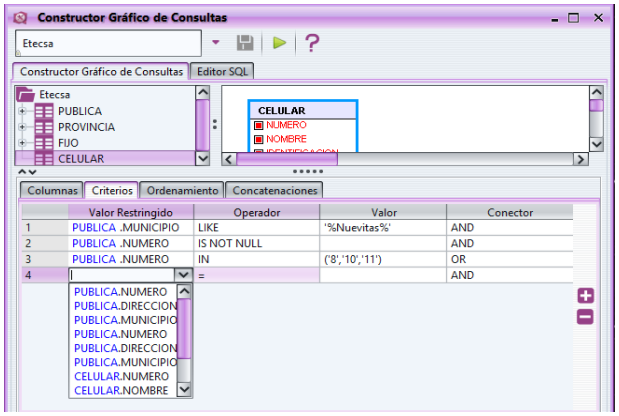
Historia de usuario	
Número: 4	**Nombre:** Gestionar criterio de selección de las tuplas resultantes de la consulta
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario agregar, modificar o eliminar criterios a tener en cuenta para la aplicación de filtros a las tuplas, eliminando aquellas que no cumplan con las condiciones especificadas	
Prototipo de interfaz: 	

Tabla 7 Historia de usuario Gestionar opción de ordenamiento ascendente o descendente al resultado

Historia de usuario	
Número: 5	**Nombre:** Gestionar opción de ordenamiento ascendente o descendente al resultado de la consulta
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario agregar, modificar o eliminar el orden (de manera ascendente o descendente) en que serán ordenadas las tuplas resultantes de la consulta, basándose en los valores de las columnas a las que se asocie la opción de ordenamiento	

Prototipo de interfaz:

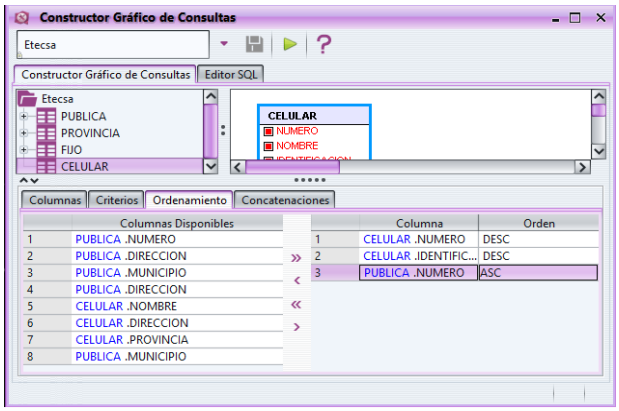


Tabla 8 Historia de usuario Gestionar concatenación entre tablas de la base de datos importada en AUDAT

Historia de usuario	
Número: 6	**Nombre:** Gestionar concatenación entre tablas de la base de datos importada en AUDAT
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario agregar, modificar o eliminar concatenación entre tablas de la base de datos importada y realizar consultas complejas que incluyen más de una tabla	
Prototipo de interfaz:	

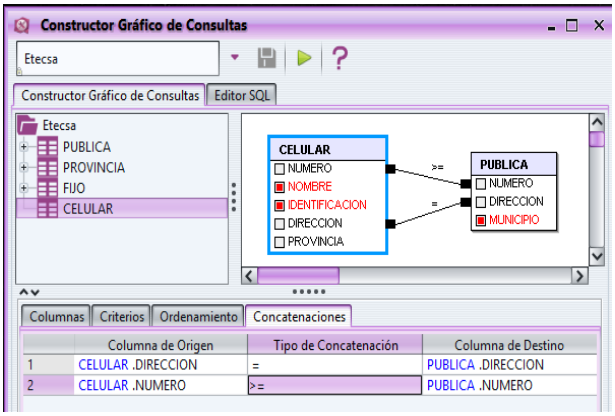


Tabla 9 Historia de Usuario Definir consulta SQL manualmente

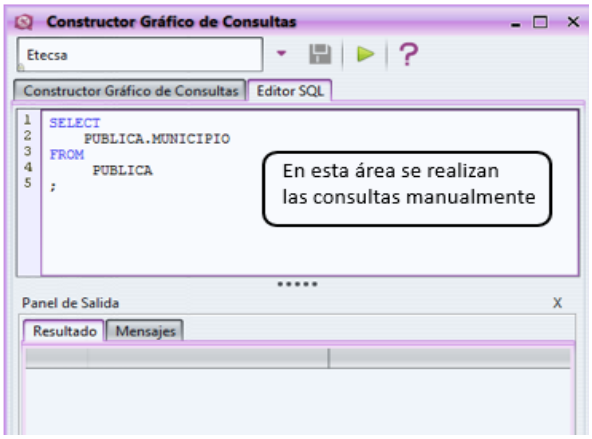
Historia de usuario	
Número: 7	**Nombre:** Definir consulta SQL manualmente
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario, con conocimientos del lenguaje SQL, la implementación de la consulta de forma manual en la pestaña Editor SQL	
Prototipo de interfaz: 	

Tabla 10 Historia de usuario Generar código SQL

Historia de usuario	
Número: 8	**Nombre:** Generar código SQL
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Después de crear la consulta de manera gráfica el código SQL es generado automáticamente en la pestaña Editor SQL	
Prototipo de interfaz:	

Tabla 11 Historia de usuario Ejecutar consulta SQL

Historia de usuario	
Número: 9	**Nombre:** Ejecutar consulta SQL
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 2 semanas
Riesgo en desarrollo: Alto	**Puntos reales:** 2 semanas
Descripción: Permite al usuario ejecutar el código resultante de la construcción gráfica de la consulta en el área de diseño o el código en el área SQL y obtener los resultados	
Prototipo de interfaz: 	

Tabla 12 Historia de usuario Mostrar resultado de la ejecución de la consulta

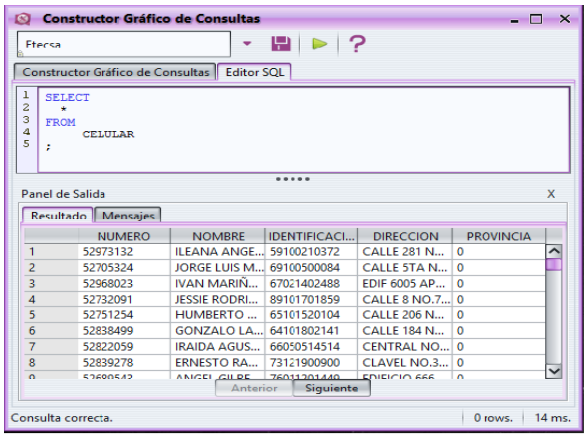
Historia de usuario	
Número: 10	**Nombre:** Mostrar resultado de la ejecución de la consulta
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario visualizar el resultado de la consulta ejecutada	
Prototipo de interfaz: 	

Tabla 13 Historia de usuario Guardar el resultado de la consulta ejecutada

Historia de usuario	
Número: 11	**Nombre:** Guardar el resultado de la consulta ejecutada
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario guardar el resultado de la consulta ejecutada para posteriores usos sobre dicho resultado	
Prototipo de interfaz:	

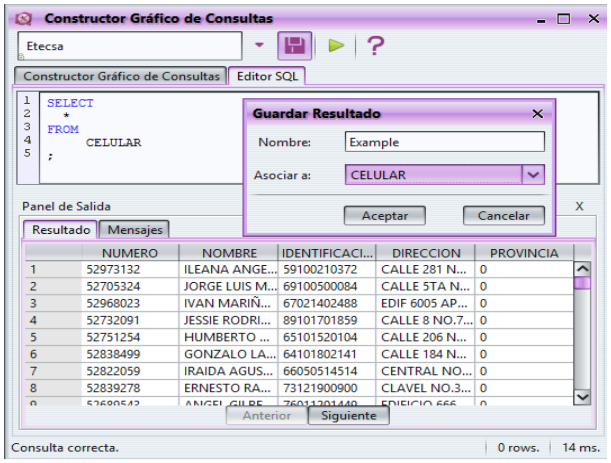


Tabla 14 Historia de usuario Importar el resultado de la consulta guardada

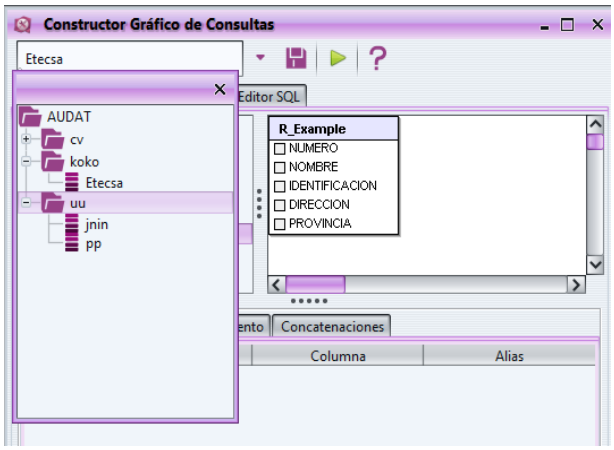
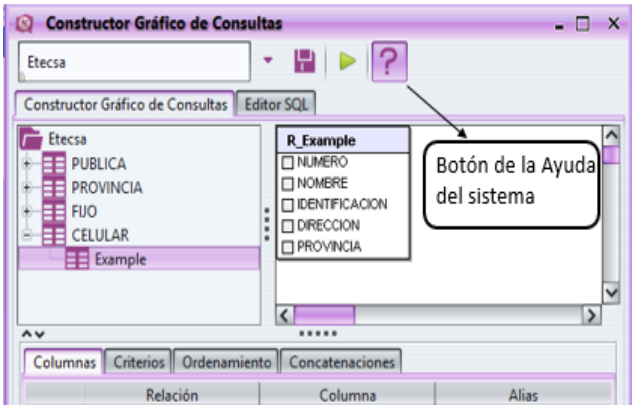
Historia de usuario	
Número: 12	**Nombre:** Importar el resultado de la consulta guardada
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Alto	**Puntos reales:** 1 semana
Descripción: Permite al usuario cargar el resultado de una consulta realizada previamente y guardada, para realizar una nueva acción sobre ella	
Prototipo de interfaz: 	

Tabla 15 Historia de usuario Consultar ayuda

Historia de usuario	
Número: 13	**Nombre:** Consultar ayuda
Cantidad de modificaciones: 0	
Usuario: Liosdanys Rodríguez Pérez	**Iteración asignada:** 3
Prioridad en negocio: Media	**Puntos estimados:** 1 semana
Riesgo en desarrollo: Bajo	**Puntos reales:** 1 semana
Descripción: Permite al usuario consultar la ayuda del Constructor Gráfico de Consultas para mitigar cualquier duda que pueda surgir en el empleo de las funcionalidades desarrolladas en la herramienta	
Prototipo de interfaz: 	

2.2.2. Lista de reserva del producto

La etapa de definición de requisitos es una tarea de suma importancia a la hora de desarrollar un sistema; esta consiste en generar una definición clara y concisa de los aspectos más relevantes del producto. La lista de reserva del producto agrupa los requisitos funcionales (organizados por prioridad según la complejidad en el negocio) y los requisitos no funcionales.

Los requisitos funcionales se definen como las capacidades o servicios que el sistema debe proporcionar, cómo debe reaccionar ante ciertas entradas y cómo debe comportarse en situaciones particulares; mientras que los requisitos no funcionales son aquellos que especifican propiedades del sistema como restricciones del entorno o de la implementación, rendimiento, facilidad de mantenimiento, extensibilidad o fiabilidad. (42)

En la tabla siguiente se muestran los requisitos funcionales y no funcionales definidos para su implementación en la propuesta, derivados de las historias de usuario acordadas con el cliente.

Tabla 16 Lista de reserva del producto

Número	Descripción	Estimación	Estimado por
Requisitos funcionales			
Prioridad: Muy Alta			
1	Mostrar en el área de diseño la tabla de la base de datos importada en AUDAT y seleccionada para la consulta	1	Analista
2	Mostrar los atributos de la tabla que formarán parte del procesamiento o resultado de la consulta	1	Analista
3	Agregar alias de una columna seleccionada	0,4	Analista
4	Modificar alias de una columna seleccionada	0.3	Analista
5	Eliminar alias de una columna seleccionada	0,3	Analista
6	Agregar criterio de selección de las tuplas resultantes de la consulta	0,4	Analista
7	Modificar criterio de selección de las tuplas resultantes de la consulta	0,3	Analista
8	Eliminar criterio de selección de las tuplas resultantes de la consulta	0,3	Analista
9	Agregar opción de ordenamiento ascendente o descendente al resultado de la consulta	0,4	Analista
10	Modificar opción de ordenamiento ascendente o descendente al resultado de la consulta	0,3	Analista
11	Eliminar opción de ordenamiento ascendente o descendente al resultado de la consulta	0,3	Analista
12	Agregar opción de concatenación entre tablas de la base de datos importada en AUDAT	0,4	Analista
13	Modificar opción de concatenación entre tablas de la base de datos importada en AUDAT	0,3	Analista
14	Eliminar concatenación entre tablas de la base de datos importada en AUDAT	0,3	Analista

15	Definir consulta SQL manualmente	1	Analista
16	Generar código SQL	1	Analista
17	Ejecutar consulta SQL	2	Analista
18	Mostrar resultado de la ejecución de la consulta	1	Analista
Prioridad: Alta			
19	Guardar el resultado de la consulta ejecutada	1	Analista
20	Importar el resultado de la consulta guardada	1	Analista
Prioridad: Media			
21	Consultar ayuda	1	Analista
Requisitos no funcionales			
Apariencia o interfaz externa			
	El Constructor Gráfico de Consultas tendrá una interfaz amigable, con una navegabilidad flexible y de fácil comprensión, y debe cumplir con las pautas establecidas por la línea de desarrollo *Xedro* bajo la que fue liberada AUDAT, donde:		
22	- Color de la interfaz: Morado		
23	- Tipografía: *Novasson Bold*		
24	- Enumerar líneas a la izquierda del área SQL		
25	- Resaltar en color azul las palabras reservadas del lenguaje SQL		
Software			
26	- Tener instalado como sistema operativo *Linux* o *Windows*		
27	- Tener instalado la máquina virtual de *Java* versión 7 o superior		
Hardware			
	El ordenador debe tener como mínimo:		
28	- Procesador Celeron Pentium 3		
29	- Capacidad libre en el disco duro de 20 Gb		
30	- 512 Mb de RAM		
Restricciones de diseño e implementación			
31	Se utilizará el lenguaje de programación *Java*, haciendo uso del IDE *Netbeans* 7.4		
Requisitos para la documentación de usuarios y ayuda del sistema			
32	El sistema contará con una ayuda que se encontrará como un enlace dentro del sistema, la que		

	debe cubrir todas las funcionalidades del Constructor Gráfico de Consultas para AUDAT

2.2.3. Plan de iteraciones

El plan de iteraciones es una planificación donde los desarrolladores y clientes establecen los tiempos de implementación ideales de las historias de usuario, la prioridad de estas y cuáles serán implementadas en cada versión del programa. Al comienzo de cada iteración los clientes deben seleccionar las historias de usuario definidas en el plan de iteraciones que serán implementadas. (43)

La implementación del sistema propuesto tendrá una duración de 14 semanas, tiempo en que se realizarán 3 iteraciones; la tabla siguiente muestra la planificación propuesta.

Tabla 17 Plan de iteraciones

Iteración	Descripción de la iteración	Orden de la HU a implementar	Duración total
1	Se implementarán las historias de usuario con prioridad en el negocio MUY ALTA	1, 2, 3, 4, 5, 6 ,7, 8, 9, 10	11 semanas
2	Se implementarán las historias de usuario con prioridad en el negocio ALTA	11, 12	2 semanas
3	Se implementará la historia de usuario de prioridad MEDIA en el negocio	13	1 semana

2.3. Modelo de Diseño

Para el diseño de aplicaciones la metodología XP no requiere la presentación del sistema mediante diagramas de clases utilizando notación UML; en su lugar se usan otras técnicas como las tarjetas Clase-Responsabilidad-Colaboración, comúnmente llamadas tarjetas CRC. No obstante, el uso de estos diagramas puede aplicarse siempre y cuando influyan en el mejoramiento de la comunicación, no sea un peso su mantenimiento y no sean extensos.

2.3.1. Tarjetas CRC

Las tarjetas CRC garantizan que el equipo completo contribuya en la tarea de diseño; el uso de estas permite al programador centrarse en el desarrollo orientado a objetos, rompiendo con la programación procedural. Se dividen en tres partes fundamentales:

- Clase: Representa una colección de objetos similares.

- Responsabilidades: Describen las funciones que debe realizar la clase.
- Colaboraciones: Describen las demás clases con las que trabaja la clase, de conjunto, para llevar a cabo sus responsabilidades.

Para la implementación de las 13 de historias de usuario definidas para dar cumplimiento a los requisitos del cliente se diseñaron 28 tarjetas CRC, de las cuales se muestran 9 en las tablas siguientes, pudiendo consultarse el resto en el Anexo 1.

Las tarjetas que se muestran en las tablas siguientes representan objetos; la clase a la que pertenece cada objeto se especifica en la parte superior de la tarjeta, en la columna de la izquierda se muestran las responsabilidades u objetivos que debe cumplir el objeto y, a la derecha, las clases que colaboran.

Tabla 18 Tarjeta CRC QueryController

Tarjeta CRC	
Clase: QueryController	
Responsabilidades	**Colaboraciones**
- Asignar la responsabilidad del manejo de los eventos del sistema a una clase - Proporcionar respuestas a las acciones que el usuario realiza sobre la aplicación	- *Table* - *JQueryFrame* - *SQLCode* - Criterio - *Join* - *Parser* - AtributoSet - *Ordering*

Tabla 19 Tarjeta CRC Table

Tarjeta CRC	
Clase: Table	
Responsabilidades	**Colaboraciones**
- Definir las propiedades de las tablas	- *GraphicObjectPosition* - Atributo - *Join*

Tabla 20 Tarjeta CRC Join

Tarjeta CRC
Clase: Join

Responsabilidades	Colaboraciones
- Definir las concatenaciones entre tablas y los atributos por los cuales se establece la concatenación	- *GraphicObject* - Atributo

Tabla 21 Tarjeta CRC DataBaseMetaData

Tarjeta CRC	
Clase: DataBaseMetaData	
Responsabilidades	**Colaboraciones**
- Obtener la información de la base de datos seleccionada por el usuario - Construir el modelo con la información de la base de datos	- *NodeInfo* - *NodeType*

Tabla 22 Tarjeta CRC QueryData

Tarjeta CRC	
Clase: QueryData	
Responsabilidades	**Colaboraciones**
- Crear todo lo que el usuario desarrolle en la aplicación gráficamente a SQL	

Tabla 23 Tarjeta CRC Criterio

Tarjeta CRC	
Clase: Criterio	
Responsabilidades	**Colaboraciones**
- Establecer las condiciones de la consulta	- *AbstractInformation* - *AtributoSet*

Tabla 24 Tarjeta CRC Ordering

Tarjeta CRC	
Clase: Ordering	
Responsabilidades	**Colaboraciones**

- Establecer el criterio de ordenamiento de la consulta	- *AbstractInformation*

Tabla 25 Tarjeta CRC Parser

Tarjeta CRC	
Clase: Parser	
Responsabilidades	**Colaboraciones**
- Validar que se haga referencia a las consultas de tipo *Select* solamente	- *Lexer* - *Token*

Tabla 26 Tarjeta CRC JQueryFrame

Tarjeta CRC		
Clase: JQueryFrame		
Responsabilidades	**Colaboraciones**	
- Establecer vínculo entre el usuario y la aplicación (Interfaz principal)	- *QueryController* - QueryPanel - *InsideTable* - *ColumnOutPut* - *Ordering*	- *CodeEditor* - *TreeDatabase* - *AtributoSet* - Criterio

2.4. Arquitectura de software

La definición oficial de arquitectura del *software* es de la IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) y plantea que la arquitectura del *software* es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos, el contexto en el que se implementarán y los principios que orientan su diseño y evolución. (44)

2.4.1. Patrones y estilos arquitectónicos

Un estilo arquitectónico es una familia de términos de un patrón de organización estructural, más específicamente, un estilo arquitectónico determina el vocabulario de los componentes y conectores que se pueden utilizar, junto con un conjunto de restricciones sobre la forma en que se pueden combinar. (45)

Existen numerosos estilos arquitectónicos entre los que se encuentra el Estilo de Llamada y Retorno y dentro de él, el patrón de arquitectura Modelo-Vista-Controlador (MVC). Este patrón de arquitectura separa la lógica de control, la interfaz de usuario y los datos de una aplicación en tres componentes.

Con el uso de este patrón se persigue mejorar la reusabilidad y la separación de conceptos, características que buscan facilitar el desarrollo de la aplicación y su posterior mantenimiento. La figura siguiente muestra cómo se emplea el patrón MVC, mediante la explicación del flujo implementado en la historia de usuario "Ejecutar consulta SQL", que demuestra su aplicación durante el desarrollo del Constructor Gráfico de Consultas.

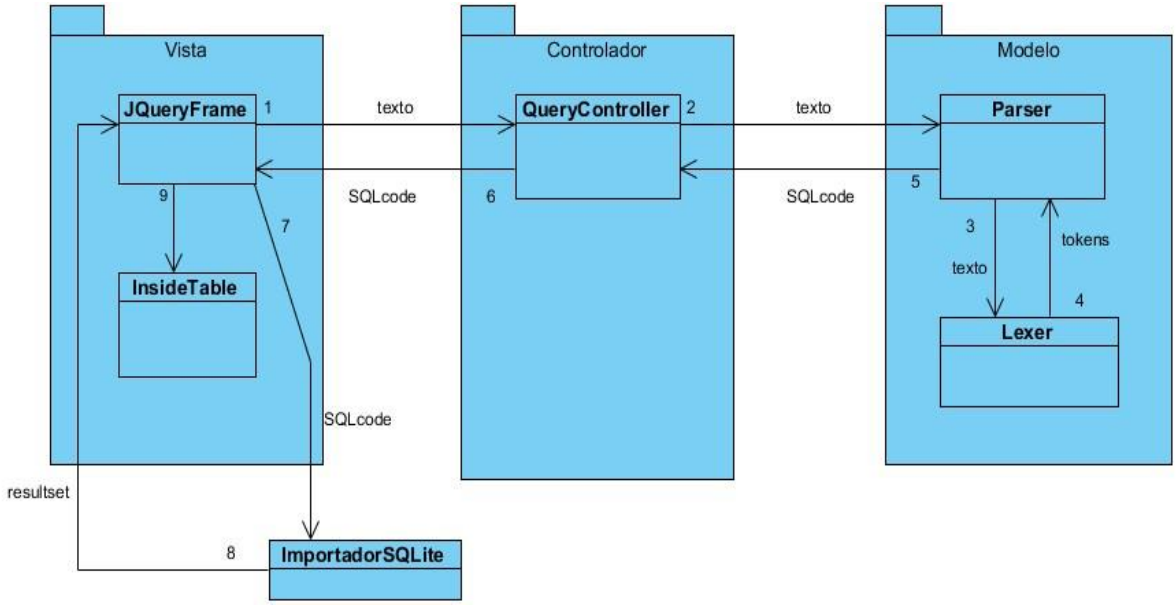


Fig. 5 Patrón arquitectónico Modelo Vista Controlador

Como muestra la figura anterior, las clases están separadas en tres paquetes, de manera que a través de la interfaz principal, implementada por la clase *JQueryFrame* (en el paquete Vista), se le envía el texto de la consulta a la clase *QueryController* (del paquete Controlador), que a su vez lo envía al analizador sintáctico *Parser* encargado de crear el código SQL de conjunto con el *Lexer* (del paquete Modelo)*;* una vez creado, dicho código es transportado por la misma vía a la interfaz principal, quien lo envía al gestor *SQLite* mediante la clase *ImportadorSQLite* (de AUDAT), la que devuelve el resultado de su ejecución, mostrándose en una tabla en la vista principal.

2.5. Patrones de diseño

Los patrones de diseño son la respuesta común a problemas comunes en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces. Son el esqueleto de las soluciones a problemas comunes en el desarrollo de aplicaciones informáticas. (46)

Un patrón de diseño brinda una solución ya probada y documentada a problemas de desarrollo de *software* que están sujetos a contextos similares.

Los patrones de diseño pretenden: (47)

- Proporcionar catálogos de elementos reusables en el diseño de sistemas *software*.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.

2.5.1. Patrones GRASP

GRASP son patrones generales de *software* para la asignación de responsabilidades (del inglés *General Responsibility Assignment Software Patterns*) que describen el principio fundamental de la asignación de responsabilidades a objetos y las colaboraciones que ocurren entre estos, expresados claramente en forma de patrones. Los patrones GRASP son: Experto, Creador, Alta Cohesión, Bajo Acoplamiento y Controlador, los que fueron aplicados durante el desarrollo del Constructor Gráfico de Consultas.

Patrón Creador

El patrón Creador guía la asignación de responsabilidades y ayuda a identificar quién debe ser el responsable de la creación de objetos. Se asigna la tarea a una clase de crear cuando contiene, añade, compone, almacena o emplea otra clase. El objetivo fundamental de este patrón es encontrar un creador que se deba conectar con el objeto producido en cualquier evento. El uso de este patrón se evidencia en la clase, *QueryPanel* la cual crea objetos de las clases *Join* y *Table* respectivamente para realizar sus responsabilidades. (Ver figura 6)

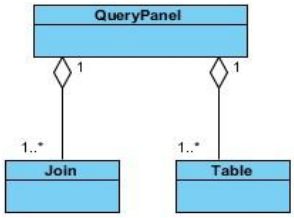


Fig. 6 Uso del patrón GRASP Creador

Patrón Experto

El patrón Experto se utiliza para asignar responsabilidades al experto en la información, o sea la clase que cuenta con la información necesaria para realizar o cumplir con la responsabilidad. Es un principio fundamental utilizado mucho en el diseño orientado a objetos. Este expresa la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen. El uso de este patrón se evidencia en la clase *QueryController* la cual asigna responsabilidades a la clase Parser para crear el código SQL. (Ver figura 7)

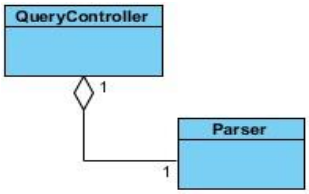


Fig. 7 Uso del Patrón GRASP Experto

Uno de los beneficios del uso del patrón Experto es que se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide.

Patrón Controlador

El patrón Controlador asigna la responsabilidad del manejo de los eventos del sistema a una clase. Este patrón sirve como intermediario entre una interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a distintas clases según el método llamado. Este patrón se evidencia entre la clase *QueryController*, que es la encargada de controlar y actualizar todos los cambios que se produzcan en la interfaz principal *JQueryFrame*. (Ver figura 8)

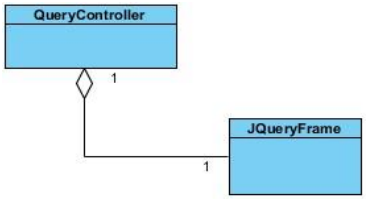


Fig. 8 Uso del Patrón GRASP Controlador

Patrón Alta Cohesión

La Alta Cohesión se debe tener presente durante todo el proceso de desarrollo de la aplicación. El alto nivel de cohesión es presentado por las clases que tienen responsabilidades moderadas en un área funcional y colaboran con otras para llevar a cabo las tareas, y la información que se almacenan en ellas debe ser coherente y debe estar, en la medida de lo posible relacionada con la clase a la que hace referencia. Este patrón se evidencia entre la clase *QueryController* y las clases *Ordering*, *Join* y Criterio, donde estas colaboran entre sí con el fin de crear las consultas según sus responsabilidades durante este proceso, donde están relacionadas únicamente con *QueryController* y no entre ellos. (Ver figura 9)

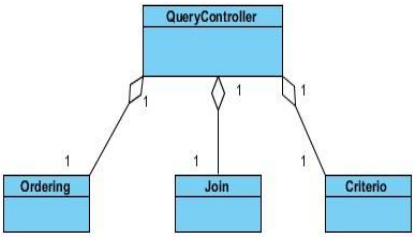


Fig. 9 Uso del Patrón GRASP Alta Cohesión

Patrón Bajo Acoplamiento

El bajo acoplamiento estimula la asignación de responsabilidades de forma tal que la inclusión de estas no incremente el acoplamiento; significa asignar una responsabilidad para mantener pocas dependencias entre clases. Se puede evidenciar el uso de este patrón en la clase *QueryController* y *Ordering*, puesto que esta tiene el mínimo de dependencias con otras clases. (Ver figura 10)

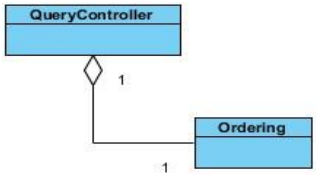


Fig. 10 Uso del Patrón GRASP Bajo Acoplamiento

2.5.2. Patrones GOF

Los patrones GOF (*Gang of Four*, Pandilla de los Cuatro) constituyen una serie de patrones de diseño que se agrupan en tres categorías: creacionales, estructurales y de comportamiento. De los diferentes patrones que ofrecen, para la modelación del Constructor Gráfico de Consultas se tuvo en cuenta el comportamiento, que contribuye a definir la comunicación entre los objetos de la aplicación, aplicándose el patrón *Observer* (observador), que define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.

El uso de este patrón se evidencia entre la clase controladora *QueryController* y la vista principal *JQueryFrame*, donde al ocurrir algún cambio en la vista principal se notifica a la controladora que es la encargada de notificar los cambios a las clases que corresponden.

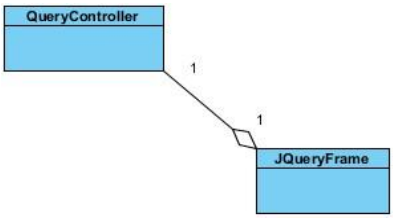


Fig. 11 Uso del Patrón GOF Observador

Conclusiones del capítulo

En el capítulo se identificaron 13 historias de usuario de las cuales se derivaron 21 requisitos funcionales y, se definieron 11 requisitos no funcionales, los que se agruparon en la lista de reserva del producto, planificándose su implementación en 14 semanas, mediante el desarrollo de 3 iteraciones comenzado por las historias de usuario de mayor prioridad para el negocio. Además, se confeccionaron 28 tarjetas CRC, se describieron el patrón arquitectónico MVC utilizado, así como los patrones de diseño Creador, Experto, Controlador, Alta Cohesión, Bajo Acoplamiento y Observador aplicados.

3 Implementación y pruebas del Constructor Gráfico de Consultas para AUDAT

En este capítulo se describe la implementación del sistema. Para darle solución a los requisitos contenidos en las historias de usuario definidas en el capítulo anterior se realizan las tareas de ingeniería, se define el estándar de codificación utilizado para el código fuente y se especifica la estrategia de pruebas a utilizar para lograr la calidad requerida del Constructor Gráfico de Consultas.

3.1. Implementación del sistema

La implementación tiene como objetivo desarrollar el sistema como un todo, así como describir la organización del código. Para llevarla a cabo se desglosan en tareas de ingeniería las historias de usuario definidas en el capítulo anterior, las cuales guían la implementación.

3.1.1. Tareas de ingeniería

La división de las historias de usuario en tareas más sencillas, tienen el propósito de que el programador obtenga una mejor visión a la hora de implementarlas, planificando el trabajo desde el punto de vista técnico.

Las tablas siguientes muestran las tareas de ingeniería números 9, 10 y 11 vinculadas a la historia de usuario “Ejecutar consulta SQL” con un tiempo estimado de desarrollo de 2 semanas, pudiendo consultarse las 12 restantes en el Anexo 2.

Tabla 27 Tarea de ingeniería Obtener el texto de la consulta creada

Tarea de ingeniería	
Número de tarea: 9	**Número de Historia de usuario:** 9
Nombre de la tarea: Obtener el texto de la consulta creada	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1 semana
Fecha de inicio: 6/11/2014	**Fecha de fin:** 12/11/2014

Programador responsable: Liosdanys Rodríguez Pérez
Descripción: Captura el texto de la consulta creada gráficamente o escrita por el usuario a través de comandos SQL

Tabla 28 Tarea de ingeniería Enviar el código SQL al gestor

	Tarea de ingeniería
Número de tarea: 10	**Número de Historia de usuario:** 9
Nombre de la tarea: Enviar el código SQL al gestor	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 0.5 semana
Fecha de inicio: 13/11/2014	**Fecha de fin:** 16/11/2014
Programador responsable: Liosdanys Rodríguez Pérez	
Descripción: Enviar el código SQL, obtenido de la validación realizada por la clase Parser, al gestor *SQLite*	

Tabla 29 Tarea de ingeniería Crear una tabla con el resultado obtenido

	Tarea de ingeniería
Número de tarea: 11	**Número de Historia de usuario:** 9
Nombre de la tarea: Crear una tabla con el resultado obtenido	
Tipo de Tarea: Desarrollo	**Puntos Estimados:** 0.5 semana
Fecha de inicio: 17/11/2014	**Fecha de fin:** 19/11/2014
Programador responsable: Liosdanys Rodríguez Pérez	
Descripción: Crear una tabla en el sistema para mostrar los resultados a partir de la respuesta del gestor *SQLite*	

3.1.2. Estándares de codificación

Una de las buenas prácticas de la metodología XP es que promueve el empleo de los estándares de programación, lo que permite la obtención de un código uniforme (como si el sistema fuera programado por una sola persona).

Los estándares de codificación, también llamados estilos de programación o convenciones de código, son convenios para escribir código fuente en ciertos lenguajes de programación. Estos estándares ayudan a mejorar el proceso de codificación, mantienen un estilo de programación, sirven como punto de referencia para los programadores y elevan la facilidad en el mantenimiento del código. (48)

Entre las ventajas que proporcionan los estándares de codificación destacan que:

- Mejoran la lectura del código, permitiendo entender aquel nuevo.
- Ayudan a generar código de alta calidad.
- Favorecen la escalabilidad.

El estándar definido para la implementación del Constructor Gráfico de Consultas para AUDAT incluye los siguientes elementos.

Margen adicional

El margen adicional del código permite que este sea más legible al ganar en organización. Para el margen del código del Constructor se deben garantizar los siguientes elementos:

- Longitud de línea: Cada línea de código debe tener menos de 80 caracteres.
- Rompiendo líneas: Cuando una línea de código tenga más de 80 caracteres se debe romper de acuerdo con los principios de:
    - Romper después de una coma.
    - Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la anterior.

Comentarios

Es conveniente dejar información que pueda ser leída después para entender qué fue lo que se hizo en el fragmento de código. Los comentarios deben ser escritos de forma clara en una sola línea, reservando los de bloque para la documentación formal o para comentar porciones de código.

Declaración de variables

Las declaraciones realizadas deben cumplir con las siguientes normas:

- Realizar una declaración por línea ya que facilita los comentarios.
- El nombre de las variables debe comenzar con letras minúsculas y cada palabra relevante por la que esté compuesta debe iniciar con letra mayúscula.

Declaración de funciones

Garantizar en las declaraciones de funciones que:

- No haya espacio en blanco entre el nombre de la función y el paréntesis izquierdo, ni entre este y la lista de parámetros.

- Haya un espacio en blanco entre el paréntesis derecho y la llave de comienzo del cuerpo de la función.
- Las sentencias del cuerpo estén en la línea siguiente.
- La llave que cierra el cuerpo de la función esté alineada con la línea de declaración de la misma.

Identificadores

Los identificadores pueden estar formados por cualquiera de las letras minúsculas o mayúsculas del alfabeto, los dígitos y el carácter subrayado ("_"). Debe evitarse el uso de caracteres no anglosajones como vocales con acentos, ñ, etc. porque no siempre pueden ser leídos o entendidos correctamente.

Sentencias

Las sentencias deben cumplir con las siguientes normas:

- Cada línea debe contener solo una sentencia simple.
- Debe haber un espacio en blanco entre las palabras reservadas y el paréntesis de la condición, así como entre el paréntesis que cierra dicha condición y las llaves de la sentencia. Además las sentencias a ejecutar dentro de esta deben aparecer en la línea siguiente, cumpliendo las reglas de identación definidas.

Espacios en blanco

Las líneas en blanco mejoran la facilidad de lectura separando secciones de código que están lógicamente relacionadas.

Se debe usar siempre una línea en blanco en las siguientes circunstancias:

- Entre funciones.
- Entre las distintas secciones lógicas de una función para facilitar la lectura.

Se deben usar espacios en blanco en las siguientes circunstancias:

- Entre una palabra clave del lenguaje y un paréntesis.
- Después de cada coma en las listas de argumentos.
- Para cualquier operador binario, excepto el punto ("."), el paréntesis que abre ("(") y el corchete que abre ("["), todos deben ser separados por un espacio entre operandos y operador.

La figura 12 muestra un ejemplo del estándar adoptado en la implementación del Constructor Gráfico de Consultas, específicamente en la clase *Parser*, que tiene como objetivo crear el código SQL a partir del texto obtenido de la interfaz principal.

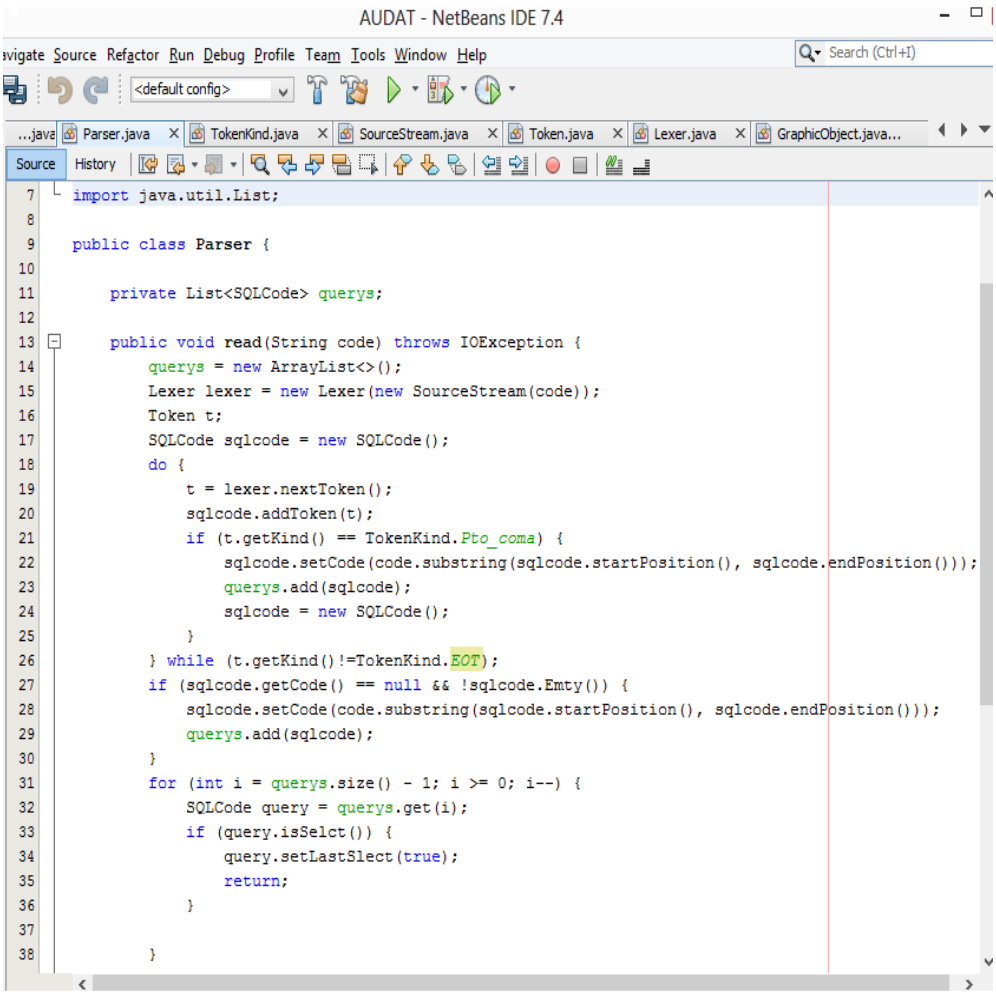


Fig. 12 Ejemplo del estándar aplicado durante la implementación

3.1.3. Interfaz de la aplicación

La aplicación desarrollada cuenta con una interfaz principal que contiene dos pestañas, una para construir las consultas gráficamente (Constructor Gráfico de Consultas) y la otra para la realización manual de consultas (Editor SQL).

Área del Constructor Gráfico de Consultas

En esta área se realizan las consultas SQL de manera visual. Está dividida en tres partes:

- Árbol de objetos: Donde se escoge la base de datos con la que se desea trabajar para luego seleccionar las tablas a utilizar en la realización de la consulta, que se arrastran hacia el área de diseño.
- Área de diseño: Muestra las tablas arrastradas y sus atributos permitiendo seleccionar aquellos necesarios para la ejecución de la consulta.
- Área de criterios: Muestra opciones para el establecimiento de alias a las columnas de salida de la consulta a ejecutar (Pestaña Columnas); de condiciones a cumplir por las tuplas resultantes (Pestaña Criterios) mediante la adición de criterios empleado columnas, operadores y valores; de criterios para el establecimiento de orden de las tuplas (Pestaña Ordenamiento) con alternativas ascendentes o descendentes y; de uniones de tablas (Pestaña Concatenaciones).

El código de la consulta construida de manera gráfica se crea automáticamente, visualizándose en la pestaña Editor SQL. (Ver figuras 13 y 14)

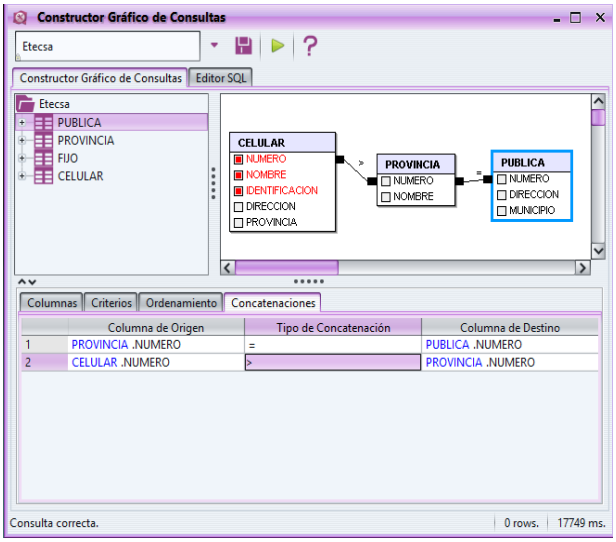


Fig. 13 Pestaña Constructor Gráfico de Consultas

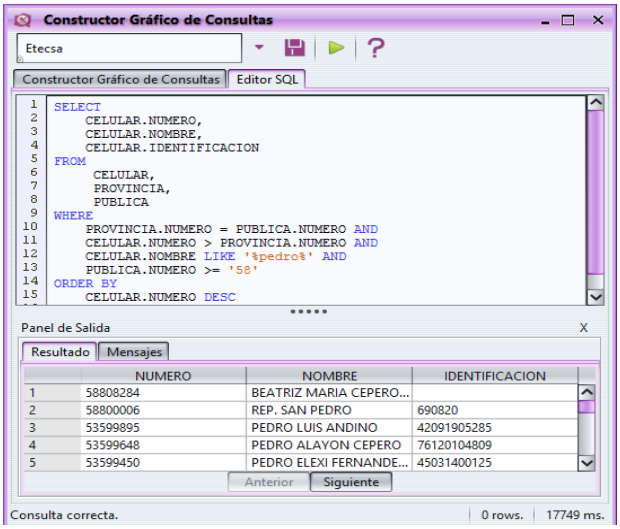


Fig. 14 Pestaña Constructor Gráfico de Consultas, consulta generada automáticamente

Área del Editor SQL

En esta área se escriben las consultas SQL que se realizan de forma manual sobre la base de datos. Está dividida en dos partes, en la superior se muestra el panel SQL (donde se implementan las consultas) y que tiene enumeradas las líneas y, en la parte inferior el panel Resultado, donde se muestran las tuplas resultantes de la consulta ejecutada o lo errores derivados de su ejecución, así como el historial de mensajes. (Ver figura 15)

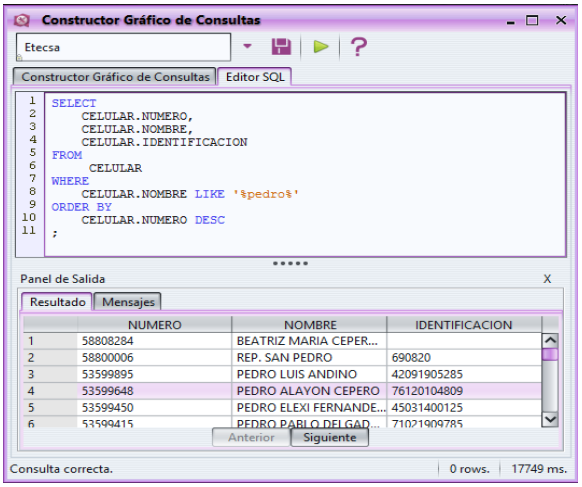


Fig. 15 Pestaña Editor SQL

3.2. Validación del sistema

La validación y comprobación del funcionamiento del sistema permite aumentar su calidad, reduciendo el número de errores no detectados y el tiempo transcurrido entre la aparición de un error y su detección.

3.2.1. Pruebas de software

Las pruebas de *software* son indispensables para evaluar y determinar su calidad; concretamente se puede definir el término pruebas de *software* como: (49)

- El proceso de ejecución de un programa con la intención de descubrir errores previos a la entrega al usuario final.
- Una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, los resultados son observados y registrados y se realiza una evaluación sobre algún aspecto suyo.

Las metodologías ágiles no consideran las pruebas como un conjunto de niveles que hay que ir superando para alcanzar la validación final del sistema que se está desarrollando; en su lugar presentan distintos enfoques del proceso de desarrollo, determinados por los tipos de pruebas que se realizan. (50)

La metodología XP, como forma de validar las necesidades de los usuarios y dirigir la implementación, fracciona las pruebas del sistema en dos grupos: (51)

- Pruebas unitarias, responsabilizadas de verificar el código de cada módulo de manera independiente y, diseñadas por los programadores para garantizar que un determinado módulo cumpla con un comportamiento antes de ser integrado al sistema.
- Pruebas de aceptación, encargadas de evaluar las funcionalidades del *software* y, definidas y diseñadas por el cliente con el objetivo de asegurar que las funcionalidades del sistema cumplan con lo que se espera de ellas.

Las pruebas unitarias utilizan la técnica de caja blanca o de caja negra; la técnica de caja blanca se utiliza, principalmente, desde una perspectiva interna en el desarrollo de *software*, mientras que las pruebas de caja negra (particiones de equivalencia, análisis de valores límites, pruebas transversales, entre otras) tienen una visión externa del producto de *software*. Las pruebas de caja negra están centradas en analizar las funcionalidades, por tanto, sus casos de prueba se basan en las diferentes entradas que puede recibir el *software* y sus correspondientes valores de salida; estas técnicas se pueden aplicar en cualquiera de

los niveles de pruebas (unitarias, integración, aceptación) con diferentes formas de abstracción en la definición de los casos de pruebas.

En las pruebas de aceptación se realizan dos tipos de pruebas, que posibilitan la detección de errores por el usuario final: (52)

- Prueba *alfa*, que se lleva a cabo por un cliente en un entorno controlado, se usa el *software* de forma natural con el desarrollador como observador del usuario y registrando los errores y los problemas de su uso.
- Prueba *beta*, que se lleva a cabo por los usuarios finales del *software* en el entorno del cliente; a diferencia de la prueba *alfa* el desarrollador no está presente, por tanto es una prueba en un entorno no controlado, siendo el cliente quien registra los problemas detectados y los informa al desarrollador.

Se decide emplear para las pruebas al Constructor Gráfico de Consultas la estrategia de pruebas aceptación de tipo *alfa* con la técnica de caja negra, ya que permitiría la comprobación de las funcionalidades del Constructor por el cliente en un entorno controlado mediante la aplicación de pruebas de caja negra, las que pretenden encontrar errores en: (53)

- Funciones incorrectas o ausentes.
- En la interfaz.
- En estructuras de datos o en accesos a bases de datos externas.
- De rendimiento.
- De inicialización y de terminación.

3.2.2. Diseño de casos de pruebas. Método seleccionado

Los casos de pruebas son importantes para la obtención de un *software* con calidad y libre de errores. Un caso de prueba bien diseñado da la posibilidad de obtener resultados más fiables y eficientes. (54)

Un caso de prueba contiene una serie de valores de entrada, condiciones de ejecución y resultados esperados, desarrollado con un objetivo tal como ejecutar una ruta determinada de un programa o verificar el cumplimiento de un requerimiento específico y; determina un conjunto de acciones con resultados y salidas previstas basadas en los requisitos del sistema.

Se pueden definir varios casos de prueba para determinar que una funcionalidad es completamente satisfactoria; teniendo en cuenta que debe existir al menos uno para cada historia de usuario.

Los casos de pruebas definidos al aplicar la técnica de caja negra pretenden comprobar que las funcionalidades son operativas, al aceptar las entradas y producir salidas coherentes con estas. El método de prueba que se decide emplear al aplicar la técnica de caja negra es Partición Equivalente. Este método divide el dominio de entrada de un programa en clases de datos, a partir de los cuales deriva los casos de prueba. Cada una de estas clases de equivalencia representa a un conjunto de estados válidos o inválidos para las condiciones de entrada.

Esta técnica trata de definir los casos de prueba que descubren clases de errores, reduciendo el número total de casos de prueba que deben ser desarrollados. Una ventaja de este enfoque es la reducción en el tiempo requerido para las pruebas de *software* debido a un menor número de casos de prueba.

La tabla 31 muestra el caso de prueba “Construir el ORDER BY de la consulta”, de la Historia de Usuario “Gestionar opción de ordenamiento ascendente o descendente al resultado de la consulta” y, la tabla 32 muestra el caso de prueba “Especificar concatenación entre tablas de la base de datos” de la Historia de Usuario **“**Gestionar concatenación entre tablas de la base de datos importada en AUDAT”, el resto de los 11 casos de prueba pueden consultarse en el Anexo 3.

Tabla 30 Caso de prueba Construir el ORDER BY de la consulta

<table>
<tr><th colspan="7">Descripción general</th></tr>
<tr><td colspan="7">El caso de prueba verifica el correcto funcionamiento de la funcionalidad “Gestionar opción de ordenamiento ascendente o descendente al resultado de la consulta”</td></tr>
<tr><th colspan="7">Condiciones de ejecución</th></tr>
<tr><td colspan="7">El usuario tiene que haber diseñado una consulta gráficamente</td></tr>
<tr><th colspan="7">Descripción de las variables</th></tr>
<tr><td colspan="7"><table>
<tr><th>No</th><th>Nombre de campo</th><th>Clasificación</th><th>Nulo</th><th>Descripción</th></tr>
<tr><td>1</td><td>Columnas disponibles</td><td>Campo de selección</td><td>No</td><td>Atributos de todas las tablas que conforman la consulta diseñada</td></tr>
<tr><td>2</td><td>Columnas</td><td>Campo de texto</td><td>No</td><td>Atributo(s) seleccionados para la especificación del orden de las tuplas resultantes de la consulta</td></tr>
<tr><td>3</td><td>Orden</td><td>Campo de selección</td><td>Sí</td><td>Opción ascendente o descendente que se le aplica a las columnas</td></tr>
</table></td></tr>
<tr><th colspan="7">Flujo de eventos</th></tr>
<tr><td colspan="7"><table>
<tr><th>Escenario</th><th>Descripción</th><th>1</th><th>2</th><th>3</th><th>Respuesta del sistema</th><th>Flujo central</th></tr>
<tr><td>EC1
Adicionar columnas para el establecimiento de opciones de</td><td>Permite adicionar columnas para la confección de la cláusula ORDER BY de la consulta</td><td>V
FIJO.NUMERO
FIJO.NOMBRE
FIJO.DIRECCION
FIJO.PROVINCIA</td><td>V
FIJO.NOMBRE</td><td>V
ASC</td><td>Genera en la pestaña Editor SQL la consulta con la cláusula ORDER BY</td><td>1. Selecciona de las columnas disponibles (todas o una a la vez) aquellas por la que desea establecer opción de ordenamiento</td></tr>
</table></td></tr>
</table>

ordenamiento	SELECT	**V**	**I**		No genera en la pestaña Editor SQL la consulta con la cláusula ORDER BY	2. Da clic en uno de los botones disponibles para pasar uno o todos los campos seleccionados a Columnas 3. Especifica (de manera opcional) la opción de ordenamiento ascendente o descendente
		FIJO.NUMERO FIJO.NOMBRE FIJO.DIRECCION FIJO.PROVINCIA	0 (No se pasa la columna seleccionada)	N/A		
		V	**I**		Genera en la pestaña Editor SQL la consulta con la cláusula ORDER BY	
		FIJO.NUMERO FIJO.NOMBRE FIJO.DIRECCION FIJO.PROVINCIA	0 (No se pasa la columna seleccionada)	N/A		
EC2 Eliminar columna de la opción de ordenamiento	Permite eliminar columnas de la cláusula ORDER BY de la consulta SELECT	**V**	**V**	**V**	Elimina de Columnas el campo seleccionado y actualiza en la pestaña Editor SQL la cláusula ORDER BY de la consulta	1. Selecciona de las columnas previamente determinadas para el establecimiento de opciones de ordenamiento aquella a eliminar (FIJO.NUMERO) 2. Da clic en uno de los botones disponibles para pasar la columna seleccionada a Columnas Disponibles
		FIJO.NUMERO FIJO.NOMBRE FIJO.DIRECCION FIJO.PROVINCIA	FIJO.NOMBRE FIJO.NUMERO	ASC DESC		
		V	**V**	**V**	No elimina de Columnas el campo seleccionado	
		FIJO.NUMERO FIJO.NOMBRE FIJO.DIRECCION FIJO.PROVINCIA	FIJO.NOMBRE FIJO.NUMERO	ASC DESC		

		V FIJO.NUMERO FIJO.NOMBRE FIJO.DIRECCION FIJO.PROVINCIA	V FIJO.NOMBRE FIJO.NUMERO	V ASC DESC	Elimina de Columnas el campo seleccionado pero no actualiza en la pestaña Editor SQL la cláusula ORDER BY de la consulta	

Tabla 31 Caso de prueba Especificar concatenación entre tablas de la base de datos

Descripción general				
El caso de prueba verifica el correcto funcionamiento de la funcionalidad “Gestionar concatenación entre tablas de la base de datos importada en AUDAT”				
Condiciones de ejecución				
El usuario tiene que arrastrar al área de diseño la tabla o las tablas con las que desea trabajar				
Descripción de las variables				
No	**Nombre de campo**	**Clasificación**	**Nulo**	**Descripción**
1	Área de diseño	Campo de diseño	No	Área donde se dibuja la unión entre tablas y se especifica el atributo a medir
2	Columna Origen	Campo de texto	No	Tabla origen a la que se le hizo la unión
3	Tipo de concatenación	Campo de selección	No	Operador lógico por el que se va a regir la unión entre tablas
4	Columna Destino	Campo de texto	No	Tabla destino a la que se le hizo la unión
Flujo de eventos				

Escenario	Descripción	1	2	3	4	Respuesta del sistema	Flujo central
EC1 Especificar la forma de concatenación entre tablas de la base de datos	Permite establecer la concatenación entre tablas para la realización de consultas complejas	**I**	**N/A**	**N/A**	**N/A**	No se establece la unión entre tablas de la base de datos	1. Arrastra al área de diseño las tablas que se quieren concatenar 2. Especifica el atributo por el cual se va a realizar la concatenación 3. La concatenación es agregada a la pestaña Concatenación 4. Especifica en el campo Tipo de Concatenación el operador lógico por el que se van a medir los atributos seleccionados para la concatenación 5. El criterio de la concatenación es adicionado a la consulta
		0 (No se arrastran al área de diseño las tablas a unir)					
		V	**V**	**V**	**V**	Se estableció la unión entre tablas de la base de datos	
		PUBLICA. NUMERO PUBLICA. FIJO	PUBLICA. NUMERO	=	PUBLICA. FIJO		
EC2 Eliminar la concatenación entre tablas	Permite eliminar la concatenación entre tablas durante la realización de la consulta	**V**	**N/A**	**N/A**	**N/A**	No se elimina la unión entre tablas en la consulta generada	1. En el área de diseño elimina una tabla o el vínculo entre las tablas seleccionadas 2. Se elimina la unión de las tablas en la pestaña Concatenaciones
		PUBLICA. NUMERO PUBLICA. FIJO					
		V	**N/A**	**N/A**	**N/A**	Se eliminó la unión entre tablas en la consulta generada	
		PUBLICA. NUMERO					

		0 (Se eliminó una tabla perteneciente a la concatena-ción)					

Las pruebas efectuadas proporcionaron resultados satisfactorios en más del 85% de los casos, derivando el 15% restante en no conformidades. La figura 16 muestra los resultados de la ejecución de las pruebas, realizadas en tres iteraciones; detectándose en la primera 5 no conformidades y 3 en la segunda; se aplicaron los casos de prueba nuevamente en una tercera iteración, en la que no fueron detectadas no conformidades.

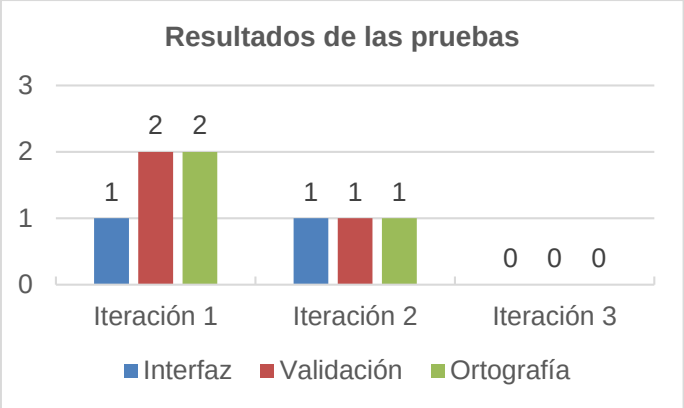


Fig. 16 Gráfica de los tipos de no conformidades encontradas en las iteraciones

La figura 17 muestra las clasificaciones de las no conformidades detectadas durante la aplicación de los casos de pruebas en cada iteración.

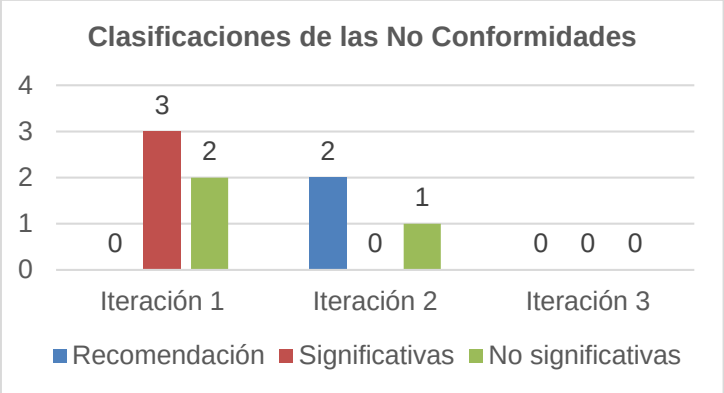


Fig. 17 Clasificaciones de las No Conformidades encontradas en las iteraciones

Conclusiones del capítulo

De las 13 historias de usuario definidas en la fase de análisis y diseño, se implementaron el 100% de las mismas. Para ello, se confeccionaron 15 tareas de ingeniería, se definió el estándar de codificación y, se

determinó como estrategia de pruebas a emplear las pruebas de aceptación, con pruebas de tipo alfa, usando la técnica de caja negra y el método partición de equivalencia. Además, se diseñaron y aplicaron 13 casos de prueba, detectándose 5 no conformidades en la primera iteración y 3 en la segunda, para un total de 8 no conformidades; de ellas 3 significativas, 3 no significativas y 2 recomendaciones; las que fueron solventadas, evidenciándose que la herramienta desarrollada cumple con las características especificadas por el cliente con la calidad requerida.

Conclusiones generales

Como resultado del trabajo de investigación realizado se arriban a las siguientes conclusiones:

- Se analizaron 4 constructores gráficos de consultas SQL para determinar el idóneo a ser integrado a AUDAT, demostrándose que ninguno cumplía con los requerimientos necesarios, fundamentalmente porque no se tenía acceso al código fuente de dos de ellos y, los restantes no estaban desarrollados en la tecnología utilizada por AUDAT lo que hacía engorrosa su posible integración; por lo que se hizo necesario la implementación de uno nuevo con el empleo de las tecnologías y herramientas definidas.
- Para la implementación de un constructor gráfico de consultas SQL para AUDAT se definieron 13 historias de usuario, derivadas en 21 requisitos funcionales y 11 requisitos no funcionales; se planificó su implementación en 3 iteraciones en un tiempo de desarrollo de 14 semanas; se confeccionaron 28 tarjetas CRC y 15 tareas de ingeniería y; se aplicaron los patrones arquitectónico MVC y de diseño Creador, Experto, Controlador, Alta Cohesión, Bajo Acoplamiento y Observador; elementos que permitieron la obtención del Constructor Gráfico.
- Se definió como estrategia de pruebas la utilización de pruebas de aceptación, con pruebas de tipo alfa, usando la técnica de caja negra y el método partición de equivalencia. Para su aplicación, fueron diseñados y aplicados 13 casos de prueba, detectándose 8 no conformidades que fueron resueltas en 3 iteraciones, permitiendo su funcionamiento con la calidad requerida.

Lográndose la implementación de un constructor gráfico de consultas SQL para AUDAT, para la creación y ejecución de consultas de manera visual, facilitando el trabajo de los auditores durante el empleo del sistema en el proceso de auditoría a bases de datos; por lo que se considera que los objetivos propuestos fueron cumplidos satisfactoriamente.

Recomendaciones

Independientemente de haberse cumplido los objetivos propuestos al inicio de la investigación, se recomienda:

- Adicionar las funcionalidades de revertir el proceso del constructor gráfico de consultas, e incorporarle otros tipos de sentencias como el INSERT, UPDATE y DELETE.
- Adicionar en Criterios, en la Pestaña Constructor Gráfico de Consultas, opciones para el tratamiento de los operadores de texto y numéricos (como el LIKE, BETWEEN, etc.) que faciliten la confección de las consultas sin emplear la sintaxis SQL necesaria para ello.

Referencias bibliográficas

1. **Flores Hurtado, Pablo Emilio.** mailxmail. *Curso elemental de auditorías.* [En línea] 29 de noviembre de 2005. [Citado el: 2014 de octubre de 10.] http://www.mailxmail.com/curso-elemental-auditoria/importancia-auditoria.

2. **Borghi, Alicia.** *Resumen de Auditoría.* México : s.n., 2010.

3. **Ingraballo, Héctor Grabiel.** *Auditoría de bases de datos.* Bogotá, Colombia : s.n., 2007.

4. **Grabiel Ingraballo, Héctor y Elizabeth Entraigas, Valeria.** *Auditorías de bases de datos.* Provincia del Chubut, Argentina : Universidad Nacional de Patagonia, 2007.

5. **Uribe Ballejo, Lucas.** *Sistema para la gestión de auditorías en bases de datos.* Medellín : s.n., 2008.

6. **Controlaría de la República de Cuba.** Contraloría. *Misión de la Contraloría de la República de Cuba.* [En línea] 29 de junio de 2009. [Citado el: 6 de octubre de 2014.] http://www.contraloria.cu/index.php/mision.

7. **Alegsa, Leandro.** Alegsa. *Editor de Consultas.* [En línea] 1998. [Citado el: 2014 de octubre de 28.] http://www.alegsa.com.ar/Dic/editor.php.

8. **Río, Gladys y Storti, Guillermo.** *Bases de datos. Consultas.* Buenos Aires, Argentina : s.n., 2009.

9. **Hernández, Eddy Ernesto Enrique.** *Módulo Editor de Consultas a la base de datos para el Sistema de Gestión de Datos Geológicos.* Habana,Cuba : s.n., 2013.

10. **Escofet Martín, Carmen.** *El lenguaje SQL.* Madrid, España : s.n., 2010.

11. **Blanco Rodrigo, Carlos.** *SQL Básico.* Madrid, España : s.n., 2006.

12. **DB-Engines.** DB-engines. *Ranking de los gestores de bases de datos más utilizados.* [En línea] Solid IT, 2015. [Citado el: 24 de febrero de 2015.] http://db-engines.com/en/ranking.

13. **Gázquez Martínez, Orelvi y Serrano García, Yadrián.** *Diseño e Implementación del módulo Diseñador de consultas del Generador Dinámico de Reportes.* Habana, Cuba : s.n., 2011.

14. **MSSQL.** Microsoft. *SQL Server. Componentes.* [En línea] 2010. [Citado el: 25 de octubre de 2014.] http://msdn.microsoft.com/es-es/library/ms187588.aspx.

15. **Rodríguez Villalobos, Alejandro y Santos Silvestre, Antonio Vicente.** *La importancia de la modularización del módulo de gestión de almacén.* Valencia, España : Escuela Politécnica de Valencia, 2006.

16. **Reingart, Mariano.** PgAdmin. *Herramienta PgAdmin.* [En línea] 2006. [Citado el: 4 de febrero de 2015.] http://www.pgadmin.org.

17. **Medina, Aldis Joan Abreu.** *Generador Dinámico de Reportes.* Habana, Cuba : s.n., 2012. ISSN: / RNPS: 2343.

18. **Cuenca Sarango, Edgar Alonso.** *Análisis comparativo de herramientas para PostgreSQL.* Chimborazo, Ecuador : s.n., 2013.

19. **Fernández Martínez, Yadiris Elena y Aguilar Chaveco, Ismel.** *Plugin editor de consultas para la herramienta de administración de bases de datos HABD.* Habana, Cuba : s.n., 2012.

20. **Fernández-Medina Patón, Eduardo.** *Metodología para el desarrollo del software: Conceptos Gnenerales.* Madrid, España : s.n., 2008.

21. **Richard Boehm, Barry y Turner.** *Balancing Agility and Discipline: A guide for the Perplexed.* California : Addison-Wesley Professional, 2003.

22. **Sandra Lorena, Anaya.** *Rup vs XP.* Habana, Cuba : Grupo_Vultur, 2010.

23. **Sanz Santos, Miguel Ángel.** *Conceptos Básicos de Lenguajes de Programación.* Madrid, España : s.n., 2000.

24. **Martínez Ladrón de Guevara, Jorge.** *Fundamentos de Programación en Java.* Madrid : Editorial EME. ISBN 978-84-96285-36-2.

25. **Universidad de Extremadura.** *Prácticas: Introducción a la programación en Java .* Caceres, España : s.n., 2012.

26. **Grazón Pérez, María Teresa.** *Sistemas Gestores de Bases de Datos.* Granada : s.n., 2010. ISSN 1988-6047.

27. **Hipp, Richard.** SQLite. *About SQLite.* [En línea] [Citado el: 12 de enero de 2015.] www.sqlite.org.

28. **Roman, Carlos Alberto.** *Lenguaje Unificado de Modelado.* Madrid, España : s.n., 2010.

29. **Hernández Orallo, Enrique.** *El lenguaje unificado de modelado (UML).* Madrid, España : s.n., 2011.

30. **López, Patricia y Ruiz, Francisco.** *Lenguaje Unificado de Modelado.* Cantabria, España : Universidad de Cantabria, 2010. IS1.

31. **González López, Pascual.** *Revista de la Facultad de Educación de Albacete.* Albacete, España : s.n., 2010. Vol. I, 12.

32. **Bustos, Guillermo.** *Visual Paradigm Community Edition.* México : s.n., 2010.

33. **Rational Software Corporation.** *Tutorials.* 2002. 800-025115-000.

34. **Caravaca Mora, Oscar Mauricio.** *Diseño de un Entorno de Desarrollo Integrado para una Unidad Controladora de procesos.* Costa Rica : Instituto Tecnológico de Costa Rica, febrero 2010.

35. **Gembeta:Dev.** Genbetadev. *IDE Netbeans.* [En línea] 09 de enero de 2014. [Citado el: 25 de octubre de 2014.] www.genbetadev.com/herramientas/netbeans-1.

36. **Gimeno, Juan Manuel y González, José Luis.** *Introducción al Netbeans.* Madrid, España : s.n., 2010.

37. **Hernández, Héctor.** *Entorno de desarrollo integrado (IDE) para aplicaciones Java.* 2014.

38. **Larman, Craig.** *UML y Patrones. 2da Edición.* s.l. : Prentice Hall, 2004. ISBN: 8420534382.

39. **Macías Mendoza, Verónica.** *Modelamiento basado en el dominio.* Guayaquil, Ecuador : s.n., 2006.

40. **Letelier, Patricio y Penadés, María Carmen.** *Metodologías ágiles para el desarrollo del software: eXtreme Programming (XP).* Valencia, España : s.n., 2012.

41. **Martínez Gutiérrez, Dalilys.** *Extensión de graficación v2.0 para PostgreSQL.* La Habana, Cuba : s.n., 2013.

42. **Quiroga, Juan Pablo.** *Requerimientos funcionales y no funcionales.* Ecuador : Universidad de los Andes, 2008.

43. **I. Jacobson, J. Rumbaugh y G. Booch.** *Proceso unificado de desarrollo de software.* California : Addison-Wesley, 1999.

44. **Barbosa Guerrero, Lugo Manuel.** *Arquitectura de software como eje temático de investigación.* México : s.n., 2006.

45. **Rodríguez Lorenzo, Amarilys y Cabrera Pedroso, Jeovany Jesús.** *Biblioteca en C++ para obtener las características de las ontologías representativas del conocimiento en la herramienta HABD.* Habana, Cuba : s.n., 2013.

46. **Tedeshi, Nicolas.** Microsoft. *¿Qué es un Patrón de Diseño?* [En línea] 2014. [Citado el: 2015 de febrero de 25.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.

47. **Tello Cáceres, Jesús.** *Patrones de diseño: ejemplo de aplicación en los Generative Learning Object.* Alcalá : Universidad de Alcalá, 2011.

48. **Miranda, Héctor Alcides Almaguer y Álvarez, Yordania Mercedes.** *Módulo para la gestión de la información de las Bases de Datos Relacionales en ontologías representativas del conocimiento.* La Habana, Cuba : s.n., 2012.

49. **Aristegui, José Luis.** *Los casos de pruebas en la prueba del software.* s.l. : Revista digital LAMPSAKOS, 2010. Vol. 3.

50. **Yague, Juan y Carbajosa, Agustín.** *Actas de los talleres de las jornadas de software y bases de datos.* Madrid : Universidad Politécnica de Madrid (UPM), 2009. Vol. 3, 4.

51. **Malfará, Dayvis y Cócaro, Fernando.** Facultad de Ingeniería. Universidad de la República de Uruguay. *Testing en eXtreme Programming .* [En línea] 2006. [Citado el: 15 de junio de 2015.] http://www.fing.edu.uy/inco/cursos/gestsoft/.../Testing%20en%20XP.doc.

52. **Mendoza Valdéz, Jorge Luis y Alfaro Medina, Jazmín.** Slideshare. *Pruebas de Sistemas y pruebas de Aceptación.* [En línea] 2013. [Citado el: 15 de junio de 2015.] http://es.slideshare.net/abnergerardo/pruebas-de-sistemas-y-aceptacion-23663195.

53. **Tenorio, Roberto Luis.** *Las pruebas de software y su importancia en las Organizaciones.* Veracruz : Universidad Veracruzana. Facultad de Contaduría y Administración, Agosto 2010.

54. **Blanco Bueno, Carlos.** *Construcción y Pruebas de Software.* Cantabria, España : Universidad de Cantabria, 2008.

55. **Sánchez, Jorge.** *Principios sobre bases de datos relacionales.* México : Creative Commons, 2004.

56. **Rodríguez, San Luis.** *Programación Declarativa Multiparadigma.* Alicante, España : Universidad de Alicante, 2006.

57. **Caseware Analytics.** Casewareanalytics. *Análisis de datos. IDEA.* [En línea] 2014. [Citado el: 25 de noviembre de 2015.] http://www.casewareanalytics.com/es/products/an%C3%A1lisis-de-datos-idea.