Universidad de las Ciencias Informáticas

FACULTAD 6

Título: Herramienta para la migración de datos entre bases de datos SQL y NO SQL.

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

Autor(es): Daniela Beltrán Sordía

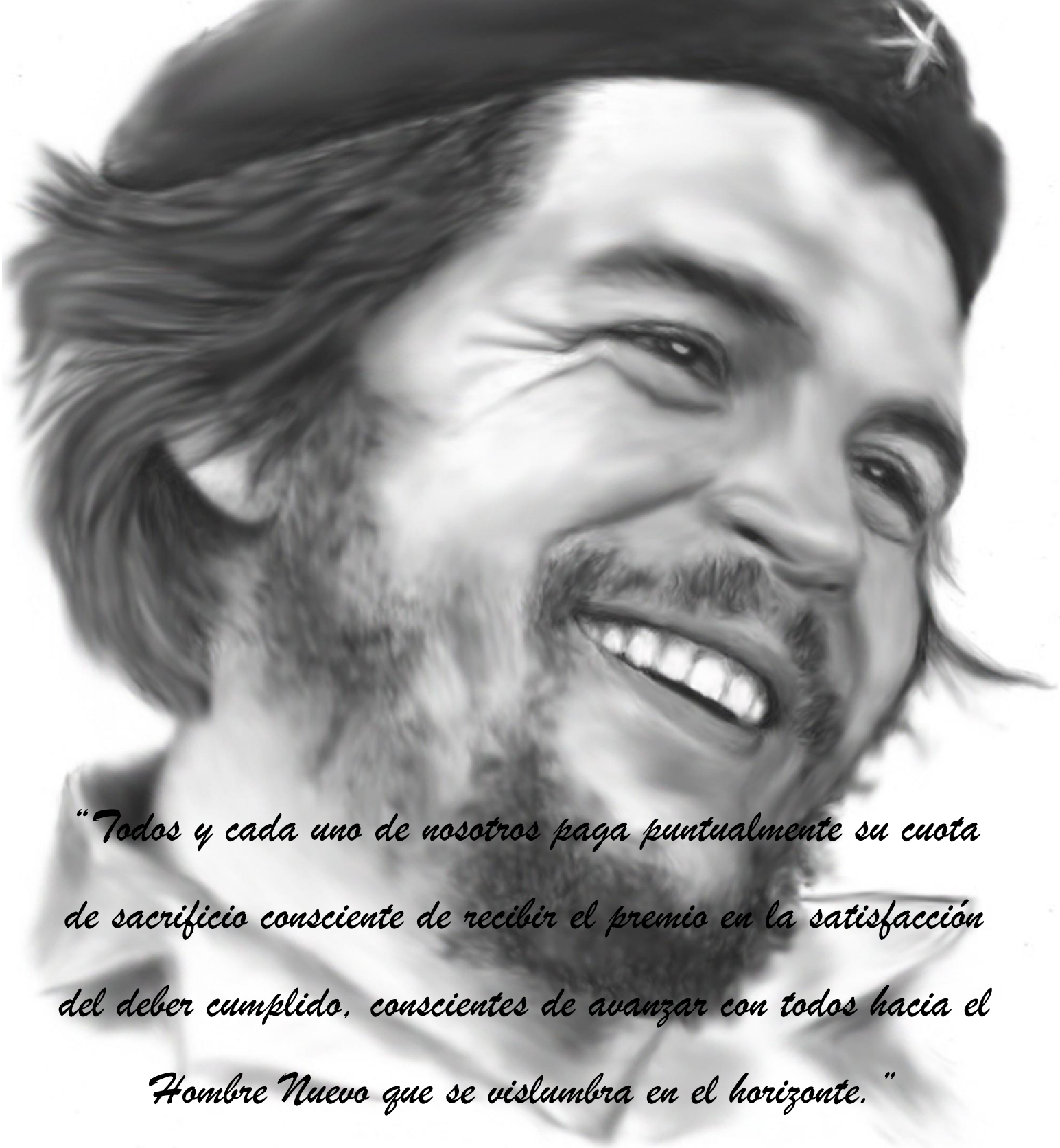
Yoel Claro Ledón

Tutor(es): Ing. Beatriz Lara Osorio

Ing. José Luis Muñoz Suárez

La Habana, Junio, 2014

Año 56 de la Revolución



"Todos y cada uno de nosotros paga puntualmente su cuota de sacrificio consciente de recibir el premio en la satisfacción del deber cumplido, conscientes de avanzar con todos hacia el Hombre Nuevo que se vislumbra en el horizonte."

Ché

DECLARACIÓN DE AUTORÍA

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

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

Daniela Beltrán Sordía

_____

Firma del Autor

Yoel Claro Ledón

_____

Firma del Autor

Ing. Beatriz Lara Osorio

_____

Firma del Tutor

Ing. José Luis Muñoz Suárez

_____

Firma del Tutor

DATOS DE CONTACTO

Tutor:

Ing. Beatriz Lara Osorio

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Email: blara@uci.cu

Tutor:

Ing. José Luis Muñoz Suárez

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Email: jlmuñoz@uci.cu

AGRADECIMIENTOS

Daniela Beltrán Sordía

Agradezco a Dios en primer lugar, a mi familia, en especial a mi madre por su abnegación y esfuerzo, a mi padre, a mis primos Clara y Pedro, a mis tíos por ser también como padres y por su esmero y confianza. Agradezco a mis tutores por su constancia y dedicación a lo largo del proceso, a mi amigo Alan, a Yoel por acompañarme incansablemente en la realización de este trabajo, a mis amigos y compañeros de aula, en especial a Jany por su amistad y preocupación, a todos los profesores que han dejado su huella en mi formación y a Miel con Limón por entenderme y apoyarme, especialmente Wilfredo, Aleida y Jesús. De manera general doy las gracias a todo el que de una forma u otra expresó su preocupación, aconsejó, y dio su apoyo, ayudándome a lograr esta meta trazada y permitiéndome cumplir uno de mis más importantes sueños. A todos y cada uno muchas gracias.

Yoel Claro Ledón

Quisiera comenzar agradeciéndole a Dios por permitirme la salud y fuerza para llegar a convertirme en profesional, a mi familia en general que siempre de una forma u otra se preocuparon por mi desarrollo en la carrera, en especial a mi mamá, mi papá, mis tíos Yoly y Juan, a mi abuela Eulalia, mi padrastro Elito ya que sin ellos no estaría hoy aquí, gracias por su confianza y apoyo incondicional. A mi hermanito Eliecer, a mi primo Ale que siempre me ha apoyado y a pesar de sus inmensas ganas de mortificarme me dejó estudiar y trabajar tranquilo cuando lo necesitaba, a mi primo Juan Carlos, mi tío Juanito, mi abuela Dalia, mis vecinos en especial Lidia y Xiomara, a mis amigos Eric y Carla que han estado siempre ahí para mí cuando los he necesitado. Le agradezco a mis hermanos Alfredo, Juan Miguel, Koisam y Adrián por aguantar mi carácter, criticarme, aconsejarme y apoyarme incondicionalmente en todo momento, a mis amigos Héctor, Luis Miguel, Alain, Batista, a mis negros Felipe y Asiel, a mis amigas Liliana, Melisa, Katy, Aimara, Betsy, Eliana, Teydi, a mis compañeros de aula y muchos más que por problemas de tiempo no puedo mencionar, gracias a todos. Le agradezco a todos los profes que de una manera u otra me han hecho crecer de manera profesional y como persona, no podría dejar de mencionar a Rachid, Karelia, Yanelis, Garnache, Glennis, Jorge Luis, a mis tutores Beatriz Lara y José Luis Muñoz que me guiaron en el camino hasta aquí, a los compañeros del tribunal y oponente, le agradezco por sus críticas constructivas y su trabajo profesional. Agradecerle a mi compañera de tesis por su dedicación y esfuerzo en esta tarea, por último y no por menos importante a mis amigos Yadian y Juan Carlos, les agradezco de corazón todo lo que hicieron por mí, sin ustedes no estaría hoy aquí.

RESUMEN

El desarrollo de las tecnologías y la informatización ha incrementado el cúmulo de información en las bases de datos de muchas empresas e instituciones, haciendo necesaria la migración de los datos hacia nuevas alternativas capaces de responder a los problemas que trae consigo dicho aumento. En la Universidad de las Ciencias Informáticas (UCI) fue desarrollada MIGDAT, un sistema para la migración entre PostgreSQL y gestores NoSQL orientados a documentos (MongoDB y CouchDB), su diseño no se adapta a los cambios, es muy complejo extenderla debido a las desventajas con las que cuenta, como son: no fueron implementados correctamente los patrones de diseño que garantizan la alta cohesión y el bajo acoplamiento, incorrecto uso del patrón experto, crecimiento cuadrático de la clase Migración al agregar nuevos gestores, clases abiertas a modificaciones, está centrada solamente en los gestores NoSQL orientados a documentos; es por todo lo planteado que se decide rehacerla. Este trabajo propone una herramienta que permita realizar la migración de datos entre los gestores: PostgreSQL, MongoDB, Cassandra y CouchDB, capaz de minimizar los problemas que trae realizar la migración de forma manual y a su vez permitir el trabajo con grandes volúmenes de datos. El uso de la herramienta contribuirá a la disminución de esfuerzo y tiempo, así como al aumento de la calidad del proceso de migración.

PALABRAS CLAVE

Bases de datos, migración de datos, gestores, PostgreSQL, MongoDB, Cassandra, CouchDB, NoSQL.

ABSTRACT

The development of technology and informatization has increased the wealth of information in the databases of many companies and institutions, necessitating the migration of data to new alternatives able to respond to the problems brought about this increase. At the Informatics Sciences University was developed MIGDAT, a system for migration between PostgreSQL and NoSQL document-oriented management systems (MongoDB and CouchDB), its design does not adapt to changes, it is very complex to extend due to the disadvantages that it has, such as: were not correctly implemented design patterns to ensure high cohesion and low coupling, incorrect use of expert pattern, quadratic growth of the Migration class when new managers are added, classes open to modifications, focuses only on NoSQL document-oriented management systems, that´s why was decided to do it over. This paper proposes a tool that allows migrating data between the following management systems: PostgreSQL, MongoDB, Cassandra and CouchDB, able to minimize the problems associated to manually migration and in turn, it allows working with large volumes of data. The use of the tool will contribute to the reduction of effort and time, and to increase the quality of the migration process.

KEYWORDS

Databases, data migration, managers, PostgreSQL, MongoDB, Cassandra, CouchDB, NoSQL.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

En la actualidad el SQL (por sus siglas en inglés Structured Query Language) es el estándar de facto de la inmensa mayoría de los Sistemas Gestores de Base de Datos (SGBD) comerciales, el SQL es un lenguaje de acceso a bases de datos (BD) que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones, los principales gestores de base de datos relacional son: Microsoft SQL Server, PostgreSQL, Microsoft Access, MySQL, InterBase y Oracle.

Desde finales de los años 70 el mercado ha operado con bases de datos relacionales; Microsoft, International Business Machines (IBM) y Oracle, cuentan con estas plataformas en su oferta. Sin embargo, la gran variedad de nuevas fuentes de información (redes sociales, imágenes, videos, etc.) han duplicado en pocos años el universo digital y provocado una oleada de datos de tal magnitud, que ha puesto en apuros a estos sistemas, muy verticales y poco escalables para afrontar este volumen.

Las restricciones mencionadas han hecho que se busquen nuevas alternativas, dando vida al movimiento NoSQL (Not Only SQL) quien permite manejar altos volúmenes de datos (big data), ya sea estructurados como no estructurados, con un buen desempeño para responder las consultas de los usuarios, es decir, con tiempos de espera mínimos. (1)

Este movimiento creció al amparo de las principales compañías de Internet, enfrentadas al boom de los datos no estructurados. Twitter y Facebook, por ejemplo, hoy utilizan Cassandra, uno de los proyectos más conocidos en la línea de NoSQL, desarrollado por la red social de Mark Zuckerberg, ejemplo de otras implementaciones lo son RavenDB, Neo4j, Big Table, Riak, Hadoop, MongoDB y CouchDB. (1)

NoSQL es una atractiva alternativa, pero no la respuesta a todo el almacenamiento de datos, deben seguir siendo las bases de datos relacionales la primera opción en muchos de los casos, las garantías de atomicidad, consistencia, aislamiento y durabilidad son importantes para diversas aplicaciones. (1)

La necesidad de proporcionar información procesada a partir de grandes volúmenes de datos, el rendimiento, la escalabilidad, la reducción de costos, nuevos procesos de negocio, entre otros escenarios, son motivos suficientes para llevar a cabo la migración de datos.

El desarrollo y evolución de la ciencia y la tecnología conlleva a que muchas empresas, instituciones o mecanismos que hagan uso de base de datos relacionales se vean afectadas por el incremento en los

datos de estas y surja la necesidad de migrar la información a un gestor de tipo NoSQL, garantizando así un mejor manejo de la información, facilidad de distribución, alta escalabilidad y excelentes tiempos de respuesta. Estas nuevas necesidades traen consigo nuevos retos a los desarrolladores y amplían el espectro de oportunidades para los que se dedican a proveer servicios de migración de datos. (2)

El proceso de migración de datos trae consigo un cierto grado de riesgo, debido a la falta de conocimientos acerca de las características y funcionalidades de los gestores en juego, ya que llevar a cabo este proceso implica un dominio de estos elementos.

En el departamento de PostgreSQL perteneciente al centro de Tecnología de Gestión de Datos (DATEC), ubicado en la Universidad de Ciencias Informáticas (UCI), se trabaja sobre líneas de investigación referentes a la migración de datos y a la aplicación de las nuevas tecnologías existentes para llevar a cabo este proceso, de ahí que constituya una excelente oportunidad de negocio para el centro contar con elementos que den soporte al proceso de migración de datos ya que la migración puede realizarse de forma manual, pero esto resultaría difícil y lento, sobre todo si la base de datos a migrar es amplia en volúmenes de datos.

Por todo lo anteriormente planteado se declara como **problema de la investigación**: ¿Cómo automatizar el proceso de migración de datos entre gestores SQL y NoSQL de manera que disminuya los riesgos que trae hacerlo de forma manual y permita el manejo de grandes volúmenes de datos?

Definiéndose como **objeto de estudio:** el proceso de migración de datos, enmarcado en el **campo de acción**: proceso de migración de datos entre gestores de bases de datos SQL y NoSQL. Para solucionar el problema planteado, se identifica como **objetivo general de la investigación:** desarrollar una herramienta informática que permita realizar la migración de los datos entre gestores de bases de datos SQL y NoSQL.

En correspondencia con el mismo se definen los siguientes **objetivos específicos:**

- Analizar las técnicas y tecnologías para la migración de datos entre gestores de bases de datos SQL y NoSQL.
- Diseñar la herramienta para la migración de datos entre gestores de bases de datos SQL y NoSQL.
- Implementar la herramienta para la migración de datos entre dichos gestores.

- Validar la herramienta.

Se hace necesaria la formulación de las siguientes **preguntas científicas:**

- ¿Cómo diseñar una herramienta que realice la migración de datos entre gestores de base de datos SQL y NoSQL?
- ¿Qué metodología y lenguaje de programación utilizar para la implementación de la herramienta de migración de datos?
- ¿Cómo automatizar el proceso de migración de datos entre gestores de base de datos SQL y NoSQL?
- ¿Cómo disminuir los riesgos que trae realizar la migración de los datos de forma manual?
- ¿Cómo permitir el manejo de grandes volúmenes de datos?

Para dar cumplimento a los objetivos específicos antes planteados, se propone la realización de las siguientes **tareas de la investigación:**

- Caracterización de las herramientas, tecnologías y metodologías a utilizar en el desarrollo de la herramienta para la migración de datos entre gestores de bases de datos SQL y NoSQL.
- Identificación de las funcionalidades de la herramienta para la migración de datos entre gestores de bases de datos SQL y NoSQL.
- Diseño de la herramienta a partir de las funcionalidades identificadas.
- Implementación de las funcionalidades identificadas.
- Diseño de los casos de prueba para la herramienta de migración de datos entre gestores de bases de datos SQL y NoSQL.
- Aplicación de los casos de prueba para validar que la herramienta responde a las necesidades del usuario final.

La investigación está sustentada en **los métodos teóricos** que facilitarán la investigación y servirán de guía para organizar mejor el trabajo y de esta forma posibilitar el entendimiento del problema, estudiarlo, analizarlo y llegar a conclusiones para la solución. A continuación se explica la utilización de los que han sido seleccionados:

Los **métodos teóricos** utilizados fueron**:**

- **Histórico/Lógico:** Este método permite establecer la necesaria correspondencia entre los elementos lógicos e históricos, con el fin de analizar la evolución histórica de los conceptos trabajados en la investigación. Se empleó debido a que se hizo una profunda investigación para analizar la trayectoria histórica de las bases de datos tanto SQL como NoSQL, elaborando el estudio del estado del arte de este tema. Obteniendo una tendencia de cómo se comporta en la actualidad.
- **Analítico/Sintético:** Este método permite analizar, comparar y confrontar las diferentes fuentes bibliográficas consultadas, en cuanto a tendencias actuales para el diseño e implementación de modelos dimensionales en base de datos NoSQL, lo que permite determinar las regularidades y así tomar decisiones. Además, analizar donde se emplearía la solución de la propuesta en cuestión**.**
- **Modelación:** A partir del cual se representará la realidad existente, modelando la necesidad actual en un diagrama de dominio que podrá sustituir al objeto de estudio definido para esta investigación. Mediante la modelación es posible obtener una mejor comprensión del objeto de estudio.

El presente trabajo de diploma está estructurado en tres capítulos:

Capítulo I: "Fundamentos teóricos de la investigación": El capítulo comprende el estudio de los aspectos teóricos referentes al proceso de migración de datos. Se realiza además, un análisis de las metodologías, herramientas y tecnologías a utilizar para el desarrollo de la herramienta de migración de datos entre los gestores de bases de datos SQL y NoSQL.

Capítulo II: "Características y diseño del sistema": En el presente capítulo se analizan cada uno de los artefactos y elementos para el diseño de la aplicación, lo que permitirá un mejor entendimiento de las necesidades de la herramienta de migración de datos a desarrollar. Además se exponen las principales características de la herramienta, su diseño, su arquitectura y los requisitos a tener en cuenta en su desarrollo.

Capítulo III: "Implementación y pruebas": En el capítulo se describen los principales artefactos generados en la fase de implementación y prueba de la herramienta de migración de datos entre los gestores de base SQL y NoSQL. Se desarrolla la descripción de la implementación del sistema y se especifican las pruebas a las que fue sometida la herramienta de migración de datos en cada una de las iteraciones. Proceso que guía la identificación y corrección de fallos cometidos en las Historias de Usuario (HU), así

como su verificación y materialización. Contribuye a elevar la calidad del producto desarrollado y la seguridad en los programadores para efectuar modificaciones.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

El capítulo comprende el estudio de los aspectos teóricos que soportan la investigación a desarrollar. Se realiza además un análisis de las metodologías, herramientas y tecnologías a utilizar para el desarrollo de la herramienta de migración de datos entre gestores de bases de datos SQL y NoSQL.

1.1 Conceptos asociados a la Investigación

1.1.1 Base de datos

Una base de datos es un “almacén” que permite guardar grandes cantidades de información de forma organizada para que luego se puedan encontrar y utilizar fácilmente. El término de bases de datos fue escuchado por primera vez en 1963, en un simposio celebrado en California, USA. Una base de datos se puede definir como un conjunto de información relacionada que se encuentra agrupada o estructurada. (3)

Desde el punto de vista informático, la base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos, cada base de datos se compone de una o más tablas que guarda un conjunto de datos, cada tabla tiene una o más columnas y filas. Las columnas guardan una parte de la información sobre cada elemento que queramos guardar en la tabla, cada fila de la tabla conforma un registro. (3)

Su definición consiste en que es un conjunto de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular. (3)

Principales características de los sistemas de bases de datos:

- Independencia lógica y física de los datos.
- Redundancia mínima.
- Acceso concurrente por parte de múltiples usuarios.
- Integridad de los datos.
- Consultas complejas optimizadas.
- Seguridad de acceso y auditoría.
- Respaldo y recuperación.
- Acceso a través de lenguajes de programación estándar. (3)

1.1.2 Sistema Gestor de Bases de Datos

Un Sistema Gestor de Base de Datos (SGBD, en inglés DBMS: Database Management System) es un sistema de software que permite la definición de bases de datos; así como la elección de las estructuras de datos necesarias para el almacenamiento y búsqueda de los datos, ya sea de forma interactiva o a través de un lenguaje de programación (3). El propósito general de los sistemas de gestión es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización.

Objetivos:
Dentro de los objetivos más importantes que debe cumplir un SGBD se encuentran:

- Definir la Base de Datos mediante un Lenguaje de Definición de Datos.
- Separar la descripción y manipulación de los datos.
- Permitir la inserción, eliminación, actualización y consulta de los datos mediante el Lenguaje de Manejo de Datos.
- Gestionar la estructura física de los datos y su almacenamiento.
- Proporcionar un mecanismo de vistas, que permita a cada usuario tener su propia vista o visión de la base de datos.
- Eliminar la redundancia de datos.
- Proveer interfaces procedimentales y no procedimentales, permitiendo la manipulación por usuarios interactivos y programadores.
- Independizar la estructura de la organización lógica de los datos (Independencia física).
- Independizar la descripción lógica de la Base de datos y las descripciones particulares de los diferentes puntos de vistas de los usuarios.
- Permitir una fácil administración de los datos.

1.1.3 Modelos de base de datos

Un modelo de base de datos es un tipo de modelo de datos que determina la estructura lógica de una base de datos y de manera fundamental determina el modo de almacenar, organizar y manipular los datos. (4) Los modelos de datos no son cosas físicas: son abstracciones que permiten la implementación

de un sistema eficiente de base de datos; por lo general se refieren a algoritmos, y conceptos matemáticos.

Los modelos de base de datos Relacionales y No Relacionales serán objeto de estudio en la realización de la herramienta para la migración de datos, además de éstos existen otros de no menos importancia como son:

Bases de datos jerárquicas.

Base de datos de red.

Bases de datos orientadas a objetos.

Bases de datos deductivas.

Bases de datos distribuidas. (4)

1.1.4 Base de datos relacional

El concepto de base de datos relacional fue definido por Edgar Frank Codd a finales de los años 60. Una base de datos relacional es una base de datos en donde todos los datos visibles al usuario están organizados estrictamente como tablas de valores, y en donde todas las operaciones de la base de datos operan sobre estas tablas. Estas bases de datos son percibidas por los usuarios como una colección de relaciones normalizadas de diversos grados que varían con el tiempo. (4)

El modelo relacional resulta en la actualidad el más utilizado para modelar problemas reales y administrar datos dinámicamente. Su idea fundamental es el uso de relaciones. Estas relaciones podrían considerarse en forma lógica como un conjunto de datos llamados "tuplas". Tratando cada relación como si fuese una tabla que está compuesta por registros, que representarían las tuplas, y campos. En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia. La información puede ser recuperada o almacenada mediante consultas que ofrecen una amplia flexibilidad y poder para administrar la información. El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL (del inglés, 'Structured Query Language'), un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales. (4)

1.1.5 Gestor de bases de datos relacional

PostgreSQL

Es un SGBD relacional que incluye características como la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. Posee código abierto y licencia BSD (Berkeley Software Distribution). Sus características técnicas la hacen una de las bases de datos más potente y robusta del mercado. Su desarrollo comenzó hace más de 16 años, y durante este tiempo, estabilidad, potencia, robustez, facilidad de administración e implementación de estándares han sido las características que más se han tenido en cuenta durante su desarrollo. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema. (5)

Durante los años de existencia del Proyecto PostgreSQL, el tamaño del mismo, tanto en número de desarrolladores, como en números de línea de código, funciones y complejidad del mismo ha ido aumentando año tras año. Para el desarrollo de MIGDAT se escoge a PostgreSQL como gestor relacional ya que es el más usado entre los relacionales, posee código abierto, es multiplataforma, además de contar con una inmensa bibliografía entre otras características antes mencionadas.

1.1.6 Bases de datos NoSQL

Originalmente el término NoSQL, que fue acuñado en 1998, se refería a una base de datos relacional de código abierto que no usaba un lenguaje de consultas SQL. Hasta 2009 estas cinco letras cayeron en el olvido, pero fue Johan Oskarsson, entonces empleado de Last.fm, quien organizó un evento para tratar las bases de datos distribuidas de código abierto no relacionales, llamándolas “NoSQL”, Not-Only SQL. (6)

Definitivamente, NoSQL hace referencia a una multitud de bases de datos que intentan solventar las limitaciones que el modelo relacional presenta en entornos de almacenamiento masivo de datos, y concretamente en las que tiene en el momento de escalar, donde es necesario disponer de servidores muy potentes y de balanceo de carga.

1.1.7 Clasificación de las bases de datos NoSQL:

BD No Relacional Orientada a Key-Value:

Llave-valor es la forma más representativa, su uso es muy parecido al de una tabla de Hash, en la que se almacena una clave, y un valor. Entre sus características principales se encuentra la rapidez para las operaciones de lectura/escritura y que son fácilmente escalables particionando la clave, de esta forma se pueden almacenar las distintas claves en distintos servidores dependiendo de su valor inicial. (7)

BD No Relacional Orientada a Documentos:

Estas almacenan la información como un documento (generalmente con una estructura simple como JSON o BSON y con una llave única. Podemos encontrar a MongoDB y CouchDB entre las más importantes de este tipo. (7)

BD No Relacional Orientada a Grafos:

En el caso de estas bases de datos almacenan la información como grafos, de forma que las relaciones entre nodos pueden tener atributos. Son recomendables para sistemas que se puedan representar en forma de red de manera sencilla. Son muy útiles para redes sociales como SourceForge y Amazon. (7)

BD No Relacional Orientada a Columnas:

Almacenan toda la información en columnas de esta forma las lecturas son muy rápidas, pero se sacrifica mucho tiempo para las escrituras, por lo que no son recomendables a no ser que el número de lecturas sea muy superior al número de escrituras. Ejemplos: HBase, Hypertable, Cassandra, Riak. Buenas en: gestión de tamaño, cargas de escrituras masivas y alta disponibilidad. (7)

1.1.8 Gestores de bases de datos NoSQL

Para el desarrollo de la herramienta se seleccionó por parte de los gestores NoSQL a MongoDB, CouchDB y Cassandra por sus características en particular y por encontrarse ubicados entre los primeros en el ranking de su tipo a nivel mundial (ver Tabla 1) y ser de los más populares en utilización y preferencia. También se tuvo en cuenta los beneficios que estos aportan al mundo de las bases de datos, entre estos beneficios está el de ser productos de código abierto y esto proporciona que sean más confiables, seguros y rápidos de implementar que las alternativas propietarias. MongoDB y CouchDB son buenas opciones para el almacenamiento de escritura con alta frecuencia, rara vez leen los datos estadísticos, tales como web y contador de visitas. Cassandra cuenta con un buen rendimiento en entornos con aplicaciones de alta disponibilidad, donde el tiempo de funcionamiento máximo es de vital importancia.

Tabla 1. Ranking mundial de gestores de bases de datos.[1]

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1468.06	-149.13
2.	2.	MySQL	Relational DBMS	1309.29	+55.02
3.	3.	Microsoft SQL Server	Relational DBMS	1205.87	-28.58
4.	4.	PostgreSQL	Relational DBMS	230.96	+40.12
5.	5.	DB2	Relational DBMS	190.61	+24.71
6.	6.	MongoDB	Document store	183.07	+21.20
7.	7.	Microsoft Access	Relational DBMS	171.67	+30.07
8.	8.	SQLite	Relational DBMS	99.50	+20.72
9.	9.	Sybase	Relational DBMS	95.28	+17.53
10.	↑ 11.	Cassandra	Wide column store	80.51	+22.93
11.	↓ 10.	Teradata	Relational DBMS	63.72	+3.60
12.	12.	Solr	Search engine	62.45	+15.92
13.	13.	Redis	Key-value store	51.78	+11.21
14.	14.	FileMaker	Relational DBMS	47.64	+12.54
15.	↑ 17.	Informix	Relational DBMS	36.60	+9.32
16.	↓ 15.	Memcached	Key-value store	36.09	+7.59
17.	↓ 16.	HBase	Wide column store	35.28	+7.70
18.	↑ 19.	Hive	Relational DBMS	26.59	+8.92
19.	↓ 18.	CouchDB	Document store	25.41	+6.07
20.	20.	Netezza	Relational DBMS	17.97	+3.79
21.	21.	Elasticsearch	Search engine	17.12	+4.22

MongoDB

(De la palabra en inglés “humongous” que significa enorme) es un sistema de bases de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto. MongoDB es un sistema de bases de datos **multiplataforma**, de esquema libre. Esto significa que cada entrada o registro puede tener un esquema de datos diferentes, con atributos o “columnas” que no tienen por qué repetirse de un registro a otro. Está escrito en C++, además, está licenciado como GNU ALP 3.0, de modo que se trata de un software de licencia libre. Funciona en sistemas operativos Windows, Linux, OS X y Solaris. (8)

Las características que más destacarían de MongoDB son su **velocidad** y su rico pero sencillo **sistema de consulta** de los contenidos de la base de datos. Alcanza un balance perfecto entre rendimiento y

[1] http://lineadecodigo.com/tag/bases-de-datos

funcionalidad, incorporando muchos de los tipos de consulta que utilizaríamos en un sistema relacional, pero sin disminuir en rendimiento. En MongoDB, cada registro o conjunto de datos se denomina **documento**. Los documentos se pueden agrupar en **colecciones**, las cuales se podría decir que son el equivalente a las tablas en una base de datos relacional (sólo que las colecciones pueden almacenar documentos con muy diferentes formatos, en lugar de estar sometidos a un esquema fijo). Se pueden crear **índices** para algunos atributos de los documentos, de modo que MongoDB mantendrá una estructura interna eficiente para el acceso a la información por los contenidos de estos atributos. (8)

Cassandra

Cassandra es un gestor de Bases de Datos NoSQL. Posee una estructura clave-valor para almacenar datos de manera escalable, consistente y distribuida. Fue creada para manejar grandes volúmenes de datos distribuidos en numerosos servidores estándar, ofreciendo alta disponibilidad.
La historia de Cassandra nace directamente de la experiencia que supuso Dynamo para Amazon. Facebook contrató a uno de sus autores y le encargó diseñar un nuevo sistema para sus datos. Este ingeniero desarrolló Cassandra y, en 2008, Facebook liberó su código. (9)

Características:

- Punto de falla: maneja grandes volúmenes de datos distribuidos en numerosos servidores estándar, ofreciendo alta disponibilidad sin ningún punto único de falla.
- Basada en clave-valor: tiene un almacén de valores de claves manejado con consistencia eventual (modelo de consistencia usado en programación paralela). Las claves mapean hacia múltiples valores que se agrupan en familias de columnas. Esas familias se definen cuando se crea una base Cassandra, pero luego se les puede agregar columnas a las diferentes familias. Las columnas se pueden agregar a claves específicas y así diferentes claves tendrán diferentes cantidades de columnas dentro de una misma familia. Los valores de una familia de columnas para cada clave se almacenan juntos, haciendo de Cassandra un híbrido entre una DBMS orientada a columnas y un almacén de datos orientado a filas.
- Soporte comercial: debido a que es un producto relativamente nuevo, recién acaba de aparecer una primera empresa ofreciendo soporte comercial para Cassandra. Se trata de Riptano y su plan consiste en usar a Cassandra como plataforma central e ir agregándole porciones propietarias para hacer a su versión comercial. Uno de sus primeros servicios podría ser el de migración para que usuarios de otras bases de datos puedan pasar su información a Cassandra.

Ventajas:

1. Dispone de consistencia eventual, al igual que el sistema Dynamo de Amazon.
2. Proporciona un modelo de datos basados en ColumnFamily, más rico que el tradicional modelo de clave/valor, al igual que el modelo BigTable de Google.
3. Es altamente escalable y distribuida. Normalmente se ejecutan en clusters de servidores formados por ordenadores baratos, por lo que la expansión del sistema es realmente sencilla.
4. Es rápida ya que elimina el cuello de botella que supone el tener que traducir las consultas a lenguaje SQL. (9)

No sólo Facebook utiliza Cassandra, empresas como Digg, Cisco, IBM, Cloudkick u Ooyala se sirven de ella para sus proyectos.

CouchDB

Couch (sofá, diván) DB (del inglés Database) es un nuevo concepto de bases de datos, de la mano de la comunidad Apache. El nombre no ha sido elegido al azar, ni de manera casuística. Su nombre evoca relax, y ésa es su filosofía. El proyecto de software libre nace en Abril de 2005 de la idea de Damien Katz de crear un nuevo motor de bases de datos orientado a documentos para entornos web. En un primer momento CouchDB estaba escrito en C++ y ya incluía algunas de las características esenciales del proyecto, tales como: la falta de esquema relacional, almacenamiento y actualizaciones atómicas. En estos inicios, el proyecto estaba muy influenciado por Lotus Notes (sistema cliente/servidor) y su idea de base de datos orientada a documentos. (10)

CouchDB no sigue el concepto de base de datos tradicional entidad-relación (modelo relacional), sino un modelo de almacenamiento documental, gestionado por un potente motor de consultas sencillísimo de utilizar. Las BD CouchDB guardan documentos nombrados de modo unívoco y se guardan en formato JSON o BSON que permite a las aplicaciones leer y modificar estos documentos. (10)

Cada documento puede tener campos no definidos en otros documentos. Los mismos no están asociados a un esquema de bases de datos estricto. Cada documento contiene metadatos (datos sobre datos) como el identificador unívoco del documento (id) y su número de revisión (rev). Los campos de un documento pueden ser de varios tipos como strings, números, booleanos, colecciones entre otros. Cuando se hacen cambios sobre un documento CouchDB se crea una nueva versión del documento, denominado revisión.

Se mantiene un historial de modificaciones gestionado automáticamente por la BD. CouchDB no dispone de mecanismos de bloqueo (locking) ante escrituras. (10)

Tras estos hechos, lo más reseñable es que en Noviembre de 2008 CouchDB pasa a ser un proyecto Apache de primer nivel, al lado de otros como Apache HTTP Server, Tomcat, Ant, entre otros. (10)

Características principales de CouchDB:

- Plataforma de bases de datos simplificada.
- Centrada en documentos.
- No sigue el modelo relacional.
- Facilita la distribución, alta escalabilidad y la tolerancia a fallos.
- Preparada para funcionar offline.
- Replicación bidireccional.
- Orientada a Internet.

CouchDB toma como referente el teorema de CAP:

Consistency: Consistencia o la capacidad de que todos los clientes vean los mismos datos, incluso con actualizaciones concurrentes.

Availability: Disponibilidad o la capacidad de que todos los clientes accedan a la misma versión de los datos.

Partition tolerance: Partición tolerante o la capacidad de que la base de datos pueda estar dividida en múltiples servidores.

Licencia de CouchDB:

CouchDB es adoptado por IBM y posteriormente por Apache Foundation con lo que la licencia pasa de ser GNU GPL (General PublicLicense) a ser Apache License. (10)

1.1.9 Migración de datos

Se denomina migración de datos, al proceso que tiene por objeto tanto la importación como la exportación de una determinada información almacenada en un sistema de bases de datos, para llevar a cabo su traspaso. (11)

Una migración de bases de datos es un proceso que se realiza para mover o trasladar los datos almacenados de un formato de datos a otro, para lo cual es indispensable que antes de empezar cualquier proceso de esta naturaleza, se tenga clara y documentada la razón por la cual se está migrando, además de elaborarse la planeación detallada de las actividades que se llevarán a cabo. (12)

Existen diversos motivos para realizar una migración, tales como: mejorar el desempeño de la BD, cumplir con nuevos requerimientos de usuario, de la aplicación o políticas de seguridad; así como la compatibilidad con otras aplicaciones, la actualización de versiones, la estandarización de la tecnología de información, la reducción de costos que se puede tener al cambiar por software libre y nuevos procesos de negocio. (12)

Según Susana Laura Corona (Dirección General de Servicios de Cómputos, Universidad Nacional Autónoma de México), el proceso de migración cuenta con una serie de fases o etapas que no se pueden violar, los cuales son conocidos como factores críticos de éxitos. (12)

1.1.9.1 Factores Críticos de Éxito de una migración de datos

Son los elementos o aspectos que resultan esenciales para que se alcancen los mejores resultados de la herramienta. Tales resultados satisfactorios aseguran un proyecto exitoso de migración de datos. A continuación se exponen dichos factores:

- Planeación de la fuente de datos: en esta etapa se deben establecer los objetivos, alcance, estrategias y fases a seguir.
- Mapeo de la Información: donde se tienen en cuenta las características de cada uno de los elementos y tablas con las cuales se va a trabajar, así como su relación e integridad origen.
- Selección de la Herramienta: es importante hacer un análisis de las diferentes alternativas existentes buscando la mejor opción considerando la relación costo-beneficio de cada una de ellas.
- Pruebas: pruebas de las aplicaciones que usarán la base de datos migrada y pruebas del proceso de migración.
- Migración: procesos de extracción, transformación y carga, los cuales permiten obtener los datos desde su origen, modificarlos para cumplir con la integridad y la consistencia e insertarlos finalmente en la BD destino.
- Resultados: medición y análisis de los resultados. (12)

1.2 Herramienta para la migración de datos MIGDAT

MIGDAT es una herramienta informática que se encarga de realizar la migración de los datos entre los gestores de base de datos PostgreSQL, MongoDB y CouchDB, la cual tiene como función automatizar el proceso de migración de datos. Permite que la pérdida de información debido al gran flujo de datos a migrar sea mínima. Su diseño pretende colaborar con la persistencia de los datos y evitar la interferencia en la confidencialidad y persistencia de la información almacenada en las base de datos. La herramienta es un producto propio del departamento de PostgreSQL, para su desarrollo se utilizó el lenguaje de programación Python (13). La herramienta cubre las siguientes funcionalidades: crear la conexión con la BD, seleccionar los gestores implicados en la migración, seleccionar el tipo de migración y realizarla.

Al ser MIGDAT un producto propio del centro DATEC, se realizó un análisis exhaustivo con el objetivo de determinar si extender sus funcionalidades era viable y de esa manera reutilizar todo lo posible con el consecuente ahorro de tiempo. Dicho análisis se dividió en dos partes, análisis de las características del sistema a partir del estudio de los requisitos funcionales y no funcionales del mismo y el análisis estructural mediante el estudio de la arquitectura y el diseño del sistema.

En el análisis de las características de MIGDAT se identificaron como principales carencias:

- No permite el mapeo de los tipos de datos de la fuente al destino, el cual constituye un paso fundamental en la metodología de migración de datos, ya que la integridad y consistencia de estos no se pueden violar.
- MIGDAT intenta migrar cada concepto de una sola vez; al no utilizar carga por lotes (paginado de datos) corre el riesgo de sobrecargar la memoria de la máquina al migrar grandes volúmenes de datos afectando de esta manera el rendimiento.
- Está centrada solamente en los gestores NoSQL basados en documentos.

En el estudio de la arquitectura del sistema, solo observando la clase Migración se obtuvo la siguiente lista de deficiencias del diseño:

1. Alto acoplamiento: la clase está muy acoplada a los distintos gestores de bases de datos y depende de objetos concretos y no de abstracciones como plantea el principio de inversión de la dependencia.

2. Baja cohesión: implementa métodos específicos para distintos gestores, violando el principio de única responsabilidad.
3. Incorrecto uso del patrón experto: la clase debe ser experta en el proceso de migración de datos indistintamente de quienes sean el origen y el destino.
4. Crecimiento cuadrático de la clase Migración (Ver Anexo 2) al agregar nuevos gestores: al poseer un método por cada permutación de los gestores soportados; si se deseara adicionar “x” gestores a los “k” existentes se debería agregar “y” nuevos métodos a la clase Migración, esto se describe por la ecuación $y = x^2 + 2kx - x$ implicando así un crecimiento cuadrático de dicha clase y dificultando el mantenimiento, para un mejor entendimiento ver Anexo 2.
5. Clase abierta a modificaciones: para ampliar los gestores soportados en MIGDAT, es necesario modificar la clase Migración, contrario a lo que aconseja el principio abierto y cerrado.

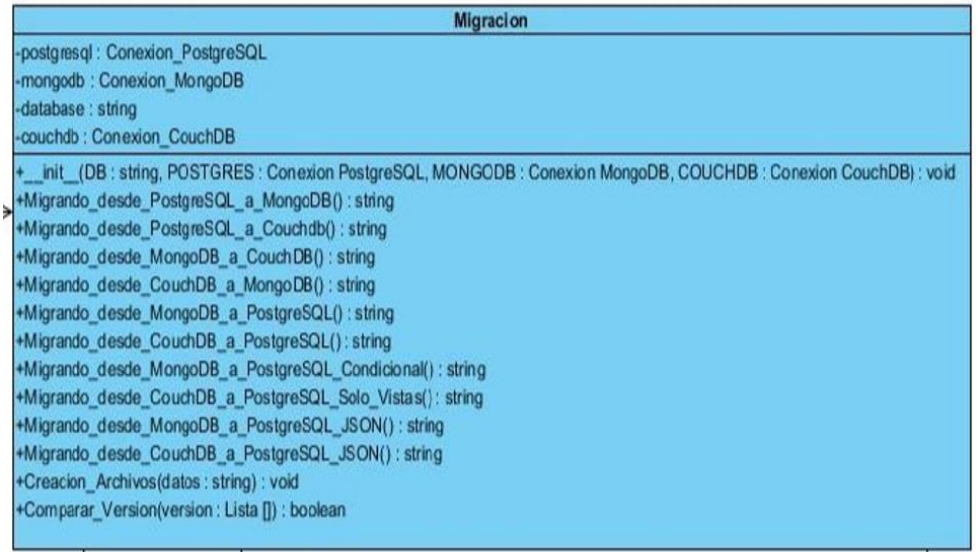


Figura 1. Clase Migración de la herramienta MIGDAT.

Luego de un análisis del funcionamiento de MIGDAT se concluyó que su diseño no se adapta a los cambios. Es muy complejo extenderla debido a la serie de desventajas con las que cuenta, es por todo lo anteriormente planteado que se decide rehacerla.

1.3 Descripción del proceso de migración en la solución

Proceso de migración

Cada gestor, tanto origen como destino conoce cómo transformar sus datos hacia la Estructura de Transferencia Intermedia (ETI) y de la ETI hacia ellos. A continuación se muestran los pasos en el proceso de migración:

- Se extrae la información del gestor de bases de datos origen hacia la ETI, transformando así su modelo en la estructura intermedia.
- El gestor de bases de datos destino toma la estructura intermedia del gestor de base de datos origen y lo transforma en su propio modelo. (Ver Anexo 1)

ETI (Estructura de Transferencia Intermedia)

La estructura de datos intermedia surge para apoyar el proceso de migración, siendo la construcción de la misma el paso intermedio de dicho proceso. Los gestores de bases de datos soportados por la herramienta de migración no tienen que conocer cómo migrar entre ellos directamente. Un gestor solo debe conocer cómo migrar su información hacia la estructura de datos intermedia y cómo migrar la información contenida en la estructura de datos hacia su propio modelo. Esto asegura que si se eliminan o se insertan gestores nuevos a la herramienta no es necesario modificar las funciones de migración de los ya soportados por la aplicación. Está compuesta por estructuras llamadas containers: ETI, GroupContainer, MainContainer y Leaf.

A continuación en la Tabla 2 se puede ver la ETI y su estructura homóloga en cada gestor:

Tabla 2. Estructura de Transferencia Intermedia

ETI	**PostgreSQL**	**MongoDB**	**Cassandra**	**CouchDB**
ETI	Nombre de Base de Datos	Nombre de Base de Datos	Nombre de Base de Datos	Nombre de Base de Datos
GroupContainer	Schema	-	-	-
MainContainer	Tables	Collections	ColumnFamily	-
Leaf	Column	Document	Row	Document

Trabajo con los tipos de datos

El diseño de la herramienta para la migración de datos permite que la pérdida de tipos de datos en el proceso de migración sea mínima, en caso de PostgreSQL se realiza un tratamiento con los tipos de datos de los valores a insertar donde se convierten éstos para que el lenguaje SQL pueda interpretarlo y así poder insertarlos en el gestor. En Cassandra es utilizado el lenguaje CQL, que se encarga de interpretar y

transformar los datos a insertar en el gestor y convertirlos para que internamente este los trabaje con arreglos de bytes. En CouchDB son tratados los tipos de datos fecha y tiempo que son convertidos a “timestamp" (tipo de dato de fecha y tiempo) para así aprovechar su estructura. En el caso de MongoDB, el driver utilizado para la comunicación con el gestor realiza estas acciones internamente sin ninguna validación manual.

Tratamiento de las relaciones en los gestores

Entre los gestores soportados por la aplicación, existen algunos que manipulan información relacional. Para poder migrar esta información, la estructura de datos intermedia cuenta con el objeto Reference, el cual declara relaciones entre los MainContainer de dicha estructura. Cada gestor migra esta información en dependencia de sus particularidades. En el caso de PostgreSQL esta es transformada en constraints (restricción) de las tablas y en el caso de MongoDB se crean objetos DBRef, los cuales son documentos especiales que permite relacionar documentos entre colecciones.

1.4 Metodología de desarrollo de software

Debido al desarrollo que ha alcanzado la producción de software a nivel mundial se ha creado una gran cantidad de documentación sobre las políticas, procesos y procedimientos para lograr la calidad del producto final. Las metodologías de desarrollo de software se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental, etc.). Adicionalmente una metodología debería definir con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto, guías para uso de herramientas de apoyo, etc. (14)

Según lo antes mencionado, una metodología puede definirse como un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software, ya que indica cómo obtener los distintos productos parciales y finales. Considerando su filosofía de desarrollo, estas se clasifican en metodologías tradicionales (no ágiles) y metodologías ágiles. Las metodologías tradicionales hacen mayor énfasis en la planificación y control del proyecto y en la especificación precisa de requisitos y modelado. Las metodologías ágiles están más orientadas a la generación de código con ciclos muy cortos de desarrollo, haciendo especial hincapié en aspectos humanos asociados al trabajo en equipo e involucrando activamente al cliente en el proceso. (15)

Entre los principales objetivos de las metodologías de desarrollo de software se encuentran cumplir con los requisitos iniciales del problema y minimizar las pérdidas de tiempo en el proceso de desarrollo, así como minimizar riesgos e incrementar las posibilidades de éxito en el desarrollo de los productos.

1.4.1 Metodologías ágiles

Las metodologías ágiles son efectivas en proyectos con requisitos muy cambiantes, donde se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad de los mismos. Es una metodología sencilla, tanto en su aprendizaje como en su aplicación, lo que posibilita la reducción de los costos de implantación en un equipo de desarrollo. Están especialmente orientadas para proyectos pequeños y entre sus características más destacables está que dan un mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Entre las más usadas están SCRUM y Extreme Programming (XP). (16)

1.4.1.1 XP (Extreme Programming)

La programación extrema es una metodología de desarrollo ligero (o ágil) basada en una serie de valores y de buenas prácticas de manera que persigue el objetivo de aumentar la productividad a la hora de desarrollar programas. Este modelo de programación se basa en una serie de metodologías de desarrollo de software en la que se da prioridad a los trabajos que dan un resultado directo y que reducen la burocracia que hay alrededor de la programación. Una de las características principales de este método de programación, es que sus ingredientes son conocidos desde el principio de la informática. Los autores de XP han seleccionado aquellos que han considerado mejores y han profundizado en sus relaciones y en cómo se refuerzan los unos con los otros. El resultado de dicha selección ha sido esta metodología única y compacta. Por esto, aunque no está basada en principios nuevos, es una nueva manera de ver el desarrollo de software. (17)

A continuación se presentan algunas características de XP que se adaptan a las necesidades de un proyecto, así como a las condiciones de trabajo:

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo Pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.

- Programación por parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera es revisado y discutido mientras se escribe.
- Frecuente interacción del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir una nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- Propiedad del código compartida, en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantiza que los posibles errores sean detectados.
- Simplicidad en el código: La programación extrema apuesta que es más sencillo hacer un código simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar funciones complicadas y quizás nunca utilizarlas. (18)

Se decidió utilizar XP como metodología de desarrollo de software, debido a que cuenta con características y ventajas que se adaptan a las necesidades actuales del desarrollo de la herramienta.

A continuación se mencionan algunas de ellas:

- Se consiguen productos usables con mayor rapidez.
- El proceso de integración es continuo, por lo que el esfuerzo final para la integración es nulo.
- Se atienden las necesidades del usuario con mayor exactitud. Esto se consigue gracias a las continuas versiones que se le ofrecen a este.
- Que el software funcione es más importante que la documentación exhaustiva. La regla a seguir es: no producir documentos a menos que sean necesarios de forma inmediata. Estos documentos deben ser cortos y centrarse en lo fundamental.
- Se consiguen productos más fiables y robustos contra los fallos gracias al diseño de pruebas de forma previa a la codificación.

1.5 Tecnologías y herramientas asociadas al desarrollo de la solución

El desempeño de un proyecto está antecedido siempre por la investigación de las tecnologías que se utilizan, así como las ventajas y desventajas que trae su aplicación. Esta investigación fue realizada, analizando las tecnologías existentes que se ajustan a la solución a desarrollar. La herramienta debe estar libre de costos adicionales relativos a pago de licencias de software. A partir de la investigación realizada se identificaron las siguientes tecnologías a utilizar en el desarrollo del sistema**.**

1.5.1 Lenguaje Unificado de Modelación (UML)

UML (del inglés Unified Modeling Language) o Lenguaje Unificado de Modelado es un estándar ampliamente utilizado en la industria del software para el modelado de este. Ayuda a los profesionales a visualizar, comunicar y aplicar sus diseños para proporcionar un entorno de modelado visual. Como UML es un lenguaje de modelado, cuenta con pautas para mezclar los elementos gráficos a través de un grupo de herramientas CASE donde se obtienen como resultado final los diferentes diagramas. (19) Es de suma importancia acentuar que este lenguaje no es una guía para el análisis y el diseño, sino que permite modelar sistemas orientados a objetos. Por lo que se hace imprescindible hacer referencia a la herramienta CASE que utiliza a UML para el modelado.

1.5.2 Herramienta CASE

Las herramientas CASE (del inglés Computer Aided Software Engineering, Ingeniería de software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad y ayudar en todos los aspectos del ciclo de vida del desarrollo del software, reduciendo el costo de las mismas en términos de tiempo y dinero. Su objetivo fundamental es automatizar los aspectos claves de todo el proceso de desarrollo de un sistema. (20)

1.5.2.1 Visual Paradigm

Visual Paradigm es una herramienta CASE que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, implementación y pruebas. Ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite construir diagramas de diversos tipos, código inverso, generar código desde diagramas y generar documentación. La herramienta

también proporciona abundantes tutoriales, demostraciones interactivas y proyectos UML. Algunas de las características de la herramienta que son utilizadas para el desarrollo de la extensión propuesta son:

- Integración con entornos de desarrollo: Apoyo al ciclo de vida completo de desarrollo de software en IDE como: Eclipse, NetBeans, Sun ONE, Oracle JDeveloper, JBuilder y otros.
- Detalles de casos de uso: Entorno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Multiplataforma: Soportado en la plataforma Java para varios sistemas operativos (Windows/Linux/ac OS X). Su versión 8.0 incluye la funcionalidad de crear y especificar perfiles UML, la cual resulta de vital importancia para la implementación y ejecución de extensiones para la herramienta. (21)

Debido a las características mencionadas y los beneficios que brinda para el desarrollo de software, especialmente referentes al modelado, se decidió utilizar Visual Paradigm for UML 8.0 para el modelado. Además de ello, se tuvo en cuenta que esta constituye la herramienta que se utiliza en la UCI y dentro de ella el centro DATEC para el desarrollo de software.

1.5.3 Lenguaje de Programación

Java es un lenguaje sencillo de aprender. Su sintaxis es la de C++ “simplificada”. Los creadores de Java partieron de la sintaxis de C++ y trataron de eliminar de este todo lo que resultase complicado o fuente de errores en este lenguaje, es un lenguaje orientado a objetos, aunque no de los denominados puros; en Java todos los tipos, a excepción de los tipos fundamentales de variables (int, char, long...) son clases. Sin embargo, en los lenguajes orientados a objetos puros incluso estos tipos fundamentales son clases, por ejemplo en Smalltalk. El código generado por el compilador Java es independiente de la arquitectura: podría ejecutarse en un entorno UNIX, Mac o Windows. (22)

Características:

- Lenguaje totalmente orientado a Objetos: Todos los conceptos en los que se apoya esta técnica, encapsulación, herencia, polimorfismo, etc., están presentes en Java.
- Disponibilidad de un amplio conjunto de bibliotecas: Como ya se mencionó anteriormente, Java es algo más que un lenguaje. La programación de aplicaciones con Java se basa no solo en el empleo del juego de instrucciones que componen el lenguaje, sino, fundamentalmente, en la

posibilidad de utilizar el amplísimo conjunto de clases que Sun pone a disposición del programador y con las cuales es posible realizar prácticamente cualquier tipo de aplicación.

- Lenguaje simple: Java posee una curva de aprendizaje muy rápida. Resulta relativamente sencillo escribir applets interesantes desde el principio. Todos aquellos familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros. Debido a su semejanza con C y C++, y dado que la mayoría de la gente los conoce aunque sea de forma elemental, resulta muy fácil aprender Java. Los programadores experimentados en C++ pueden migrar muy rápidamente a Java en poco tiempo.
- Distribuido: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets, establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.
- Interpretado y compilado a la vez: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time).
- Robusto: Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo de los punteros.
- Seguro: Dada la naturaleza distribuida de Java, donde las applets se bajan desde cualquier punto de la red, la seguridad se impuso como una necesidad de vital importancia. Así que se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.
- Indiferente a la arquitectura: Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows Nt, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos disímiles. Para acomodar requisitos de ejecución tan diversos, el compilador de Java genera bytecodes: un formato intermedio indiferente a la arquitectura diseñada para transportar el código eficientemente a múltiples plataformas de hardware y software. El resto de los problemas son solucionados por el intérprete de Java.

- Portable: La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas. Estas dos últimas características se conocen como la Máquina Virtual Java (JVM).
- Alto rendimiento.
- Asíncrono: Hoy en día ya se ven como terriblemente limitadas las aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (multithilado) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.
- Dinámico: El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden vincular nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la red. (23)

El lenguaje de programación seleccionado para el desarrollo de la herramienta de migración es Java debido a su robustez, ya que los errores se detectan en el momento en que se producen, lo que facilita la depuración. Es unos de los lenguajes más usados y cuenta con una extensa bibliografía.

1.5.4 Entorno de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) también conocido como entorno de diseño integrado, entorno integrado de depuración o entorno de desarrollo interactivo, está compuesto por un conjunto de herramientas útiles para un desarrollador. Es una aplicación de software que proporciona servicios integrales a los programadores de computadoras para el desarrollo de software. Puede ser exclusivo para un lenguaje de programación o utilizarse para varios. Generalmente trae incluido un editor de código, un compilador, un depurador y en algunos casos un constructor de interfaz gráfica. (24)

1.5.4.1 NetBeans

NetBeans es un IDE (Entorno Integrado de Desarrollo) de Java. Esta versión incluye herramientas para crear aplicaciones Java web y empresariales, así como los servidores GlassFish OSE 4.0 y Apache Tomcat 7.0.41. NetBeans IDE le permite a Java desarrollar aplicaciones de escritorio, móviles y web, así como aplicaciones HTML5 con HTML, JavaScript y CSS, de forma rápida y fácil. El IDE también

proporciona un gran conjunto de herramientas para desarrolladores de PHP y C / C + +. Es gratuito y de código abierto y tiene una gran comunidad de usuarios y desarrolladores de todo el mundo. (25)

Características:

- Mejor soporte para las modernas tecnologías Java: NetBeans IDE proporciona un amplio soporte de primera clase para las últimas tecnologías Java y las actuales mejoras de la especificación de Java antes de otros IDE. Es el primer IDE que proporciona soporte gratuito para JDK 8 avances, JDK 7, Java EE 7, incluyendo sus accesorios relacionados con HTML5 y JavaFX 2. NetBeans IDE establece el estándar para el desarrollo de tecnologías de vanguardia.
- Fast & Smart Edición de código: Un IDE es mucho más que un editor de texto. También proporciona plantillas de código, consejos de codificación y herramientas de refactorización. El editor soporta varios lenguajes de Java, C / C + +, XML y HTML, de PHP, Groovy, Javadoc, JavaScript y JSP. Debido a que el editor es extensible, se puede conectar en el apoyo a muchos otros idiomas.
- Manejo fácil y eficiente de los proyectos: Mantener una visión clara de las aplicaciones grandes, con miles de carpetas y archivos, y millones de líneas de código, es una tarea de enormes proporciones. NetBeans provee diferentes vistas de los datos, de múltiples ventanas del proyecto a herramientas útiles para la creación de sus aplicaciones y la gestión de manera eficiente, lo que le permite profundizar en sus datos de forma rápida y sencilla.
- Desarrollo rápido de la interfaz de usuario: Diseño de interfaces gráficas de usuario para Java SE, HTML5, Java EE, PHP, C / C + + y Java ME aplicaciones de forma rápida y sin problemas mediante el uso de editores y herramientas de arrastrar y soltar en el IDE. Para aplicaciones Java SE, el NetBeans GUI Builder se encarga automáticamente del espaciamiento y alineación correcta. El constructor de interfaz gráfica de usuario es tan fácil de usar e intuitivo que se ha utilizado para crear prototipos de interfaces gráficas de usuario en vivo en presentaciones a clientes.
- Escribe código libre de errores: NetBeans proporciona herramientas de análisis estático, especialmente la integración con la herramienta FindBugs, ampliamente utilizado para identificar y solucionar problemas comunes en código Java. Además, el depurador del NetBeans le permite colocar puntos de interrupción en el código fuente. (25)

El IDE seleccionado para el desarrollo de este trabajo es el NetBeans7.4 por las ventajas que brinda su uso, además de que asegura robustez y rendimiento.

Conclusiones del capítulo

En el presente capítulo se realizó un análisis de los principales aspectos y conceptos relacionados con: Base de Datos, principales gestores SQL Y NoSQL, PostgreSQL, MongoDB, CouchDB, Cassandra, migración de datos, entre otros. A partir de la investigación realizada acerca de las tecnologías y herramientas, se hará uso de la metodología XP para guiar el proceso de desarrollo de la herramienta de migración de datos y como herramienta para el modelado Visual Paradigm 8.0, empleando el lenguaje de modelado UML. Durante la implementación se utilizará el IDE de desarrollo Net Beans 7.4, utilizando como lenguaje de programación Java.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

En el presente capítulo se analizan cada uno de los artefactos y elementos para el diseño de la aplicación, lo que permitirá un mejor entendimiento de las necesidades de la herramienta a desarrollar. Además se expondrán las principales características de la herramienta, su diseño, su arquitectura y los requisitos a tener en cuenta en su desarrollo.

2.1 Modelo del dominio

La metodología XP se inspira en la simplicidad, basada en desarrollar sólo el sistema que realmente se necesita, sin embargo su gran adaptabilidad permite el empleo de diagramas UML, siempre y cuando influyan en el mejoramiento de la comunicación. Debido a ello se decide realizar un modelo de dominio ,debido a que este le permite a los usuarios, clientes, desarrolladores e interesados, utilizar un vocabulario común para lograr un efectivo entendimiento del contexto en que se encuentra el sistema. De esta manera el modelo de dominio proporciona una perspectiva conceptual de los objetos implícitos en dicho contexto, realizando su descripción a través de un diagrama de clases UML.

Según Ivar Jacobson, Grady Booch, y James Rumbaugh, “un modelo de dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las cosas que existen o los eventos que suceden en el entorno en el que trabaja el sistema”. (26) Es una representación visual de clases conceptuales o de objetos reales en un dominio de interés.

Definición de las clases del modelo del dominio:
Usuario: Persona que interactúa con la herramienta.
Herramienta de Migración: Aplicación encargada de realizar todo el proceso de migración de los datos de un gestor a otro.
Gestor de BD Origen: Gestor donde se encuentran las bases de datos a migrar.
Gestor de BD Destino: Gestor donde se almacenarán las bases de datos migradas.
Base de Datos: Conjunto de datos.

Diagrama Modelo del dominio

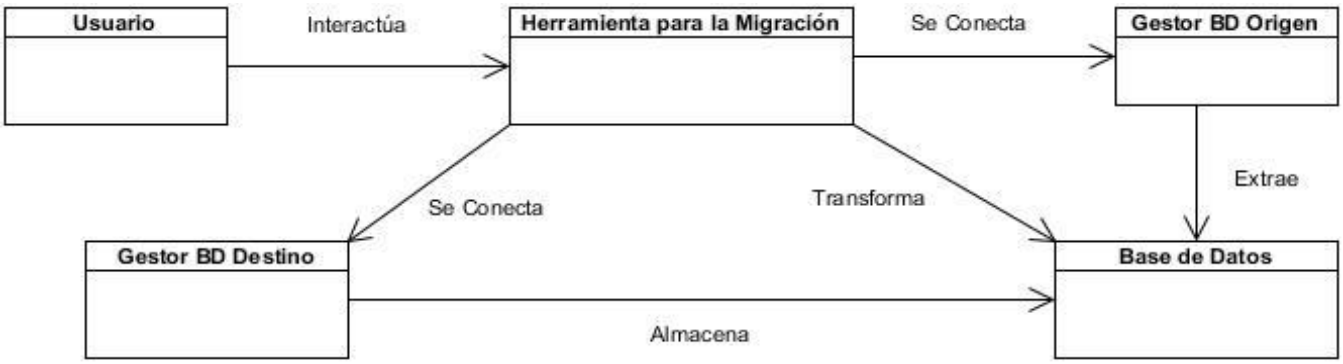


Figura 2. Modelo de dominio.

2.2 Descripción del sistema propuesto

Para solucionar el problema de investigación planteado se decide desarrollar una herramienta de migración de datos entre gestores de bases de datos SQL y NoSQL con las siguientes características:

- Permite que la pérdida de información debido al gran flujo de datos a migrar sea mínima.
- Su diseño pretende colaborar con la persistencia de los datos y evitar la interferencia en la confidencialidad de la información almacenada en las bases de datos.
- Flexibilidad ante la necesidad de incorporar nuevos gestores de bases de datos para llevar a cabo el proceso de migración.

La implementación de la herramienta estará condicionada por el siguiente procedimiento:

- **Selección de los Gestores de Bases de Datos**: el usuario selecciona el Gestor de BD origen y destino con los cuales desea realizar la migración de datos.
- **Conexión a las bases de datos**: el usuario introduce todos los datos requeridos para poder realizar la conexión a las bases de datos (Base de datos, Usuario, Contraseña, Host y Puerto).
- **Mapeo de los datos entre los gestores**: una vez seleccionado los gestores de bases de datos y haber realizado la conexión correcta, se procede a transformar los datos del gestor origen al destino.
- **Reportes**: durante el proceso, se obtienen detalles de la migración a través de reportes.

Para llevar a cabo la implementación de la herramienta es necesario realizar la especificación de requisitos, los detalles del tiempo que requiere la implementación y la estimación del riesgo. Para ello en

la fase de Exploración de la metodología XP se utiliza un período de tiempo en el que se realizan las Historias de Usuarios (HU), uno de los artefactos elementales que genera la metodología XP.

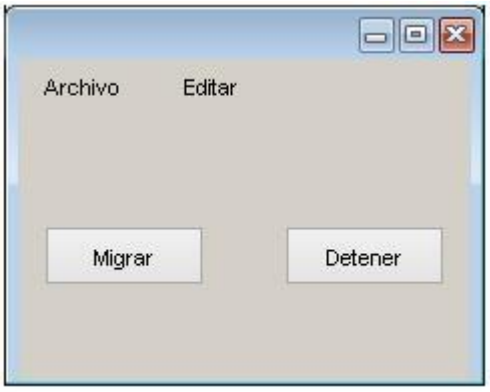
2.3 Historias de usuarios

Las historias de usuarios son las técnicas utilizadas en la metodología XP para especificar y administrar los requisitos del software de una forma eficaz. Son descripciones cortas y escritas en el lenguaje del usuario sin terminología técnica, debido a ello son lo suficientemente comprensibles y delimitadas para que los programadores puedan implementarlas en unas semanas. Permiten responder rápidamente a los requisitos cambiantes pues su tratamiento es dinámico y flexible.

Para el presente trabajo de diploma se obtienen un total de ocho HU que son implementadas en cuatro iteraciones. A continuación en la Tabla 3 se muestra una de las HU desarrollada: “Migrar Datos de Origen a Destino”, el resto se encuentran en el Expediente de Proyecto.

Tabla 3.Historia de Usuario ´´Migrar datos de origen a destino´´.

Historia de Usuario	
Número: 6	**Nombre de la Historia de Usuario:** Migrar datos de origen a destino
Cantidad de modificaciones a la Historia de Usuario: Ninguna	
Usuario: Yoel Claro Ledón	**Iteración asignada:** 1
Prioridad en negocio: Muy Alta	**Puntos estimados: 1 semana**
Riesgo en desarrollo: Muy Alta	**Puntos reales: 0.9 semana**
Descripción: Realiza el proceso de Extracción, Transformación y Carga (ETC) desde la BD origen hacia la de destino.	
Observaciones:	
Prototipo de interfaces	



Las HU describen las funcionalidades que debe realizar la herramienta de migración de datos. Proveen información acerca de la prioridad de la funcionalidad a implementar así como el tiempo estimado de duración de dicha implementación. A continuación las HU son relacionadas en el artefacto Lista de Reserva del Producto en la cual se referencia información relevante acerca de las mismas.

2.4 Lista de reserva del producto

La lista de reserva del producto es una tabla que contiene los requisitos funcionales que debe cumplir la herramienta que se desea realizar, ordenados según la prioridad en el negocio (Muy alta, Alta, Media y Baja). Se indica de cada uno de ellos la estimación, su implementación por semanas y el rol que lo estimó. Contiene por último los requisitos no funcionales que requiere el sistema a desarrollar.

Tabla 4.Lista de reserva del producto.

Ítem *	**Descripción**	**Estimación**	**Estimado por**
Prioridad: Muy Alta			
1	Migrar datos de origen a destino	1 semana	Analista
Prioridad: Alta			
2	Crear conexión origen	0.8 semana	Analista
3	Comprobar conexión origen	0.8 semana	Analista
4	Crear conexión destino	0.8 semana	Analista
5	Comprobar conexión destino	0.8 semana	Analista

Prioridad: Media			
6	Mapear modelos	0.7 semana	Analista
Prioridad: Baja			
7 8	Generar log de errores Generar log de operaciones	0.4 semana 0.4 semana	Analista Analista
Requisitos No Funcionales			
1	**Usabilidad:** El sistema debe ser capaz de darle al usuario la menor cantidad de responsabilidades y de facilitar el trabajo de este con la aplicación, utilizando tooltips (información sobre la herramienta) y un manual de usuario		
2	**Portabilidad:** La aplicación será un sistema multiplataforma, permitiendo la disponibilidad en cualquier sistema operativo.		
3	**Seguridad:** El sistema garantiza la seguridad para el acceso a la aplicación, autenticando al usuario mediante una contraseña en las BD origen y destino.		
4	**Software:** Para poder hacer uso de esta aplicación debe encontrarse instalada la máquina virtual de Java en su séptima versión.		
5	**Hardware:**		

	Los clientes deberán tener como mínimo 2 GB de RAM , un microprocesador Dual Core de 1.5 GHz.		
6	**Apariencia o interfaz externa:** Debe ser compatible con resoluciones de 800x600 píxeles en adelante.		

2.5 Tareas de ingeniería

Después de haberse realizado la definición de las Historias de Usuarios, el equipo de desarrollo divide por cada una de ellas una serie de tareas de ingenierías que contribuyen al desarrollo de las HU. A continuación se presenta un ejemplo de las tareas de ingeniería desarrolladas durante la realización de la herramienta y el resto se encuentran en el Expediente de Proyecto.

Tabla 5. Tarea de ingeniera Crear conexión origen.

Tarea de Ingeniería	
Número Tarea: 1	**Número Historia de Usuario:** 1
Nombre Tarea: Crear conexión origen.	
Tipo de Tarea : Desarrollo	**Puntos Estimados:** 0.8 semana
Fecha Inicio: 2-2-2014	**Fecha Fin:** 20-2-2014
Programador Responsable: Daniela Beltrán Sordía	
Descripción: Crea la conexión origen.	

2.6 Plan de iteraciones

Las HU después de ser descritas, identificadas y estimado el esfuerzo propuesto para la realización de cada una de ellas, se especifica cuáles serán implementadas en cada iteración del sistema por lo que se procede a la realización de un plan de iteraciones.

La metodología XP aporta mayor valor a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Al comienzo de cada ciclo, se realiza una reunión de planificación de

la iteración. El plan de iteraciones es empleado para la planificación, donde el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas.

Tabla 6.Plan de iteraciones.

No	Descripción de la iteración	Historias a implementar	Duración total
1	Se desarrollan las funcionalidades que permiten la migración de datos de origen a destino.	6	1 semana
2	Se desarrollan las historias de usuario relacionadas con la creación de las conexiones origen y destino y la comprobación de estas.	1,2,3,4	3.2 semana
3	Se desarrollan las funcionalidades que permiten el mapeo de modelos.	5	0.7 semana
4	Se desarrollan las funcionalidades referentes a la generación de log de errores y de operación	7, 8	0.8 semana

2.7 Modelo del diseño

En la fase de diseño, XP propone mejorar la comunicación entre todos los integrantes del equipo, al crear una visión global y común de lo que se quiere desarrollar para lograr un diseño sencillo, pues Kent Beck refiere que“en cualquier momento el diseño adecuado para el software es aquel que: supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención de implementación de los programadores y tiene el menor número posible de clases y métodos” (27). Además Pressman asegura: *“el diseño es el lugar donde se fomentará la calidad del software”* (28), ya que un mal trabajo en la etapa de diseño traería consigo problemas en el resto de las etapas de desarrollo del software, lo que podría implicar baja calidad del software, demora en la entrega de este y un mal funcionamiento de la aplicación.

Es necesario describir qué clases están presentes y cómo se relacionan, para lograr esto la metodología XP utiliza técnicas como Tarjeta Clase, Responsabilidad y Colaboración (CRC). Esta metodología también permite usar diagramas UML siempre y cuando influyan en el mejoramiento de la comunicación y se enfoquen en la información elemental. Por dicha razón se realiza un diagrama de las clases a implementar para mostrar sus relaciones y dependencias utilizando notación UML.

2.7.1 Diagrama de clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema representando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. (29)

A continuación en la Figura 4 se muestra el diagrama de clases del diseño para la herramienta de migración de datos a nivel de paquetes debido a la extensibilidad del diagrama original, para verlo en su totalidad diríjase al Expediente de Proyecto.

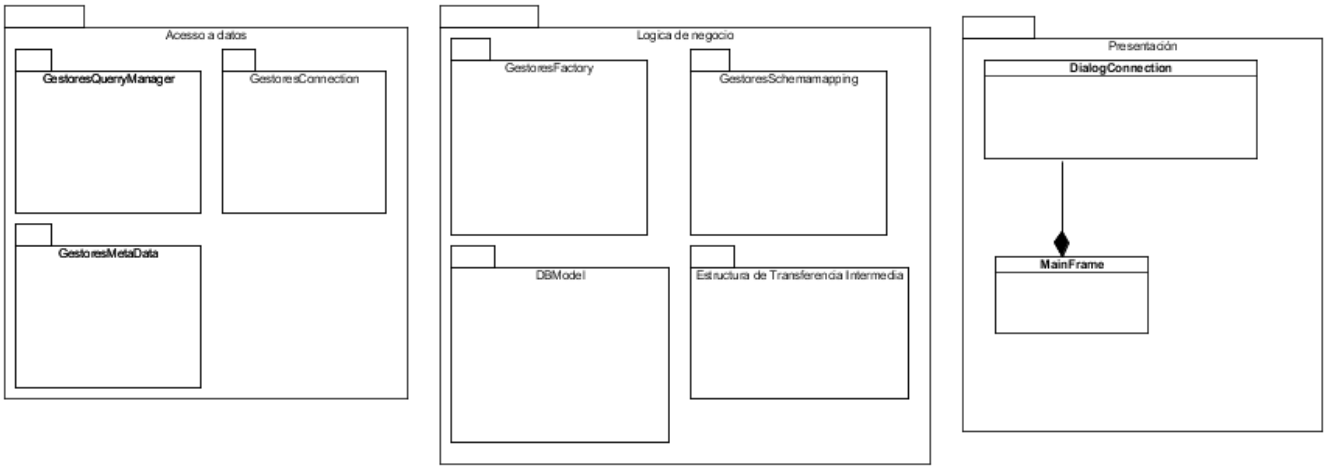


Figura 3. Diagrama de Clases del Diseño.

2.7.2 Tarjetas Clase, Responsabilidad y Colaboración

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

A continuación se muestra en la tabla 4 la tarjeta CRC SGFactory generada para el diseño de la herramienta de migración de datos, las demás se encuentran en el expediente de proyecto.

Tabla 7.Tarjeta CRC SGFactory.

Tarjeta CRC	
Clase: SGFactory	
Responsabilidades	**Colaboraciones**
Crear los objetos necesarios para manipular un gestor de BD.	DBModel

2.8 Patrones de arquitectura

Los patrones de arquitectura de software son patrones de diseño que constituyen una vía en la solución de problemas de arquitectura de software. Los mismos poseen un nivel de abstracción mucho mayor que los patrones de diseño. Además brindan una descripción de los elementos y el tipo de relación que tienen, así como las restricciones para su uso. Los patrones se especifican describiendo los componentes, con sus responsabilidades, relaciones, y las formas en que colaboran. (30)

2.8.1 Arquitectura en capas

Se puede decir que:

- Resulta útil cuando se pueden identificar distintas clases de servicios que logran ser articulados jerárquicamente.
- Los componentes de cada capa consisten en conjuntos de procedimientos.
- Las interacciones entre capas usualmente proceden por invocación de procedimientos.

- Por definición, los niveles de abajo no pueden usar una funcionalidad ofrecida por los de arriba.

La herramienta para la migración de datos creada en esta investigación se apoya para su diseño en el estilo arquitectónico en Capas, estructurando el trabajo en tres capas, una de ellas representa la capa Presentación, la segunda capa la lógica del negocio y la tercera el acceso a datos.

A continuación una breve descripción de cada capa:
La capa de presentación: Esta capa se encarga de proveer una interfaz entre el sistema y el usuario. Básicamente, se responsabiliza de que se le comunique información al usuario por parte del sistema y viceversa, manteniendo una comunicación exclusiva con la capa de negocio. Además dentro de esta entraría aquello que el usuario "ve" cuando se conecta a la aplicación**.**
La capa de negocio: Es la capa que contiene los procesos a realizar con la información recibida desde la capa de presentación, dichos procesos son responsables de que se le envíen las respuestas adecuadas a la capa de presentación. Se podría ver como una capa intermedia, entre la capa de presentación y la capa de datos, puesto que se relaciona con ambas y por supuesto, procesa también la información devuelta por la capa de datos.
La capa de acceso a datos: Se encarga mediante la capa de negocio, de ofrecer, modificar, almacenar, borrar y recuperar datos, a través del gestor (o los gestores) de base de datos que la aplicación requiera.

2.9 Patrones de diseño

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software, en otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. (31)

Un patrón de diseño provee un esquema para refinar los componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (32)

2.9.1 Patrones GRASP

Una muy buena práctica a la hora de desarrollar aplicaciones orientadas a objetos es el uso de los patrones GRASP. Dichos patrones describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. GRASP es un acrónimo del inglés

General Responsibility Assignment Software Patterns (Patrones de Software para la Asignación General de Responsabilidad). El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos. (33)

Patrón Experto: Surge como principio fundamental que hay que tener en cuenta siempre cuando se esté asignando una responsabilidad a una clase. La respuesta es asignar la responsabilidad a la clase que contenga la información necesaria para cumplir el encargo, o sea la clase debe ser la experta en la información.

La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). El patrón experto se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen.

El cumplimiento de una responsabilidad requiere a menudo de información distribuida en varias clases de objetos. Lo que significa que hay muchos expertos "parciales" que colaboran en la tarea.

Los principales beneficios que se obtienen al usar este patrón son:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión. A continuación un ejemplo de la clase de SchemaMapping la cual contiene todos los métodos para el mapeo y trasformación de los datos por lo que es la clase especializada en realizar todos los procesos de esta naturaleza.

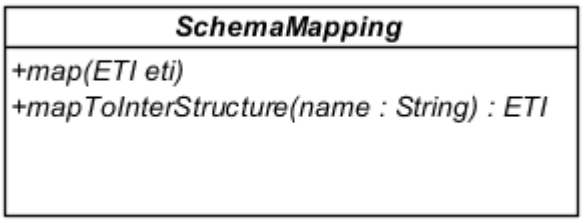


Figura 4. Ejemplo de uso del patrón experto.

Patrón Creador: La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella. Se asigna la responsabilidad de que una clase B cree un objeto de la clase A.

La clase B crea las instancias de A si:

- B agrega los objetos de A
- B contiene los objetos de A
- B registra las instancias de los objetos de A.
- B tiene los datos de inicialización que serán enviados a la clase A cuando este objeto sea creado.

Este patrón es muy simple y su mayor beneficio es que contribuye a soportar el bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. A continuación se presenta un ejemplo (Figura 6) evidenciando el patrón creador donde la clase DBModel crea instancias de los objetos DbInfo y SchemaMapping para implementar sus funcionalidades.

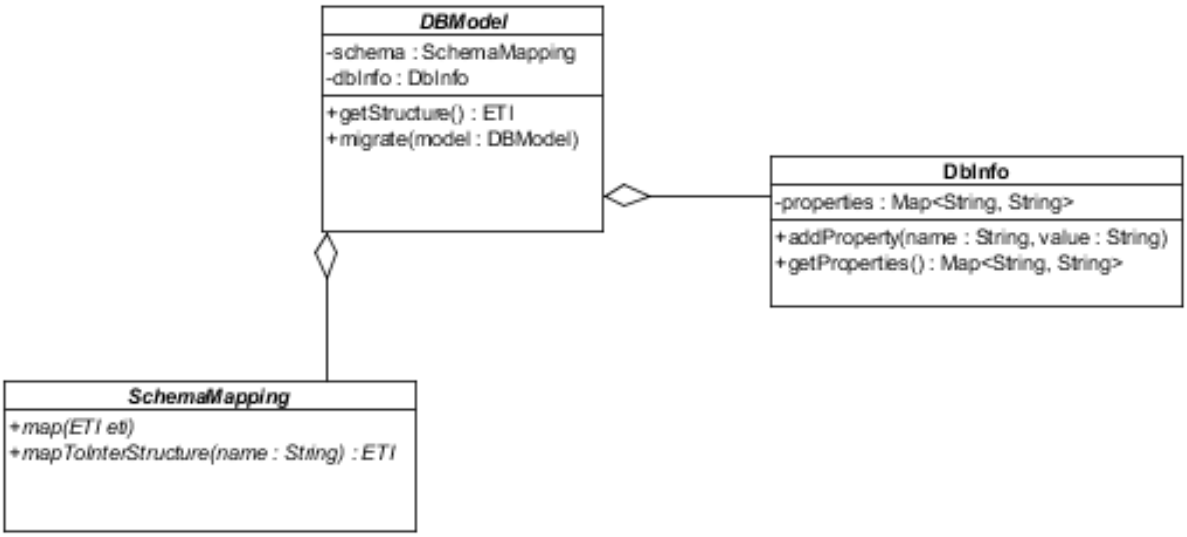


Figura 5. Ejemplo de uso del patrón creador.

Alta Cohesión: Cada elemento del diseño debe realizar una labor única dentro del sistema, lo cual expresa que la información que almacena una clase debe de ser coherente y está en la mayor medida de lo posible relacionada con la clase (33). Mantener una alta cohesión en las clases de la herramienta presentada en este trabajo, permite que el mantenimiento de la misma pueda ser más eficiente y facilita la reutilización de sus componentes en otras soluciones. Este patrón se evidencia en todas las clases de la aplicación.

Bajo Acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas.

- Acoplamiento bajo significa que una clase no depende de muchas clases.
- Acoplamiento alto significa que una clase recurre a muchas otras clases.

Esto presenta los siguientes problemas:

- Los cambios de las clases afines ocasionan cambios locales.
- Difíciles de entender cuando están aisladas.
- Difíciles de reutilizar puesto que dependen de otras clases. (33)

Lograr un Bajo Acoplamiento en las clases de la herramienta, permite mejorar la claridad y facilidad con la que se entiende el diseño, simplificar el mantenimiento y las mejoras de funcionalidad y da soporte a una mayor capacidad de reutilización. Este patrón está presente en todas las clases de la aplicación.

2.9.2 Patrones GOF

El catálogo de patrones más famoso es el contenido en el libro Design Patterns: Elements of Reusable Object-Oriented Softwarell, también conocido como el LIBRO GOF (Gang-Of-Four Book). Según el libro GOF estos patrones se clasifican según su propósito en: creacionales, estructurales y de composición, mientras que respecto a su ámbito se clasifican en clases y objetos.

Clasificación respecto a su propósito:

- De Creación: Abstraen el proceso de creación de instancias. Resuelven problemas relativos a la creación de objetos.
- Estructurales: Se ocupan de cómo clases y objetos son utilizados para componer estructuras de Mayor tamaño. Resuelven problemas relativos a la composición de objetos.
- De Comportamiento: Atañen a los algoritmos y a la asignación de responsabilidades entre objetos. Resuelven problemas relativos a la interacción entre objetos.

Patrones Creacionales

- Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. Proporciona una interfaz para crear familias de objetos que dependan entre sí, sin especificar sus clases concretas.
- Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella. (34)

Patrón de Comportamiento

- Observer (Observador)**:** Un efecto muy frecuente en aquellos sistemas que se fragmentan en un conjunto de clases que cooperan, es la necesidad de mantener la consistencia entre los distintos objetos interrelacionados. Para no recurrir a soluciones fuertemente acopladas (que reducen la posibilidad de reutilización), este patrón define una dependencia “uno a muchos” entre objetos,

para que, cuando uno de ellos cambie su estado, todos los que dependan de él sean avisados y puedan actualizarse convenientemente. (34)

Para desarrollar la herramienta a través de un diseño flexible y acorde con la solución propuesta se decidió utilizar el patrón de diseño GOF **Abstract Factory** ya que este patrón provee una interfaz para la creación de una familia de objetos relacionados o dependientes. En caso de la herramienta de migración de datos se manipula la información de varios sistemas gestores de bases de datos. De los gestores se debe conocer principalmente como se conectan (la clase conexión) y como extraen los metadatos de la misma (nombre de las tablas, atributos, etc.), para así realizar las consultas y las trasformaciones para el proceso de migración.

En este caso las familias serían los tipos de sistemas gestores de bases de datos, y se debe garantizar que si se está trabajando con un gestor en específico todas las clases que se creen deben pertenecer a ese gestor. Abstract Factory provee una interfaz para construir los objetos, la construcción se lleva a cabo en cada clase concreta que hereda de la factoría, que serían las familias, ayudando esto a una mayor organización de código ya que a la hora de agregar o modificar elementos de dicha familia solo se trabajaría con la clase Factory, efectuando las operaciones internamente y no en todo el código, las clases involucradas en la familia serían, “Conection”,”QuerryManager”, “SchemaMaping” y “MetaData”. A continuación se puede apreciar un ejemplo del uso del patrón Abstract Factory (Figura 7).

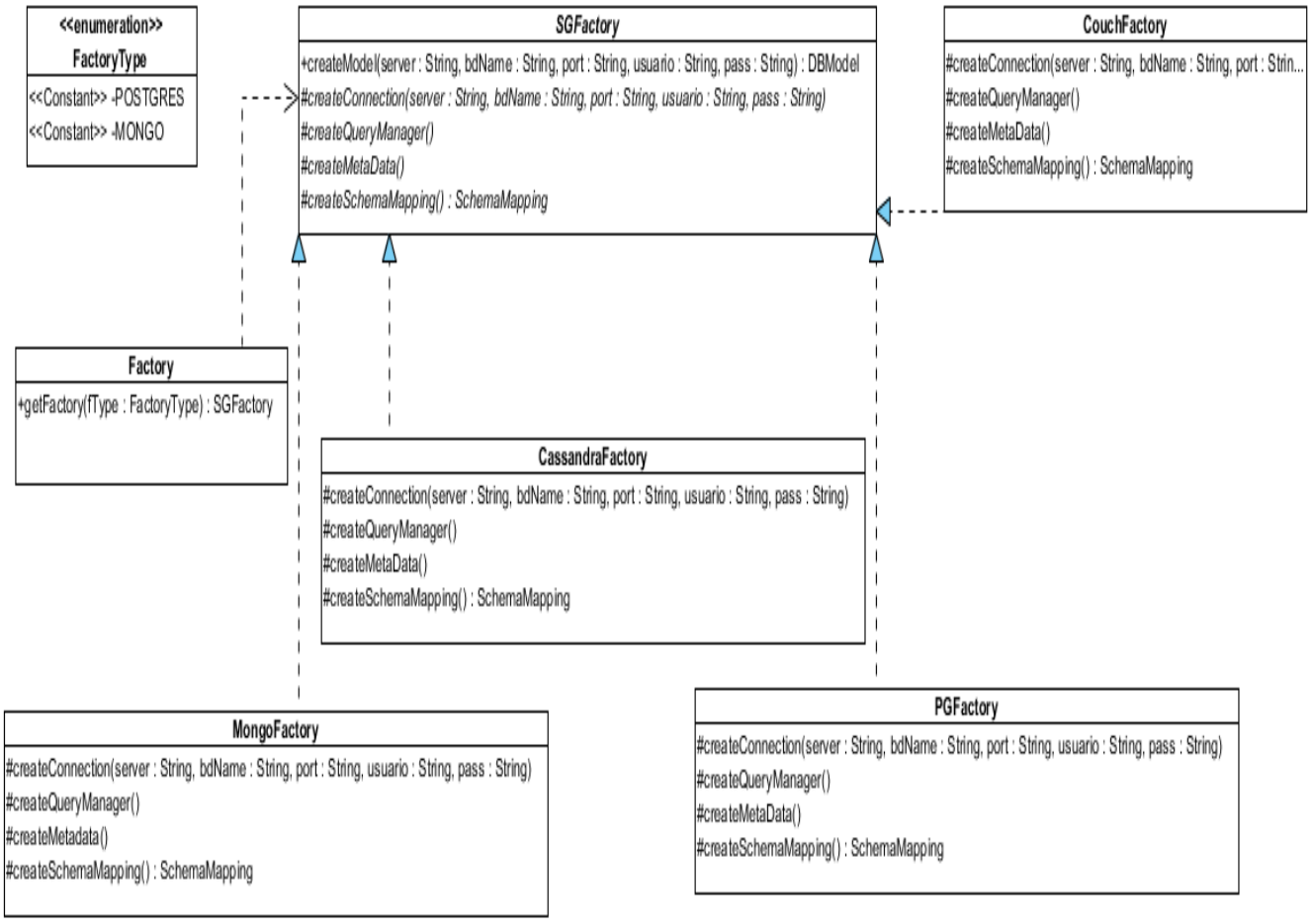


Figura 6. Ejemplo de uso del patrón Abstract Factory.

Se decidió utilizar el patrón Singleton y el Observer en conjunto para el manejo de los reportes del proceso de migración de datos a través de la clase MigLogger. Utilizando el patrón Singleton se asegura que existe una única copia en memoria del objeto responsable de gestionar los mensajes de la aplicación (MigLogger) de esta manera todos los objetos que deban reportar algún mensaje accederían a esta única instancia, combinando este patrón con el patrón Observer, se asegura que los componentes responsables de visualizar los mensajes estén actualizados con la ocurrencia de los mismos (Figura 8).

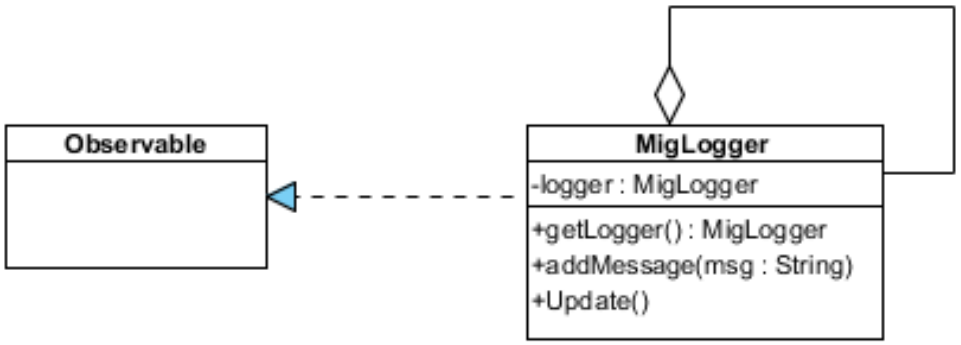


Figura 7. Ejemplo de uso del patrón Singleton y Observer.

Conclusiones del capítulo

Se identificaron ocho HU las cuales responden a los requisitos funcionales de la aplicación, se definió la iteración en la que será implementada cada HU. Se especificaron los requisitos no funcionales a tener en cuenta a la hora de utilizar la aplicación. Para el diseño de la aplicación se identificaron 31 tarjetas CRC, donde se utilizó para la modelación del sistema el estilo arquitectónico MVC y los patrones de diseño que ofrece GRASP Y GOF, facilitando el trabajo de diseño de la aplicación para la migración de datos.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

Las pruebas de software, permiten la verificación de la calidad de un producto. Son utilizadas para identificar posibles fallos de una aplicación, básicamente es una fase en el desarrollo de software que consiste en probar la aplicación construida. En el presente capítulo se describen los principales artefactos generados en la fase de implementación y prueba de la herramienta de migración de datos entre los gestores de bases de datos SQL y NoSQL. Se desarrolla la descripción de la implementación del sistema y se especifican las pruebas a las que fue sometida la herramienta de Migración de Datos en cada una de las iteraciones. Proceso que guía la identificación y corrección de fallos cometidos en las HU, así como su verificación y materialización. Contribuye a elevar la calidad del producto desarrollado y la seguridad en los programadores para efectuar modificaciones.

3.1 Estándares de codificación

Las convenciones o estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. XP enfatiza la comunicación de los programadores a través del código y plantea que estos pueden realizar cambios en cualquier parte del código en cualquier momento, por lo cual es indispensable que se sigan ciertos estándares de programación manteniendo el código legible para los miembros del equipo, de forma tal que se faciliten los cambios.

El estándar asegura que todos los programadores del proyecto trabajen de forma coordinada y comprensible para obtener un código fuente legible, pues repercute directamente en lo bien que se comprende un sistema de software, aspecto que es indispensable para lograr estabilidad del código, ya que con la misma el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. (35)

Al programar en un leguaje en específico se deben seguir reglas que permitirán que cualquier persona que se desempeñe como codificador de dicho lenguaje, pueda interpretar de manera eficiente la escritura del código, a continuación se describe el estándar de codificación que utiliza Java. (36)

3.1.1 Comentarios

Los comentarios se utilizan para describir el código ("el cómo"), y en ellos se incluye información relacionada con la implementación, tales como la descripción de la función de variables locales, fases lógicas de ejecución de un método, captura de excepciones con el objetivo del que el propio programador u otras personas cuando lean el código puedan entenderlo, por lo que se deben escribir de manera clara, resumida y objetiva. (36)

A continuación un ejemplo del código de fuente donde se evidencian los comentarios de dos funcionalidades.

```
/**
 * Define la conexión cuando el gestor se va a conectar como fuente de la migración
 */
protected abstract boolean connectSource();
/**
 * Define la conexión cuando el gestor se va a conectar como destino de la migración
 */
protected abstract boolean connectTarget();
```

Figura 8. Ejemplo de comentarios.

3.1.2 Declaración de variables

Declaración por línea: Se recomienda el uso de una declaración por línea, promoviendo así el uso de comentarios. Ejemplo:

int idUnidad; // Identificador de la unidad organizativa

String[] funciones; // Funciones de la unidad

Inicialización: Toda variable local tendrá que ser inicializada en el momento de su declaración, salvo que su valor inicial dependa de algún valor que tenga que ser calculado previamente. Ejemplo:

int idUnidad = 1;

String[] funciones = { "Administración", "Intervención"};

Localización: Las declaraciones deben situarse al principio de cada bloque principal en el que se utilicen, y nunca en el momento de su uso. Ejemplo:

public void Método(){

 int contador=0; //inicio del método

 ...

} (36)

A continuación un ejemplo del código fuente donde se evidencia la declaración de variables.

```
public class PgSchemaMapping implements SchemaMapping
{  //se crean variables de tipo PGQuerryManager y PgMetadata
    private PostgresQueryManager pgQuery ;
    private PgMetaData pgMeta;
```

Figura 9. Ejemplo de declaración de variables.

3.1.3 Identificadores

Las variables se escribirán siempre en minúsculas. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúsculas. Las variables nunca podrán comenzar con el carácter "_" o "$". Los nombres de variables deben ser cortos y sus significados tienen que expresar con suficiente claridad la función que desempeñan en el código. Debe evitarse el uso de nombres de variables con un sólo carácter, excepto para variables temporales. (36) Ejemplos:

Unidad unidad;

Agencia agencia;

Tramite tramite;

3.1.4 Sentencias

Cada línea debe contener como máximo una sentencia. Ejemplo:

int contador++;

int variable--;

Las sentencias pertenecientes a un bloque de código estarán tabuladas un nivel más a la derecha con respecto a la sentencia que las contiene. El caracter inicio de bloque "{" debe situarse al final de la línea que inicia el bloque. El caracter final de bloque "}" debe situarse en una nueva línea tras la última línea del bloque y alineada con respecto al primer caracter de dicho bloque.

Todas las sentencias de un bloque deben encerrarse entre llaves "{...}", aunque el bloque conste de una única sentencia. Esta práctica permite añadir código sin cometer errores accidentalmente al olvidar añadir las llaves. Ejemplo:

```
if(condición){
 variable++;
}
```

La sentencia "try/catch" siempre debe tener el formato siguiente:

Try {sentencias;}

Catch (ClaseException e)

{Sentencias;}

En el bloque "catch" siempre se imprimirá una traza de error indicando el tipo de excepción generada y posteriormente se elevará dicha excepción al código invocante, salvo que la lógica de ejecución de la aplicación no lo requiera.

Siempre se utilizará el bloque "finally" para liberar recursos y para imprimir trazas de monitorización de fin de ejecución.

Try {sentencias;}

Catch (ClaseException e)

{Sentencias;}

Finally

{Sentencias;}

Sentencias Simples:

Cada línea debe contener como máximo una sentencia. Se debe tener en cuenta que una sentencia de asignación puede resultar literal en la asignación de una función o de un objeto.

```
//conexion con el servidor
String connectionServer = "jdbc:postgresql://" + this.server+":"+this.port;
//preparando conexion a la base de datos
String connectionDB     = "jdbc:postgresql://" + this.server+":"+this.port+"/"+this.dbName;
try
{
    //conectandome con la base de datos
    con = DriverManager.getConnection(connectionDB, this.usuario, this.pass);
}
```

Figura 10. Ejemplo de sentencias simples.

Sentencias Compuestas:

Las sentencias compuestas son aquellas sentencias que contienen una lista de estas. Contienen estructura de control como if o for. Esto facilita agregar nuevas sentencias sin la introducción accidental de errores. (36)

```
    //En estos 2 bucles anidados construyo la estructura intermedia con los metadatos de postgreSQL.
    for (String schemaName : schemaList)
     {
          GrouperContainer gc = new GrouperContainer(eti,schemaName);
          List<String> tableList     = this.pgMeta.getTables(gc.getName());

          for(String tableName : tableList)
          {
              MainContainer current = new MainContainer(gc,tableName.toLowerCase());
              constraintMainContainer(current,gc.getName(),tableName.toLowerCase());
              populateMainContainer(current,schemaName);
              gc.addItem(current);
          }
          eti.addItem(gc);
     }
     return eti;
}
```

Figura 11. Ejemplo de sentencias compuestas

3.2 Descripción de los principales métodos implementados

A continuación se muestran algunos métodos de la implementación del sistema así como una breve descripción de ellos para un mejor entendimiento.

3.2.1 Método mapToInterStructure

Este método se encuentra en la clase PgSchemaMapping el cual se encarga construir la ETI[2] usando los metadatos de PostgreSQL, trasformando así la estructura de datos de este a una homóloga en la estructura intermedia. Por ejemplo, Schemas en GroupContainer y Tablas en MainContainer.

[2] Estructura de transferencia intermedia

```
public InterStructure mapToInterStructure(String name)
{
    InterStructure eti = new InterStructure(name);
    List<String> schemaList      = this.pgMeta.getSchema();
    //En estos 2 cilos anidados construyo la estructura intermedia con los metadatos de postgreSQL.
    for (String schemaName : schemaList)
     {
         GrouperContainer gc = new GrouperContainer(eti,schemaName);
         List<String> tableList     = this.pgMeta.getTables(gc.getName());

         for(String tableName : tableList)
         {
             MainContainer current = new MainContainer(gc,tableName.toLowerCase());
             constraintMainContainer(current,gc.getName(),tableName.toLowerCase());
             populateMainContainer(current,schemaName);
             gc.addItem(current);
         }
         eti.addItem(gc);
     }
     return eti;
}
```

Figura 12. Ejemplo del método mapToInterStructure.

3.2.2 Método populateMainContainer

Este se encuentra en la clase PgSchemaMapping, usando los metadatos[3] de PostgreSQL se encarga de cargar la información de las columnas de las tablas en los Leaf de los MainContainer.

```
private void populateMainContainer(MainContainer container, String schemaName)
{
    try
    {
        List<String> columnsName = this.pgMeta.getColumnsName(schemaName, container.getName());
        ResultSet resultSet      = this.pgMeta.getData(schemaName, container.getName());

        while(resultSet.next())
        {
            List<Object> values = new ArrayList<>();
            for(int i = 0 ; i < columnsName.size(); i++)
            {
                values.add(resultSet.getObject(columnsName.get(i)));
            }
            Leaf currentLeaf = new Leaf(columnsName, values);
            container.addItem(currentLeaf);
        }
    }
    catch (SQLException ex)
    {
        Logger.getLogger(PgSchemaMapping.class.getName()).log(Level.SEVERE, null, ex);
    }
```

Figura 13. Ejemplo del Método populateMainContainer

3.2.3 Método getDocuments

Se encuentra en la clase MongoMetaData y se devuelve una lista con los documentos del gestor MongoDB.

[3] Datos que describen otros datos

```
public List<Map> getDocuments(String cName)
{
    DBCollection collection  = this.mHandler.getDb().getCollection(cName);
    DBCursor cursor          = collection.find();
    ArrayList<Map> documents = new ArrayList<>();
    while(cursor.hasNext())
    {
        DBObject document = cursor.next();
        documents.add(document.toMap());
    }
    return documents;
}
```

Figura 14. Ejemplo del método getDocuments.

3.2.4 Método insertElements

Este método se encuentra en la clase PostgresQuerryManager y su función es realizar las consultas para la inserción de datos en el gestor PostgreSQL.

```
public void insertElements(String scName, String tableName, List<Object>values)
{
    try
    {
        StringBuffer sql;
        if(scName.equals("") || scName == null)
            sql = new StringBuffer("INSERT INTO "+tableName+" VALUES ()");
        else
            sql = new StringBuffer("INSERT INTO "+scName+"."+tableName+" VALUES()");
        for(int i = 0; i<values.size(); i++)
        {
            if(i == values.size()-1)
                sql.insert(sql.length()-1, "?");
            else
                sql.insert(sql.length()-1, "?,");
        }
        PreparedStatement pStatement = this.pgConnection.getPgHandler().prepareStatement(sql.toString());
        for(int i = 0; i<values.size(); i++)
        {
            pStatement.setObject(i+1, values.get(i));
        }
        pStatement.executeUpdate();
    }
    catch (SQLException ex)
    {
        Logger.getLogger(PostgresQueryManager.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Figura 15. Ejemplo del método insertElements.

3.3 Interfaz principal de la aplicación

La Herramienta de Migración de Datos (Calidris) consta de una Interfaz Principal que le permite al usuario realizar diferentes tipos de migraciones de datos. La interfaz de Calidris está compuesta por componentes

que permiten una correcta navegación en la misma. En la parte izquierda de la aplicación se le brinda al usuario la opción de selección de los gestores origen y destinos respectivamente entre los que se va a realizar la migración y la introducción de todos los datos para realizar la migración de datos y la conexión a dichos gestores. En la parte superior de la interfaz se encuentra una tabla con datos (propiedad y valor) de las bases de datos origen y destino. En el centro de la interfaz se encuentran las tareas de migración, le permite al usuario ir añadiendo a una lista tantas migraciones quiera realizar para después realizar el proceso de migración de estás una por una. En la parte inferior se muestran reportes de información acerca del proceso de migración así como una barra del progreso de este.

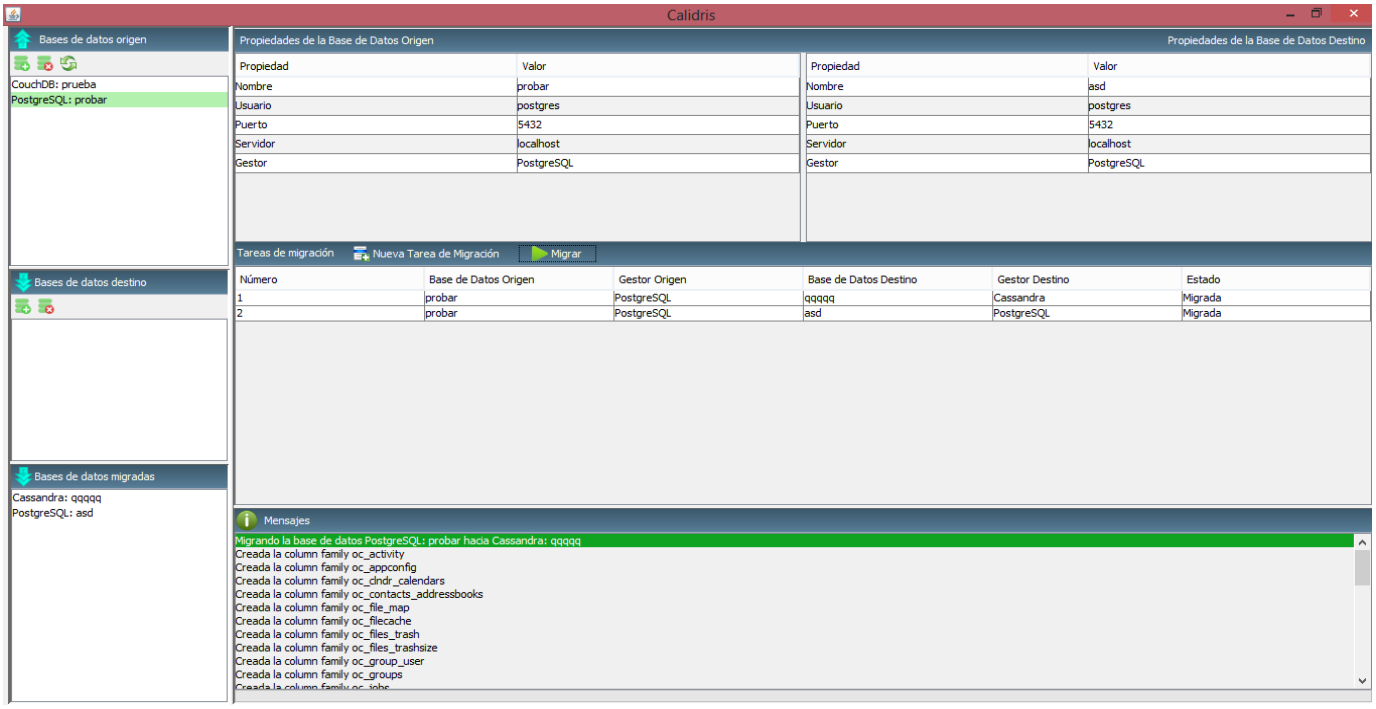


Figura 16. Imagen de la interfaz de la aplicación.

3.4 Pruebas

La fase de pruebas es una de las fases fundamentales del desarrollo de una aplicación. El objetivo de cada una de las pruebas no es el de prevenir errores sino de detectarlo basándose en técnicas y estrategias empleadas en cada una de las pruebas. Además son utilizadas para identificar posibles fallos de implementación, calidad o usabilidad de un programa.

3.4.1 Niveles de pruebas

A la hora de evaluar dinámicamente un sistema software se debe comenzar por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar todo el software en su conjunto.

Las pruebas se aplican durante todo el ciclo de desarrollo del software para diferentes objetivos y en distintos niveles de trabajo, dentro de estos se distinguen: (37)

- Pruebas de Desarrollador
- Pruebas Independientes
- Pruebas de Unidad
- Pruebas de Integración
- Pruebas de Sistema
- Pruebas de Aceptación

En el desarrollo de la fase de pruebas de la herramienta de migración de datos se aplicarán las pruebas de desarrollador y de aceptación para probar que el sistema cumpla con las historias de usuarios previamente definidas en la fase de análisis y diseño.

3.4.2 Técnicas de pruebas

Las técnicas de pruebas facilitan el cómo diseñar una serie de casos de prueba que tengan una alta probabilidad de encontrar errores, estas técnicas proporcionan directrices sistemáticas para pruebas de diseño que:

- Comprueben la lógica interna y las interfaces de todo componente del software.
- Comprueben los dominios de entrada y salida del programa para descubrir errores en su función, comportamiento y desempeño.

Funcionales

- Prueba funcional: Asegurar el trabajo apropiado de las historias de usuarios, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados. Además se prueba una funcionalidad completa, donde pueden estar implicadas una o varias clases y la propia interfaz de usuario. (37)

3.4.3 Pruebas de aceptación

Las pruebas de aceptación son creadas sobre la base de las HU. Las mismas son consideradas como "pruebas de caja negra", están estrechamente relacionadas con los requisitos funcionales de la aplicación y el cumplimiento de estos. El cliente debe especificar uno o diversos escenarios para comprobar que las HU han sido correctamente implementadas. Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. En caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una HU no se considera terminada hasta que se complete las pruebas de aceptación. Para asegurarse que la aplicación desarrollada cumple sus requisitos, a continuación se representan algunas de las pruebas de aceptación realizadas a las HU, el resto se encuentran en el expediente de proyecto en la planilla: Casos de pruebas de aceptación.

Tabla 8. Caso de prueba de aceptación 1.

Caso de prueba de aceptación	
Código:HU6_P1	**Historia de Usuario:6**
Nombre: Migrar datos de origen a destino.	
Descripción: Evaluar que se realice la migración de datos correspondiente.	
Condiciones de ejecución: Deben estar seleccionados los gestores de bases tanto origen como destino y la base de datos que se quiere migrar.	
Entrada/Pasos de ejecución: Se introducen los datos necesarios para garantizar la conexión a los gestores de bases de datos, se comprueba la conexión, se presiona el botón conectar y se selecciona en el gestor origen la base de datos a migrar. Luego de estar conectados los gestores origen y destino y seleccionado la base de datos a migrar se añade el proceso de migración en la lista de tareas y se presiona el botón migrar.	
Resultados Esperados: La herramienta muestra a través de mensajes los reportes del proceso de migración alcanzado exitosamente.	
Evaluación de la prueba: Satisfactoria	

A continuación se presentan ejemplos de la prueba de aceptación 1:

Ejemplo de base de datos a migrar desde PostgreSQL

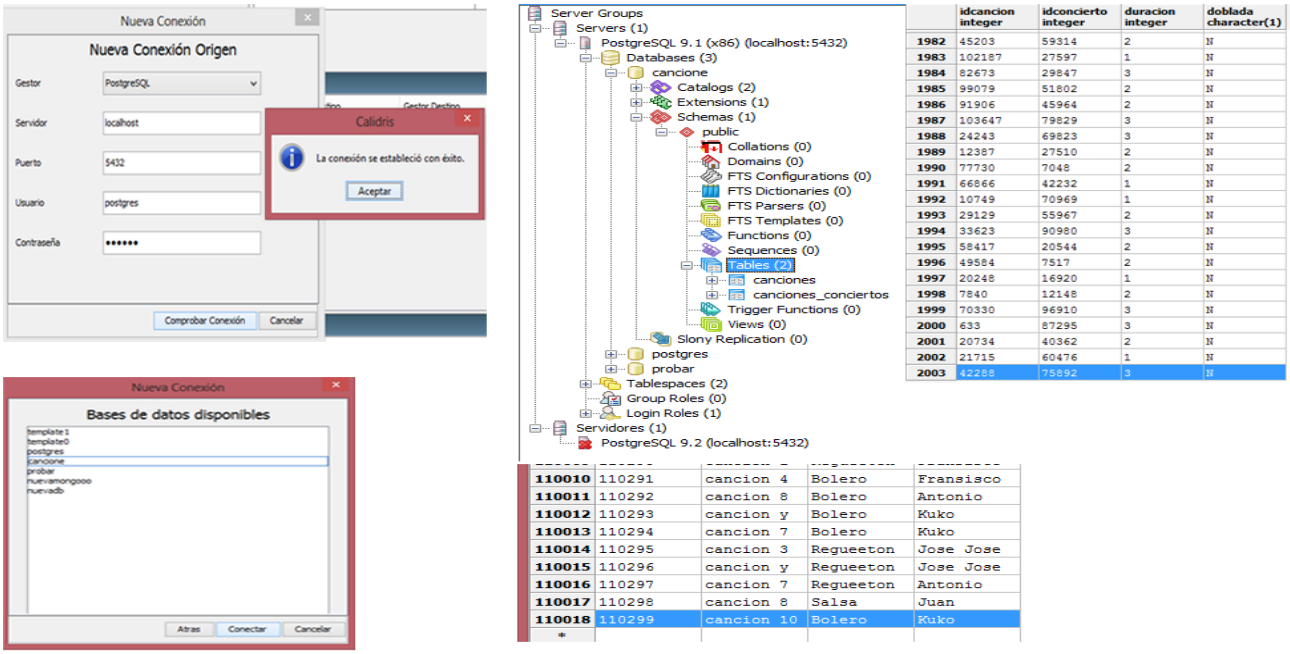


Figura 17. Ejemplo 1 de prueba de aceptación_1.

Migrando desde PostgreSQL a CouchDB

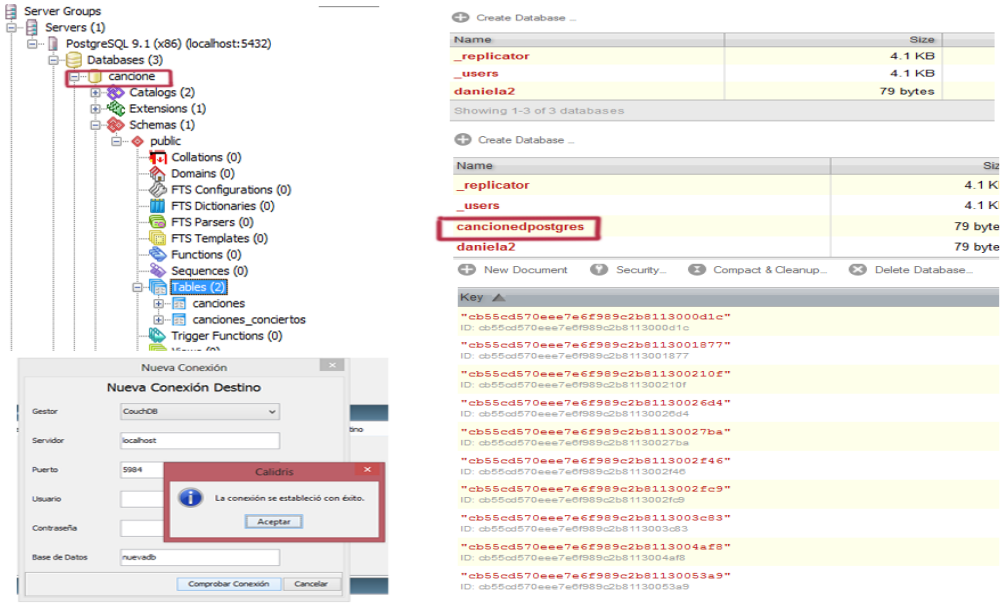


Figura 18. Ejemplo 2 de prueba de aceptación_1.

Migrando de PostgreSQL a MongoDB

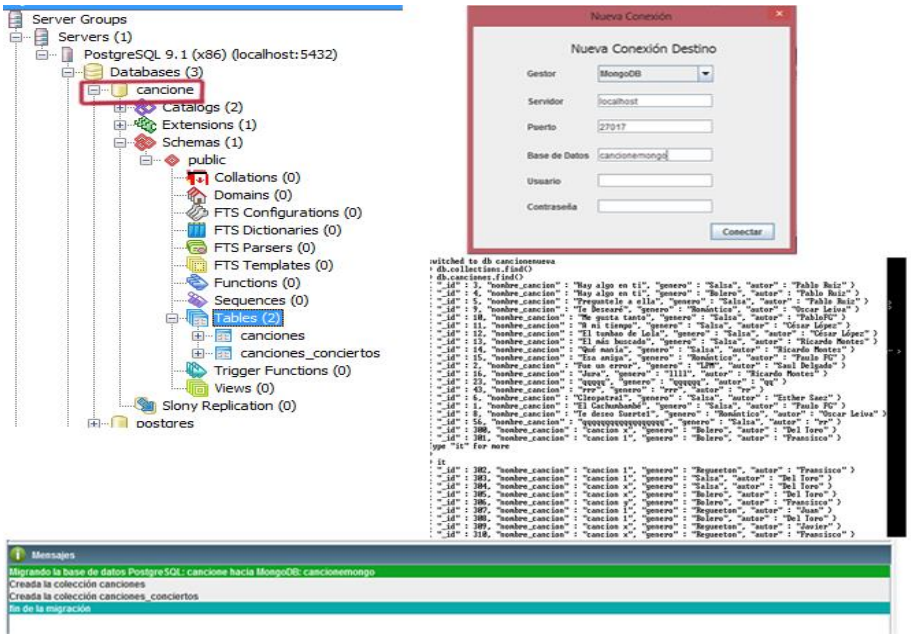


Figura 19. Ejemplo 3 de prueba de aceptación_1.

Migrando desde MongoDB a Cassandra

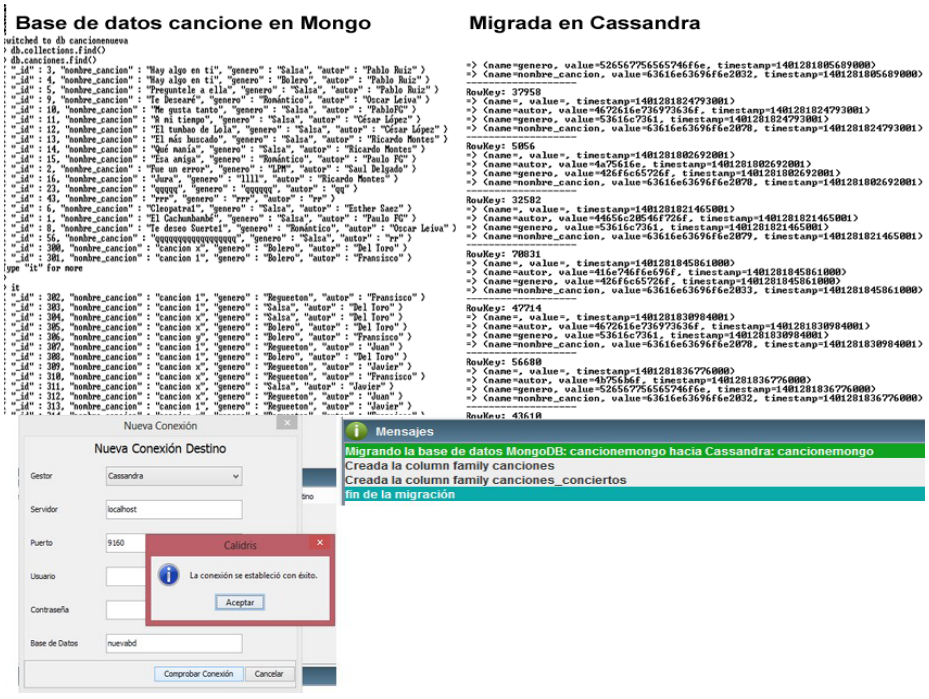


Figura 20. Ejemplo 4 de prueba de aceptación_1.

3.4.4 Métodos de pruebas

3.4.4.1 Caja negra

También conocidas como Pruebas de Comportamiento, se concentran en los requisitos funcionales de software, es decir, permitiendo al ingeniero de software derivar conjuntos de condiciones de entrada que ejercitarán por completos todos los requisitos funcionales de un programa. (37)

Estas pruebas tratan de encontrar errores en las siguientes categorías:

- Funciones incorrectas o faltantes.
- Errores de interfaz.
- Errores en estructura de datos o en acceso a bases de datos externas.
- Errores de comportamiento o desempeño.
- Errores de inicialización o término. (37)

Las pruebas de caja negra pretenden demostrar que las funciones del software son operativas y que funcionan correctamente aceptando de forma adecuada la entrada de datos y produciendo una salida correcta. Este tipo de prueba permite demostrarle al cliente que la aplicación puede satisfacer las necesidades del mismo. Se decide aplicar la técnica de caja negra debido a que es más importante el que el cliente este satisfecho y conforme con el producto al comprobar que las funcionalidades del programa funcionan correctamente satisfaciendo así las necesidades de este.

Técnica utilizada para el método de caja negra

Partición de Equivalencia: La partición de equivalencia es un método de prueba de Caja Negra que divide el campo de entrada de un programa en variables con juegos de datos de entrada y salida. En esencia, esta técnica intenta dividir el dominio de entrada de un programa en un número finito de variables de equivalencia. De tal modo que se pueda asumir razonablemente que una prueba realizada con un valor representativo de cada variable es equivalente a una prueba realizada con cualquier otro valor de dicha variable. (38)

Las variables de equivalencia representan un conjunto de estados válidos y no válidos para las condiciones de entrada de un programa. Éstas se identifican examinando cada condición de entrada (normalmente una frase en la especificación) y dividiéndola en dos o más grupos. Se definen dos tipos de

variables de equivalencia, las válidas, que representan entradas válidas al programa, y las no válidas, que representan valores de entrada erróneos, aunque pueden existir valores no relevantes a los que no sea necesario proporcionar un valor real de dato. (38)

3.4.5 Casos de pruebas basados en historias de usuario

En la actividad Diseño de los Casos de Prueba, usando técnicas de caja negra se desarrollan los casos de prueba. Esta actividad incluye diseñar las pruebas e identificar los datos de prueba. Para cada funcionalidad a probar, se crea una matriz que muestra su correspondencia con los casos de prueba, conociéndose de esta forma que ítem cubren los casos de pruebas. A través de los casos de pruebas se verifica si la aplicación realmente funciona, en ellos se describen los diferentes escenarios y se indica la respuesta correcta que el sistema debe mostrar en cada escenario. Se debe dejar bien claro en cada caso de prueba el flujo central de la aplicación que no es más que los pasos a seguir para trabajar en la aplicación a la hora de probar cada historia de usuario.

En la siguiente tabla se detallan las variables que se encuentran asociadas a la Historia de Usuario: Migrar datos de origen a destino.

Tabla 9. Variables asociadas a la HU Migrar datos de origen a destino.

No.	Nombre del campo	Clasificación	Nulo	Descripción
1	Servidor	Campo de texto	No	Host de la BD origen
2	Puerto	Campo de texto	No	Puerto de la BD origen
3	Base de datos	Campo de texto	No	Nombre de la BD origen
4	Usuario	Campo de texto	No	Nombre del usuario
5	Contraseña	Campo de texto	No	Contraseña del usuario
6	Servidor	Campo de texto	No	Host de la BD destino
7	Puerto	Campo de texto	No	Puerto de la BD destino
8	Base de datos	Campo de texto	No	Nombre de la BD destino
9	Usuario	Campo de texto	No	Nombre del usuario
10	Contraseña	Campo de texto	No	Contraseña del usuario

Esta descripción posibilitó que se realizara una matriz de datos, donde se evaluó y probó la validez de cada uno de los datos introducidos en el sistema, específicamente en la sección que se estuvo probando.

Utilizando un juego de datos válidos e inválidos se identificó el empleo de la técnica de partición de equivalencia.

Nombre de la HU: Migrar datos de origen a destino.

Tabla 10. Ejemplo de caso de prueba de caja negra_1.

ID del escenario	Descripción	Variables										Respuesta del sistema	Flujo central
		1	2	3	4	5	6	7	8	9	10		
EC 1.1 Introducir Datos Correctos.	Se introducen datos correctamente para conectarse a las base de datos origen y destino respectivamente.	*v*	*V*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	El sistema muestra un mensaje afirmando el éxito de la conexión.	El usuario introduce los datos necesarios para la conexión y luego pulsa el botón conectar.
		(localhost)	*(5432)*	*(MiBD)*	*(postgres)*	*(ysedon)*	*(localhost)*	*(27017)*	*(MiBD)*	*(admin)*	*(admin)*		
EC 1.2 Introducir datos incorrectos.	Se introducen datos incorrectos.	*v*	*l*	*v*	*v*	*l*	*v*	*v*	*v*	*v*	*v*	EL sistema muestra un mensaje de error señalando el campo donde se ingresó el dato inválido.	El usuario introduce los datos necesarios para la conexión y luego pulsa el botón conectar.
		(localhost)	*(1232)*	*(MiBD)*	*(postgres)*	*(juan)*	*(localhost)*	*(27017)*	*(MiBD)*	*(lolo)*	*(admin)*		
EC 1.3 Campos Vacíos.	Se tratan de insertar los datos pero se dejan campos vacíos.	*v*	*V*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	*v*	EL sistema muestra un mensaje de error que falló la conexión señalando el campo donde se ingresó el dato inválido.	
		(localhost)	*()*	*(MiBD)*	*()*	*(juan)*	*(localhost)*	*(27017)*	*(MiBD)*	*()*	*(admin)*		

3.4.6 Presentación de los resultados de las pruebas funcionales

Después de realizar las pruebas de Caja Negra mediante los casos de prueba asociados a cada historia de usuario, se comprobó el correcto funcionamiento de la herramienta y la correcta validación de los campos, verificando que solo se acepten los caracteres válidos para los mismos.

Conclusiones del capítulo

En este capítulo se realizó un estudio de las técnicas, tipos y métodos de pruebas aplicadas al sistema, en la cual se decidió aplicar a nivel de desarrollador los tipos de pruebas de aceptación y funcionales basado en el método de caja negra, aplicando la técnica partición de equivalencia. Fueron probadas todas las funcionalidades, lográndose obtener una aplicación que responde a todas las historias de usuarios identificadas. Se definió el estándar de codificación que propone la metodología XP e imágenes de la aplicación así como ejemplos de algunos de los métodos de la implementación.

CONCLUSIONES

- Se obtuvo una herramienta que realiza la migración de datos entre los gestores PostgreSQL, Cassandra, MongoDB y CouchDB, nombrada Calidris, permitiendo el manejo de grandes volúmenes de datos y cumpliendo con las buenas prácticas del diseño de la POO.
- El diseño de una aplicación que garantiza la migración satisfactoria, reduce los riesgos que trae hacerlo de forma manual, y permite extenderse a otros gestores con adecuaciones mínimas.
- Las validaciones efectuadas mediante las pruebas con los criterios establecidos, permitieron probar todas las funcionalidades, lográndose obtener una aplicación que responde a las historias de usuarios identificadas.

RECOMENDACIONES

Luego de haber cumplido los objetivos planteados, y a partir de los resultados obtenidos, se recomienda:

- Continuar el desarrollo del sistema con el objetivo de implementar otras funcionalidades que se deseen para próximas versiones y facilitar la extensibilidad de la herramienta para su uso con otros gestores.

REFERENCIAS BIBLIOGRÁFICAS

1. El Costo Oculto de la Migración de Datos. [En línea] 2008. [Citado el: 13 de Noviembre de 2013.] http://www.acronis.com.uy/documentos/pdf/Data_Migration_wp.es.pdf.

2. Automatizacion de Unidades de Información. [En línea] 03 de 10 de 2008. [Citado el: 5 de Octubre de 2013.] http://www.bibliotecologia.filo.uba.ar.

3. Corona, Susana Laura. Factores Críticos de Éxitos en la migración de datos. [En línea] 2008. [Citado el: 14 de Octubre de 2013.]

4. Aizaga, Andrés. Modelos de Base de Datos. [En línea] 2007. [Citado el: 8 de Diciembre de 2013.] https://tecnologiaeinformaticacji.files.wordpress.com.

5. Gestor de Base de Datos. [En línea] [Citado el: 23 de Octubre de 2013.] http:// www.um.es/geograf/sigmur/sigpdf/temario_9.pdf.

6. Base de Datos. [En línea] [Citado el: 22 de Noviembre de 2013.] https://cursos.aiu.edu/Base%20de%20Datos/pdf/Tema%202.pdf.

7. NoSQL. [En línea] [Citado el: 13 de Noviembre de 2013.] http://www.nosql.es/blog/nosql/que-es-nosql.html.

8. NoSQL Database. [En línea] 2009. [Citado el: 21 de Noviembre de 2013.] https://sites.google.com/site/proyectosinosql/.

9. postgresql-es. [En línea] 2003 . [Citado el: 25 de Octubre de 2013.] http://www.postgresql.org.es/sobre_postgresql.

10. Pirtle, Mitch. MongoDB for Web Development (1st edición). [En línea] 3 de Marzo de 2011. [Citado el: 7 de Diciembre de 2013.] http://docs.mongodb.org/manual/reference/mongostat.

11. Cassandra. [En línea] [Citado el: 29 de Octubre de 2013.] http:// www.ibm.com/developerworks/.

12. Hernandez Pérez, Rafael. CouchDB. [En línea] 2012. [Citado el: 9 de Diciembre de 2013.] http://couchdb.apache.org/.

13. Wilson Moré, Raidel y Cordero Vazquez, Marcos Raúl. Herramienta de Migración de Datos entre los Gestores de Base de Datos PostgreSQL, MongoDB y CouchDB. 2013.

14. Coimbra, V.H.G. Modelos de procesos de desarrollo, ciclo de vida de desarrollo de software, MSF, MOF, ITIL. 2009.

15. Méndez, A.V. Metodologías de Desarrollo de Software. 2010.

16. Patricio Letelier, C.P. Metodologías ágiles para el desarrollo de software: eXtreme Programming(XP). 2009.

17. Metodología XP. [En línea] [Citado el: 29 de Octubre de 2013.] http://recursosbiblioteca.utp.edu.co/dspace/bitstream/0053E18cp.pdf.

18. Metodologías ágiles. [En línea] [Citado el: 30 de Octubre de 2013.] http://www.casadellibro.com/libro-ingenieria-del-software-7-ed/1775061.
19. Nobrega, Maria de. Herramientas CASE: Rational Rose. [En línea] 4 de Junio de 2005. [Citado el: 29 de Octubre de 2013.]
20. Programación en Java.Características del lenguaje. [En línea] [Citado el: 29 de Octubre de 2013.] http://education.oracle.com/.
21. Visual Paradigm. [En línea] 30 de Noviembre de 2011. [Citado el: 26 de Octubre de 2013.] http://www.visual-paradigm.com/.
22. Entrenamiento Profesional en la Tecnología Java. [En línea] 26 de Enero de 2012. [Citado el: 23 de Noviembre de 2013.] http://globalmentoring.com.mx/cursos-java/java-fundamentos/que-es-un-ide/.
23. Lenguajes de Programación. [En línea] 2012. [Citado el: 11 de Enero de 2013.] http://www.casadellibro.com/libro-Java-9-ed/1885091.
24. Jacobson, Ivar , Booch, Grady y Rumbaugh, James. Proceso Unificadado de Desarrollo de Software. 2012.
25. Letelier, Patricio. Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). valencia : s.n., 2006.
26. Pressman, Roger. Ingeniería del Software: Un enfoque práctico. Séptima Edición. 2005.
27. Diagrama de Clases. [En línea] [Citado el: 30 de Enero de 2014.] http://www.scribd.com/doc/19776998/DIAGRAMA-DE-CLASES.
28. Microsoft Developer Network. [En línea] [Citado el: 21 de Febrero de 2014.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.
29. Gamma, Erich. Design patterns: elements of reusable object-oriented software.
30. Larman, Craig. Introducción al análisis y deseño orientado a objetos. México : s.n., 2002.
31. Departamento De Informática Universidad Técnica Federico Santa María. [En línea] [Citado el: 21 de Febrero de 2014.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.
32. autores, Varios. GOF Gang-Of-Four Book. 2000.
33. Larman, Craig. Introducción al análisis y deseño orientado a objetos. México : s.n., 2002.
34. Buschmann, y otros, y otros. Pattern-Oriented Software Architecture. 2000.
35. Letelier, Patricio. Métodologías ágiles para el desarrollo de software. Valencia : s.n., 2006.
36. Estandares de cofificacion java. [En línea] http://www.aves.edu.co/ovaunicor/recursos/view/265.
37. Pressman, Roger. Ingeniería del Software: Un enfoque práctico. Séptima Edición. 2005.
38. Somerville, Ian. Ingenieria de Software. 2005.

BIBLIOGRAFÍA

1. El Costo Oculto de la Migración de Datos. [En línea] 2008. [Citado el: 13 de Noviembre de 2013.] http://www.acronis.com.uy/documentos/pdf/Data_Migration_wp.es.pdf.

2. Automatizacion de Unidades de Información. [En línea] 03 de 10 de 2008. [Citado el: 5 de Octubre de 2013.] http://www.bibliotecologia.filo.uba.ar.

3. Corona, Susana Laura. Factores Críticos de Éxitos en la migración de datos. [En línea] 2008. [Citado el: 14 de Octubre de 2013.]

4. Aizaga, Andrés. Modelos de Base de Datos. [En línea] 2007. [Citado el: 8 de Diciembre de 2013.] https://tecnologiaeinformaticacji.files.wordpress.com.

5. Gestor de Base de Datos. [En línea] [Citado el: 23 de Octubre de 2013.] http:// www.um.es/geograf/sigmur/sigpdf/temario_9.pdf.

6. Base de Datos. [En línea] [Citado el: 22 de Noviembre de 2013.] https://cursos.aiu.edu/Base%20de%20Datos/pdf/Tema%202.pdf.

7. NoSQL. [En línea] [Citado el: 13 de Noviembre de 2013.] http://www.nosql.es/blog/nosql/que-es-nosql.html.

8. NoSQL Database. [En línea] 2009. [Citado el: 21 de Noviembre de 2013.] https://sites.google.com/site/proyectosinosql/.

9. postgresql-es. [En línea] 2003 . [Citado el: 25 de Octubre de 2013.] http://www.postgresql.org.es/sobre_postgresql.

10. Pirtle, Mitch. MongoDB for Web Development (1st edición). [En línea] 3 de Marzo de 2011. [Citado el: 7 de Diciembre de 2013.] http://docs.mongodb.org/manual/reference/mongostat.

11. Cassandra. [En línea] [Citado el: 29 de Octubre de 2013.] http:// www.ibm.com/developerworks/.

12. Hernandez Pérez, Rafael. CouchDB. [En línea] 2012. [Citado el: 9 de Diciembre de 2013.] http://couchdb.apache.org/.

13. Wilson Moré, Raidel y Cordero Vazquez, Marcos Raúl. Herramienta de Migración de Datos entre los Gestores de Base de Datos PostgreSQL, MongoDB y CouchDB. 2013.

14. Coimbra, V.H.G. Modelos de procesos de desarrollo, ciclo de vida de desarrollo de software, MSF, MOF, ITIL. 2009.

15. Méndez, A.V. Metodologías de Desarrollo de Software. 2010.

16. Patricio Letelier, C.P. Metodologías ágiles para el desarrollo de software: eXtreme Programming(XP). 2009.

17. Metodología XP. [En línea] [Citado el: 29 de Octubre de 2013.] http://recursosbiblioteca.utp.edu.co/dspace/bitstream/0053E18cp.pdf.

18. Metodologías ágiles. [En línea] [Citado el: 30 de Octubre de 2013.] http://www.casadellibro.com/libro-ingenieria-del-software-7-ed/1775061.
19. Nobrega, Maria de. Herramientas CASE: Rational Rose. [En línea] 4 de Junio de 2005. [Citado el: 29 de Octubre de 2013.]
20. Programación en Java.Características del lenguaje. [En línea] [Citado el: 29 de Octubre de 2013.] http://education.oracle.com/.
21. Visual Paradigm. [En línea] 30 de Noviembre de 2011. [Citado el: 26 de Octubre de 2013.] http://www.visual-paradigm.com/.
22. Entrenamiento Profesional en la Tecnología Java. [En línea] 26 de Enero de 2012. [Citado el: 23 de Noviembre de 2013.] http://globalmentoring.com.mx/cursos-java/java-fundamentos/que-es-un-ide/.
23. Lenguajes de Programación. [En línea] 2012. [Citado el: 11 de Enero de 2013.] http://www.casadellibro.com/libro-Java-9-ed/1885091.
24. Jacobson, Ivar , Booch, Grady y Rumbaugh, James. Proceso Unificadado de Desarrollo de Software. 2012.
25. Letelier, Patricio. Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). valencia : s.n., 2006.
26. Pressman, Roger. Ingeniería del Software: Un enfoque práctico. Séptima Edición. 2005.
27. Diagrama de Clases. [En línea] [Citado el: 30 de Enero de 2014.] http://www.scribd.com/doc/19776998/DIAGRAMA-DE-CLASES.
28. Microsoft Developer Network. [En línea] [Citado el: 21 de Febrero de 2014.] http://msdn.microsoft.com/es-es/library/bb972240.aspx.
29. Gamma, Erich. Design patterns: elements of reusable object-oriented software.
30. Larman, Craig. Introducción al análisis y deseño orientado a objetos. México : s.n., 2002.
31. Departamento De Informática Universidad Técnica Federico Santa María. [En línea] [Citado el: 21 de Febrero de 2014.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.
32. autores, Varios. GOF Gang-Of-Four Book. 2000.
33. Larman, Craig. Introducción al análisis y deseño orientado a objetos. México : s.n., 2002.
34. Buschmann, y otros, y otros. Pattern-Oriented Software Architecture. 2000.
35. Letelier, Patricio. Métodologías ágiles para el desarrollo de software. Valencia : s.n., 2006.
36. Estandares de cofificacion java. [En línea] http://www.aves.edu.co/ovaunicor/recursos/view/265.
37. Pressman, Roger. Ingeniería del Software: Un enfoque práctico. Séptima Edición. 2005.
38. Somerville, Ian. Ingenieria de Software. 2005.
39. Acronis.sa El Costo Oculto de la Migración de Datos [Citado el: 10 de Enero de 2014.] 2008 http://www.acronis.com.uy/documentos/pdf/Data_Migration_wp.es.pdf. [En línea].

ANEXOS

Anexo 1: Proceso de migración de datos

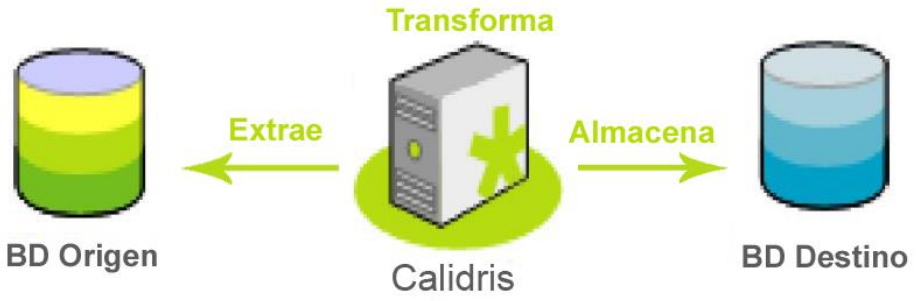


Figura 21. Proceso de migración de datos

Anexo 2: Demostración del crecimiento cuadrático de la clase Migración

Al formar todos los dúos posibles de un conjunto de k elementos donde el orden importa y sin repeticiones se tiene que la cantidad de dúos posibles es igual a la variación de k elementos tomados en grupos de a 2:

$V_2^k = \frac{k!}{(k-2)!} = \frac{k(k-1)\cancel{(k-2)!}}{\cancel{(k-2)!}} = k(k-1)$ (Desarrollando y simplificando términos semejantes)

Se tiene entonces que la cantidad (C) posible de dúos que se pueden formar de un conjunto de k elementos es de:

$C = k(k-1)$

Se puede deducir que al adicionar x elementos nuevos al conjunto de k se tendría que $k+x$ sería el nuevo tamaño del conjunto y $C = (k+x)(k+x-1)$ la cantidad de dúos que se pueden formar a partir de este.

Sea $C_0 = k(k-1)$ la cantidad inicial de dúos iniciales y $C = (k+x)(k+x-1)$ los que se obtendrían al adicionar x elementos nuevos al conjunto. Entonces la cantidad de nuevos dúos que se pueden formar al adicionar x elementos nuevos a un conjunto de k elementos está dado por:

- $y = C - C_0 = (k+x)(k+x-1) - k(k-1)$ (Desarrollando la multiplicación)
- $y = \cancel{k^2} + kx - \cancel{k} + kx + x^2 - x - \cancel{k^2} + \cancel{k}$ (Sumando términos semejantes)
- $y = x^2 + 2kx - x$

Por tanto la cantidad de nuevos dúos al adicionar x elementos a los k existentes está dada por la ecuación cuadrática:

$y = x^2 + (2k-1)x$