**Universidad de las Ciencias Informáticas**

**Facultad 6**

Trabajo de Diploma para Optar por el Título de
Ingeniero en Ciencias Informáticas

# Título: Generador de Datos para Bases de Datos NoSQL

**Autor:** Eduarluis Espinosa Gato

**Tutores:** MsC. Anthony Rafael Sotolongo

MsC. Tonysé De La Rosa Martín

**La Habana, 2013**

**Año 56 de la Revolución**

# DECLARACIÓN DE AUTORÍA

Declara ser autor de la presente tesis que tiene por título: Generador de Datos para Bases de Datos NoSQL y reconozco 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 _____.

**Eduarluis Espinosa Gato**

_____

Firma del Autor

**MsC. Anthony Rafael Sotolongo**

_____

Firma del Tutor

**MsC. Tonysé De La Rosa Martín**

_____

Firma del Tutor

# DATOS DE CONTACTO

**Tutor:**

MsC. Anthony Rafael Sotolongo León

Universidad de las Ciencias Informáticas, La Habana, Cuba

Correo electrónico: asotolongo@uci.cu

**Tutor:**

MsC. Tonysé De La Rosa Martín

Universidad de las Ciencias Informáticas, La Habana, Cuba

Correo electrónico: tdelarosa@uci.cu

# AGRADECIMIENTOS

*Le agradezco principalmente a mi madre querida por haberme apoyado siempre en mis decisiones y en cada paso que daba en la vida y por haberme guiado siempre por el camino correcto.*

*A mis dos padres, al biológico y al que ha estado gran parte de mi formación como ser humano a mi lado, a los dos gracias por el apoyo incondicional que siempre me han dado.*

*Un agradecimiento especial a dos personas que hoy no se encuentran con nosotros físicamente pero que son para mí como mis padres ya que fueron los que me criaron en mis primeros años de vida, a Mima y a Abebo muchas gracias y que dios los tenga en la Gloria.*

*A mi abuela Cuca por quererme y por atenderme tan bien cada vez que voy a verla, gracias.*

*A mi tutor Anthony por guiar mis pasos en mi formación como Ingeniero y por tenerme en cuenta siempre que hay que hacer algún trabajito del proyecto.*

*Al Laffo mi primer amigo y compañero de cuarto cuando llegue a esta escuela. Gracias por apoyarme en las buenas y en las malas.*

*A Elizabeth gracias por soportarme gran parte de tu vida y darme siempre tu cariño incondicional.*

*A dos personas que desde hace algún tiempo ocuparon un lugar en mi corazón desde que nos conocimos, a Anet y a Laura, gracias por soportarme.*

*A los iracundos Julio, Líván, Linares y Mario por acompañarme y ser buenos amigos desde que nos conocimos.*

*A Glennys y Ovi por ser mis amigos y brindarme su apoyo incondicional en las buenas y en las malas. A Glennys por ser profesora y amiga y a Ovi por darme una idea genial para la optimización de la aplicación.*

*Al piquete del comité de base 106106, les agradezco por tenerme en cuenta cuando había que formarla en cualquier lugar y por ayudarme cuando me hacía falta algo.*

# Agradecimientos

*A Claudia por ser buena amiga y aguatarme mis pesadeces.*

*A Claudiña por colarse en mi corazón en tan poco tiempo.*

*A la gente del aula gracias por confiar en mi cada vez que tenían alguna duda en algo, y por ayudarme cada vez que me hacía falta algo.*

*A Marcos por ayudarme en algo cada vez que me hace falta y por ser buen amigo.*

*A todos muchas gracias por influir de una manera u otra e mi formación como Ingeniero Informático.*

# DEDICATORIA

*Este trabajo de diploma va dedicado a mi madre por quererme y por su fiel guía en todos mis pasos, Te quiero.*


Carpe Diem

# RESUMEN

Al realizar pruebas de software es necesario que la base de datos utilizada por la aplicación cuente con información almacenada. El llenado de estas bases de datos se realiza utilizando herramientas específicas para llevar a cabo este proceso. Las gestores NoSQL han tomado un gran auge en los últimos años, por lo que están siendo utilizadas cada vez más en el desarrollo de aplicaciones, esto trae consigo que se haga necesario el poblado de las bases de datos de estos gestores para poder realizar pruebas a las aplicaciones que los utilizan.

Este trabajo de diploma brinda como propuesta un sistema de generación de datos para gestores NoSQL, con el objetivo de realizar el poblado de las bases de datos de estos gestores, para agilizar el proceso de pruebas al software que utilicen gestores NoSQL. Como resultado de la investigación se obtuvo la herramienta *NoSQL Generator* la cual permite realizar el poblado de bases de datos para gestores NoSQL.

Para el desarrollo de esta solución se seleccionó XP (eXtreme Programming) como metodología de desarrollo, usando UML como lenguaje de modelado y como herramienta CASE el Visual Paradigm. La implementación de la solución fue realizada con el lenguaje Python usando como IDE de desarrollo Aptana Studio.

**Palabras claves:** Gestores NoSQL, Generación de datos, Aleatorio.

# ABSTRACT

When testing software it is necessary that the database used by the application contains stored information to perform the tests and check the operation of the application. The filling of these databases is performed using tools to carry out this process. The NoSQL managers have become very popular in recent years, so they have been increasingly used in application development; this brings populating these databases a necessity in order to perform tests to the applications that use them.

This thesis proposes as a solution a data generation system for NoSQL managers, with the objective of making the NoSQL manager's databases populate, that speeds up the testing process on software that uses this kind of database managers. As the result of this investigation it was obtain the NoSQL Generator tool wish allows make the databases populated.

To develop this application eXtreme Programming (XP) was selected as the development methodology, using UML as the modeling language and Visual Paradigm as the CASE tool. The solution was implemented using Python as the programming language and the Aptana Studio as the Integrated Development Environment (IDE).

**Keywords**: NoSQL managers, Data generation, Random.

# ÍNDICE DE CONTENIDO

# ÍNDICE DE TABLAS

# ÍNDICE DE FIGURAS

# INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones TIC, surgieron con el objetivo de informatizar muchos de los procesos que se estaban llevando a cabo en la sociedad. Actualmente la información constituye uno de los pilares fundamentales en el desarrollo de cualquier país, la necesidad de almacenarla de forma rápida y fácil se convierte en una prioridad para cualquier institución, pues tenerla organizada permite que esta pueda ser accedida en cualquier momento por todas aquellas personas que la necesiten. Con el desarrollo de la informática se comienzan a crear sistemas de información que con el transcurso del tiempo y el perfeccionamiento de las técnicas se ha llegado a lo que hoy se conoce como base de datos.

En la actualidad la mayoría de los sistemas que son creados hacen uso de las bases de datos debido a que estos manejan gran cantidad de información. Los datos de las bases de datos son manejados a través de los Gestores de Bases de Datos, los cuales se componen por un conjunto de programas no visibles al usuario final, encargados de la privacidad, la integridad, la seguridad de los datos y la interacción del mismo con el sistema operativo. Este a su vez propicia una interfaz entre los datos, los programas que los manejan y los usuarios finales.

Con el cursar del tiempo las bases de datos relacionales han sido las más utilizadas para el almacenamiento de los datos que se usan en aplicaciones informáticas, dado a la seguridad e integridad de la información que este modelo brinda. Con el paso de los años y la evolución de los sistemas la cantidad de información a almacenar en las bases de datos aumentó. Según el sitio oficial de las CSC *(Computer Sciences Corporation*, por sus siglas en ingles), el volumen de información almacenada en el año 2012 es de aproximadamente de 7.9 ZettaByte [1], por lo que las consultas en estas bases de datos relacionales se tornan lentas y más cuando existen millones de usuarios haciendo búsquedas en millones de registros. Además el modelo relacional no permite almacenar de forma correcta estructuras de programación avanzadas que a menudo consisten en tipos de datos complejos o datos jerárquicos. A raíz de estos inconvenientes que vinieron presentando las bases de datos relacionales surge una nueva forma de guardar la información, las bases de datos NoSQL.

NoSQL (*Not Only SQL* por sus siglas en inglés), son gestores de bases de datos que difieren del modelo de almacenamiento de datos relacional. El aspecto más destacado de estos gestores es que no

usan SQL como el principal lenguaje de consultas. Los datos almacenados no requieren estructuras fijas como tablas, y habitualmente escalan bien horizontalmente. Estas se agrupan en cuatro grupos fundamentales: las orientadas a documentos, las orientadas a grafos, las columnares y las que almacenan la información en forma de atributo-valor.

La Universidad de las Ciencias Informáticas (UCI), específicamente el departamento de PostgreSQL perteneciente al Centro de Tecnologías de Gestión de Datos (DATEC), se especializa en el trabajo con los gestores de bases de datos. Actualmente el departamento se encuentra dando los primeros pasos en cuanto al uso de los gestores NoSQL, se utilizó MongoDB en el proyecto NAIRE y se están haciendo pruebas con CouchDB. Estos estudios realizados en el departamento tributan a la futura utilización de estos gestores en aplicaciones desarrolladas por el departamento.

Una de las labores fundamentales en el desarrollo de aplicaciones son las pruebas al software, lo que permite comprobar el comportamiento del sistema en un ambiente semejante a donde va a ser desplegado. Las pruebas de software son de gran importancia para garantizar la calidad de un sistema. Un proyecto promedio tiene semanas destinadas a las pruebas, principalmente en las semanas antes del despliegue, para así poder entregar al cliente un sistema lo más robusto y libre de errores posibles. Pruebas como, resistencia *(Stress)* o de rendimiento, son realizadas a las aplicaciones para comprobar en tiempo real su comportamiento en condiciones anormales.

Para poder realizar estas pruebas es necesario que las bases de datos cuenten con información almacenada, para que el sistema pueda funcionar correctamente. Para ello se hace necesario llenar estas bases de datos antes de comenzar el proceso de pruebas. Este proceso de llenado, o poblado de las bases de datos puede ser realizado manualmente, lo que conlleva un gasto de tiempo y recursos que puede llegar a atrasar el despliegue del sistema.

En la actualidad existen herramientas utilizadas para poblar las bases de datos, en algunos casos implementadas con tecnologías privativas, pero se aboga por la utilización de herramientas libres, y en otros desarrollada para los gestores relacionales. Actualmente los gestores NoSQL están siendo usados cada vez más en el desarrollo de aplicaciones y la universidad no se queda atrás en cuanto al uso de estos. Existen algunas herramientas desarrolladas en la universidad que hacen uso de estos gestores, como se dijo anteriormente el proyecto NAIRE es una de ellas, proyecto desarrollado por el Departamento de PostgreSQL y el buscador bibliográfico Sunshine. A medida que se vaya desarrollando el conocimiento

sobre este tipo de gestores en la universidad más van a ser usados, por lo que cuando se quiera realizar pruebas a estas aplicaciones sería necesario poblar estas bases de datos NoSQL.

Ion Lungu profesor del Departamento de Informática Económica de la facultad de cibernética de la Academia de Estudios Económicos de Bucarest, en conjunto con el profesor asistente Bogdan George Tudorica de la Universidad Petroleum-Gas de Ploiesti, Romania, desarrollaron una herramienta para realizar estudios estadísticos de comportamiento entre los gestores MongoDB y MySQL, en cuanto a tiempo de demora en operaciones de lectura y escritura. Esta investigación llevaba implícita la generación de datos, por lo que se considera el resultado de la misma como una herramienta para la generación de datos para gestores NoSQL. Pero esta tiene como inconvenientes que está desarrollada solo para un gestor NoSQL, MongoDB, y además que tiene implementada solo una técnica de generación de datos, en este caso la técnica aleatoria.

Luego de haber realizado una investigación detallada sobre las herramientas de generación de datos, no se encontró ninguna que permita realizar este proceso para gestores NoSQL. Lo anteriormente planteado trae consigo que se dificulte y se haga más trabajoso el almacenamiento de la información en bases de datos NoSQL, para realizar pruebas de software, debido al gasto de tiempo y de recursos empleado en la realización manual de este proceso.

Debido a lo antes expuesto se define como **problema a resolver** ¿Cómo lograr la generación de datos para las bases de datos NoSQL?

A partir del problema antes planteado se puede delimitar como el **objeto de estudio** el proceso de generación de datos.

Para darle solución al problema se define como **objetivo general** desarrollar una aplicación que permita la generación de datos para las bases de datos NoSQL con el objetivo de agilizar las pruebas de software, derivándose como **campo de acción** el proceso de generación de datos para las bases de datos NoSQL.

Para alcanzar el objetivo general se definieron los siguientes **objetivos específicos:**

- Realizar un estudio bibliográfico sobre las técnicas y herramientas para la generación de datos.
- Diseñar una herramienta para realizar la generación de datos para los gestores NoSQL.

- Implementar el sistema de generación de datos.
- Validar el sistema de generación de datos.

Para cumplir con los objetivos planteados se trazaron las siguientes **tareas de la investigación**:

- Caracterización de las técnicas y herramientas relacionadas con la generación de datos para bases de datos.
- Caracterización de los gestores de bases de datos NoSQL.
- Definición de las tecnologías a utilizar.
- Identificación y descripción de las funcionalidades con las que el sistema debe cumplir.
- Implementación de las funcionalidades identificadas.
- Selección de las técnicas y métodos para la validación de la solución.
- Realizar la validación de la solución.

Guiados por lo expuesto anteriormente y para apoyar la investigación se plantean las siguientes **preguntas científicas:**

- ¿Qué características tiene una herramienta de generación de datos?
- ¿Qué herramientas se deben utilizar para el desarrollo del sistema?
- ¿Qué gestores NoSQL utilizar para realizar el poblado de bases de datos?
- ¿Cómo diseñar el sistema de generación de datos?
- ¿Cómo validar el sistema de generación de datos desarrollado?

Para el desarrollo de esta investigación se utilizan los siguientes métodos científicos:

**Analítico-sintético:** Será utilizado para realizar el análisis de toda la información que se obtenga durante todo el proceso de desarrollo de la investigación y el mismo permitirá obtener una síntesis de los contenidos fundamentales que puedan resultar relevantes para este trabajo de diploma.

**Histórico-lógico:** Se realizará una investigación que permitirá tener conocimiento de los gestores NoSQL más utilizados a nivel mundial con el objetivo de realizar una selección de estos para realizar la generación de datos para estos gestores, así como la existencia de herramientas capaces de realizar este proceso.

La presente investigación se encuentra estructurada en tres capítulos divididos de la siguiente manera:

**Capítulo 1- Fundamentación teórica.**

En este capítulo se realiza toda la fundamentación teórica de la investigación, en la que se abordan los principales conceptos asociados al problema general y se realiza un estudio de las herramientas existentes que permiten realizar el proceso de generación, además de seleccionar las que serán utilizadas para realizar la implementación de la solución.

**Capítulo 2- Características del sistema.**

En este capítulo se realiza el diseño de la solución propuesto para darle cumplimiento al objetivo general. Se describen los artefactos que especifica la metodología de desarrollo seleccionada, los cuales fueron realizados para tener un mejor entendimiento del problema. Se definieron los patrones a utilizar durante la implementación de la solución.

**Capítulo 3- Implementación y pruebas.**

Este capítulo muestra el resultado obtenido luego de haber realizado la implementación de la solución. Se describen las pruebas realizadas para validar el sistema y contar con una herramienta funcional.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## Introducción

En este capítulo se abordarán los conceptos principales necesarios para entender el problema y la posible solución a este. Se seleccionarán los gestores de bases de datos que formarán parte de la solución final, la metodología a seguir durante el ciclo de desarrollo, así como la selección de las tecnologías necesarias para llevar a cabo la implementación de la aplicación.

## 1.1 Base de datos

Un concepto aportado por el Dr. Christopher J. Date, define a las bases de datos como: un conjunto de datos persistentes que es utilizado por los sistemas de aplicación de alguna empresa dada [2].

El Glosario IEEE de Ingeniería del Software (IEEE 1990) define el término base de datos de la siguiente forma: Una colección de datos interrelacionados almacenados conjuntamente en uno o más ficheros de computadora [3].

Visto estos conceptos de varios autores se puede definir como base de datos a un conjunto de datos almacenados referentes a un mismo tema, variables en el tiempo, a los cuales se puede acceder de forma rápida y sencilla. Existen varios modelos de almacenamiento de datos, dentro de los que se encuentran el modelo relacional y el NoSQL.

## 1.2 Gestores de bases de datos NoSQL

Existen varios tipos de gestores NoSQL los cuales son clasificados dependiendo del tipo de dato que almacena cada uno, están los orientados a documentos y a grafos, los que almacenan los datos en forma de atributo-valor *(Key-Value)* y las llamadas base de datos columnares u orientadas a columnas.

### *1.2.1 Orientados a Documentos*

Este tipo de almacenamiento de datos se guarda con un formato definido de forma tal que el servidor pueda entenderlo. Aunque cada base de dato orientada a documento a menudo difiere del formato en general todas asumen el encapsulamiento de documentos y la codificación de los datos. Los formatos

más comúnmente utilizados son JSON, YAML y XML. Entre las bases de datos de este tipo, están los más conocidos de la familia NoSQL: MongoDB y Apache CouchDB.

**MongoDB:** es un poderoso, flexible y escalable gestor de bases de datos. Combina la habilidad de escalar muchas de las características más útiles de las bases de datos relacionales, tales como índices secundarios, consultas de rango y ordenamiento. MongoDB, cuenta con variadas características útiles tales como soporte integrado para agregaciones de estilo MapReduce e índices geoespaciales [4].

MongoDB es un gestor de bases de datos orientada a documento, no uno relacional. La principal razón para migrar del modelo relacional es para hacer más fácil la escalabilidad, pero también hay otras ventajas. MongoDB remplaza el concepto de fila por uno más flexible, el "documento". Este nuevo formato permite empotrar documentos y arreglos, lo que hace posible representar relaciones jerárquicas complejas con un solo registro. MongoDB además no guarda los datos en esquemas lo que se conoce como *schema-free*, esto posibilita a los desarrolladores una mayor flexibilidad a la hora de trabajar con los modelos de datos.

**Apache CouchDB:** es un sistema gestor de bases de datos relativamente nuevo, diseñado para adaptarse a las aplicaciones de software modernas que tienden a ser basadas en la web (web-based), orientada a documentos *(document-oriented)* y distribuidas [5]. Este es un gestor distribuido bajo la licencia de código abierto de Apache. También es *schema-free* por lo que a diferencia de los gestores SQL, no existen las tablas ni columnas, llaves primarias o foráneas, uniones o relaciones.

CouchDB es un gestor de base de datos orientado a documentos, por lo que almacena sus datos en documentos en formato JSON (JavaScript Object Notation). Los valores que pueden ser almacenados en los documentos pueden ser, numéricos, fechas, booleanos, listas, mapas u otros tipos de datos. Todos los documentos en la base de datos tienen un identificador único además no existe límite en la cantidad de documentos a almacenar en la base de datos así como el tamaño de los valores almacenados.

### *1.2.2 Almacenamiento Atributo-Valor*

Estas son las bases de datos más simples en cuanto a su uso, ya que simplemente almacenan valores identificados por una clave. Normalmente, el valor guardado se almacena como un arreglo de bytes. De esta forma el tipo de contenido no es importante para la base de datos, solo la clave y el valor que tiene asociado.

**Redis:** es un gestor de bases de datos estructurado con un conjunto de datos en memoria para aumentar la velocidad. Es llamado un gestor de bases de datos estructurado y no solo un servidor que almacena en forma de atributo-valor porque implementa estructuras de datos que permiten a las claves que contengan cadenas binarias, hashes, conjuntos y conjuntos ordenados, así como listas. Esta combinación de flexibilidad y rapidez hacen que Redis sea la herramienta ideal para muchas aplicaciones [6].

### *1.2.3 Orientada a columnas*

Como su nombre lo indica, guardan los datos en columnas en lugar de filas. Este tipo de almacenamiento es eficiente en lecturas, ya que permite consultar un número reducido de columnas de forma rápida, pero no es eficiente para realizar escrituras. Por ello este tipo de soluciones es usado en aplicaciones con un índice bajo de escrituras pero muchas lecturas.

**Cassandra:** es un gestor de bases de datos de código abierto, distribuido, descentralizado, elásticamente escalable, altamente disponible, con tolerancia a fallos, orientado a columnas, que basa su diseño en la distribución de Dynamo de Amazon y su modelo de datos en Bigtable de Google. Creado en Facebook, ahora se utiliza en algunos de los sitios más populares en la web.

Frecuentemente es conocido como un gestor de bases de datos orientado a columnas, no es relacional y representa su estructura de datos en hashtable multidimensional esparcido. “Esparcido” quiere decir que para cualquier fila dada puede haber una o más columnas, pero cada fila no necesita tener todas las mismas columnas. Cada fila tiene una llave única, que hace que los datos sean accesibles. Cassandra almacena sus datos en lo que puede ser conocido como un hashtable multidimensional. Esto significa que no se tiene que decidir por adelantado como tu estructura de dato va a lucir o que campos los registros van a necesitar. Esto puede ser muy útil si se está empezando y se está agregando o modificando características con frecuencia [7].

### *1.2.4 Orientada a Grafos*

Una base de datos orientada a grafo almacena los datos en un grafo, la más genérica de las estructuras de datos, capaz de representar cualquier tipo de datos de una manera altamente accesible. Los datos en este tipo de bases de datos son representados mediante nodos y relaciones a los cuales se les puede asociar propiedades.

**Neo4j:** es un gestor de bases de datos orientado a grafo, es de código abierto apoyado por *Neo Technology*. Este gestor almacena los datos en nodos conectados entre sí por relaciones, las cuales contienen propiedades. Este tipo de bases de datos también son conocidas como Grafos de propiedad.

Entre las principales características de este gestor se encuentran [8]:

- Intuitivo, utilizando un modelo gráfico para la representación de datos.
- Seguro, con transacciones ACID completos.
- Duradero y rápido, usando un motor de almacenamiento nativo basado en disco personalizado.
- Alta disponibilidad, cuando se distribuye a través de múltiples máquinas.
- Rápido, con un potente framework de recorrido para consultas a alta velocidad en los grafos.

Estos gestores descritos previamente fueron seleccionados dentro de la amplia gama de gestores NoSQL que existen por ser los más utilizados mundialmente en su clasificación ya que se encuentran en las primeras posiciones del ranking publicado en [9], aunque existen otros conocidos como Memcached, HBase, Riak y DynamoDB.

## 1.3 Generación de datos

Generar datos no es más que crear un dato con consistencia pero sin valor real alguno, el cual una vez almacenado en la base de datos permite simular un entorno lo más parecido a la realidad del sistema. Este proceso de generación de datos es importante debido a que para poder realizar las pruebas a la base de datos es importante que esta cuente con información almacenada y hacer esto de otra manera conllevaría a un gasto de tiempo y recursos innecesarios.

**Técnicas para generar datos**

Existen dos técnicas fundamentales para generar datos, una es de manera aleatoria en la cual se pueden controlar la forma en que se genera:

**Para generar numéricos:**

- Mediante rangos.

- Mediante cantidad de cifras.

**Para generar cadenas:**

- Mediante la longitud de la cadena.
- Mediante rangos de caracteres.
- Combinación de ambas.

**Para generar fechas:**

- Mediante rangos de fechas.
- Mediante incremento de fechas.

**Para generar listas:**

- Mediante especificación de la longitud de la lista y de los datos que se almacenarán en ella.

Existen variedades de formas cuando se quiere generar datos aleatorios. Otra técnica común de generar datos es mediante una fuente existente, donde el usuario especifica la fuente que va a usar y el sistema utiliza la información contenida en la fuente para insertarla en la base de datos. En la investigación se van a utilizar estas dos técnicas de generación de datos ajustándolas a las necesidades del sistema.

## 1.4 Herramientas para generación de datos

**forSQL Data Generator:** es un generador de datos automático para pruebas a base de datos y aseguramiento de la calidad a gran escala. Ayuda a los desarrolladores y probadores de base de datos a generar datos automáticamente para llenar la base de datos con datos de prueba lógicos y realistas. Ahorra contables horas e inclusive días que de otra manera se hubiesen desperdiciado realizando esta tarea manualmente [10].

Lo mejor en forSQL Data Generator es su carácter universal. No solo funciona con solo un servidor, el programa también puede funcionar con varios servidores que usen ODBC[1]. Otra buena función en forSQL Data Generator es la habilidad de llenar los campos usando varios métodos diferentes que van desde

[1] Open DataBase Connectivity es un estándar de acceso a bases de datos.

valores aleatorios hasta llenarlos con datos reales desde una plantilla. Esta herramienta lee automáticamente la información acerca de la estructura de la base de datos, llaves primarias y foráneas, lo que simplifica considerablemente la configuración de la base de datos.

Este gestor tienes implementadas varias formas de generar los datos, como son [10]:

- Genera datos desde archivos con datos reales. (ej.: lista de provincias de Cuba).
- Genera datos aleatorios.
- Genera datos por formato.
- Genera datos constantes.
- Genera datos desde una lista.
- Genera datos incrementales.
- Genera datos como resultado de un script SQL.

**Spawner Data Generator:** es una herramienta de código abierto para generar datos para poblar las bases de datos, existe una versión para Linux y otra para Windows. Es un generador de datos de prueba para bases de datos. Puede ser configurado como salida de texto delimitado o como una declaración de insert de SQL. También puede insertar los datos directamente en las bases de datos de MySQL. Incluye varios tipos de campos, la mayoría pueden ser configurados [11].

Spawner Data Generator puede crear aleatoriamente números, cadenas y fechas, pero también puede hacer algunas cosas inteligentes con estos como secuencias, rangos, número de palabras, número de caracteres, nombres, emails, direcciones. También puede generar direcciones IPv4.

**Generatedata:** es una herramienta libre y de código abierto escrita en JavaScript, PHP y MySQL que permite generar rápidamente grandes volúmenes de datos personalizados en una variedad de formatos para su uso en pruebas de software, rellenar bases de datos, etc. Ofrece una versión en línea para que el usuario interactúe con él y tenga una idea de lo que hace el programa, las características que ofrece y cómo funciona. Brinda además una versión para su instalación en el sistema del cliente para que este haga uso de el sin tener que conectarse a internet.

Permite al usuario escoger entre una inmensa cantidad de formatos para los cuales el sistema genera los datos; ej.: código HTML, formato JSON, lenguajes de programación. Es compatible con varios gestores de bases de datos como son MySQL, PostgreSQL, SQLite. Además brinda disímiles tipos de datos que pueden ser usados para generar los datos aleatorios [12].

El funcionamiento básico de estas herramientas es el mismo, dan la posibilidad al usuario de escoger que gestor de bases de datos desea poblar y cual base de dato en específico. Luego de estar conectadas al gestor seleccionado estas herramientas permiten la configuración de los datos que van a ser insertados en la base de datos. Cada una de estas tiene sus particularidades en la configuración de los datos ya que unas cuentan con soporte para más cantidad de datos que otras, pero en general el proceso es el mismo. Una vez que se configuraron todos los parámetros de los datos se procede a la creación del dato y a la inserción de este en la base de datos seleccionada por el usuario.

Todos estos generadores de datos explicados anteriormente son de gran utilidad para la realización de pruebas de software a bases de datos. Estos son capaces de generar un gran cúmulo de datos aleatoriamente para luego insertarlos en la base de datos. El inconveniente de estas herramientas identificadas es que son funcionales principalmente para los gestores de bases de datos relacionales como son PostgreSQL, MySQL, SQLite, etc.

En el caso de la herramienta Generatedata genera el tipo de dato JSON que es el utilizado en los gestores documentales MongoDB y CouchDB, esto pudiera utilizarse para generar los datos, pero estos se exportan en texto plano y no son insertados directamente en el gestor por la herramienta. Debido a este inconveniente sería necesario realizar una aplicación que cargara estos datos JSON generados por Generatedata y los insertara en la base de datos. Esta opción no es factible ya que habría que implementar otra herramienta y lo mejor sería contar con solo una aplicación que realice todo el proceso.

**Herramientas de generación de datos para gestores NoSQL**

Debido a la reciente popularidad de los gestores NoSQL no se encontraron herramientas especializadas para la generación de datos para estas bases de datos. En caso de MongoDB que es el gestor que cuenta con más prestigio y popularidad mundialmente, ya que se encuentra ubicado en el primer lugar del ranking de las bases de datos NoSQL, lo que se sugiere es que cuando se le quiera realizar alguna prueba a la base de datos, se tome una ya existente con datos almacenados y se copien estos en la base de datos

para la realización de las pruebas, y una vez terminado de hacer estas pruebas eliminar estos datos que fueron copiados.

En el caso de los otros gestores como Redis, Neo4j y CouchDB, no se halló tampoco herramientas para poblar sus bases de datos, por lo que se puede asumir que para la realización de pruebas se puede ejecutar el mismo procedimiento que con el gestor MongoDB. En cuanto al gestor Cassandra, cuenta con la herramienta Cassandra-Unit que de por si no genera datos automáticamente, sino que agiliza el proceso de creación de tuplas a las tablas en la base de datos de Cassandra ya que permite cargar datos desde un fichero .yaml que ya contiene todos los datos generados aleatoriamente dentro de las tablas.

Luego de haber realizado una investigación detallada sobre las herramientas de generación de datos para gestores NoSQL, no se encontró ninguna que permita realizar el proceso de poblado de bases de datos para estos gestores. Por lo que cuando se quiera realizar pruebas a los software que utilicen bases de datos NoSQL esto sería un inconveniente, ya que habría que realizar el poblado de la base de datos de manera manual y cuando sea posible, realizar una copia de otros datos existentes. Lo ideal sería contar con una aplicación informática que realice esta operación de forma automática, para agilizar el proceso de pruebas.

## 1.5 Metodología de desarrollo de software

Una metodología de desarrollo de software es un conjunto de pasos a seguir y tener en cuenta en el desarrollo de una aplicación informática. Esta guía todo el proceso de desarrollo desde que se hace la entrevista con el cliente hasta que se le entrega el producto ya realizado. Escoger una buena metodología para el desarrollo de la aplicación es de vital importancia ya que esta selección conlleva al éxito o al fracaso del proyecto.

Cada metodología define diferentemente como llevar a cabo el ciclo de desarrollo del software, pero todas tienen como objetivo fundamental obtener un producto final con calidad y que cumpla con todas las especificaciones descritas por el cliente. Estas especificaciones o, como también se les llama, requisitos en algunos casos tienden a variar durante el transcurso del desarrollo de la aplicación, por lo que han surgido diferentes metodologías de desarrollo y se han clasificado en dos grupos, las robustas y las ágiles.

De manera general las metodologías robustas definen un ciclo de vida más amplio y con bastante documentación que realizar para construir el software por lo que se hace necesario contar con abundante

personal para llevar a cabo el ciclo de vida del proyecto. Por otro lado las metodologías ágiles no definen gran cantidad de documentación a realizar para poder implementar el software y son más fiables de utilizar cuando se cuenta con un número reducido de personal y cuando el tiempo de entrega de la solución es corto.

Dentro de las metodologías agiles se encuentra Extreme Programming (XP) la cual tiene como objetivo satisfacer las necesidades del cliente. La metodología XP brinda retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas [13].

- **Entregas pequeñas.** Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

- **Diseño simple.** Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

- **Pruebas.** La producción de código está dirigida por las pruebas unitarias. Estas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

- **Refactorización *(Refactoring)*.** Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

- **Cliente in-situ.** El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Este es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el

trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.

- **Estándares de programación.** XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

Se selecciona la metodología de desarrollo XP ya que es la que más se adecúa a las necesidades del proyecto y además por ser la metodología de trabajo predefinida por el departamento.

## 1.6 Tecnologías y herramientas

Para poder lograr una mejor implementación del sistema, se hace necesario hacer una investigación de las herramientas existentes que permiten desarrollar software. Una vez investigado se hace necesario comprobar que estas herramientas son las adecuadas para la implementación de la solución.

### *1.6.1 Lenguaje de Modelado*

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios[14].

Algunas de las propiedades de UML como lenguaje de modelado estándar son:

- Concurrencia, es un lenguaje distribuido y adecuado a las necesidades de conectividad actuales y futuras.
- Ampliamente utilizado por la industria.
- Reemplaza a decenas de notaciones empleadas con otros lenguajes.
- Modela estructuras complejas.
- Las estructuras más importantes que soporta tienen su fundamento en las tecnologías orientadas a

objetos, tales como: objetos, clases, componentes y nodos.

- Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario.
- Comportamiento del sistema: casos de uso, diagramas de secuencia y de colaboración, que sirven para evaluar el estado de las máquinas.

### *1.6.2 Herramienta CASE*

CASE (*Computer Aided Software Engineering)* por sus siglas en inglés, traducido al español como *Ingeniería de Software Asistida por Computadoras*. Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ser de mucha ayuda en los aspectos del ciclo de vida de desarrollo del software en tareas como el diseño de proyectos, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras. Es un sistema de software que intenta proporcionar ayuda automatizada a las actividades del proceso de desarrollo de software.

Una de estas herramientas es el Visual Paradigm for UML, esta comparte las características de las herramientas CASE. Soporta el ciclo de vida completo del software: análisis y diseño, implementación, pruebas y despliegue.

**Ventajas de usar Visual Paradigm for UML [15]:**

- Integración con la mayoría de los IDEs de desarrollo.
- Es la herramienta UML más fácil de usar.
- Es multiplataforma.
- Tiene soporte para tarjetas CRC.
- Conversión instantánea de código fuente y ejecutables a modelos.
- Diseño de diagramas de forma automática.

Dada estas ventajas Visual Paradigm se escoge como una potente herramienta para ser usada en la construcción de la aplicación.

### *1.6.3 Lenguaje de programación*

Es el lenguaje codificado usado por los programadores para escribir instrucciones que una computadora puede entender para hacer lo que el programador quiera. El lenguaje de computadora más básico, o llamado también de bajo nivel, es el lenguaje de máquina el cual usa código binario (“1” y “0”) el cual la computadora puede ejecutar rápidamente sin tener que usar un traductor o intérprete para hacerlo, pero este tipo de codificación es muy compleja. Los lenguajes de alto nivel son mucho más simples de usar, pero necesitan usar otro programa para convertir el lenguaje de alto nivel en lenguaje de máquina, y son un poco más lentos. Existen docenas de lenguajes de programación y se están desarrollando algunos nuevos continuamente. También llamados lenguajes de computadoras [16].

Los lenguajes de programación brindan una solución práctica a los seres humanos a la hora de hacer que un ordenador haga lo que el usuario desea. Este facilita las tareas de programación debido a que brindan una manera de escribir y leer fácilmente un programa, los lenguajes de programación son independientes del tipo de ordenador que se está usando.

Dentro de los lenguajes de alto nivel se encuentra Python el cual es un lenguaje de programación, orientado a objetos y con semántica dinámica. Su estructura de alto nivel combinada con su tipado dinámico lo hacen muy atractivo para el desarrollo rápido de aplicaciones (RAD por sus siglas en ingles). Python es simple, fácil de aprender, su sintaxis enfatiza la legibilidad y por lo tanto reduce el costo en el mantenimiento del programa. Soporta módulos y paquetes, lo que estimula a la programación modular y el reutilizamiento de código.

Python contiene las bibliotecas necesarias para el trabajo con los gestores de bases de datos NoSQL seleccionados y además estas bibliotecas se encuentran bien documentadas y las compañías que las desarrollan brindan soporte a estas. Estas bibliotecas actúan como intermediario entre el lenguaje Python y el gestor de base de datos, por lo que hacen posible hacer peticiones y consultas a la base de datos desde Python de manera eficiente y rápida sin tener que usar el lenguaje del gestor para realizar estas tareas. Por lo que este lenguaje es el indicado para utilizarlo en la implementación del sistema.

### *1.6.4 Entorno de desarrollo*

Entorno de desarrollo integrado IDE por sus siglas en inglés, es una aplicación compuesta por una serie de herramientas que le facilitan el trabajo al programador. Esta puede estar desarrollada para soportar un solo lenguaje de programación o para soportar varios. Los IDE son editores de código o constructores de interfaz gráfica que pueden ser aplicaciones independientes o ser parte de aplicaciones que ya existen. Proveen un ambiente de trabajo muy amigable para la mayoría de los lenguajes de programación.

El IDE seleccionado es el Aptana Studio, este soporta múltiples lenguajes de programación dentro de los que se encuentra Python y haciendo uso del complemento Pydev y Extensiones Pydev, brinda la posibilidad de realizar de forma rápida y sencilla una aplicación en este lenguaje. Pydev es un plugin de terceros para Aptana Studio, utilizado para programar en Python basado en el apoyo de refactorización de código y depuración gráfica. Ciertas características avanzadas tales como análisis de código, arreglos rápidos, se reservan para la versión libre Pydev la cual se empleará en el desarrollo de la herramienta.

## 1.7 Conclusiones del capítulo

En el capítulo se realizó un estudio de las diferentes herramientas de generación de datos existentes actualmente, lo que permitió además tener un conocimiento de este proceso en general. Se demostraron los inconvenientes de estas herramientas para hacer uso de ellas en la generación de datos para bases de datos NoSQL. Se seleccionaron los gestores NoSQL MongoDB, CouchDB, Cassandra, Neo4j y Redis los cuales serán utilizados por el sistema para realizar la generación de los datos. Además se hizo una investigación sobre las herramientas de generación de datos para gestores NoSQL no encontrándose ninguna que realizara este proceso para este tipo de gestores. Se seleccionó la metodología de desarrollo de software XP, la cual guiará el proceso de construcción de la aplicación en todo su ciclo de vida. Se escogieron las herramientas y tecnologías para el desarrollo de la aplicación, UML 2.0 como lenguaje de modelado y Visual Paradigm como herramienta CASE. Para la implementación del sistema se seleccionó Python como lenguaje de programación y Aptana Studio como IDE de desarrollo.

# CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

## Introducción

En el presente capítulo se hace una descripción detallada de la propuesta de solución para el problema de la investigación identificado. Se determinan los requisitos no funcionales y las Historias de Usuario con los que deberá cumplir la aplicación desarrollada. Además se generan los artefactos que propone la metodología de desarrollo de software seleccionada. Se definen los patrones de arquitectura y diseño que se utilizarán en el desarrollo del software, así como la estructura de codificación a seguir en la implementación.

## 2.1 Descripción de la solución propuesta

Con el objetivo de dar solución al problema de la investigación identificado se propone el desarrollo de una herramienta que posibilite el poblado de las bases de datos de los gestores NoSQL. Esta herramienta brindará la posibilidad al usuario de seleccionar el gestor de base de datos NoSQL que desee poblar. Una vez seleccionado el gestor se le ofrecerá la posibilidad de escoger de qué forma se desea generar los datos, ya sea de manera aleatoria o utilizando datos almacenados en una fuente. Si selecciona la técnica aleatoria el usuario podrá configurar la forma en que se generarán los datos en dependencia de su tipo.

**Tipos de datos**

Las herramientas de generación de datos identificadas generan datos de diferentes formatos. En el caso de forSQL Data Generator, genera datos aleatorios, datos por formato, datos constantes, datos desde una lista y datos incrementales. Estos datos son creados usando los tipos de datos: números, cadenas y fechas. Spawner Data Generator permite crear aleatoriamente números, cadenas y fechas. De este estudio realizado se escogen como los tipos de datos a usar en la herramienta:

1- Números

- Enteros
- Reales

2- Cadenas

3- Fechas

4- Listas

Haciendo uso de estos tipos de datos la herramienta será capaz de crear los datos usados en cada gestor en específico. En el caso de los gestores documentales MongoDB y CouchDB el tipo de dato que se almacena es JSON *(JavaScript Object Notation)*, el gestor Cassandra utiliza como tipo de dato las columnas las cuales se insertan varias por cada fila, el gestor orientado a grafos Neo4j lo que almacena son nodos y las relaciones entre estos y el gestor Redis almacena su información con datos de tipo llave-valor donde a cada llave se le asocia un valor en específico. Todos estos tipos de datos específicos que almacenan los gestores NoSQL seleccionados serán creados por la herramienta mediante la técnica aleatoria haciendo uso de los tipos de datos mencionados anteriormente.

En el caso de la generación de los datos utilizando una fuente existente se realizará cargando los datos desde una fuente de texto plano. Esta fuente deberá contener el dato conciso que use el gestor especificado, almacenando un dato por línea. Los datos específicos de cada gestor son expuestos a continuación.

Para los gestores documentales utilizados por el sistema, MongoDB y CouchDB, el dato almacenado en el fichero de texto debe de estar en forma de diccionario, a continuación se muestra un ejemplo de cómo se encuentra estructurado este tipo de dato:

- {"Nombre”: “Eduarluis”, “Apellidos”: “Espinosa Gato”, “Edad”: 23}

En el caso del gestor Redis el dato debe de estar compuesto de una lista con dos elementos donde el primero es la llave y el segundo el valor, a continuación se muestra un ejemplo de este dato:

- [“Nombre”, “Eduarluis”]

Para el gestor Cassandra el tipo de dato que debe contener al fichero es un diccionario como el de los gestores documentales, pero en este caso este diccionario debe de contener una estructura determinada, la misma es mostrada a continuación:

- {'fila1': {'Nombre': 'Eduarluis', 'Apellidos': 'Espinosa Gato'}, 'fila2': {'Edad': ‘23’}}

El gestor Neo4j almacena nodos y relaciones entre estos, por lo que el fichero de texto puede contener dos tipos de datos diferentes, ya sea para insertar un solo nodo o para insertar un conjunto de nodos y sus relaciones. En caso de insertar nodos simplemente se usa el tipo de dato diccionario y para relacionar nodos se usan las tuplas, a continuación se muestra un ejemplo de cada caso:

- {"Nombre": "Eduarluis", "Apellidos": "Espinosa Gato", "Edad": 23}
- ({"Nombre": "Eduarluis"}, {"Nombre": "Oswaldo"}, (0, "Conoce", 1, {"desde": 2009}))

Utilizando estos tipos de datos descritos anteriormente la aplicación será capaz de insertarlos en la base de datos correspondiente. En caso de existir algún error en el formato de los datos almacenados en el fichero de texto el sistema almacenará en el log de errores del sistema el dato que se encuentra con error, e insertara los demás.

## 2.2 Modelo de dominio

El Modelo de Dominio (MD) es un artefacto que se genera en la fase de análisis de un proyecto, este se crea con el fin de representar el vocabulario y los conceptos clave del dominio del problema. El modelo de dominio también identifica las relaciones entre todas las entidades comprendidas en el ámbito del dominio del problema, y comúnmente identifica sus atributos. La metodología XP no genera precisamente este artefacto, pero se decidió hacerlo porque mediante el mismo se agiliza el proceso de entendimiento de las clases conceptuales, de la forma más fácil para aquellas personas que interactúen con la aplicación.

La figura 1 muestra el modelo de dominio de la aplicación, donde el usuario que realiza pruebas al software accede al gestor de base de datos, para poder poblar manualmente las bases de datos NoSQL utilizadas por el software. De esta manera puede realizar las pruebas de software una vez que tenga la base de datos poblada.

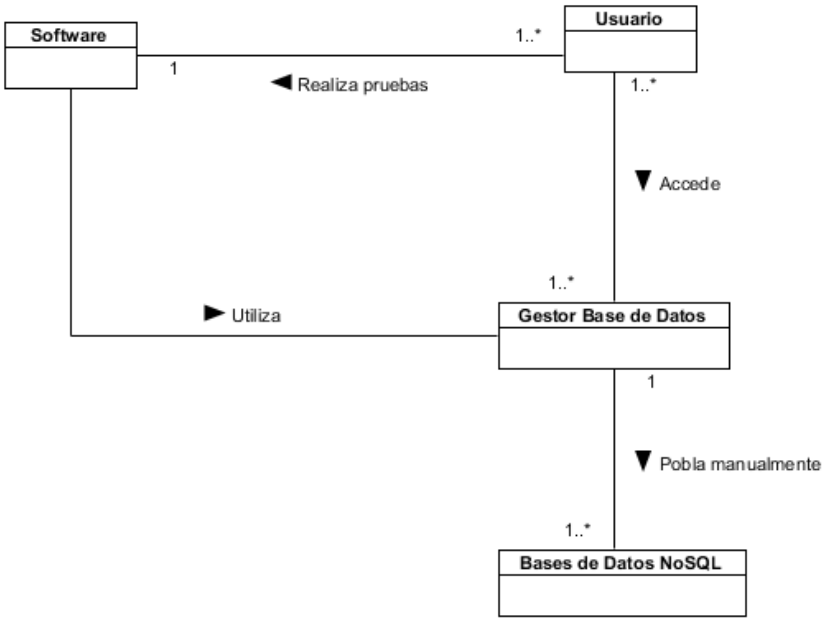



**Figura 1. Modelo de Dominio.**

Los conceptos involucrados en el dominio se encuentran descritos en la tabla 1.

**Tabla 1. Descripción de los conceptos del modelo de dominio.**

| Concepto | Descripción |
|---|---|
| Usuario | Persona que realiza las pruebas de software. |
| Software | Aplicación a la cual el usuario le realiza las pruebas. |
| Gestor de Bases de Datos | Herramienta mediante la cual el usuario se conecta a la base de datos para insertar manualmente la información. |
| Bases de Datos NoSQL | Base de datos donde se encuentran almacenados los datos usados por el software para su correcto funcionamiento. |

## 2.3 Historias de usuarios

Las Historias de Usuario (HU) representan una breve descripción del comportamiento del sistema, estas no son descritas en lenguaje técnico sino que emplean terminología del cliente. Se realiza una HU por cada característica principal del sistema.

Son la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [14].

Luego del análisis realizado con el objetivo de determinar los requisitos funcionales que corresponden a la aplicación que se necesita desarrollar, se determinaron 14 Historias de Usuario las cuales fueron documentadas. A continuación se muestran algunas de las más importantes, el resto se encuentran en el expediente de proyecto en la “Planilla de Historias de Usuario”.

**Tabla 2. HU Conectar con el gestor de bases de datos MongoDB.**

| **Historia de Usuario** | |
|---|---|
| **Numero: 1** | **Nombre de la Historia de Usuario:** Conectar con el gestor de bases de datos MongoDB. |
| **Cantidad de modificaciones a la Historia de Usuario: 1** | |
| **Usuario: Eduarluis Espinosa Gato** | **Iteración asignada:** 1ra iteración |
| **Prioridad en negocio: Muy Alta** | **Puntos estimados: 0.6 semanas** |
| **Riesgo en desarrollo: Muy Alto** | **Puntos reales: 0.6 semanas** |
| **Descripción: El usuario deberá seleccionar el gestor MongoDB y especificar los parámetros de conexión requeridos para que el sistema se conecte.** | |

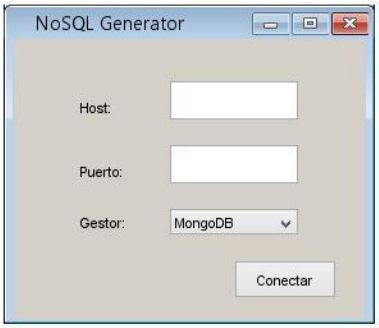
| |
|---|
| **Observaciones:** |
| **Prototipo de interfaces:**   |

**Tabla 3. HU Configurar parámetros de generación de datos.**

| **Historia de Usuario** | |
|---|---|
| **Numero: 7** | **Nombre de la Historia de Usuario:** Configurar parámetros de generación de datos. |
| **Cantidad de modificaciones a la Historia de Usuario: 1** | |
| **Usuario: Eduarluis Espinosa Gato** | **Iteración asignada:** 2da iteración |
| **Prioridad en negocio: Alta** | **Puntos estimados: 1.6 semanas** |
| **Riesgo en desarrollo: Alto** | **Puntos reales: 1.6 semanas** |
| **Descripción: El sistema debe permitir la configuración de los datos que van a ser generados especificando el tipo de dato y los rangos de cada dato.** | |
| **Observaciones:** | |
| **Prototipo de interfaces:** | |


## 2.4 Lista de Reserva del Producto

La Lista de Reserva del Producto (LRP) es uno de los artefactos que genera la metodología XP, en el cual se recogen los requisitos funcionales y no funcionales del sistema. Las funcionalidades son ordenadas descendentemente por prioridad de implementación: Muy Alta, Alta, Media y Baja. Los requisitos no funcionales son agrupados a continuación. En la misma también se especifica la estimación en semanas necesaria para la realización de cada uno de los requisitos funcionales, así como la persona que realiza esta estimación.

La tabla 4 muestra los requisitos funcionales y no funcionales con los que debe cumplir el sistema. Estos requisitos son los que permitirán contar con una herramienta capaz de realizar la generación de datos para las bases de datos NoSQL, además de permitirle al usuario realizar las configuraciones necesarias para la realización de este proceso.

**Tabla 4. Lista de reserva del producto.**

| Ítem * | Descripción | Estimación (semanas) | Estimado por |
|---|---|---|---|
| **Prioridad: Muy Alta** | | | |
| 1 | Conectar con el gestor de bases de datos MongoDB. | 0.6 | Analista |
| 2 | Conectar con el gestor de bases de | 0.6 | Analista |

| | | | |
|---|---|---|---|
| | datos CouchDB. | | |
| 3 | Conectar con el gestor de bases de datos Redis. | 0.6 | Analista |
| 4 | Conectar con el gestor de bases de datos Cassandra. | 0.6 | Analista |
| 5 | Conectar con el gestor de bases de datos Neo4j. | 0.6 | Analista |
| 6 | Realizar la autenticación de cada uno de los gestores seleccionados. | 0.8 | Analista |
| **Prioridad: Alta** | | | |
| 7 | Configurar parámetros de generación de datos. | 1.6 | Analista |
| 8 | Generar datos para el gestor de bases de datos MongoDB. | 0.4 | Analista |
| 9 | Generar datos para el gestor de bases de datos CouchDB. | 0.4 | Analista |
| 10 | Generar datos para el gestor de bases de datos Cassandra. | 0.4 | Analista |
| 11 | Generar datos para el gestor de bases de datos Redis. | 0.4 | Analista |
| 12 | Generar datos para el gestor de bases de datos Neo4j. | 0.4 | Analista |
| **Prioridad: Media** | | | |
| 13 | Obtener datos desde una fuente. | 0.8 | Analista |
| 14 | Generar reporte de errores | 0.4 | Analista |

| Requisitos no funcionales | | |
|---|---|---|
| 1 | **Software:** Sistema Operativo: Multiplataforma. Lenguaje: Python. Servidor de Bases de datos: MongoDB, CouchDB, Redis, Cassandra, Neo4j. Bibliotecas de Python: pymongo, python-redis, pycassa, py2neo, python-couchdbkit. | |
| 2 | **Apariencia o interfaz externa**: La aplicación debe contener una interfaz que resulte intuitiva, amigable y fácil de aprender y utilizar por parte de los usuarios que hagan uso de la misma. Debe ser compatible con resoluciones 1024x768 o 800x600 píxeles respectivamente. | |
| 3 | **Portabilidad:** La aplicación será multiplataforma, permitiendo la disponibilidad del mismo en cualquier sistema operativo. | |
| 4 | **Hardware:** Se necesita 1 MB de memoria RAM como mínimo, 4GB de espacio libre en el disco duro y el micro a 1.6 GHz. | |

## 2.5 Modelo de diseño

La metodología XP debido a su enfoque ágil hace especial énfasis en diseños simples, debido a que los diseños simples son implementados más fácilmente y con mayor agilidad. Con el diseño se quiere modelar una solución que pueda ser transformada en código fuente y construir un sistema fuerte, simple y extensible.

### *2.5.1 Tarjetas Clase-Responsabilidades-Colaboración (CRC)*

Las tarjetas CRC son otra herramienta con la cual es posible asignar responsabilidades e indicar una colaboración con otros objetos. Las tarjetas CRC son tarjetas de índices, una por cada clase, sobre las cuales se abrevian las responsabilidades de la clase y se anota una lista de los objetos con los que colaboran para desempeñarlas [17].

El uso de las tarjetas CRC permite al programador centrarse y apreciar el desarrollo orientado a objetos. Durante el proceso de desarrollo de la aplicación se elaboraron un total de 38 tarjeras CRC. A continuación se muestran algunas referentes a las clases más importantes de la aplicación.

**Tabla 5. Tarjeta CRC para la clase Principal.**

| **Tarjeta CRC** | |
|---|---|
| **Clase: Principal** | |
| **Responsabilidades** | **Colaboraciones** |
| *Clase visual que se encarga de conectar al sistema con el gestor de base de datos seleccionado por el usuario.* | *Progress*<br>*ConfigCouch*<br>*ConfigNeo*<br>*ConfigCassandra*<br>*ConfigRedis*<br>*ConfigMongo*<br>*Controladora_CouchDB*<br>*Controladora_Redis*<br>*Controladora_MongoDB*<br>*Controladora_Neo4j*<br>*Controladora_Cassandra* |

**Tabla 6. Tarjeta CRC para la clase Controladora.**

| **Tarjeta CRC** | |
|---|---|
| **Clase: Controladora** | |
| **Responsabilidades** | **Colaboraciones** |

| | |
|---|---|
| *Clase genérica que contiene los métodos para llevar a cabo las funciones principales del sistema, y además es la que controla todos los procesos en la aplicación.* | *NoSQLFactory*<br>*Controladora_CouchDB*<br>*Controladora_Redis*<br>*Controladora_MongoDB*<br>*Controladora_Neo4j*<br>*Controladora_Cassandra*<br>*Generador* |

### *2.5.2 Diagrama de Clases*

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces (las de Java, por ejemplo) en una aplicación. Normalmente este diagrama contiene información referente a las clases, asociaciones, atributos y métodos. El diagrama de clases contiene las definiciones de las entidades del software [17].

Este artefacto no es propio de la metodología XP, si lo es de las metodologías robustas como RUP *(Rational Unified Process)*. Se decide incorporarlo para mostrar más detalladamente y de una manera entendible la solución del problema de la investigación. En la figura 2 se muestra el diagrama de clases en el cual se muestran las clases utilizadas para la implementación de la solución, así como las relaciones existentes entre ellas. En el diagrama de clases se describen las clases utilizadas para llevar a cabo la implementación del sistema, se cuenta con un total de 38 clases. En él se ilustran las clases controladoras y visuales utilizadas en el sistema, así como las clases encargadas de la generación de los datos y del trabajo con la conexión con los gestores NoSQL.

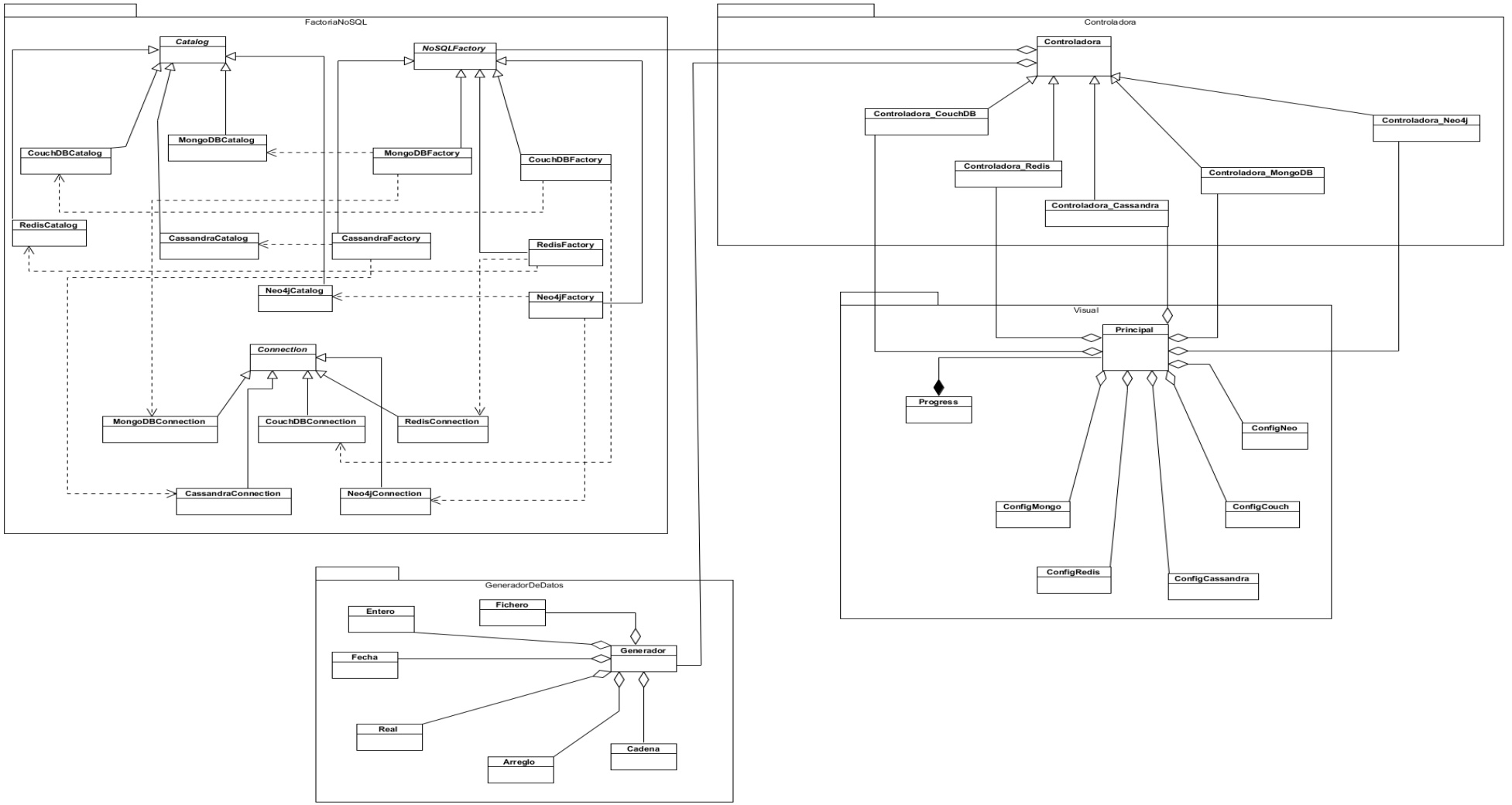



**Figura 2. Diagrama de clases.**

## 2.6 Patrón Arquitectónico

Se entiende por Patrón Arquitectónico a la descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma como estos colaboran entre sí [18].

**Patrón Modelo Vista Controlador (MVC)**

El patrón arquitectónico MVC divide la aplicación interactiva en tres componentes. El modelo *(model)* el cual contiene la información central y los datos. Las vistas *(view)* que despliegan información al usuario. Los controladores *(controlers)* que son los que capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario [18].

MVC divide las aplicaciones en tres niveles de abstracción:

**Modelo**

Representa la lógica de negocios. Es el encargado de acceder de forma directa a los datos actuando como “intermediario” con la base de datos.

**Vista**

Es la encargada de mostrar la información al usuario de forma gráfica y “humanamente legible”.

**Controlador**

Es el intermediario entre la vista y el modelo. Es quien controla las interacciones del usuario solicitando los datos al modelo y entregándolos a la vista para que ésta, lo presente al usuario, de forma “humanamente legible” [19].

La figura 3 muestra cómo se evidencia el patrón MVC en la aplicación. Donde dentro de las clases asociadas al modelo se encuentran las usadas para el trabajo con los gestores de bases de datos, así como las usadas para la generación de los datos. En la vista se encuentran implementadas las clases visuales encargadas de permitirle al usuario la realización de las configuraciones para la generación de los

datos. En el controlador se encuentran las clases controladoras específicas de cada gestor NoSQL usados en la implementación, las cuales se encargan de realizar todo el proceso de generación e inserción de los datos en las bases de datos usando las configuraciones especificadas en las clases visuales y los datos generados por las clases ubicadas en el modelo.

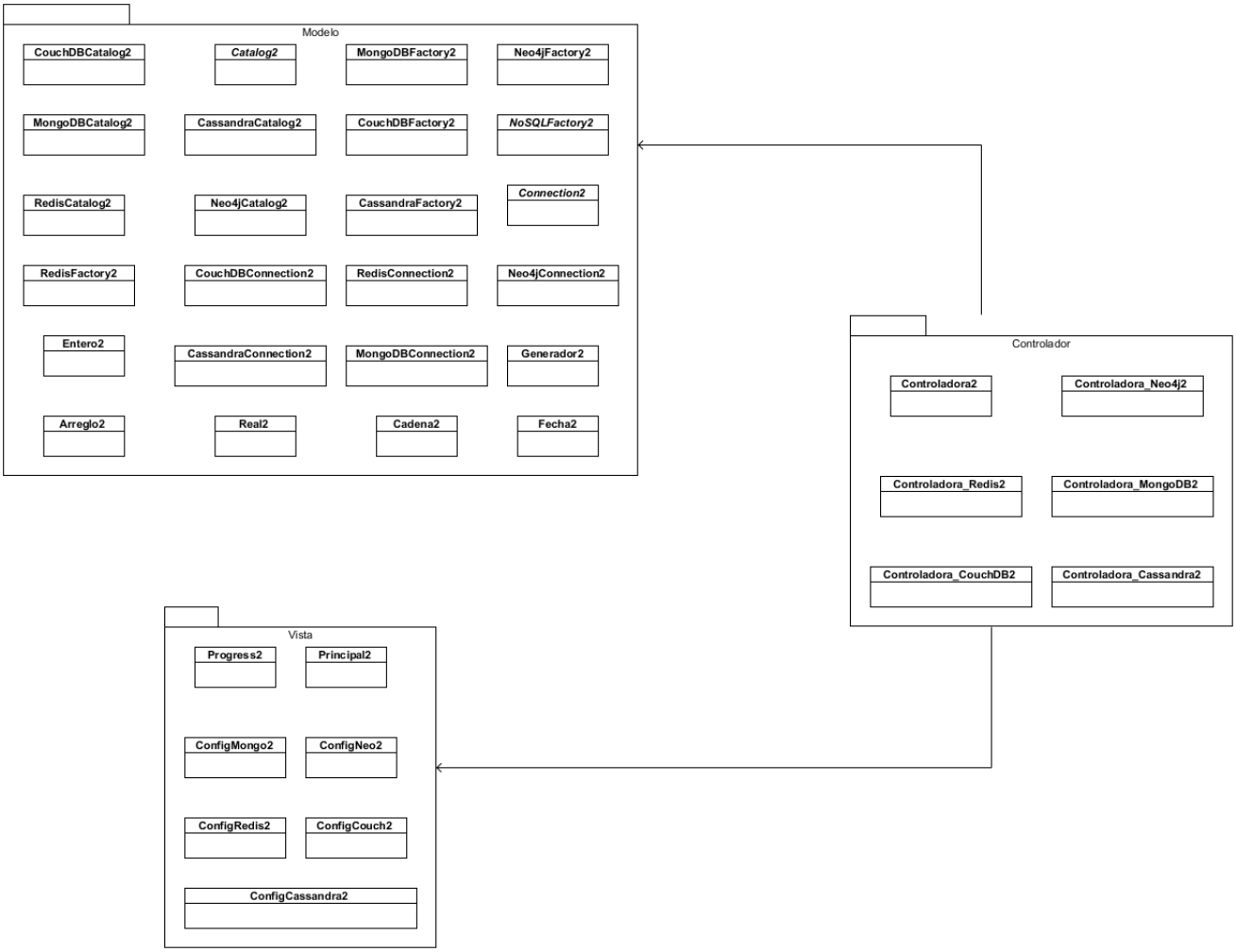



**Figura 3. Modelo-Vista-Controlador.**

## 2.7 Patrones de diseño

Un patrón de diseño provee un esquema para refinar los subsistemas o 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. Son menores en escala que los patrones arquitectónicos, y tienden a ser independientes de los lenguajes y paradigmas de programación. Su aplicación no tiene efectos en la estructura fundamental del sistema, pero sí sobre la de un subsistema [18].

### Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones [17]**.**

A continuación se describen los patrones GRASP que se usaron en el desarrollo de la aplicación:

**Bajo acoplamiento:** Se deben asignar las responsabilidades de forma tal que cada clase se comunique con el menor número de clases, minimizando el nivel de dependencia.

**Alta cohesión:** Asignar a las clases responsabilidades que trabajen sobre una misma área de la aplicación y que no tengan mucha complejidad, evitando así que una clase sea la única responsable de muchas tareas en áreas funcionales muy heterogéneas.

Ambos patrones se pueden apreciar en la aplicación en las clases Generadoras de datos, donde existe una clase que genera cada tipo de dato (Fecha, Real, Entero, Cadena, Arreglo) aplicando el patrón alta cohesión. Dichas clases no se relacionan entre sí, sino que se relacionan directamente con la clase Generador, poniendo en práctica el patrón bajo acoplamiento.

**Experto:** Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para realizar la tarea. Este patrón se evidencia en las clases generadoras de datos las cuales son las encargadas de generar cada tipo de dato.

**Creador:** Este patrón ayuda a responder la pregunta de qué clase debe ser la responsable de crear nuevos objetos (instancias de alguna clase). Este patrón se pone de manifiesto en la aplicación en la clase Principal la cual crea objetos de las clases visuales de configuración de cada gestor en específico.

**Controlador:** Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. En la aplicación se evidencia este patrón de diseño en las clases Controladora y sus hijas las cuales se encargan de controlar el flujo de información dentro del sistema.

### Patrones GOF

**Fábrica Abstracta / Abstract Factory:** Proporciona una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin especificar sus clases concretas [20]. Este patrón se evidencia en la solución en las clases que realizan el trabajo con los gestores de bases de datos donde se crea una fábrica *(Factory)* que es la encargada de crear y devolver los objetos de las clases de conexión y catálogo de cada gestor en específico.

## 2.8 Estándares de Codificación

Al ser XP una metodología ágil los programadores se enfocan en un módulo del sistema por separado. Para que exista una buena comunicación entre los programadores, y para hacer más rápida la realización del software, es necesario establecer un estándar de codificación para que el código sea entendible por todos. Los estándares de programación mantienen el código legible para los miembros del equipo, facilitando los cambios.

El estándar de codificación que se utiliza es el definido por el lenguaje de programación Python, debido a que este lenguaje tiene su propio estilo ante los demás lenguajes. El estándar de codificación que se utiliza es el definido a continuación [21].

**Indentación:** En la aplicación se usa 4 espacios por cada nivel de indentación.

**Ej.**
```
def fichero(self):
    if self.form.fichero.isChecked():
        self.form.cant_doc.setEnabled(False)
        self.form.frame_4.setEnabled(False)
```

```
        self.form.frame.setEnabled(False)
    else:
        self.form.cant_doc.setEnabled(True)
        self.form.frame_4.setEnabled(True)
        self.form.frame.setEnabled(True)
```

**Tamaño máximo de línea:** Limita todas las líneas a un máximo de 79 caracteres. La forma de dividir líneas largas es utilizando la característica de Python de continuar las líneas de forma implícita dentro de paréntesis, corchetes y llaves.

**Ej.**
```
self.rel_nodo.insert(self.form.listarelacion.currentRow(),
                     [str(self.form.listarelacion.selectedItems()[0].text()),
                      str(self.form.listarelacion.selectedItems()[1].text())])
```

**Imports:** Los imports son colocados en distintas líneas. Se colocan siempre en la parte superior del archivo, justo después de cualquier comentario o cadena de documentación del módulo, y antes de las variables globales y las constantes del módulo.

**Ej.**
```
import sys
from Controladora import Controladora
```

**Comentarios:** Existen dos tipos de comentario en Python, el comentario de una línea y el de bloque. El comentario de línea generalmente se aplica al código que se encuentra a continuación, y se indenta al mismo nivel que dicho código. Cada línea de un comentario de bloque comienza con un # y un único espacio.

**Ej.**
```
self.dicc_nodo = {}  # Guarda los nodos y sus propiedades
```

Los comentarios de bloque se realizan encerrando los párrafos entre tres comillas simples.

**Ej.**
```
‘’’ este es el comentario de un bloque ‘’’
```

**Nombre de Clases:** Casi sin excepciones, los nombres de clases usan la convención CapWords. Las clases de uso interno tienen además un guion bajo al principio del nombre.

**Ej.** class ConfigNeo:

**Nombre de Funciones:** Los nombres de funciones se escriben en letra minúsculas, con palabras separadas mediante guiones bajos según sea necesario para mejorar la legibilidad y a partir de la segunda palabra se comienza con mayúscula.

**Ej.** def add_Relacion(self):

**Nombre de variables:** Las variables son declaradas con letra minúscula y de ser necesario con un guion bajo para dividir las palabras.

**Ej.** cant = 0

## 2.9 Conclusiones del capítulo

En el capítulo se analizaron los procesos fundamentales que están relacionados con la solución informática que se debe desarrollar. Se describió la solución propuesta para el problema de la investigación. Se identificaron 14 HU y los requisitos no funcionales, elementos fundamentales para la creación de la aplicación. Fueron realizados la lista de reserva del producto y las tarjetas CRC[2] artefactos requeridos por la metodología para lograr un mejor entendimiento de la solución, así como otros artefactos que facilitan el entendimiento del desarrollo de la aplicación: el diagrama de dominio y el diagrama de clases. Se seleccionó el patrón de diseño modelo vista controlador para hacer uso de este en la implementación del sistema. El uso de patrones, ya sean de arquitectura o de diseño, en la aplicación, permiten que la implementación del software sea de manera ágil y sencilla.

[2] Tarjetas de Clase Responsabilidad Colaboración.

# CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

## Introducción

Una vez realizado el diseño de la solución se procede a la implementación y validación del sistema. Este capítulo se centra en mostrar el resultado obtenido en el desarrollo de la aplicación y en el diseño y aplicación de las pruebas al sistema.

## 3.1 Tareas de la Ingeniería

El equipo de desarrollo tiene la responsabilidad de revisar, analizar y evaluar cada una de las funcionalidades o HU correspondientes a la aplicación que se desea elaborar. Esta tarea es realizada con el objetivo de dividir cada una de las HU en tareas más pequeñas, que de conjunto tributen al correcto desempeño de la funcionalidad.

Las tareas de ingeniería (TI) describen las tareas realizadas sobre el proyecto. Cada tarea está relacionada a una HU, pero una HU puede ser dividida en varias tareas de ingeniería. En estas se especifica la fecha de inicio y fin de la tarea y el programador responsable de cumplirla así como la estimación en semanas para darle cumplimiento a la misma. En la tabla 5 se muestran las tareas de ingeniería 1 y 6 correspondientes a las HU número 1 y 6 respectivamente. Para la aplicación que se desea desarrollar se determinaron un total de 16 TI. Aunque solo se exponen dos de ellas, las restantes pueden ser consultadas en el expediente de proyecto.

**Tabla 7. TI Implementar la conexión con el gestor MongoDB.**

| **Tarea de Ingeniería** | |
|---|---|
| **Número Tarea:1** | **Número Historia de Usuario: 1** |
| **Nombre Tarea:** Implementar la conexión con el gestor MongoDB. | |
| **Tipo de Tarea :** Desarrollo | **Puntos Estimados:** 0.6 semanas |
| **Fecha Inicio:** 3/2/2014 | **Fecha Fin:** 6/2/2014 |

| |
|---|
| **Programador Responsable:** Eduarluis Espinosa Gato |
| **Descripción:** Esta clase va a permitir la conexión con el gestor MongoDB. |

**Tabla 8. TI Diseñar la interfaz de autenticación para los gestores NoSQL.**

| **Tarea de Ingeniería** | |
|---|---|
| **Número Tarea: 6** | **Número Historia de Usuario: 6** |
| **Nombre Tarea:** Diseñar la interfaz de autenticación para los gestores NoSQL. | |
| **Tipo de Tarea :** Desarrollo | **Puntos Estimados:** 0.2 semanas |
| **Fecha Inicio:** 6/3/2014 | **Fecha Fin:** 7/3/2014 |
| **Programador Responsable:** Eduarluis Espinosa Gato | |
| **Descripción:** Interfaz que permite la autenticación en los gestores NOSQL. | |

## 3.2 Plan de iteraciones

Este plan es utilizado por los desarrolladores para planificar el tiempo idóneo para la implementación de cada una de las HU. Este tiempo es acordado en estrecha relación entre el cliente y el desarrollador. Se realizan varias iteraciones antes de ser entregado el sistema. Cada iteración no debe de sobrepasar las tres semanas. Es en la primera iteración donde se trata de complementar una arquitectura del sistema para poder trabajar en base a esta durante el desarrollo del proyecto. Esto se logra escogiendo las historias de mayor prioridad las cuales van a influir en la creación de esta arquitectura. Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable [22].

La tabla 7 muestra el plan de iteraciones acordado entre el cliente y el equipo de desarrollo. En él se muestran las HU a desarrollar en cada una de las iteraciones y además el tiempo estimado para la finalización de la iteración. Esta distribución de las HU por iteraciones permite que al concluir ese tiempo se cuente con la primera versión funcional de la aplicación.

**Tabla 9. Plan de iteraciones.**

| No | Descripción de la iteración | Historias a implementar | Duración total |
|---|---|---|---|
| 1 | En esta iteración se implementaran las Historias de Usuario con prioridad muy alta ya que son las que mayor incidencia tienen en la aplicación. | HU #1<br>HU #2<br>HU #3<br>HU #4<br>HU #5<br>HU #6 | 4 semanas |
| 2 | En la 2da iteración serán desarrolladas las Historias de Usuario con prioridad alta ya que son las que complementan a las anteriores para tener una arquitectura funcional de la aplicación. | HU #7<br>HU #8<br>HU #9<br>HU #10<br>HU #11<br>HU #12 | 3.6 semanas |
| 3 | En la 3ra iteración serán implementadas las Historias de Usuario de menor importancia para la aplicación, estas son de prioridad media. | HU #13<br>HU #14 | 1.2 semanas |

## 3.3 Interfaces de la aplicación

Luego de concluida la implementación de las funcionalidades identificadas se obtuvo como resultado la herramienta *"NoSQL Generator"*. Para hacer uso de la aplicación el usuario debe de especificar los parámetros necesarios para la conexión con el gestor de bases de datos. La figura 4 muestra la interfaz principal de la aplicación donde el usuario especifica estos parámetros mencionados anteriormente.

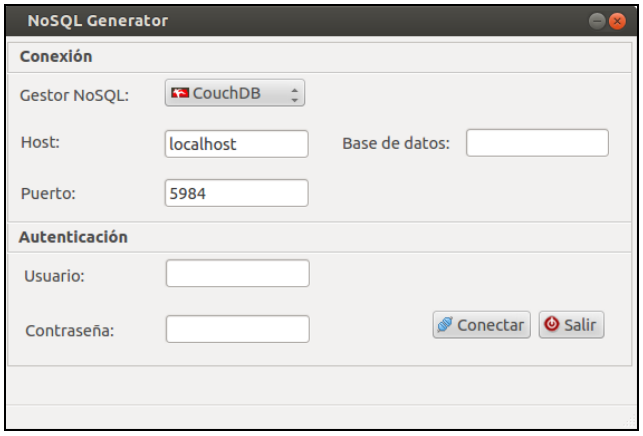



**Figura 4. Interfaz Principal.**

Una vez especificado estos parámetros se procede a realizar la conexión, de no existir ningún error en los parámetros introducidos el sistema muestra la interfaz de configuración para la generación de los datos específica del gestor seleccionado. Esta interfaz permite configurar al usuario la manera en que van a ser insertados los datos, ya sea desde una fuente con datos existentes o de manera aleatoria. El usuario puede además configurar los campos y los tipos de datos de cada uno para conformar el dato que va a ser insertado. En la figura 5 se muestra la interfaz de configuración para el gestor CouchDB.

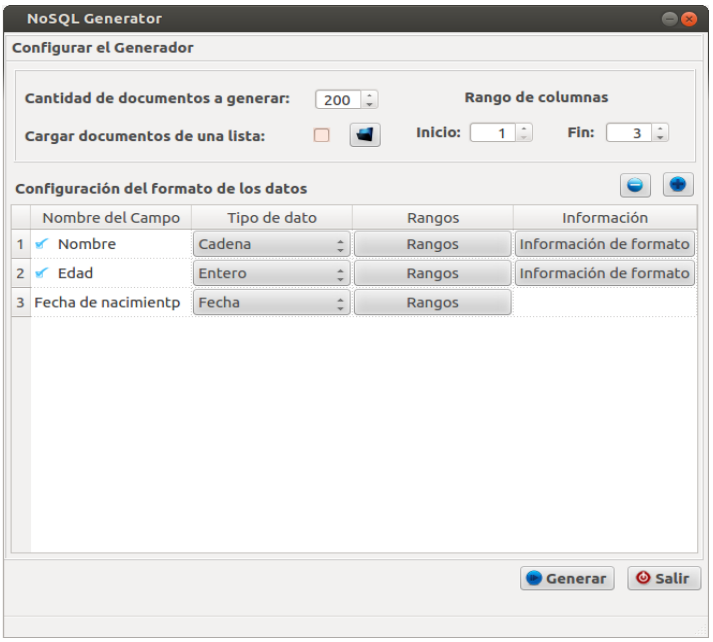



**Figura 5. Interfaz de configuración para el gestor CouchDB.**

Una vez que el usuario configure todos los parámetros necesarios para poder realizar la generación de los datos, el sistema realiza todo este proceso de creación e inserción de estos y una vez concluido se le informa al usuario que la operación concluyó con éxito dándosele la posibilidad de salir de la aplicación o de generar más datos en las bases de datos del gestor tal y como se muestra en la figura 6, de existir algún error en el proceso se le informará al usuario del error ocurrido.

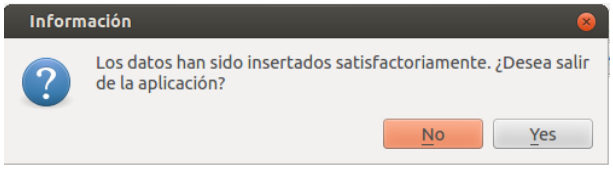



**Figura 6. Mensaje de fin del proceso de generación.**

## 3.4 Resultados de la generación de datos

Con el objetivo de demostrar los resultados obtenidos por la aplicación una vez realizada la generación de los datos para los diferentes gestores de bases de datos seleccionados, se exponen a continuación los resultados obtenidos en la generación de datos para cada uno de los gestores seleccionados. En la figura 7 se muestra la ventana de configuración para el gestor CouchDB en la cual se puede apreciar los

nombres de los campos y sus respectivos tipos de datos, estos son especificados por el usuario a su consideración. Por cada campo se puede especificar los rangos correspondientes a cada tipo de dato, en este caso se muestra el diálogo de configuración de rangos del campo *"Fecha de nacimiento"* cuyo tipo de dato es fecha.

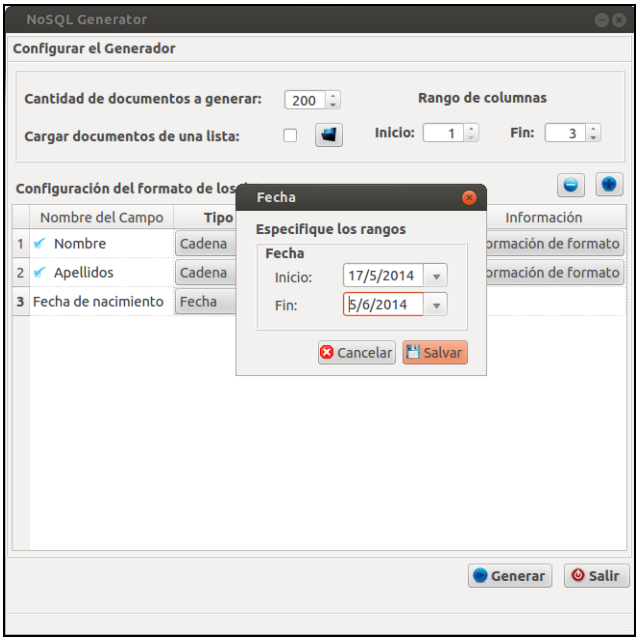
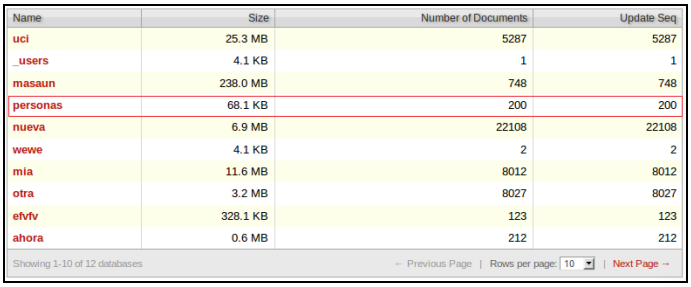



**Figura 7. Configuración para el gestor CouchDB.**

En este caso se indicó que se insertaran doscientos datos dentro del gestor CouchDB, estos datos se insertarán en la base de datos *"personas"* la cual fue especificada en la interfaz principal de la aplicación. Una vez concluida la configuración para la generación de los datos se procede a realizar el proceso. El sistema captura las configuraciones, genera los datos y los inserta en la base de datos. La figura 8 muestra una vista de las bases de datos almacenadas en el gestor CouchDB donde se puede apreciar la base de datos personas con los doscientos datos almacenados.



| Name | Size | Number of Documents | Update Seq |
|---|---|---|---|
| uci | 25.3 MB | 5287 | 5287 |
| _users | 4.1 KB | 1 | 1 |
| masaun | 238.0 MB | 748 | 748 |
| personas | 68.1 KB | 200 | 200 |
| nueva | 6.9 MB | 22108 | 22108 |
| wewe | 4.1 KB | 2 | 2 |
| mia | 11.6 MB | 8012 | 8012 |
| otra | 3.2 MB | 8027 | 8027 |
| efvfv | 328.1 KB | 123 | 123 |
| ahora | 0.6 MB | 212 | 212 |

Showing 1-10 of 12 databases ← Previous Page | Rows per page: 10 | Next Page →

**Figura 8. Lista de las bases de datos de CouchDB.**

En la figura 9 se muestra la interfaz de configuración para el gestor Neo4j en la cual, además de permitir realizar las mismas configuraciones de datos posibles en la configuración de CouchDB, se le agregan las relaciones entre los nodos, esto es debido a que el gestor Neo4j es un gestor orientado a grafos y el cual permite almacenar nodos y relaciones.

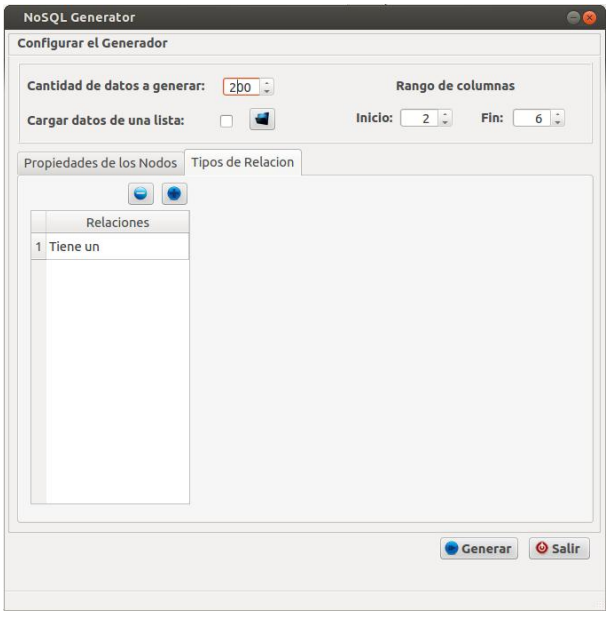



**Figura 9. Configuración para el gestor Neo4j.**

La comprobación de la inserción de los datos en el gestor Neo4j se puede hacer de dos formas, una es visualizando el grafo creado luego haber insertados los nodos con sus relaciones. Otra vía es observar la cantidad de nodos almacenados en el gestor antes de realizado el proceso de generación de los datos y después de realizado el mismo. En este caso se muestra en la figura 10 el grafo generado luego de haber insertado los datos.

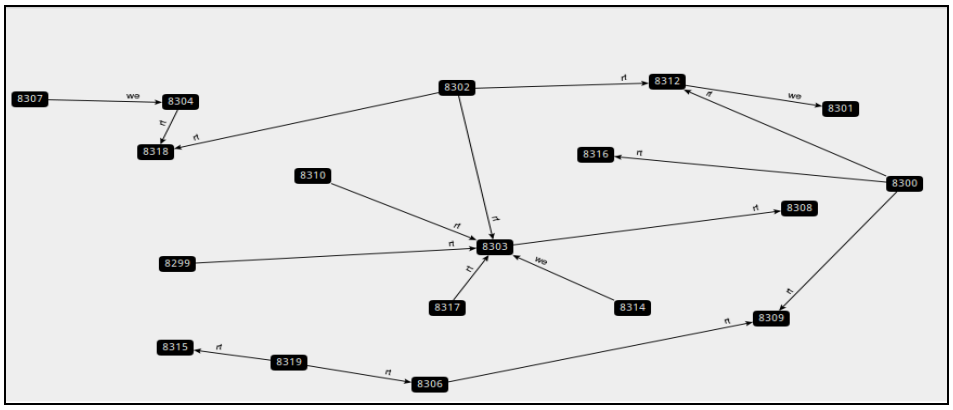



**Figura 10. Grafo generado con los datos insertados.**

En el caso específico de este ejemplo se generaron 20 nodos al gestor. Este procedimiento permitió comprobar, en conjunto con el otro método mencionado, que los nodos fueron insertados en el gestor sin inconvenientes y con los parámetros especificados en la configuración.

Para el caso de los gestores restantes no cuentan con un administrador de bases de datos visual por lo que se hizo necesario mostrar los datos insertados en consola. La figura 11 muestra la configuración realizada para probar la generación de los datos para el gestor MongoDB.

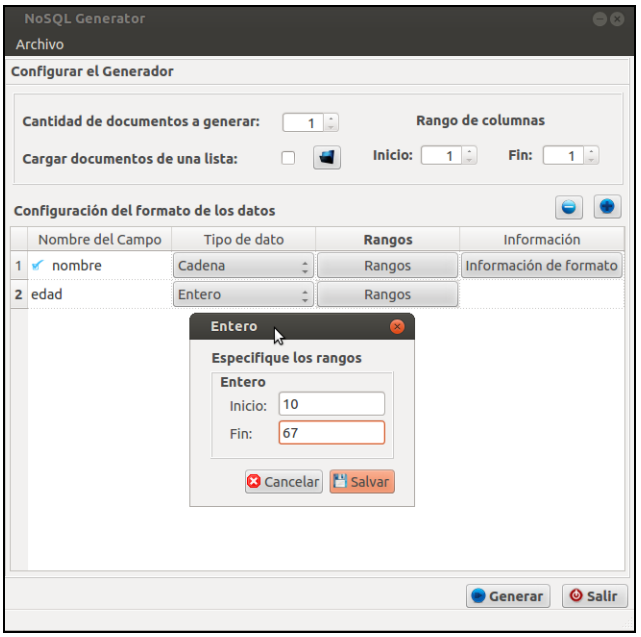



**Figura 11. Interfaz de configuración para MongoDB.**

Para este gestor se conformaron los datos utilizando las columnas *"nombre"* y *"edad"* mostrándose en este caso la ventana de configuración de los rangos para el tipo de dato *"Entero"*. En aras de lograr una mejor visualización de los datos en la consola solo se generaron 10 datos para insertarlos. En la figura 12 se muestran los datos almacenados en la base de datos y la colección especificada en la interfaz principal. Esta consulta realizada en la base de datos permitió comprobar que los datos fueron insertados correctamente.

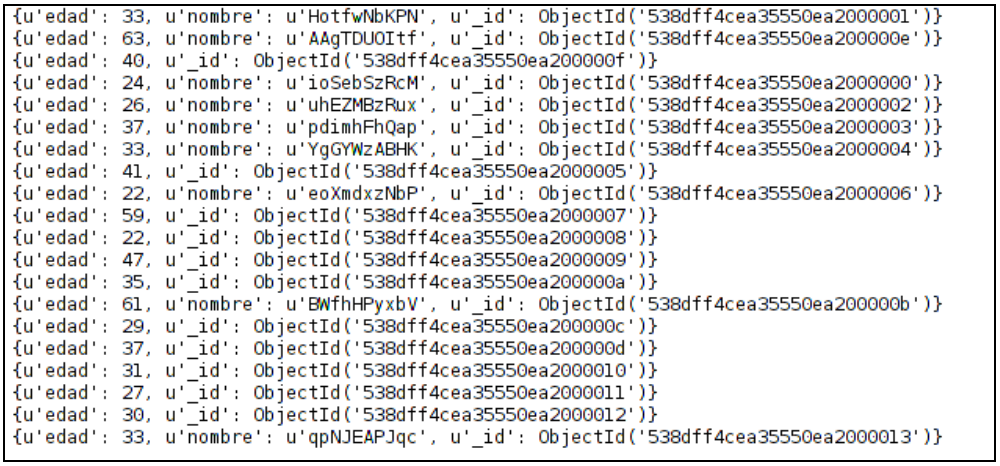



**Figura 12. Datos almacenados en la base de datos de MongoDB.**

Se muestra a continuación en la figura 13 la interfaz de configuración para el gestor Redis. En este caso se insertaron 20 datos utilizando la llave *"nombre"* la cual fue modificada por el sistema para poder insertar la cantidad de datos especificada, debido a que de mantenerse el mismo nombre solo se insertaría un solo dato en la base de datos. En esta interfaz se muestra la configuración de tipo de dato *"Cadena"* especificándose en este caso que las cadenas a almacenar contaran de una longitud de 10 caracteres.

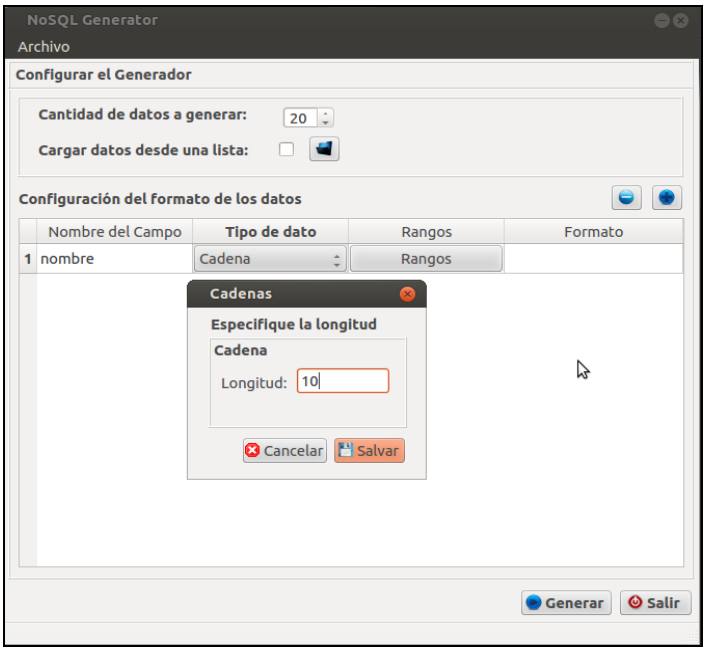



**Figura 13. Interfaz de configuración para el gestor Redis.**

Los datos insertados en la base de datos son mostrados en la figura 14, en esta se puede apreciar el cambio de nombre realizado para poder insertar todos los datos especificados.

```
nombre0: WWvBdUKBsr
nombre1: NaTmcPVZSV
nombre2: BBqSWwxyAE
nombre3: navzhdNgax
nombre4: RVREYlosBL
nombre5: wIfGuLrSYY
nombre6: lwgGPvZBIW
nombre7: TcVHLYnhiG
nombre8: SjovHDPgCf
nombre9: OdLbvthqaB
```

**Figura 14. Datos almacenados en la base de datos de Redis.**

Por último se muestra la generación de datos realizada para el gestor Cassandra, mostrándose en la figura 15 la configuración realizada para este gestor. Para este gestor serán insertados 20 datos para poder tener una mejor visualización de estos en la consola. Se especificaron para almacenar en cada fila insertada en la base de datos de Cassandra las columnas *"nombre"* y *"fecha de nacimiento"* con tipos de datos *"cadena"* y *"fecha"* respectivamente.

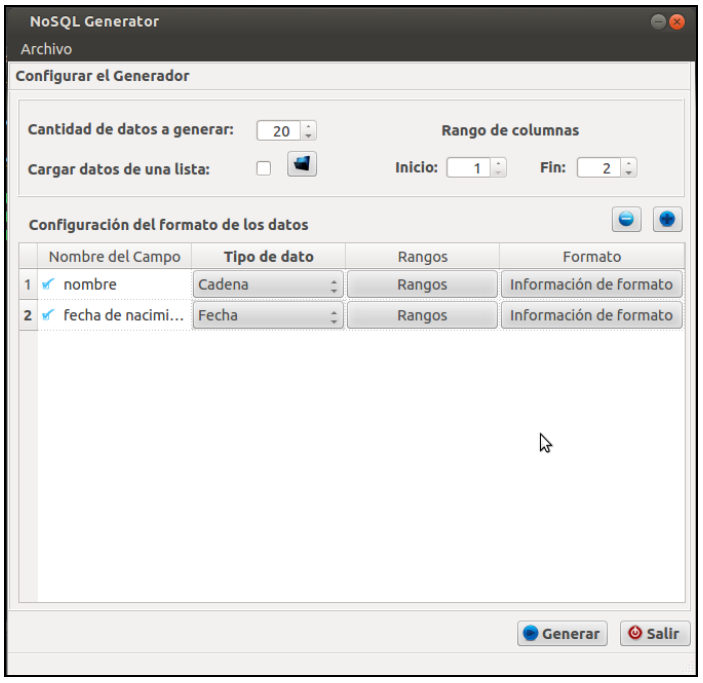



**Figura 15. Interfaz de configuración para el gestor Cassandra.**

En la figura 16 se muestran los resultados de la generación de datos realizada para el gestor Cassandra, donde se evidencia la inserción de las columnas especificadas en la configuración.

```
row0: OrderedDict([('dos', 'dqz'), ('fecha de nacimiento', '24/03/2014'), ('nombre', 'ruLfqYnxgQ')])
row1: OrderedDict([('dos', 'ZrM'), ('fecha de nacimiento', '15/12/2014'), ('name1', 'val3'), ('name2'
row2: OrderedDict([('dos', 'pva'), ('fecha de nacimiento', '21/03/2015'), ('fo34o', 'b5ar'), ('foo',
row3: OrderedDict([('dos', 'Gyc'), ('fecha de nacimiento', '12/10/2014'), ('nombre', 'mcnVzUloQW')])
row4: OrderedDict([('dos', 'iZH'), ('fecha de nacimiento', '30/04/2015'), ('nombre', 'MfSflnRXJu')])
row5: OrderedDict([('dos', 'MGr'), ('fecha de nacimiento', '20/04/2014'), ('nombre', 'hSodYgoQRc')])
row6: OrderedDict([('dos', 'rbH'), ('fecha de nacimiento', '17/03/2015'), ('nombre', 'fsLluEofAR')])
row7: OrderedDict([('dos', 'bDp'), ('fecha de nacimiento', '16/10/2014')])
row8: OrderedDict([('dos', 'KIu'), ('fecha de nacimiento', '05/09/2014'), ('nombre', 'HcflxcTTnx')])
row9: OrderedDict([('dos', 'DMl'), ('fecha de nacimiento', '24/09/2014')])
row10: OrderedDict([('fecha de nacimiento', '06/08/2014')])
row11: OrderedDict([('fecha de nacimiento', '25/01/2015')])
row12: OrderedDict([('fecha de nacimiento', '14/10/2014')])
row13: OrderedDict([('fecha de nacimiento', '06/10/2014'), ('nombre', 'PcZuXqgooV')])
row14: OrderedDict([('fecha de nacimiento', '18/02/2015')])
row15: OrderedDict([('fecha de nacimiento', '03/04/2015')])
row16: OrderedDict([('fecha de nacimiento', '01/08/2014'), ('nombre', 'KbXBJVmCIa')])
row17: OrderedDict([('fecha de nacimiento', '07/03/2015'), ('nombre', 'TctipaQqwM')])
row18: OrderedDict([('fecha de nacimiento', '24/09/2014')])
row19: OrderedDict([('fecha de nacimiento', '07/04/2014'), ('nombre', 'nKExGVBaLK')])
```

**Figura 16. Datos almacenados en la base de datos de Cassandra.**

Estas pruebas fueron realizadas para comprobar la efectividad de la herramienta en la generación de datos para los gestores NoSQL seleccionados. Luego de haber realizado estas pruebas se pudo constatar

que la herramienta puede ser utilizada en la generación de datos para poblar las bases de datos de los gestores NoSQL seleccionados.

## 3.5 Validación del sistema

Uno de los aspectos fundamentales de la metodología XP es el proceso de pruebas. Este permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones.

XP divide el proceso de pruebas en dos grupos, (1) Pruebas Unitarias y (2) Pruebas de aceptación.

**Tipos de pruebas**

- **Pruebas Unitarias:** Al desarrollar un nuevo software o sistema de información, la primera etapa de pruebas a considerar es la etapa de pruebas unitarias o también llamada pruebas de caja blanca, estás pruebas también son llamadas pruebas modulares ya que nos permiten determinar si un módulo del programa está listo y correctamente terminado, estas pruebas no se deben confundir con las pruebas informales que realiza el programador mientras está desarrollando el módulo. Este tipo de pruebas debe ser realizado por personal especializado en las pruebas de software, el cual debe estar familiarizado en el uso de herramientas de depuración y pruebas, así mismo deben conocer el lenguaje de programación en el que se está desarrollando la aplicación [23].

- **Pruebas de aceptación:** Las pruebas de aceptación son creadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada.

Las pruebas de aceptación son consideradas como “pruebas de caja negra”. Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Así mismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación. Dado que la responsabilidad es grupal, es recomendable publicar los resultados de las pruebas de aceptación, de manera que todo el equipo esté al tanto de esta información [24].

**Técnicas de pruebas**

Cualquier producto de ingeniería puede probarse de una de estas dos formas: (1) conociendo la función específica para la que fue diseñado el producto, se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa y, al mismo tiempo buscando errores en cada función; (2) conociendo el funcionamiento del producto, se pueden desarrollar pruebas que aseguren que "todas las piezas encajan", o sea, que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada. El primer enfoque de prueba se denomina "prueba de caja negra" y el segundo, "prueba de caja blanca" [25].

- **Pruebas de caja blanca:** La prueba de caja blanca es un método de diseño de casos de prueba que usa la estructura de control de diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que (1) garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo; (2) ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; (3) ejecuten todos los bucles en sus límites y con sus límites operacionales; y (4) ejerciten las estructuras internas de datos para asegurar su validez [25].

- **Pruebas de caja negra:** Las pruebas de caja negra se centran en los requisitos funcionales del software. 0 sea, la prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de caja negra no es una alternativa a las técnicas de prueba de caja blanca. Más bien se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores que los métodos de caja blanca. La prueba de caja negra intenta encontrar errores de las siguientes categorías: (1) funciones incorrectas o ausentes, (2) errores de interfaz, (3) errores en estructuras de datos o en accesos a bases de datos externas, (4) errores de rendimiento y (5) errores de inicialización y de terminación [25].

**Método de prueba de caja negra:**

- **Partición de equivalencia:** La partición equivalente es un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores, que de

otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar [25].

Para la validación del sistema se selecciona como Tipo de prueba, las pruebas de aceptación, como Técnica de prueba, las pruebas de caja negra y dentro de estas el método de partición de equivalencia.

## 3.6 Casos de pruebas basadas en HU

Para el desarrollo de las pruebas que validarán el correcto funcionamiento de la solución, se realizará un flujo completo que comprende un caso de prueba para cada uno de los escenarios generales que dan cumplimiento a cada una de las Historias de Usuarios identificadas. Este flujo describirá el proceso necesario para desarrollar las pruebas con el objetivo de verificar cada escenario por los que está compuesto dicha solución. Para el correcto funcionamiento de cada uno de los escenarios de prueba el usuario debe seguir el flujo central de operaciones a fin de lograr el resultado de la prueba seleccionada.

La realización de las pruebas al sistema fueron guiados por un total de 7 casos de pruebas, los cuales fueron diseñados para probar cada una de las funcionalidades realizadas por la aplicación. Estos casos de pruebas arrojaron no conformidades la cuales fueron agrupadas en las tablas de no conformidades correspondiente a cada caso de prueba. La tabla 10 muestra un ejemplo de las no conformidades encontradas en el sistema luego de aplicar los casos de pruebas. Esta tabla tiene como atributos, entre otros, el nombre de la no conformidad encontrada, la clasificación ya sea: significativa (S), no significativa (NS) o recomendación (R) y la respuesta del equipo de desarrollo.

**Tabla 10. Tabla de ejemplo de no conformidades detectadas**

| No. | NC | Aspecto asociado | Etapa de detección | Clasificación | Estado NC | Respuesta |
|---|---|---|---|---|---|---|
| 1 | Acepta el campo Host con valor vacío generándose un error. | Conexión con el gestor MongoDB | Pruebas | S | 29/04/2014 PD 30/04/2014 RA | Corregido |

## 3.7 Pruebas del sistema

La prueba del sistema, está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas [25].

Como se muestra anteriormente las pruebas del sistema están enfocadas en verificar que el sistema funcione correctamente. Dentro de esta clasificación de pruebas se encuentran las de recuperación, las pruebas de seguridad y las pruebas de resistencia. Para comprobar el funcionamiento del sistema en condiciones anormales se decide realizar las pruebas de resistencia.

**Prueba de resistencia *(Stress)***

Las pruebas de resistencia están diseñadas para enfrentar a los programas con situaciones anormales. En el caso específico de la herramienta *"NoSQL Generator"* se decide aplicar este tipo de pruebas para comprobar la respuesta del sistema ante condiciones no esperadas que pueden ocurrir durante la ejecución de la aplicación. Para realizar esta prueba se identificó como principal anomalía que pudiese ocurrir durante el proceso de generación realizado por la aplicación, la pérdida de la conexión con el gestor de bases de datos con el cual el sistema se encuentra conectado.

Durante la realización de esta prueba se especificó en el sistema la generación e inserción de grandes volúmenes de datos en cada uno de los gestores de bases de datos utilizados. Durante el proceso de generación de los datos llevado a cabo por la aplicación, se detuvo el servidor de bases de datos quedando inactivo en ese instante. El sistema en algunos gestores no respondió correctamente ante la anomalía proporcionada detectándose una no conformidad en cada caso, en otros casos el sistema dio la respuesta esperada mostrando un mensaje de error y escribiendo en el archivo de errores de la aplicación el error ocurrido en este caso, en la figura 17 se evidencia el mensaje de error mostrado por el sistema cuando ocurre la pérdida de la conexión con el servidor.

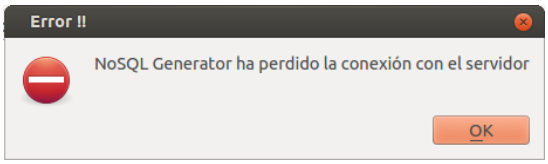



**Figura 17. Mensaje de error.**

## 3.8 Análisis de los resultados de las pruebas

Para comprobar el correcto funcionamiento e implementación de las Historias de Usuarios se aplicaron pruebas específicas para validarlas. La figura 18 muestra una gráfica en la cual se recogieron los datos correspondientes a las no conformidades encontradas durante las iteraciones de pruebas realizadas.

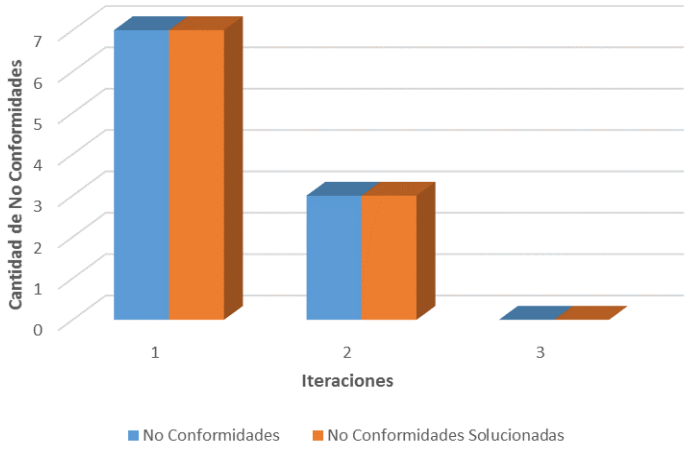



**Figura 18. Resultado de las pruebas por iteración.**

En esta gráfica se muestran las iteraciones realizadas para completar el proceso de pruebas. Por cada iteración se contabilizaron las no conformidades detectadas así como las que fueron resueltas. En la última iteración no fueron encontradas No Conformidades, concluyéndose el proceso de pruebas con un total de 10 no conformidades encontradas y todas fueron resueltas por el equipo de desarrollo.

## 3.9 Conclusiones del capítulo

En este capítulo se mostraron algunas interfaces visuales utilizadas en la aplicación, lo que permitió tener una idea visual de cómo se encuentra concebida la aplicación. Se mostraron algunos resultados obtenidos

en el uso de la herramienta, comprobándose la eficacia de la misma. Se definieron las pruebas que se aplicarían a la aplicación seleccionándose las pruebas de aceptación y la técnica de caja negra haciendo uso del método de partición de equivalencia como las pruebas hábiles para guiar el proceso de pruebas. También se seleccionó dentro de las pruebas del sistema las pruebas de resistencia, lo que permitió comprobar la respuesta del sistema ante situaciones anormales. Se aplicaron las pruebas seleccionadas permitiendo contar luego de 3 iteraciones con una versión estable de la aplicación.

# CONCLUSIONES GENERALES

Al concluir el desarrollo de la presente investigación y descritos cada uno de los capítulos en los que se encuentra dividido se arriban a las siguientes conclusiones:

- Se realizó un estudio sobre las herramientas existentes para la generación de datos en bases de datos, no encontrándose alguna que realizara el proceso para los gestores de bases de datos NoSQL.
- El proceso de desarrollo fue guiado por la metodología de desarrollo de software eXtreme Programming (XP). Como lenguaje de modelado se seleccionó UML y Visual Paradigm como herramienta CASE. La solución fue implementada utilizando el lenguaje de programación Python y el IDE de desarrollo Aptana Studio.
- Se obtuvo como resultado la herramienta *"NoSQL Generator"* la cual cumple con las funcionalidades identificadas.
- Se realizaron pruebas de aceptación utilizando la técnica caja negra mediante el método partición de equivalencia, además se le realizaron la herramienta pruebas del sistema, específicamente las pruebas de resistencia, lo que permitió contar con una herramienta funcional.

De manera general se cumplió el objetivo planteado al inicio de la investigación, contándose con una herramienta capaz de poblar los gestores de bases de datos NoSQL. Dicha herramienta se encuentra a disposición de la comunidad Universitaria para hacer uso de esta cuando se necesite.

# RECOMENDACIONES

Con el objetivo de mejorar y extender el alcance de la herramienta *NoSQL Generator* se recomienda:

- Extenderla a otros gestores NoSQL que vayan ganando en auge, ejemplos de estos pueden ser HBase, CouchBase y Memcached.
- Permitir la generación de los datos aleatorios utilizando fuentes de datos como por ejemplo, listas de nombres para el tipo de dato *Cadena.*

## REFERENCIAS BIBLIOGRÁFICAS


1. **CSC.** Big Data Universe Beginning to Explode. [En línea] 2012. http://www.csc.com.

2. **J Date, Christopher.** *An Introduction to Database Systems.* Octava Edicion. 2008.

3. **Connexions.** El Glosario IEEE de Ingeniería del Software. [En línea] 2013. http://cnx.org/content/m17423/latest.

4. **Chodorow, Kristina y Dirolf, Michael.** *MongoDB, The Definitive Guide.* Primera Edicion. s.l. : O'Reilly, 2010.

5. **Lennon, Joe .** *Beginning CouchDB.* United States of America : s.n., 2009.

6. **Macedo, Tiago y Oliveira, Fred .** *Redis Cookbook.* Primera Edición. s.l. : O'Reilly, 2011.

7. **Hewitt, Eben.** *Cassandra: The Definitive Guide.* [ed.] O'Reilly. Primera Edición. 2011.

8. **Neo Technology, Inc.** What is Neo4j? [En línea] 2014. http://www.neo4j.org/learn/neo4j.

9. **DB-Engines.** DB-Engines Ranking. [En línea] 2013 . http://db-engines.com/en/ranking.

10. **forSQL, Ltd.** forSQL Data Generator. [En línea] 2008. http://www.forsql.com/.

11. **Blog, Dave's Testing.** Populating Databases with random data using Spawner. [En línea] 2009. http://blog.karit.geek.nz/2009/05/populating-databases-with-random-data.html.

12. **Generatedata.** Generatedata.com. [En línea] http://www.generatedata.com.

13. **Canós, José H., Letelier, Patricio y Penadés, Mª Carmen.** *Métodologías Ágiles en el Desarrollo de Software.* Valencia : s.n., 2012.

14. **Rumbaugh, James y Booch, Grady y Jacobson, Ivar.** *El lenguaje unificado de modelado. Manual de Referencia.* 2000.

15. **Paradigm, Visual.** Boost Productivity with Innovative and Intuitive Technologies. [En línea] http://www.visual-paradigm.com/aboutus/10reasons.jsp.

16. **Dictionary, Business.** Business Dictionary. [En línea] http://www.businessdictionary.com/definition/programming-language.html.

17. **Larman, Craig .** *UML y Patrones.* 2004.

18. **Camacho, Erika , Cardeso, Fabio y Nuñez, Gabriel .** *Arquitecturas de Software.* 2004.

19. **Bahit, Eugenia .** *POO y MVC en PHP.* 2011.

20. **Madrid, Facultad de informática - Universidad Politécnica de.** *Patrones del "Gang of Four".*

21. **van Rossum, Guido y Warsaw, Barry.** Guía de estilo del código Python. [En línea] 2007 . http://mundogeek.net/traducciones/guia-estilo-python.htm.

22. **Sánchez, Carlos .** Ciclo de vida de un proyecto XP. [En línea] 2004. http://oness.sourceforge.net/proyecto/html/ch05s02.html.

23. **Oré, Ing. Alexander .** Quality Assurance & Software Testing. [En línea] 2009. http://www.calidadysoftware.com.

24. **Joskowicz, Ing. José .** *Reglas y Prácticas en eXtreme Programming.* 2008.

25. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico.* Quinta Edición.

# BIBLIOGRAFÍA

1. **van Rossum, Guido y Warsaw, Barry.** Guía de estilo del código Python. [En línea] 2007 . http://mundogeek.net/traducciones/guia-estilo-python.htm.

2. **Sánchez, Carlos .** Ciclo de vida de un proyecto XP. [En línea] 2004. http://oness.sourceforge.net/proyecto/html/ch05s02.html.

3. **Rumbaugh, James y Booch, Grady y Jacobson, Ivar.** *El lenguaje unificado de modelado. Manual de Referencia.* 2000.

4. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico.* Quinta Edición.

5. **Paradigm, Visual.** Boost Productivity with Innovative and Intuitive Technologies. [En línea] http://www.visual-paradigm.com/aboutus/10reasons.jsp.

6. **Oré, Ing. Alexander .** Quality Assurance & Software Testing. [En línea] 2009. http://www.calidadysoftware.com .

7. **O'Higgins, Niall .** *MongoDB & Python.* s.l. : O'Reilly, 2011.

8. **Neo Technology, Inc.** What is Neo4j? [En línea] 2014. http://www.neo4j.org/learn/neo4j.

9. **Madrid, Facultad de informática - Universidad Politécnica de.** *Patrones del "Gang of Four".*

10. **Lennon, Joe .** *Beginning CouchDB.* United States of America : s.n., 2009.

11. **Larman, Craig .** *UML y Patrones.* 2004.

12. **Joskowicz, Ing. José .** *Reglas y Prácticas en eXtreme Programming.* 2008.

13. **J Date, Christopher.** *An Introduction to Database Systems.* Octava Edicion. 2008.

14. **Hewitt, Eben.** *Cassandra: The Definitive Guide.* [ed.] O'Reilly. Primera Edición. 2011.

15. **González Duque, Raúl .** *Python para todos.* 2008.

16. **Generatedata.** Generatedata.com. [En línea] http://www.generatedata.com.

17. **forSQL, Ltd.** forSQL Data Generator. [En línea] 2008. http://www.forsql.com/.

18. **Downey, Allen B. .** *Think Python.* s.l. : O'Reilly, 2012.

19. **Dictionary, Business.** Business Dictionary. [En línea] http://www.businessdictionary.com/definition/programming-language.html.

20. **DB-Engines.** DB-Engines Ranking. [En línea] 2013 . http://db-engines.com/en/ranking..

21. **CSC.** Big Data Universe Beginning to Explode. [En línea] 2012. http://www.csc.com.

22. **Connexions.** El Glosario IEEE de Ingeniería del Software. [En línea] 2013. http://cnx.org/content/m17423/latest.

23. **Chodorow, Kristina y Dirolf, Michael.** *MongoDB, The Definitive Guide.* Primera Edicion. s.l. : O'Reilly, 2010.

24. **Canós, José H., Letelier, Patricio y Penadés, Mª Carmen.** *Métodologías Ágiles en el Desarrollo de Software.* Valencia : s.n., 2012.

25. **Camacho, Erika , Cardeso, Fabio y Nuñez, Gabriel .** *Arquitecturas de Software.* 2004.

26. **Blog, Dave's Testing.** Populating Databases with random data using Spawner. [En línea] 2009. http://blog.karit.geek.nz/2009/05/populating-databases-with-random-data.html.

27. **Bahit, Eugenia .** *POO y MVC en PHP.* 2011.

28. **Macedo, Tiago y Oliveira, Fred .** *Redis Cookbook.* Primera Edición. s.l. : O'Reilly, 2011.

29. **Chodorow, Kristina.** *Scaling MongoDB.* s.l. : O'Reilly, 2011.