Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

PROPUESTA DE ARQUITECTURA PARA REPORTERÍA EN PARADIGMA NoSQL (BASE DE DATOS BASADO EN DOCUMENTO)

Oscar Raúl Salguero Ventura
Asesorado por el Ing. Bitzel Enrique Cortez Sic

Guatemala, marzo de 2015

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

PROPUESTA DE ARQUITECTURA PARA REPORTERÍA EN PARADIGMA NoSQL (BASE DE DATOS BASADO EN DOCUMENTO)

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

OSCAR RAÚL SALGUERO VENTURA
ASESORADO POR EL ING. BITZEL ENRIQUE CORTEZ SIC

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, MARZO DE 2015

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paíz Recinos
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Narda Lucía Pacay Barrientos
VOCAL V	Br. Walter Rafael Véliz Muñoz
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paíz Recinos
EXAMINADOR	Ing. Marlon Antonio Pérez Türk
EXAMINADOR	Ing. Pedro Pablo Hernández Ramírez
EXAMINADOR	Ing. Marlon Francisco Orellana López
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

PROPUESTA DE ARQUITECTURA PARA REPORTERÍA EN PARADIGMA NoSQL (BASE DE DATOS BASADO EN DOCUMENTO)

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 14 de julio de 2014.

Oscar Raúl Salguero Ventura

Guatemala, 22 de septiembre de 2014

Ingeniero

Luis Fernando Espino Barrios

Respetable Ingeniero Espino:

Por este medio le informo, que como asesor del trabajo de graduación del estudiante universitario de la carrera de Ingeniería en Ciencias y Sistemas, Oscar Salguero, Carné 200714484, he revisado el protocolo, el marco teórico y el o los capítulos de aporte del trabajo de graduación titulado: "Propuesta de arquitectura para reportería en paradigma NoSQL (Base de datos basado en documento)". Y a mi criterio el mismo está completo y cumple a totalidad con los objetivos propuestos en el protocolo para su desarrollo.

Agradeciendo su atención a la presente,

Atentamente,

Ing. Bitzel Enrique Cortez Sic
Asesor de trabajo de graduación
Colegiado: 12725

Bitzel Enrique Cortez Sic
Ingeniero en Ciencias y Sistemas
Colegiado No. 12725
USAC

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 28 de Noviembre de 2014

Ingeniero
Marlon Antonio Pérez Türk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **OSCAR RAÚL SALGUERO VENTURA** con carné **2007-14484,** titulado: **"PROPUESTA DE ARQUITECTURA PARA REPORTERÍA EN PARADIGMA NoSQL (BASE DE DATOS BASADO EN DOCUMENTO)",** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
INGENIERIA EN CIENCIAS Y SISTEMAS
COORDINACION TESIS

ESCUELA DE CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación ***"PROPUESTA DE ARQUITECTURA PARA REPORTERÍA EN PARADIGMA NoSQL (BASE DE DATOS BASADO EN DOCUMENTO)"****, realizado por el estudiante OSCAR RAÚL SALGUERO VENTURA, aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

Ing. Marlon Antonio Pérez Türk
Director, Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 17 de marzo 2015

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 119.2015

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **PROPUESTA DE ARQUITECTURA PARA REPORTERÍA EN PARADIGMA NoSQL (BASE DE DATOS BASADO EN DOCUMENTO),** presentado por el estudiante universitario **Oscar Raúl Salguero Ventura,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Murphy Olympo Paiz Recinos
Decano

Guatemala, 17 de marzo de 2015

/gdech

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

ACTO QUE DEDICO A:

Dios Por ser guía y quien ha iluminado mi camino desde el inicio de este largo viaje, hasta el final. Por nunca dejarme.

Mis padres Oscar y Liliam Ventura de Salguero, por darme la vida, su apoyo económico y moral. Su cariño y buenos principios enseñados, que me han llevado a plantearme y cumplir este logro.

Mis hermanas Astrid y Andrea Salguero, porque son parte importante en mi vida y personas muy especiales. Más que mis hermanas son mis amigas y quiero compartir este logro con ellas.

Mi hermano Nilson Salguero, por su inmenso cariño y apoyo incondicional en todos los momentos de mi vida.

Mis amigos Josue Collado, Sergio Batz, Jerry Osorio, Bitzel Cortez y Estuardo Ventura, por todas las experiencias compartidas que fortalecieron nuestra amistad, se que puedo contar con ustedes. Espero que nuestra amistad prevalezca.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser mi alma mater, brindándome las herramientas necesarias para realizarme en el área profesional, contribuyendo con mi país.
Facultad de Ingeniería	Por brindarme los conocimientos, siendo un profesional con valores éticos y morales. Por la experiencia inolvidable de mi vida.
Mi esposa	Por estar a mi lado y ser la compañía perfecta en todo momento.
Mi hijo	Por traer a mi vida gran felicidad y ser la motivación para seguir adelante y luchar en la vida.
Mis amigos	Por todos los desvelos, risas, éxitos y fracasos compartidos, que hicieron de este un camino menos difícil. También gracias a sus familias que me recibieron en sus hogares mientras trabajábamos en tareas y proyectos**.**
Ing. Luis Espino	Por su apoyo en la realización de este trabajo de graduación.
Ing. Bitzel Cortez	Por todo su apoyo y asesoría en el presente.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

GLOSARIO

AJAX Asynchronous JavaScript and XML *(*JavaScript asincrónico y XML*)*. Es una técnica de desarrollo web para crear aplicaciones interactivas o RIA

AOP Aspect-oriented Programming (programación orientado a aspectos). Es un paradigma de programación que tiene como objetivo aumentar la modularidad, permitiendo la separación de preocupaciones transversales. AOP constituye una base para el desarrollo de software orientado a aspectos.

HTML HyperText Markup Language (Lenguaje de Marcado de Hipertexto). Es el lenguaje de marcado predominante para la elaboración de páginas web.

iBATIS Es un marco de código abierto basado en capas desarrollado por Apache Software Foundation, se ocupa de la capa persistencia. Puede ser implementado en Java y .NET

Informática Es la ciencia que trata la gestión automática de la información, utilizando dispositivos electrónicos y sistemas computacionales.

Internet	Es un conjunto de redes interconectadas por un protocolo TCP/IP que funciona como una red lógica, única de alcance mundial.
IOC / IoC	Investment of Control (inversión de control). Es un principio abstracto que describe un aspecto de algunos diseños de arquitectura de software en el que se invierte el flujo de control de un sistema en comparación con la programación procedimental.
JDBC	Java Database connectivity, controlador de bases de datos para lenguaje de programación Java.
POJO	Plain Old Object (objeto plano) es un acrónimo creado por Martin Fowler, Rebecca Parsons y Josh McKenzie en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un *framework* en especial.
POO	Programación Orientada a Objetos.
RIA	Rich Internet Application (aplicaciones ricas de internet) es una aplicación web que tiene muchas de las características de las aplicaciones de escritorio, por lo general entregados, ya sea a través de un navegador de sitios específicos, de un navegador *plug-in*, sistemas independientes o máquinas virtuales.

SQL	Structured query languaje (lenguaje de consulta estructurado). Lenguaje de consulta de base de datos.
www	El world wide web comúnmente conocido como la web, es un sistema de hipertexto enlazado y accesibles a través de internet.

RESUMEN

En la actualidad el uso de reportes es comúnmente utilizado para la toma de decisiones en base al uso de indicadores dentro de las organizaciones. La información es muy importante para estas organizaciones en el análisis de datos. Pero en la actualidad esta información se incrementa de forma exponencial, a tal punto que se está llegando a los límites de la capacidad de los repositorios donde se almacenan. Su acceso se vuelve mucho más complicado y se tienen problemas de performance. Esto hace que los reportes generados para la toma de decisiones sea muy lentos y esto no puede darse, ya que como se mencionó son para toma de decisiones y estas se dan en cualquier momento.

En un modelo relacional los datos son presentados en un conjunto de tablas en las cuales existen relaciones entre sí. Con lo que realizar consultas de la información involucra uniones de conjuntos, los cuales equivalen a un costo en cuanto a recursos. La información es representada de forma horizontal, por lo que esto se basa en el concepto de registro, los cuales contienen información como: número de cliente, nombre, dirección, teléfono, entre otros.

De esta forma se trabajan los ambientes de bases de datos, con lo que en este paradigma la información de los registros es cargada a memoria, teniendo acceso a todos hasta obtener el resultado deseado. Con esto se tiene una gran ineficiencia, ya que al final se leen todos los campos de un registro, cuando algunos de ellos son irrelevantes para solucionar una pregunta de negocio.

OBJETIVOS

General

Proponer una arquitectura de inteligencia de negocios con el fin de mostrar una forma rápida de generar reportes en bases de datos NoSQL. Se busca también, que sirva como alternativa para los analistas de sistemas que se encuentran con el problema de performance al momento de generar reportes.

Específicos

1. Hacer un análisis de la base de datos MongoDB para la creación de un repositorio basado en el paradigma NoSQL, buscando acoplamiento entre capas, fácil configuración y la mayor disponibilidad y seguridad posibles.

2. Hacer una guía de arquitectura para la carga masiva de información a la base de datos con paradigma NoSQL MongoDB, para su posterior análisis en la toma de decisiones.

3. Mostrar en detalle una alternativa para la generación de reportes de inteligencia de negocio sobre una base de datos no relacional y basada en documentos, buscando que sea rápida y efectiva, y que los reportes sean dinámicos.

4. Establecer los criterios que pueden servir para la elección del paradigma NoSQL como base de una arquitectura. Así como, la adopción de la arquitectura propuesta como propia de la aplicación.

5. Establecer los criterios que pueden servir para la elección de MongoDB y iReport como herramientas para el uso y manipulación eficiente de datos en la creación de reportes para la toma de decisiones, utilizando la arquitectura propuesta, proveyendo atributos de calidad esenciales para una aplicación óptima, orientado a administradores y desarrolladores de bases de datos

INTRODUCCIÓN

La manipulación de reportes para áreas de *marketing* y la alta gerencia básicamente trata del uso de datos en una empresa para facilitar la toma de decisiones. Abarca la comprensión del funcionamiento actual de la empresa, la anticipación de acontecimientos futuros, con el objetivo de ofrecer conocimientos para respaldar las decisiones empresariales.

Las herramientas de inteligencia de negocios se basan en la utilización de sistemas que se forman con información extraída de bases de datos, con información relacionada con la empresa o sus ámbitos y con datos económicos.

Para poder construir los informes finales que ven los usuarios, es necesario tener la información almacenada en una base de datos. Esto no es más que un repositorio de datos que puede ser accedido con facilidad, alimentado de numerosas fuentes, transformando información sobre temas específicos de negocios para permitir nuevas consultas, análisis, generación de reportes y toma de decisiones.

La arquitectura propuesta en este documento será una opción para los analistas de sistemas, quienes podrán contar con una alternativa a los problemas que representa realizar una base de datos tradicional RDBMS, al momento de almacenar o consultar volúmenes de datos demasiado grandes, que por lo regular se ven las deficiencias de los sistemas tradicionales. NoSQL es un movimiento a favor de un grupo muy bien definido de sistemas no-relacionales. Estos almacenes de datos rompen con una larga historia de bases

de datos relacionales. Estos almacenes de datos pueden no requerir esquemas fijos de tabla. Por lo general son sistemas escalares horizontales.

Existen bases de datos que utilizan este paradigma, algunas son de uso libre, mientras hay otras que deben ser compradas, entre las que tienen licencia *open source* se encuentra MongoDB, la cual es un cruce entre una clave/valor y la base de datos relacional. Ofrece gran capacidad de almacenamiento y acceso rápido, así como también a documentos incorporados. Es rápido, flexible y de alta escalabilidad, entre otras características.

1. PARADIGMA SQL

1.1. Similitudes entre paradigmas

Hay muchas diferencias entre una arquitectura de base de datos relacional y una con paradigma NoSQL, tanto en rendimiento como en metodología, seguridad y agilidad. Recientemente hay bases de datos como MongoDB que está proporcionando una interface de consulta similar a RDBMS, buscando así que RDBMS Y NoSQL vayan al mismo destino por caminos diferentes. NoSQL es una base de datos no relacional con un esquema flexible, todo lo contrario a MySQL, Oracle o PosgreSQL, que requieren relaciones, normalización y siempre con un esquema fijo.

NoSQL no está hecha para consultas complejas, pero si permite recuperar rápidamente gran cantidad de datos planos. Por lo tanto, esta base de datos se recomienda usarla para almacenar datos que contengan mucho texto plano como *blogs*. Por ejemplo: Twitter y Facebook, usan bases de datos NoSQL.

NoSQL presenta ausencia de esquema en los registros de datos, esto quiere decir que los datos pueden ser de una forma diferente cada vez, pudiendo almacenar los datos de la forma que quiera.

Otra característica es que son de escalabilidad horizontal, esto quiere decir que se puede aumentar el rendimiento solamente añadiendo más nodos.

Las bases de datos NoSQL se ejecutan en grupos de máquinas baratas, estos sistemas no requieren de mucho rendimiento en comparación con los sistemas gestores de bases de datos tradicionales y basados en SQL, por lo que se pueden montar en máquinas de un costo más reducido y en mayor número, gracias a su nivel de escalabilidad.

El paradigma NoSQL no usa sentencias SQL tan complicadas, por ejemplo el JOIN en este tipo de bases de datos no existe. Pueden manejar enormes cantidades de datos, esto se debe a su propia estructura distribuida.

Casi todos los sistemas por no decir todos son de código abierto. Las bases de datos NoSQL son mucho más rápidas en caso de peticiones simultáneas. Los datos manejados en NoSQL cumplen con transacciones BASE (coherencia eventual flexible básicamente disponible) y SQL ACID.

En un RDBMS se tendría que guardar la información en diferentes tablas y luego usar un lenguaje de programación en el cliente. En NoSQL al no estar relacionado se puede guardar la información que se requiera en cualquier tabla.

1.2. Ventajas en el uso de NoSQL *versus* SQL y modelos relacionales para la creación de reportes

Hay muchas formas de trabajar reportería con los datos cargados en una base de datos NoSQL, una de ellas es volcar a un RDBMS tradicional. Esta técnica no es atractiva para el analista que desea generar reportes, ya que representa doble trabajo, pero es ideal en muchos casos. Se deberán crear procesos de ETL para obtener los datos de MongoDB y transferirlos a PostgreSQL, MySQL, Vertica, Teradata, Netezza, o cualquier base de datos

relacional. Desde allí se tiene una gran cantidad de herramientas para disponer de los datos de una forma relacional o para el diseño de informes.

Otra forma para generar informes es por medio de PHP, se puede crear consultas con código PHP hacia MongoDB, a través del conector MongoDB PHP.

Si se desea utilizar herramientas avanzadas para análisis de datos como los usados comúnmente en *marketing* o en BI, el mayor obstáculo es que las bases de datos grandes (incluyendo MongoDB) no son accesibles a través de métodos estándar como JDBC. Eso significa que herramientas tradicionales de Business Intelligence (BI) no pueden acceder directamente a los datos de fuentes de datos grandes.

El *framework* Jaspersoft ha abordado el problema del manejo de reportes en varias bases de datos no relacionales, incluyendo MongoDB. Jaspersonft ofrece conectores a una gran variedad de fuentes de datos.

2. TIPOS DE BASES DE DATOS

2.1. Diferentes formas de almacenar información

Las bases de datos pueden clasificarse de varias maneras de acuerdo al contexto que se esté manejando la utilidad de las mismas o las necesidades que satisfagan.

2.2. Según la variabilidad de los datos almacenados

La información almacenada en las bases de datos puede variar con el tiempo y estos cambios pueden darse de distintas formas, y por diferentes razones. Es por ello que existen distintas clasificaciones de bases de datos según como estas cambian.

2.2.1. Bases de datos estáticas

Las bases de datos de lectura son estáticas, por lo regular se utilizan para guardar datos históricos que después se utilizan para analizar el comportamiento de un conjunto de información, a través del tiempo, tomar decisiones, realizar proyecciones y realizar análisis de datos para el uso que le quiera dar la empresa, son utilizadas principalmente en el área de *marketing* y negocios.

2.2.2. Bases de datos dinámicas

Esta información es modificable en el tiempo, permite realizar operaciones en las tablas como actualización de información, eliminar e insertar datos, además de consultas sobre las tablas en la base de datos. Un ejemplo de esto puede ser la base de datos utilizada en un sistema de información, por ejemplo tiendas o súper mercados.

2.3. Según el contenido

Las bases de datos pueden almacenar información de distintos tipos y distintas procedencias, la información puede ser cuantitativa, literaria, entre otros tipos. Es por eso que se puede categorizar según su contenido.

2.3.1. Bases de datos bibliográficas

Sólo contienen referencias a publicaciones científicas, podrían llegar a ser utilizadas para cualquier persona que tenga interés en estos recursos. Un registro típico de una base de datos bibliográfica, contiene información sobre la editorial, título, edición, autor, fecha de publicación, de una determinada publicación. Pueden llegar a tener un soporte físico como por ejemplo fichas impresas. Puede contener un extracto de la publicación original, pero no el texto completo, porque si no, se estaría en presencia de una base de datos a texto completo, esto saturaría la base de datos haciéndola poco práctica. Como su nombre lo indica, el contenido son cifras o números.

2.3.2. Bases de datos de texto completo

Existe una diferencia entre este tipo de base de datos y las regulares, las bases de datos de texto completo permiten ver el texto completo, en línea o en otro sitio. Almacenan las fuentes primarias como por ejemplo, el contenido de todas las ediciones de un libro.

2.3.3. Directorios

Un ejemplo son las agendas electrónicas de los teléfonos celulares, las cuales almacenan información que se desea consultar constantemente. El resultado de la búsqueda debe ser rápido y eficiente para que sea útil al usuario.

2.4. Modelos de bases de datos

Las bases de datos no solo se pueden clasificar por la función de las mismas, de acuerdo a su modelo de administración de datos también existe una clasificación.

Básicamente un modelo de datos es una descripción de un contenedor de datos (un repositorio de información que permite operaciones entre tablas y pueden consultarse), así como de los métodos para almacenar y recuperar información de estos contenedores. Los modelos de datos no son físicos, más bien son abstracciones que permiten la implementación de un sistema eficiente de base de datos; por lo general son algoritmos y teorías matemáticas.

2.4.1. Bases de datos jerárquicas

Esta base de datos se representa de forma similar a un árbol visto al revés, por la forma en que se organiza la información, gráficamente un nodo padre puede tener uno o varios nodos hijos. El nodo principal, el que no tiene padres es llamado raíz, y las hojas son los nodos que no tienen hijos.

La utilidad de las bases de datos jerárquicas es visible cuando se utilizan en casos de aplicaciones que manejan unos grandes volúmenes de información y datos muy compartidos permitiendo crear estructuras de gran rendimiento y estables.

Existen limitantes en este modelo, la principal es su poca capacidad de representar eficientemente la redundancia de datos.

2.4.2. Base de datos de red

La diferencia con el modelo jerárquico es fundamentalmente la modificación del concepto de nodo, ya que permite que un mismo nodo tenga muchos padres (esto no es posible en el modelo jerárquico).

Con respecto al modelo jerárquico es una gran mejora, ya que ofrece una eficiente solución al problema de redundancia de datos que es muy común. Pero aún así la dificultad que implica administrar la información en una base de datos de red, ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales.

2.4.3. Bases de datos transaccionales

El fin de estas bases de datos es enviar y recibir información a gran velocidad, las bases de datos transaccionales son poco comunes y por lo regular están dirigidas al entorno de análisis de calidad, industria y datos de producción. Es importante entender que su fin único es recolectar y recuperar los datos a la mayor velocidad posible, por lo tanto la redundancia y duplicación de información no es un problema como con las demás bases de datos. Por lo general para poderlas aprovechar al máximo, permiten algún tipo de conectividad a bases de datos relacionales.

Un ejemplo habitual de transacción es la compra de productos por medio de páginas web. Por lo regular se realiza mediante dos tipos de operaciones, una en la que se realizan cobros del saldo de la cuenta origen y otra en la que se incrementa el saldo de la cuenta destino. Para garantizar la atomicidad del sistema (es decir, para que no aparezca o desaparezca dinero), las dos operaciones deben ser atómicas, es decir, el sistema debe garantizar que, bajo cualquier circunstancia (incluso una caída del sistema), el resultado final es que, o bien se han realizado las dos operaciones, o bien no se ha realizado ninguna.

2.4.4. Bases de datos relacionales

Este es el modelo utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. La idea fundamental es el uso de relaciones entre conjuntos de datos, rápidamente se consolido como un paradigma en los modelos de base de datos. Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados tuplas. Pese a que esta es la teoría de las bases de datos relacionales creadas por Codd la mayoría de las veces se conceptualiza de una manera más fácil de imaginar.

Fue pensado en como relacionar los grupos de datos llamados registros en tablas, que representarían las tuplas y campos (las columnas de una tabla), buscando la forma de decir que un grupo de información tiene relación directa o indirecta con otro grupo de información.

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. La información puede ser recuperada o almacenada mediante consultas, que ofrecen una amplia flexibilidad y poder para administrar la información.

El lenguaje que se utiliza más comúnmente para construir las consultas a bases de datos relacionales es SQL, también llamado: lenguaje estructurado de consultas. Un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

Cuando se hace el diseño de una base de datos, esta pasa por un proceso al que se le llama normalización de una base de datos, esto en los modelos relacionales. Durante muchos años la aparición de las bases de datos digitales, produjo una revolución en los lenguajes de programación y sistemas de administración de datos. Aunque nunca debe olvidarse que dBase no utilizaba SQL como lenguaje base para su gestión.

2.4.5. Bases de datos multidimensionales

Estas bases de datos fueron pensadas para desarrollar aplicaciones muy concretas, como creación de cubos OLAP. Por lo regular no se diferencian demasiado de las bases de datos relacionales (una tabla en una base de datos relacional podría serlo también en una base de datos multidimensional), la diferencia está más bien a nivel conceptual. En las bases de datos multidimensionales los campos o atributos de una tabla pueden ser de dos tipos, o bien representan dimensiones de la tabla, o bien representan métricas que se desean estudiar.

2.4.6. Bases de datos orientadas a objetos

Es un modelo de base de datos bastante reciente, y está diseñado para modelos informáticos orientados a objetos. Trata de almacenar en la base de datos los objetos completos (estado y comportamiento).

Una base de datos orientada a objetos es una base de datos que incorpora todos los conceptos importantes del paradigma de objetos:

- Encapsulación: propiedad que permite ocultar la información al resto de los objetos, impidiendo así accesos incorrectos o conflictos.
- Herencia: propiedad a través de la cual los objetos heredan comportamiento dentro de una jerarquía de clases.
- Polimorfismo: propiedad de una operación mediante la cual puede ser aplicada a distintos tipos de objetos.

En bases de datos orientadas a objetos, los usuarios pueden definir operaciones sobre los datos como parte de la definición de la base de datos,

una operación (llamada función) se especifica en dos partes. La interfaz (o signatura) de una operación, incluye el nombre de la operación y los tipos de datos de sus argumentos (o parámetros). La implementación (o método) de la operación se especifica separadamente y puede modificarse sin afectar la interfaz. Los programas de aplicación de los usuarios pueden operar sobre los datos invocando a dichas operaciones a través de sus nombres y argumentos, sea cual sea la forma en la que se han implementado. Esto podría denominarse independencia entre programas y operaciones.

SQL: 2003, es el estándar de SQL92 ampliado, soporta los conceptos orientados a objetos y mantiene la compatibilidad con SQL92.

2.4.7. Bases de datos documentales

Permiten la indexación de texto completo y realiza búsquedas en líneas generales más efectivamente. Se puede utilizar Tesaurus, que es un sistema para crear y utilizar índices optimizados para este tipo de bases de datos.

2.4.8. Bases de datos deductivas

Este tipo de base de datos también llamado deductiva es un sistema similar a los que se han visto anteriormente, con la diferencia de que este permite hacer deducciones a través de inferencias. Se basa principalmente en reglas y hechos que son almacenados en la base de datos. Las bases de datos deductivas son también llamadas bases de datos lógicas, a raíz de que se basa en lógica matemática. Este tipo de base de datos surge debido a las limitaciones de la base de datos relacional de responder a consultas recursivas y de deducir relaciones indirectas de los datos almacenados en la base de datos.

3. ARQUITECTURA DE BASES DE DATOS NOSQL

3.1. ETL en una base de datos multidimensional con paradigma NOSQL

El ETL es la extracción, transformación y carga de datos a un repositorio de información, por lo regular este repositorio de información es una base de datos tipo estrella desmoralizada. Las bases de datos tipo estrella no están relacionadas al igual que las bases de datos NoSQL.

3.1.1. Integrando Spring Batch y MongoDB para ETL

Es el proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos y limpiarlos, y cargarlos en otra base de datos, Data Mart, o Data Warehouse para analizar, o en otro sistema operacional para apoyar un proceso de negocio.

3.1.1.1. Procesamiento por lotes

La principal ventaja de las aplicaciones de lotes es que no requiere ninguna intervención manual. Como resultado se pueden programar para que se ejecute en momentos cuando los recursos no están siendo utilizados. A modo de ejemplo, una herramienta ETL que se ejecuta en modo Batch, se va a analizar en una base diaria, para ir a buscar la información útil requerida. Los archivos de entrada son extraídos y procesados para obtener la información requerida, y la salida de datos se carga a una base de datos. Todo este proceso se lleva a cabo en modo de lote.

Los procesos por lotes se ocupan principalmente de grandes cantidades de datos en una serie de programas que corren para alcanzar el objetivo deseado. Estos programas se pueden ejecutar uno después del otro, o pueden funcionar en paralelo para acelerar la ejecución, dependiendo de los requisitos. El procesamiento por lotes permite compartir los recursos, estos procesos se ejecutan principalmente hacia el final del día cuando otros procesos no se encuentran consumiendo recursos.

3.1.1.2. Lotes Spring

El marco de lotes Spring está diseñado para atender a las aplicaciones de proceso por lotes que se ejecutan a diario en las organizaciones empresariales. Ayuda a aprovechar los beneficios de la infraestructura Spring junto con los servicios avanzados. Lote Spring se utiliza principalmente para procesar gran volumen de datos, ofrece un mejor rendimiento y es altamente escalable mediante la optimización de diferentes técnicas de partición. También proporciona ventaja sobre el registro y rastreo, manejo de la gestión de transacciones, las estadísticas de procesamiento de trabajos, el reinicio de trabajo, escaleras y recursos. Al utilizar el modelo de programación Spring se puede escribir la lógica de negocio y dejar que el marco cuide de la infraestructura.

Lote Spring incluye tres componentes: la aplicación de lote, entorno de ejecución de lotes y lotes de infraestructura. El componente de aplicación contiene todos los trabajos por lotes y código personalizado escrito, usando lotes de primavera.

El componente de aplicación contiene todos los trabajos por lotes y código personalizado escrito, usando lotes de primavera.

El componente básico contiene las clases de tiempo de ejecución necesarias para poner en marcha y controlar un trabajo por lotes. Esto incluye cosas tales como: una JobLauncher, Job, y las implementaciones de paso. Tanto la aplicación y Core son construidos en la cima de una infraestructura común.

La infraestructura incluye lectores, escritores y servicios que son utilizados tanto por la aplicación y el marco básico en sí. Estos incluyen cosas como ItemReader, ItemWriter y MongoTemplate. Para utilizar el marco de lotes de primavera, basta configurar y personalizar los archivos XML. Todos los servicios básicos existentes deben ser fáciles de reemplazar o extender sin ningún impacto a la capa de infraestructura.

3.1.2. MongoDB

El sistema bajo el cual funciona la base de datos MongoDB es NoSQL, orientado a documentos, el concepto bajo el cual se desarrolló es de código abierto.

MongoDB es una base de datos no relacional y es parte del nuevo paradigma de sistemas de base de datos NoSQL. En lugar de guardar la información en tablas como se hace en las base de datos relacionales, MongoDB, guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

En octubre de 2007 empezó el desarrollo de MongoDB por la compañía de software 10gen, ahora MongoDB es una base de datos lista para la producción y con muchas características. Esta base de datos es altamente

utilizada en las industrias. MTV Network, 2 Craiglist3 y Foursquare4 son algunas de las empresas que utilizan esta base de datos.

El código binario está disponible para los sistemas operativos Windows, Linux, OS X y Solaris.

3.1.2.1. Características principales

Es importante describir cuales son las principales características de esta base de datos, aunque de forma general se sabe que es del tipo no relacional y que es de licencia libre. En MongoDB prácticamente es posible trabajar las mismas cosas que en una base de datos de tipo relacional.

3.1.2.1.1. Consultas AD HOC

En la base de datos MongoDB se pueden realizar búsquedas de todo tipo, como por ejemplo; expresiones regulares, campos y consultas de rangos. Es posible que algunas consultas devuelvan un campo específico del documento, pero también puede ser una función JavaScript definida por el usuario.

3.1.2.1.2. Indexación

En un principio puede pensarse que al no ser bases de datos relacionales, no es posible indexar campos, pero el indexar campos en un documento es posible en MongoDB, al igual que es posible hacer índices secundarios. El concepto de índices en MongoDB es similar a los encontrados en base de datos relacionales.

3.1.2.1.3. Replicación

La replicación maestro-esclavo es soportada en MongoDB. El maestro puede ejecutar comandos de lectura y escritura. El esclavo puede copiar los datos del maestro y sólo se puede usar para lectura o para copia de seguridad, pero no se pueden realizar escrituras. El esclavo tiene la habilidad de poder elegir un nuevo maestro en caso de que se caiga el servicio con el maestro actual.

3.1.2.1.4. Balanceo de carga

Utilizando el concepto de Shard en MongoDB es posible escalar de forma horizontal. Para determinar como serán distribuidos los datos en un sistema con nodos, el desarrollador debe elegir una llave Shard. Los datos son divididos en rangos (basado en la llave Shard) y distribuidos a través de múltiples Shard. Un Shard es un maestro con uno o más esclavos. MongoDB tiene la capacidad de ejecutarse en múltiple servidores, balanceando la carga y/o duplicando los datos para poder mantener el sistema funcionando en caso que exista un fallo de hardware. La configuración automática es fácil de implementar bajo MongoDB y nuevas máquinas pueden ser agregadas a MongoDB con el sistema de base de datos corriendo.

3.1.2.1.5. Almacenamiento de archivos

Considerando la capacidad que tiene MongoDB, puede ser utilizado con un sistema de archivos, ya que existe una clara ventaja de la capacidad que tiene MongoDB para el balanceo de carga y también en la replicación de datos utilizando múltiples servidores para el almacenamiento de archivos. La función llamada GridFS11 está incluida en los drivers de MongoDB como una

funcionalidad y disponible para los lenguajes de programación que soporta MongoDB. Esta base de datos expone funciones para la manipulación de archivos y contenido a los desarrolladores. En un sistema con múltiple servidores, los archivos pueden ser distribuidos y copiados entre los mismos varias veces y de una forma transparente, de esta forma se crea un sistema eficiente que maneja fallos y balanceo de carga.

3.1.2.1.6. Agregación

Existe una función llamada MapReduce, la cual puede ser utilizada en MongoDB para operaciones de agregación y el procesamiento por lotes de datos. Esta función permite que los usuarios puedan obtener el tipo de resultado que se obtiene cuando se utiliza el comando SQL *group-by*.

3.1.2.1.7. Ejecución de JavaScript del lado del servidor

Uno de los beneficios de MongoDB es la capacidad que tiene para hacer consultas utilizando JavaScript, las sentencias son enviadas de forma directa a la base de datos, luego pueden ser utilizadas para hacer consultas y operaciones en las tablas.

3.1.2.2. Casos de uso

MongoDB es una base de datos que se utiliza en muchos casos, la misma es adecuada para:

- Realizar registro de eventos y almacenamiento
- Para contenido y en sistemas de manejo de documentos

- Se puede usar en comercio electrónico
- Para almacenar datos de video juegos
- Volúmenes altos de lectura
- Aplicaciones de teléfonos móviles
- Almacén de datos operacional de una página web
- Manejo de contenido
- Almacenamiento de comentarios

    - Votaciones
    - Registro de usuarios
    - Perfiles de usuarios
    - Sesiones de datos

- En proyectos que utilizan metodologías iterativo o ágiles.
- Manejo de estadísticas en tiempo real.

MongoDB es utilizado para uno o varios de estos casos por varias empresas.

Ejemplo de la estructura BSON en la que MongoDB guarda los datos:

```
{
   "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
   "Last Name": "PELLERIN",
   "First Name": "Franck",
   "Age": 29,
   "Address": {
      "Street": "1 chemin des Loges",
      "City": "VERSAILLES"
```

```
    }
}
```

4. ARQUITECTURAS PARA LA CREACIÓN DE REPORTES EN NOSQL

4.1. Generación de reportes

Los avances recientes de la industria para responder a la pregunta de ¿cómo pueden las empresas poner herramientas tradicionales de reportes y Business Intelligence (BI) en la parte superior de los sistemas de datos NoSQL? Es un cuestionamiento que todavía está lleno de muchos problemas y ciertamente no es una unión perfecta todavía.

Incluso con el crecimiento de NoSQL y sus instancias de producción diferentes que se ejecuta en muchos sistemas de datos grandes, no hay información suficiente para realizar un análisis exhaustivo. No hay miles de encuestas a empresas realizadas, integraciones completas, numerosas historias de nuevas adopciones y otros datos necesarios para decir que la unión de reporteria tradicional y NoSQL es un éxito rotundo.

La información más confiable viene a través de autores de libros y de firmas consultoras, que están ayudando a los sistemas de datos grandes con sus integraciones NoSQL y aplicaciones de manejo de reportes. La falta de evidencia sustancial con las dos principales partes de esta unión emergente y la industria tiene una visión conflictiva. Por un lado se tiene el almacenamiento y análisis de datos, el aporte de BI, los gestores y los analistas que tratan de trabajar con sus viejos sistemas SQL. Por otro lado están las personas que crean aplicaciones e integraciones en sistemas nuevos NoSQL.

Los dos grupos no están de acuerdo. En primera instancia se ven enormes brechas en sus capacidades de reporteria de BI, se desean mejores sistemas de información. Mejoras en el servicio de AdHoc de productos básicos, como cuadros de mando instantáneos, rápidas consultas y otras herramientas convencionales SQL/BI. El segundo grupo ve la promesa de NoSQL como la respuesta a los problemas cada vez mayores de SQL basados en los sistemas de datos grandes. Para este segundo grupo el manejo de grandes cantidades de datos, ya es un éxito que deber reconocerse.

Algunos de estos incluyen:

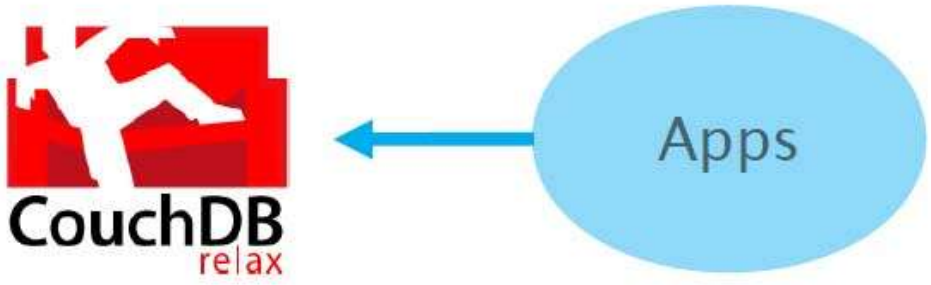
- La capacidad de almacenar más datos.
- Costo de uso efectivo y análisis.
- Richer analytical tools que permite el filtrado estándar, y el grupo de declaraciones rendidas por SQL, como el análisis gráfico, recorridos, los marcos MapReduce y otros.
- Estructuras de datos flexibles y fiables tolerancias de fallo.
- Almacenamiento a largo plazo eficaz y más fiable y escalabilidad.

En este punto en la unión entre NoSQL y las herramientas de análisis, hay esencialmente seis conjuntos arquitectónicos principales empleados por las empresas que trabajan integraciones de datos NoSQL con grandes volúmenes de datos, centradas en sistemas SQL. Probablemente ninguna de las seis están siendo utilizadas exclusivamente para sí, sino más bien son los casos esenciales de uso general o las maneras principales de hacerlo.

4.2. Informes NoSQL

Esta técnica obliga a prestar los servicios de desarrolladores para crear aplicaciones para los informes en la parte superior del sistema de NoSQL. Tiene todas las ventajas de NoSQL, pero es caro debido a la necesidad de contratar un desarrollador. Utiliza las tecnologías actuales, es totalmente personalizable, ya que es un hecho singular, es exclusivo, ya que sólo será utilizado por la empresa para la cual se ha creado. Es sólo una de las fuentes de datos, sin latencia, puede escribir trabajos MapReduce y no está limitado por otras aplicaciones. No viene con ninguna herramienta de BI de los productos básicos, tiene funcionalidad de autoservicio en la parte comercial, ya que cualquier creación de informes debe ser realizada por el promotor y es difícil y costoso para integrar otras aplicaciones en el mismo.

Figura 1. **Aplicaciones para generar reportes**



Fuente: http://couchdb.apache.org/. Consulta: 1 de diciembre de 2014.

4.3. NoSQL de arriba abajo

Similar a la primera alternativa mencionada, la arquitectura NoSQL de arriba abajo tiene todas las ventajas de la tecnología NoSQL, mientras que es

flexible. También se requiere el pago de un desarrollador para construir esa flexibilidad, por lo que es más caro, toma el primer enfoque mencionado y añade la construcción de un generador de consultas dinámicas en el sistema de presentación de informes.

Por ejemplo, permite la creación de reportes adicionales en un repositorio que pueden ser creados a partir de filtros dinámicos. Así que en lugar de tener sólo un atributo para realizar consultas, un usuario tendría varios atributos para cada tipo de reporte en particular, mediante la especificación de los mismos, esto permite realizar consultas dinámicas sobre la información.

Esta es una *guía ad hoc* del sistema y ofrece cuadros de mando más dinámicos y resumenes, pero todavía tiene los inconvenientes enumerados en el primer enfoque mencionado. Esta arquitectura permite acceder a mucha más información en las bases de datos NoSQL, pero tiene complicación al integrarse con otros sistemas de información como los CRM convencional.

Figura 2. **Flexibilidad en la generación de informes**

Fuente: Professional NoSQL, Shashank Tiwari.

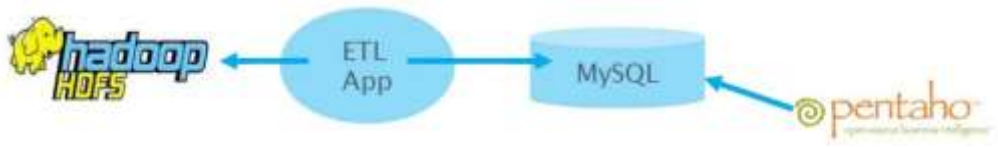
4.4. NoSQL más MySQL

Esta tercera alternativa utiliza un método de combinación también conocido como: quitar NoSQL y ponerlo en MySQL. No requiere los altos costos

de contratar a un desarrollador para crear una funcionalidad flexible, permite el uso de las herramientas de BI como Ricos Pentaho, Jasper, Tableau, Crystal Reports o cualquier otra herramienta de BI básica para informar sobre los datos, ya que se exporta la información desde el sistema NoSQL. Asimismo, permite consultas e informes dinámicos según la herramienta que se esté utilizando. Esta es una forma práctica y sencilla de resolver el problema de la generación de reportes, ya que hay herramientas que son muy dinámicas y amigables para el usuario.

Sus desventajas principales son la lentitud en actualización de datos, ya que el tiempo de retraso es 24 horas aproximadamente. Las ventajas que ofrece NoSQL en el manejo de datos en el sistema se pierde en el entorno, las herramientas de BI sólo se puede conectar al sistema de MySQL.

Figura 3. **NoSQL con MySQL**



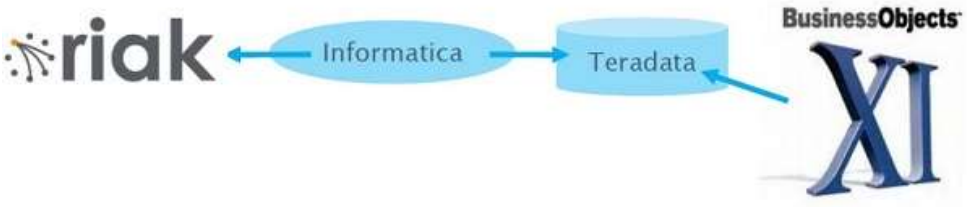
Fuente: Professional NoSQL, Shashank Tiwari.

4.5. NoSQL como ETL Data Source

El cuarto enfoque es una perspectiva novedosa para el manejo de datos NoSQL. En lugar de verlo como algo separado dentro del almacén de datos, todos los datos NoSQL se entienden como otra fuente de datos ETL. Los datos se extraen del sistema de datos NoSQL, son puestos en el almacén de datos y se integran con otros datos que ya están allí. Por lo tanto, esta es la primera arquitectura propuesta que permite que los datos estén integrados.

A continuación se pueden consultar con herramientas estándar de BI. Gran parte de esta cuarta alternativa es similar al enfoque anterior (NoSQL más MySQL), sin embargo, las grandes ventajas que ofrece NoSQL y por las cuales se trabaja con este paradigma se ven mermadas al momento de hacer la integración. Hay un alto costo de desarrollo de ETL para hacer el trabajo de integración. Las herramientas tradicionales de almacenamiento de datos también son costosos y la escalabilidad NoSQL se pierde.

Figura 4. **Datos NoSQL como fuente de datos**



Fuente: Professional NoSQL, Shashank Tiwari.

4.6. Programas NoSQL en herramientas de BI

Esta arquitectura requiere nuevamente de los servicios de un programador, el cual deberá crear aplicaciones nuevas, lo que representa costos de desarrollo. Sin embargo, es una alternativa más económica y práctica que las anteriores que se han visto. Un programador debe escribir un software para una herramienta de BI estándar y este deberá ser enviado en un informe. El desarrollador no tiene que gastar tiempo en factores del informe como los márgenes y colores, sino que escribe una aplicación que conecta la herramienta de BI a los datos NoSQL. Esto pasa por alto la necesidad de SQL, herramientas de acceso AdHoc basados en la web y metadatos.

Es más simple en su diseño, más barato que escribir un montón de informes personalizados, sigue utilizando el lenguaje NoSQL, puede escribir trabajos MapReduce, tiene acceso al 100 por ciento de la base de datos. Pero tiene desventajas que incluyen velocidades más lentas para agregaciones y resúmenes, algunos costos de desarrollo, la falta de integración con otros sistemas y no es de acceso Ad Hoc.

Figura 5. **Desarrollo de conector**

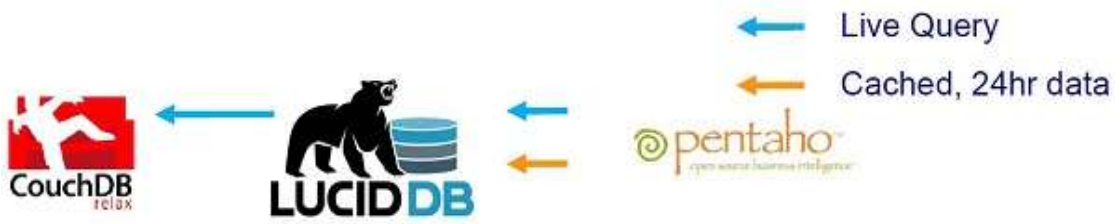
Fuente: Professional NoSQL, Shashank Tiwari.

4.7. NoSQL BI a través de bases de datos (SQL)

Otra alternativa más de arquitectura para generación de reportes en bases de datos NoSQL es el uso de herramientas creadas por empresas que se dedican al desarrollo he investigación de las mismas. Estas herramientas actúan como un intermediario para comunicar los datos en NoSQL y las herramientas estándares de BI. Este enfoque permite la integración de datos con otros sistemas, el ETL es simple con Insert, tiene acceso ad Hoc, hace uso del cache para el almacenamiento de datos, hace uso de lo mejor de ambas tecnologías.

Este enfoque sigue usando las ventajas de NoSQL, pero para la interfaz de usuario se siguen utilizando las herramientas tradicionales de BI.

Figura 6. **Uso de herramientas tradicionales para reporteria en NoSQL**



Fuente: Professional NoSQL, Shashank Tiwari.

5. PROPUESTA DE ARQUITECTURA

5.1. Casos de estudio: informes y análisis en MongoDB

A continuación se dará una descripción ilustrada de cómo generar reportes en una arquitectura de base de datos MontoDB y el diseñador de reportes iReport. También se verá una descripción de los comandos básicos para la generación de los reportes.

5.1.1. Creación de informes de iReport en un origen de datos MongoDB

iReport es un diseñador de reportes de distribución libre de código abierto para JasperReports. Crea diseños muy sofisticados que contienen gráficos, imágenes, subinformes, tablas de contingencia y mucho más. Accede a los datos a través de JDBC, TableModels, JavaBeans, XML, Hibernate, CSV, y fuentes personalizadas. Publica los informes en PDF, RTF, XML, XLS, CSV, HTML, XHTML, texto, DOCX, o OpenOffice. iReport puede descargarse de http://community.jaspersoft.com/project/ireport-designer

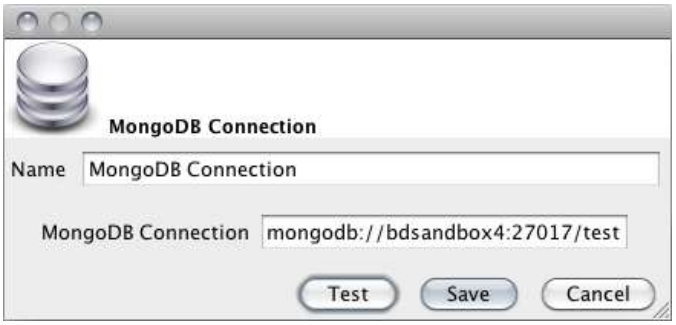
MongoDB es la base de datos con la que se realiza el caso de estudio, puede descargarse de http://www.10gen.com/

También es necesario el conector para MongoDB, el cual se puede descargar de http://community.jaspersoft.com/big-data

5.1.2. MongoDB conector

La instalación del conector es simple, se incluyen en la documentación que viene con el conector. Una vez instalado el *plugin* se debe reiniciar iReport, con ello el siguiente paso sera definir la conexión. Para realizar la conexión debe configurase el *host*, el puerto y base de datos que se va a conectar. Esto se expresa en una URL JDBC, como se muestra a continuación.

Figura 7. **Conexión entre iReport y MongoDB**



Fuente: Hadoop, the definitive guide. Consulta: 3 marzo 2014.

Con esa pequeña configuración y con MongoDB e iReport instalados, todo está listo para crear informes. El siguiente paso es seleccionar: archivo nuevo para iniciar el asistente de informes, elegir la plantilla, luego hacer clic en: Iniciar Asistente para informes. Al llegar al paso del asistente se requiere una consulta en el lenguaje MongoDbQuery para hacer la primera prueba, pero MongoDB no tiene un lenguaje de consulta, entonces para el primer informe se debe utilizar la consulta más simple posible:

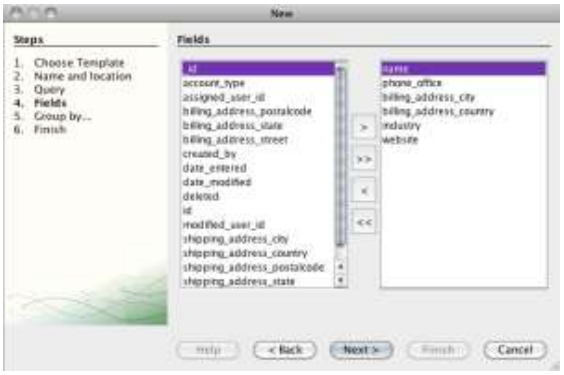
```
{'CollectionName': 'myCollection'}
```

Figura 8. **Consulta simple**

Fuente: Hadoop, the definitive guide. Consulta: 3 marzo 2014.

Luego se debe hacer clic en: Siguiente. Luego iReport ejecuta la consulta. Se conecta a MongoDB, lee cada documento de la colección especificada, encuentra cada campo y su tipo de datos y presenta los campos. Los documentos en MongoDB no necesariamente tienen los mismos campos. La lista de campos presentados es la unión de todos los campos que se encuentran en cada documento.

Figura 9. **Selección de campos**



Fuente: Hadoop, the definitive guide. Consulta: 3 marzo 2014.

Si se está trabajando con una colección muy grande, es probable que no se quiera esperar a iReport para leer todos los registros. Es posible que se configure para que lea sólo los primeros 100 documentos como este:

{'CollectionName': 'myCollection',
'rowsToProcess': 100}

Nota: el campo 'rowsToProcess' es de solo lectura. Esto significa que cuando se ejecuta, el informe seguirá procesando todos los documentos de la colección.

Al finalizar el asistente de informes se mostrará una vista previa del reporte. Se acaba de crear un informe con la informacion de una base de datos MongoDB sin tener que vaciar primero los datos en MySQL.

Figura 10. **Reporte visual**

Fuente: Hadoop, the definitive guide. Consulta: 5 marzo 2011.

5.1.3. Filtros para los reportes

La realización de informes por lo regular requiere filtros, un ejemplo de cómo realizarlos en la arquitectura de iReport es la siguiente:

```
{
  'CollectionName': "Cuentas"
  'FindQuery': {'nombre': 'M & Y Comunicaciones Takemura, Ltd'}
}
```

Un reporte avanzado no lleva la información de los filtros quemada, sino que debe llevar parámetros, la forma correcta para parametrizar es la siguiente:

```
{
  'CollectionName': "Cuentas"
  'FindQuery': {'nombre': '$ {P} CUSTOMER_NAME'}
}
```

5.1.4. Selección de campos y notación Dot

Los documentos en MongoDB pueden ser grandes, ya que ese es su objetivo. Las consultas anteriores devuelven cada campo en el documento. Cuando esto no es adecuado, es fácil crear una consulta para devolver solo los campos que se necesite.

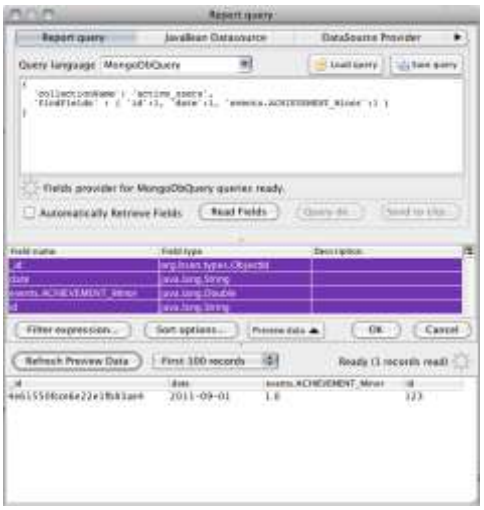
Por ejemplo, dado el caso en el que se tienen muchos usuarios y se quiere un informe que muestre que los usuarios habían registrado ciertos eventos. Podría haber grandes cantidades de eventos, pero solo hay un evento

en particular que importa en este informe. En lugar de devolver el documento completo para cada usuario se deben devolver los datos que interesan.

```
{
  'CollectionName': '' active_users,
  "FindFields": {"id": 1, 'date': 1, 'events.ACHIEVEMENT_Minor': 1}
}
```

Es posible escribirse esta sentencia en el campo de consulta iReport, y luego iReport es capaz de ejecutar la consulta y devolver solo los campos que interesan.

Figura 11. **Filtros en reportes**



Fuente: Hadoop, the definitive guide. Consulta: 5 marzo 2014.

Esto podría combinarse con un filtro, si solo se quiere regresar ciertos usuarios.

```
{
  'CollectionName': '' active_users,
```

```
  "FindFields": {"id": 1, 'date': 1, 'events.ACHIEVEMENT_Minor': 1},
  'FindQuery': {'events.ACHIEVEMENT_Major': 1}
}
```

6. PROCESOS AUTOMÁTICOS CON BASE EN DATOS MONGODB

El uso de procesos automáticos que ejecutan tareas en las bases de datos es muy común y muy útil, por lo que también hay formas de trabajar ETL sobre las bases de datos con paradigma NoSQL, por lo cual se da una descripción del uso de esta tecnología sobre MongoDB.

6.1. La integración de Spring Batch y MongoDB para ETL

Para la realización de ETL en MongoDB se debe realizar el proceso de integración de extremo a extremo, entre Spring Batch y MongoDB para aprovechar sus beneficios.

En las empresas de hoy en día la mayoría de aplicaciones son interactivas o corren en modo Batch. Las aplicaciones interactivas son como aplicaciones web en que requieran la intervención del usuario. Por el contrario las aplicaciones que necesitan iniciarse automáticamente sin intervención del usuario, a través de configuraciones que determinan el inicio y al finalizar realizan acciones dependiendo de los resultados de la ejecución anterior, son llamadas aplicaciones Batch, estas no requieren de frecuentes intervenciones manuales. Las aplicaciones Batch procesan enormes cantidades de datos. Estos incluyen cualquier herramienta ETL que extrae, transforma y carga datos a través de procesos por lotes.

A continuación se muestra un caso de estudio en el cual un marco ETL es aprovechando junto con las ventajas de Spring Batch y MongoDB, para

finalmente mostrar el contenido en un reporte configurado con iReport, lo que permite la carga por lotes a través de las bases de datos NoSQL.

La principal ventaja de las aplicaciones Batch es que no requiere ninguna intervención manual. Como resultado se pueden programar para que se ejecute en momentos cuando los recursos no están siendo utilizados, a modo de ejemplo, se realizó una herramienta ETL que se ejecuta en modo Batch para analizar Weblogs. Se analizó en una base diaria para ir a buscar la información útil requerida. Los archivos de entrada son extraídos y procesados para obtener la información requerida, y la salida de datos se carga a una base de datos.

Todo este proceso se lleva a cabo en modo de Batch, los procesos Batch se ocupan principalmente de grandes cantidades de datos en una serie de programas que corren para alcanzar el objetivo deseado. Estos programas se pueden ejecutar uno después del otro, o pueden funcionar en paralelo para acelerar la ejecución dependiendo de los requisitos. El procesamiento por lotes permite compartir los recursos, estos procesos se ejecutan principalmente hacia el final del día, cuando los no se, están trabajando con los datos y hay más recursos disponibles.

El marco de Spring Batch está diseñado para atender a las aplicaciones de proceso Batch que se ejecutan a diario en las organizaciones empresariales. Le ayuda a aprovechar los beneficios de la infraestructura Spring junto con los servicios avanzados. Spring Batch se utiliza principalmente para procesar gran volumen de datos, ofrece un mejor rendimiento y es altamente escalable mediante la optimización de diferentes técnicas de partición. También proporciona ventaja sobre el registro/rastreo, manejo de la gestión de transacciones, las estadísticas de procesamiento de trabajos, el reinicio de

trabajo, escaleras y recursos. Al utilizar el modelo de programación Spring, se puede escribir la lógica de negocio.

Batch Spring incluye tres componentes: la aplicación de lote, entorno de ejecución de lotes y lotes infraestructura.

El componente de aplicación contiene todos los trabajos por lotes y código personalizado escrito usando lotes de Spring.

El componente básico contiene las clases de tiempo de ejecución básicas necesarias para poner en marcha y controlar un trabajo por lotes. Esto incluye cosas tales como: una JobLauncher, Job, y las implementaciones de Paso. Tanto la aplicación y Core son construidos en la infraestructura común.

La infraestructura incluye lectores, escritores y servicios que son utilizados tanto por la aplicación y el marco básico en sí. Estos incluyen cosas como ItemReader, ItemWriter y MongoTemplate. Para utilizar el marco de lotes de Spring basta configurar y personalizar los archivos XML. Todos los servicios básicos existentes deben ser fáciles de reemplazar o extender, sin ningún impacto a la capa de infraestructura.

6.2. Spring Batch – integración con MongoDB

A continuación se demostrará la integración de Spring Batch con MongoDB para procesos automáticos.

6.2.1. Dividir un archivo de datos de gran tamaño

A medida que el archivo de datos de entrada es más grande, se puede dividir antes de cargarlo. Si se intenta cargar el archivo enorme secuencial, tardará mucho tiempo. Por lo tanto, es aconsejable dividir el archivo grande en partes pequeñas, el archivo grande puede dividirse utilizando cualquier lógica y múltiples partes se pueden cargar a los servidores de diferentes formas presentes en el clúster, de manera que las partes diferentes de archivos se pueden cargar en paralelo para una ejecución más rápida.

Se muestra un ejemplo de código para la FileSplitter, que toma la ruta de archivo de datos y el número de piezas que desea crear para ese archivo. También requiere que se designe a la carpeta de salida donde desea almacenar los archivos de partes divididas. Idealmente se asume que el número de piezas será el mismo que el número de servidores presentes en el *cluster*.

Primero se crea los objetos de muchos archivos con la ruta de la carpeta de salida y los almacena en fileNamesList. Sus objetos BufferedWriter correspondientes se crean y se almacenan en una lista vector. Luego lee el archivo línea por línea y escribe los datos en archivos divididos diferentes que fileNamesList, que se ha creado usando sus objetos BufferedWriter correspondientes de la lista vector. Después que todos los archivos divididos se crean, se puede transferir los archivos a través de servidores distintos mediante la clase FileTransfer en el mismo lugar que se ha dado en la carpeta de salida.

Aquí se ha asumido que sólo hay dos equipos del clúster y dos partes se crearan. Una parte se mantiene en el mismo servidor en el que ejecute el FileSplitter y el otro se transfiere a la máquina cuyos detalles se dan en la clase FileTransfer. A partir de ahora se han codificado los datos del segundo servidor

en la clase FileTransfer, pero se puede configurar la información del servidor mediante la lectura desde los archivos de propiedades. Por ejemplo, si el archivo principal es enorme employee.txt, la pieza se crea en la carpeta de salida llamada employeePart1.txt y employeePart2.txt.

FileSplitter.java

```
public class FileSplitter
{
     public static void main(String args[])
     {
int noParts = new Integer(args[0]); // no. of parts
          String inputFile = args[1]; // input file path
          String outputFolder = args[2]; // output folder path
          List<String> fileNamesList = new ArrayList<String>();
          FileInputStream fstream;
          fstream = new FileInputStream(inputFile);

          // Get the object of DataInputStream
          DataInputStream in = new DataInputStream(fstream);
          BufferedReader br = new BufferedReader(new InputStreamReader(in));
          String strLine;
          Vector<BufferedWriter> vList = new Vector<BufferedWriter>();
          for(int i=0; i<noParts; i++)
          {
               int lastIndex = inputFile.lastIndexOf("\\");
               int pointIndex = inputFile.indexOf(".");
               String justFileName = inputFile.substring(lastIndex, pointIndex);
               FileWriter fWriter = new FileWriter(outputFolder + "\\" +
justFileName + "Part"+i+".txt");
String fileName = outputFolder + "\\" + justFileName +
                                                  "Part"+i+".txt";
               fileNamesList.add(fileName);
               BufferedWriter bWriter = new BufferedWriter(fWriter);
               vList.add(bWriter);
          }
//Read File Line By Line
          int partCounter = noParts;
          int noOfPart = 0;
          while ((strLine = br.readLine()) != null)
          {
               if(noOfPart == partCounter)
                    noOfPart = 0;
               if(noOfPart < partCounter)
               {
                    vList.get(noOfPart).write(strLine);
                    vList.get(noOfPart).newLine();
               }
               noOfPart++;
          }
          for(int j=0; j<noParts; j++)
          {
               vList.get(j).close();
          }
          in.close();
          FileTransfer ft = new FileTransfer();
          for(int j=0; j<noParts; j++)
          {
               if(Math.IEEEremainder(new Double(j), new Double(2)) == 0)
```

```
                {
                     ft.transferFile(fileNamesList.get(j), outputFolder);
                     new File(fileNamesList.get(j)).delete();
                }
           }
      }
}
```

FileTransfer.java

```
public class FileTransfer
{
   public void transferFile(String fileName, String outputFolder)
   {
         String username = "some_username";
          String host = "some_host_name_ip_address";
           String pwd = "some pwd";

       JSch jsch = null;
       Session session = null;
       Channel channel = null;
       ChannelSftp c = null;

jsch = new JSch();
           session = jsch.getSession(username, host, 22);
          session.setPassword(pwd);
java.util.Properties config = new java.util.Properties();
       config.put("StrictHostKeyChecking", "no");
          session.setConfig(config);
         session.connect();

          channel = session.openChannel("sftp");
           channel.connect();
         c = (ChannelSftp) channel;
String fsrc = fileName, fdest = outputFolder;
        c.put(fsrc, fdest);
 c.disconnect();
       session.disconnect();
   }
}
```

Esto es sólo una muestra FileSplitter, pero puede usar varias lógicas disponibles para dividir los archivos. Ahora se verá la integración real de los Spring Batch con MongoDB para llevar a cabo el proceso de carga.

6.2.2. Configuración del archivo job-repository.xml

Spring Batch requiere un repositorio de trabajo para almacenar los detalles de la aplicación y también otra información relacionada con trabajo y los pasos. Este depósito puede crearse en una base de datos o en la memoria.

Se va a utilizar el repositorio de trabajo basado en memoria en esta demostración.

JOB-REPOSITORY.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2
001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springfra
mework.org/schema/beans/spring-beans-2.0.xsd">
<bean id="jobRepository" class="org.springframework.batch.core.repository.support.SimpleJ
obRepository" >
          <constructor-arg>
<bean class="org.springframework.batch.core.repository.dao.MapJobInstanceDao" />
          </constructor-arg>
          <constructor-arg>
<bean class="org.springframework.batch.core.repository.dao.MapJobExecutionDao" />
          </constructor-arg>
          <constructor-arg>
<bean class="org.springframework.batch.core.repository.dao.MapStepExecutionDao"/>
          </constructor-arg>
          <constructor-arg>
<bean class="org.springframework.batch.core.repository.dao.MapExecutionContextDao"/>
          </constructor-arg>
     </bean>
<bean id="asyncTaskExecutor" class="org.springframework.core.task.SimpleAsyncTaskExecutor
"/>
<bean id="jobRepository-
transactionManager" class="org.springframework.batch.support.transaction.ResourcelessTran
sactionManager"/>
<bean id="jobLauncher" class="org.springframework.batch.core.launch.support.SimpleJobLaun
cher" >
          <property name="jobRepository" ref="jobRepository"/>
     </bean>
</beans>
```

6.2.3. Configuración job.xml para cargar los datos desde un único archivo de colección MongoDB (tabla)

En primer lugar, definir el Job.xml (FileToMongoTableJob.xml en esta demostración). En este archivo se especifica el FlatFileItemReader, que es una clase del marco de Batch Spring. Se puede especificar el recurso al FlatFileItemReader como la ruta del archivo de entrada, aquí se indica que el valor de los recursos es del archivo: d:\data\employee.csv, es decir; la ubicación del archivo de entrada employee.csv. También se define el delimitador, que en este caso es un separador de coma a través de la clase DelimitedLineTokenizer. Luego se define la propia clase

EmployeeFieldSetMapper, que implementa el marco de Batch Spring la clase FieldSetMapper, esta clase se une a los valores de ResultSet para los campos de la tabla. Si hay cualquier cálculo o proceso en cuestión que puede atender a través de EmployeeProcessor, que implementa a ItemProcessor.

Después de esto se puede especificar los detalles en MongoDB, al mencionar el nombre del *host* donde está instalada la base de datos y también el número de puerto. Acceder a la base de datos a través de la MongoTemplate, que toma la referencia de base de datos de los detalles mencionados a través de la ID (es decir, Mongo como el argumento). En la MongoTemplate también se pasa el otro argumento (es decir, el nombre de la base de datos que va a trabajar con el interior de la MongoDB), y en este caso es nueva. Ahora se define la clase propia, MongoDBItemWriter, que es la extensión de la ItemWriter clase en Bach Spring. Esta clase lee el MongoTemplate para obtener los detalles de la base de datos.

A continuación se especifica el DynamicJobParameters clase, que implementa la JobParametersIncrementer del Batch Spring. Esto funciona como el incrementador para el trabajo.

FileToMongoTableJob.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:batch="http://www.springframework.org/schema/batch"
     xmlns:beans="http://www.springframework.org/schema/beans"
     xmlns:aop="http://www.springframework.org/schema/aop"
     xmlns:tx="http://www.springframework.org/schema/tx"
     xmlns:p="http://www.springframework.org/schema/p"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:mongo="http://www.springframework.org/schema/data/mongo"
     xsi:schemaLocation="
          http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
     http://www.springframework.org/schema/batch
     http://www.springframework.org/schema/batch/spring-batch-2.0.xsd
     http://www.springframework.org/schema/aop
     http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
     http://www.springframework.org/schema/tx
     http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
```

```
     http://www.springframework.org/schema/data/mongo
     http://www.springframework.org/schema/data/mongo/spring-mongo-1.0.xsd">

     <beans:import resource="JOB-REPOSITORY.xml"/>

     <bean id="employeeFileItemReader"      class="org.springframework.batch.item.file.Fl
atFileItemReader">
          <property name="resource" value="file:d:\data\employee.csv" />
          <property name="lineMapper">
<bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper">

               <property name="lineTokenizer">
<bean class="org.springframework.batch.item.file.transform.DelimitedLineTokenizer">
                         <property name="delimiter" value=","/>
<property name="names" value="id,name,city,designation,joiningYear,terminationYear" />
                    </bean>
               </property>
               <property name="fieldSetMapper">
<bean class="com.infosys.springbatch.mongo.example.EmployeeFieldSetMapper" />
               </property>
          </bean>
     </property>
     </bean>

<bean id="employeeProcessor" class="com.infosys.springbatch.mongo.example.EmployeeProcess
or"/>

     <mongo:mongo id="mongo" host="localhost" port="27017">
         </mongo:mongo>

<bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">
           <constructor-arg ref="mongo"/>
           <constructor-arg name="databaseName" value="new"/>
     </bean>

<bean  id="mongoDBItemWriter" class="com.infosys.springbatch.mongo.example.MongoDBItemWri
ter">
          <property name="mongoTemplate" ref="mongoTemplate"/>
     </bean>

<bean id="dynamicJobParameters" class="com.infosys.springbatch.mongo.example.DynamicJobPa
rameters" />

<batch:job id="employeeProcessorJob" job-
repository="jobRepository" incrementer="dynamicJobParameters">
          <batch:step id="step1">
<batch:tasklet transaction-manager="jobRepository-transactionManager" >
<batch:chunk reader="employeeFileItemReader" processor="employeeProcessor" writer="mongoD
BItemWriter"  commit-interval="10" />
               </batch:tasklet>
          </batch:step>
     </batch:job>
</beans>
```

6.2.4. Los archivos de clase utilizados en la definición de la Jobs.xml

A continuación se muestra la clase Employee POJO, que contiene los detalles y atributos del empleado con sus correspondientes métodos de *getter/setter*, que no se muestran aquí.

Employee.java

```
package com.infosys.springbatch.mongo.example;

import java.io.Serializable;

public class Employee implements Serializable
{
private static final long serialVersionUID = 1L;
     private String id;
     private String name;
     private String city;
     private String designation;
     private int joiningYear;
     private int terminationYear;
     private int tenure;
}
```

A continuación los mapas de clase, teniendo en cuenta los datos *fieldset* atribuyen al empleado y crea un objeto empleado.

EmployeeFieldSetMapper.java

```
package com.infosys.springbatch.mongo.example;

import org.springframework.batch.item.file.mapping.FieldSetMapper;
import org.springframework.batch.item.file.transform.FieldSet;

public class EmployeeFieldSetMapper implements FieldSetMapper<Employee>
{
     public Employee mapFieldSet(FieldSet fs)
     {
          if(fs == null)
          {
               return null;
          }
          Employee employee = new Employee();
          employee.setId(fs.readString("id"));
          employee.setName(fs.readString("name"));
          employee.setCity(fs.readString("city"));
          employee.setDesignation(fs.readString("designation"));
          employee.setJoiningYear(fs.readInt("joiningYear"));
          employee.setTerminationYear(fs.readInt("terminationYear"));
          return employee;
```

6.2.5. La ejecución de los trabajos mencionados en FileToMongoTableJob.xml y MultipleFileToMongoTableJob.xml

Para ejecutar los trabajos, se tienen que crear configuraciones de ejecución para cada uno de los puestos de trabajo:

Para cargar datos desde una sola fila a la tabla MongoDB es necesario crear y ejecutar la configuración donde la clase principal es org.springframework.batch.core.launch.support.CommandLineJobRunner y los argumentos son el archivo XML de definición, identificando del trabajo mencionado en el xml, y el trabajo incrementales FileToMongoTableJob.xml employeeProcessorJob employee.id = 1001.

6.3. Casos de estudio: implementación exitosa y fallida

Es importante hacer énfasis en casos reales que hayan sido implementados exitosamente y funcionen a gran escala, de esta forma se puede comparar la nueva tecnología propuesta en este trabajo de graduación con las tecnologías existentes.

6.3.1. Maximizando el valor de los procesos que funcionan con ERP

Con una larga experiencia (Manuel Escudero, 2014) implementando bases de datos con MongoDB y iReport, se considera a MongoDB como la mejor base de datos disponible actualmente. Se trata de un proyecto OpenSource de base de datos no relacional pero que engloba varias de las características y comportamientos de las bases de datos relacionales a las que

muchos están acostumbrados. Conviene usar MongoDB, porque es una base de datos donde en lugar de guardar records en base a tablas y otros esquemas organizativos que pueden suponer bastante trabajo extra, los guarda en documentos, describiéndolos prácticamente como son, evitándo así el crear archivos extra.

Es bastante bueno también porque tiene excelente rendimiento y escalabilidad, hacer relaciones y queries de datos complejas es bastante sencillo y práctico, además la DB es dinámica. Esto quiere decir que conforme cambie el documento de base de datos se podrá añadir nuevas columnas, nuevos datos, y otros, reflejando los cambios en la aplicación al instante.

Al utilizar iReport se puede acceder de la forma más simple a los datos y generar reportes con la misma facilidad que se haría con herramientas que trabajan con bases de datos relacionales. La ventaja es que es mucho más rápido que trabajar con bases de datos relacionales, y es por ello que se recomienda este paradigma al momento de trabajar volúmenes de datos muy grandes.

6.4. Entrevistas: un acercamiento profesional

Las experiencias compartidas por profesionales que han trabajado con el nuevo paradigma y la nueva tecnología, es otro parámetro que se debe considerar en el análisis que se le da al trabajo de graduación, en busca de obtener resultados que permitan facilitar la toma de decisión en cuando a la metodología que se implementara antes de iniciar un nuevo proyecto.

6.4.1. Mejoramiento de los procesos y del nivel de desempeño de las organizaciones, orientado a los beneficios

El ingeniero César Trigo Esteban[1] indica que como responsable del BackEnd en Gigigo se encarga principalmente de la coordinación y dirección del equipo de BackEnd. Siendo esta una compañía especializada en el desarrollo y *marketing* móvil, que sustenta sus soluciones sobre grandes sistemas de BackEnd, que aportan gran agilidad y flexibilidad a los proyectos móviles.

Dados los tiempos que corren en el panorama móvil y los miles de millones de datos que hoy en día generan las Apps, el Big Data está muy presente en los negocios de Gigigo Group. Esta empresa cuenta con un equipo de Backend con gran experiencia y altamente especializado en tecnologías NoSQL, principalmente en MongoDB, lo cual permite acompañar a los clientes a largo plazo, integrando las aplicaciones nativas con sus estrategias y aportando un valor extra a su negocio.

Por otro lado, el ingeniero César Trigo indica que es el encargado del departamento de HTML5, en el que se ha apostado por MEAN Stack para los desarrollos y que cada vez toma más peso gracias al crecimiento y adopción del HTML5, y la agilidad proporcionada por tecnologías como Node.JS y MongoDB del lado de servidor.

Concretamente en lo que a Big Data se refiere, lo más importante es saber diferenciar qué datos son realmente importantes para el cliente y cuáles no, pero también conocer las necesidades técnicas.

[1] Ing. César Trigo Esteban. Director de desarrollo de los equipos de Backend y HTML5 en Gigigo Group, MongoDB Master, fundador de MongoDBSpain (*www.mongodbspain.com*) y coorganizador del MUG Madrid. (entrevista Julio 2014)

Aunque el NoSQL es algo que lleva mucho tiempo, su popularización viene dada por la heterogeneidad de los datos que hoy en día manejan las empresas. Es una necesidad que tarde o temprano debía manifestarse para el manejo de cantidades ingentes de información no relacionada y cada vez son más los que apuestan por esta filosofía.

La popularidad de MongoDB se ve especialmente potenciada debido a su sencilla curva de entrada y la inmensa cantidad de recursos existentes para los nuevos desarrolladores que, tras muchos años se han construido sistemas relacionales, que deben cambiar radicalmente su forma de pensar. La principal ventaja que aporta es la flexibilidad de almacenamiento de los datos, permitiendo trabajar con ellos de forma aislada o conjunta, a gusto del consumidor y dependiendo de cada escenario, contando además con un gran ecosistema de herramientas y productos de que facilitan las tareas de análisis.

6.4.2. MongoDB Master

Se trata de un grupo de profesionales que de una forma u otra han colaborado de forma activa en el crecimiento de la comunidad MongoDB en todo el mundo. Este grupo cuenta con numerosas ventajas para sus integrantes, siendo una de las más interesante la posibilidad de compartir foro con los mejores expertos de MongoDB en el mundo, teniendo además acceso a las últimas novedades y previews de las versiones de MongoDB y pudiendo disponer siempre de un lugar de consulta y aprendizaje de la mano de grandes profesionales que han adoptado esta tecnología en situaciones del mundo real. Lo que se espera es el soporte a la comunidad en cada una de las zonas locales.

6.4.3. En qué casos es recomendable hacer uso de NoSQL-MongoDB

Es algo que depende absolutamente del tipo de proyecto, pero a la vez también es importante evaluar en qué se convertirá ese proyectos a medio / largo plazo.

Se podría decir que NoSQL está especialmente pensado para todos aquellos casos en los que se manejan cantidades de información muy grande, que no necesariamente debe estar relacionada entre si y por tanto su organización mediante los sistemas de base de datos más tradicionales es compleja. Además de esto suele enfocarse para aquellos casos en los que el tiempo es oro y por tanto se requiere un rendimiento y capacidad de análisis extremos. En estos aspectos, MongoDB ayuda cuando existen las necesidades de crecimiento muy grandes e inesperadas, permitiendo adaptarse fácilmente a los cambios que puedan surgir a lo largo de su ciclo de vida. Además cuenta de base con determinadas herramientas de análisis sin necesidad de acudir a terceros productos.

No obstante, es totalmente recomendable como soporte de almacenamiento para pequeños proyectos en los que simplemente se quieren almacenar algunos datos. En este sentido al no requerir que sus datos sigan un esquema concreto, MongoDB permite empezar mucho antes las fases de desarrollo e ir adaptando el esquema de datos sobre la marcha según las necesidades del proyecto.

El mayor error suele ser trasladar sistemas claramente relacionales a bases de datos NoSQL, lo que supone principalmente el desaprovechamiento

de las ventajas de rendimiento que un sistema NoSQL tiene sobre algunos relacionales.

6.4.4. Mejoras que se ven venir en MongoDB

Desde el punto de vista técnico, una de las grandes mejoras que todos están esperando fue anunciada en el MongoDB World, que tuvo lugar el verano 2014 en New York y supondrá un antes y un después en el uso de MongoDB y sus herramientas de reporteria. Se trata del bloqueo a nivel de documento (document-level locking), que verá la luz con la versión 2.8 de MongoDB antes del final de 2014 y que se centra en las operaciones de escritura de base de datos, incrementando la capacidad de concurrencia de dichas operaciones de forma exponencial y por tanto mejorando en la misma medida el rendimiento de las aplicaciones que implementan MongoDB.

Por otro lado, se deberá afrontar el gran reto de consolidarse como uno de los proyectos Open Source más grandes del mundo y todo lo que ello supone, principalmente desde el punto de vista del modelo de negocio. En este sentido ya se han dado grandes pasos como el recién anunciado soporte a entornos de producción para aquellos que usen la versión Community, privilegio que de momento estaba reservado para las versiones Enterprise.

6.5. Razones por las cuales utilizar MongoDB en lugar de RDBMS

RDBMS ha gobernado el espacio de almacenamiento durante décadas, es por ello que surge la pregunta del porqué de repente necesito utilizar MongoDB.

En un cierto conjunto de industrias el almacenamiento y gestión de datos tan grandes se ha convertido en un reto, y los RDBMS tradicionales no podían hacer frente a las necesidades. Como una alternativa, las bases de datos NoSQL llegaron para sustituirlas. Como su nombre lo indica, NoSQL no tiene ningún lenguaje de consulta, y la base de datos no tiene ningún esquema de tabla fija. Estas bases de datos generalmente almacenan los datos como pares clave-valor, big table, almacén de documentos, gráficos. Son de código abierto, distribuido y escalar a diferencia de las bases de datos relacionales. Este tipo de base de datos perfeccionan y aprovechar los nuevos nodos y fueron diseñados con hardware de bajo costo en mente. Proporcionan una alta escalabilidad, mejoran el rendimiento, replican fácil y una mayor optimización en la consulta de datos e inserciones.

MongoDB es una base de datos NoSQL, tal que es de código abierto y orientado al documento. En lugar de almacenar los datos en tablas, como en cualquier base de datos relacional, las tiendas de MongoDB estructuran data como JSON como documentos con esquemas dinámicos. MongoDB da apoyo a las consultas *ad hoc*, indización, replicación de datos y balanceo de carga. Se puede utilizar como un sistema de archivos y los usuarios pueden beneficiarse de su replicación y balanceo de carga para almacenar los archivos de varios servidores.

CONCLUSIONES

1. Con NoSQL y MongoDB, las bases de datos a nivel mundial han experimentado cambios en rendimiento. En un principio se pensó que las bases de datos con el paradigma NoSQL vendrían a reemplazar a las bases de datos relacionales. Como se ha demostrado en este trabajo una tecnología no puede remplazar a la otra. Se ha hecho un balance entre ambas tecnologías, determinando que las bases de datos NoSQL se deben utilizar en casos en los cuales se tendrán grandes volúmenes de datos y sus entidades no necesariamente están relacionadas. MongoDB permite realizar consultas más rápidas cuando se posee grandes volúmenes de información sin normalizar.

2. Estos paradigmas no buscan una rivalidad de eficiencia entre ellos, al contrario gracias al transcurso de los años se han desarrollado técnicas bastante comunes para realizar normalización en la medida de lo posible, escalarlas según crece la demanda, y utilizarlas como sistema de persistencia para almacenar información desde nuestro lenguaje procedural u orientado a objetos (entre otros). La generalidad en el uso de software como SQLite, MySQL, PostgreSQL u Oracle, por poner cuatro ejemplos de sistemas conocidos, es muy alta, encontrándose en varios de los desarrollos modernos.

3. Los sistemas basados en el paradigma NoSQL atacan de frente el problema de la baja escalabilidad, proponiendo una estructura de almacenamiento más versátil. Esto trae como consecuencia la pérdida de algunas funcionalidades de las bases de datos relacionales, tales como

las transacciones que engloban operaciones en más de una colección de datos, o la incapacidad de ejecutar el producto cartesiano de dos tablas (también llamado JOIN) teniendo que recurrir a la desnormalización de datos.

4. El paradigma NoSQL no es perfecto y ninguna herramienta de software lo es, pero en determinados entornos donde se ha de escalar rápidamente es una solución muy buena, en especial por el alto rendimiento que ofrece. Hoy en día es utilizada con mucha frecuencia, no solo para almacenamiento primario, sino también como sistema de persistencia para guardar cachés, analíticas de uso, y otros sistemas de información para los que lo que la velocidad es de suma importancia.

5. La carga masiva de datos para su posterior uso es un tema que se ha abordado y ejemplificado, pues es de suma importancia para la generación de reportes en el ambiente de MongoDB. Spring Batch, ofrece las características necesarias para realizar esta tarea al hacer cargas masivas por lotes, facilitando así la transformación de los datos para su posterior uso en la toma de decisiones a través de los reportes que se generan posteriormente.

6. La creación de reportes en este tipo de modelo de datos es complicada, pero existe varias formas de hacerlo como se vio a lo largo de este trabajo de graduación, algunas de las técnicas para generar reportes en MongoDB son costosas, otras son complicadas. Existen técnicas para generar reportes con las cuales se pierden muchas de las características del paradigma NoSQL, por lo cual se ha buscado una alternativa que permita conservar las bondades de NoSQL evitando el alto costo de desarrollo. Esto se ha logrado haciendo uso de una librería llamada

iReport, cuyo uso se ha ejemplificado en este documento. NoSQL no substituye a los modelos relacionales sino más bien viene a complementar el trabajo de las bases de datos para la demanda actual, y la forma de generar reportes también ha evolucionado con ellos.

7. La librería JasperReport proporciona una herramienta llamada iReport de soporte, que permite editar informes complejos mediante un editor gráfico permitiendo la inclusión de subinformes, gráficos e imágenes. Esta desarrollada en código Java y es de libre distribución. Entre las principales características de la herramienta se encuentra que es escrito en Java. Además, OpenSource y gratuito, todas estas son características que hacen de iReport la mejor opción para generación de reportes en MongoDB.

RECOMENDACIONES

1. Se recomienda el uso de NoSQL para modelos tipo estrella, los cuales no están normalizados, ya que es en este tipo de esquemas en el que mejor trabaja NoSQL. Este paradigma también funciona mejor con grandes volúmenes de datos. A lo largo de este trabajo de graduación se vieron muchas técnicas para la creación de reportes, pero ampliamente se recomienda utilizar la herramienta iReport en MongoDB. Utilizando el conector descrito en el trabajo de graduación, es la forma más sencilla y práctica de hacerlo. Además todo lo visto en este trabajo de graduación es Open Source, esto quiere decir que no hay problemas de licencia al momento de utilizarlo en áreas comerciales y de lucro.

2. Otras de las características por las cuales se recomienda el uso de iReport, es porque esta herramienta es de fácil aprendizaje y de interface amigable para el usuario, permite la edición de los informes complejos mediante un editor gráfico, permitiendo la inclusión de gráficos, imágenes y subinformes. Las principales características de la herramienta son que es escrito en Java y además OpenSource y gratuito, maneja el 90 por ciento de las etiquetas de JasperReports.

3. El uso de iReport permite diseñar reportes con sus propias herramientas, entre las posibilidades se incluye rectángulos, líneas, eclipses, campos de los textfields, cartas, subinformes, además soporta JDBC.

4. Para el aprendizaje de iReport y MongoDB se recomienda el sitio web MongoDB University (https://university.mongodb.com/), un completo sitio de aprendizaje en el que se ofrecen numerosos cursos de formación totalmente gratuitos orientados a diferentes perfiles y niveles. Existen cursos para DBAs, administradores de sistemas y desarrolladores sobre diferentes tecnologías. Los cursos consisten en una sucesión de lecciones en vídeo, prácticas, ejercicios y una prueba final con la que demostrar los conocimientos adquiridos.

5. Para las personas que ya han trabajado con MongoDB y lo que buscan es profundizar más en determinadas materias o evaluar casos de uso reales. Existen otras alternativas como las numerosas presentaciones y Webinars que se organizan y que quedan recogidas en su web oficial (eventos, presentaciones, Webinars).

BIBLIOGRAFÍA

1. FERNÁNDEZ, Daniel. *Definición de una arquitectura, software para el diseño de aplicaciones web basadas en tecnología java J2EE* [en línea]. Universidad de Oviedo, España. htt://www.di.uniovi.es/dflanvin/ doctorado/ ArquitecturaJ2EE. [Consulta: 24 de diciembre de 2014].

2. FLOWER, Martin. *Contenedores de inversión de control y el patrón de inyección de dependencias* [en línea]. Chicago, IL. http://www.programacion.com/contenedores_de_iversion_de_control_y_el_patron_de_inyecciion_de_dependencias_304#1_componentes. [Consulta: 15 de enero de 2014].

3. MEDIN, Juan. *Hacia una arquitectura con JSF, Sprin, Hibernate y otros Frameworks* [en línea]. Tenimap, Sevilla, España. 2013. http://administracionelectronica.gob.es/archivos/pae_000002295.pdf. [Consulta: 15 de enero de 2014].

4. O'REGAN, Graham. *Introduction to Aspect-Oriented Programing.* [en línea]. htpp://tim.orelly.com/pub/a/onja va/2004/01/14 /aop.html?page=2 [Consulta: 15 de enero de 2014].

5. RAMNIVAS, Laddad. *I Want AOP* [en línea] USA, JavaWorld.com, http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html? page =2. [Consulta: 10 de febrero de 2014].