



UNIVERSIDAD DEL AZUAY
FACULTAD DE CIENCIAS DE LA
ADMINISTRACIÓN

ESCUELA DE INGENIERÍA DE SISTEMAS Y TELEMÁTICA

**“Análisis de rendimiento entre la base de datos relacional: MySQL y una
base de datos no relacional: MongoDB”**

**Trabajo de Titulación Previo a la Obtención del título de Ingeniero de
Sistemas y Telemática**

Autor:

César Augusto Vele Zhingri

Director:

Ing. Marcos Orellana Cordero

Cuenca – Ecuador

2015

DEDICATORIA

Esta tesis está dedicada a mis amados padres Víctor y Martha, quienes han sido una fuente constante de apoyo y estímulo, y quienes han hecho sacrificios por nosotros, sus hijos, para que pudiéramos tener las oportunidades que ellos nunca tuvieron. Estoy verdaderamente agradecido por tenerlos conmigo.

A mi gran amigo y mentor Marcelo Bueno, quien me inició en el campo de las TI, lo cual me llevó a escoger esta grandiosa carrera.

AGRADECIMIENTOS

Esta investigación no hubiera podido ser realizada sin la ayuda y apoyo de muchas personas:

A mi director de tesis, el Ing. Marcos Orellana, por estar ahí desde el inicio hasta el final de la investigación proporcionándome guía y consejo. Estoy especialmente agradecido por haberme dado la libertad y el espacio para explorar y descubrir las cosas por mí mismo, mientras se mantuvo pendiente de que no me desviara de los objetivos de la investigación.

A mis compañeros y grandes amigos: Darío y Francisco, por su paciencia y apoyo en algunos temas en los cuales yo desconocía totalmente. Su ayuda no será olvidada pronto.

A mi hermano Víctor, que siempre estuviste ahí cuando más lo necesitaba y nunca me pediste nada a cambio. Siempre admiraré tu capacidad muy superior a la mía.

Gracias

Índice de Contenidos

1	Sistematización del problema.....	9
1.1	Planteamiento del problema.....	9
1.2	Objetivos	11
1.2.1	Objetivo General	11
1.2.2	Objetivos Específicos.....	11
1.3	Justificación	11
1.4	Alcances y Limitaciones.....	12
1.4.1	Los Alcances.....	12
1.4.2	Las Limitaciones.....	12
	CAPÍTULO 2 - MARCO TEÓRICO	13
2.1	Bases de datos relacionales.....	13
2.1.1	Introducción.....	13
2.1.2	El modelo relacional.....	13
2.1.3	Estructura de las Bases de Datos Relacionales	13
2.1.4	Esquema de la base de datos	15
2.1.5	Restricciones de Integridad	15
2.2	Bases de datos no relacionales.....	16
2.2.1	Introducción.....	16
2.2.2	Porqué utilizar NoSQL?	16
2.2.3	Características	18
2.2.4	Tipos de Bases de Datos NoSQL	21
2.2.5	Comparación entre Bases de Datos NoSQL.....	25
2.2.6	Modelos de consultas.....	27
2.2.7	Seguridad	30
2.3	Diferencias conceptuales entre RDBMS y NoSQL DBMS	31
2.4	MySQL.....	32
2.4.1	Arquitectura del Sistema	33
2.4.2	Catálogo del Sistema	36
2.4.3	Administración de Conexiones y Seguridad	36
2.4.4	Ejecución y Optimización de Consultas	37
2.4.5	Control de Concurrencia	37

2.4.6 Transacciones	41
2.4.7 Motores de Almacenamiento.....	42
2.5 MongoDB	47
2.5.1 Modelo de Datos	50
2.5.2 Modelo de Consultas	52
2.5.3 Indexación.....	53
2.5.4 Motores de Almacenamiento.....	55
2.5.5 BSON.....	55
2.6 Comparación entre MongoDB y MySQL	56
2.6.1 Terminología y conceptos	56
2.6.2 Lenguajes de Consulta.....	57
2.6.3 Relaciones.....	57
2.6.4 Conjunto de Características.....	58
2.6.5 Definición de Esquema.....	58
CAPÍTULO 3 - INSTALACIÓN	59
3.1 Análisis de Requisitos.....	59
3.2 Instalación de MySQL	59
3.2.1 Diseño de Instalación por defecto en Windows	59
3.2.2 Determinación de la versión a instalar	59
3.2.3 Primeros pasos.....	60
3.3 Instalación de MongoDB	62
3.3.1 Primeros pasos.....	63
CAPÍTULO 4 - ANÁLISIS DE RENDIMIENTO.....	67
4.1 Descripción del problema	67
4.2 Plataforma Hardware.....	67
4.3 Plataforma Software	67
4.4 Esquema de la base de datos	68
4.5 Consultas a los gestores de bases de datos	69
4.6 Métricas.....	72
4.7 Pruebas de Rendimiento.....	73
4.7.1 Operación: Inserción masiva de registros.....	73
4.7.2 Operación: Consulta masiva	73
4.7.3 Operación: Eliminación Masiva.....	75
4.8 Análisis de Resultados.....	75
4.9 Tendencia a futuro de los sistemas gestores de bases de datos.....	79
CONCLUSIONES.....	81

ÍNDICE DE ABREVIATURAS	83
GLOSARIO	84
ANEXO 1 – SCRIPTS UTILIZADOS PARA LAS PRUEBAS DE RENDIMIENTO	89
ANEXO 2 – RESULTADOS DE LAS PRUEBAS DE RENDIMIENTO	101

Índice de Ilustraciones

Figura 1 Tabla o Relación "Productos"	14
Figura 2 Representación Simbólica de NoSQL	16
Figura 3 Transacciones Big Data con Interacciones y Observaciones (Zaki, 2013).....	18
Figura 4 El teorema CAP, indica que se puede realizar dos operaciones a la vez (Alarcon, 2014).	21
Figura 5 Almacenamiento Clave-Valor (Antiñanco, 2013)	22
Figura 6 Colección de documentos complejos con formatos de datos arbitrarios, anidados y un formato de “registros variables” (MarkLogic, 2014)	23
Figura 7 La familia de bases de datos orientadas a columnas como Cassandra organiza los datos a través de un clave de fila que es asociada con cualquier número de columnas (MarkLogic, 2014).	23
Figura 8 Ejemplo de una base de datos orientada a grafos (Moniruzzaman & Hossain, 2013). 24	
Figura 9 Estado actual de las bases de datos NoSQL (Moniruzzaman & Hossain, 2013).....	25
Figura 10 Comparación de cuatro categorías de bases de datos NoSQL en base a atributos como diseño, integridad, indexación, distribución, sistema (Moniruzzaman & Hossain, 2013).26	
Figura 11 Posibilidades de Consulta (Hecht & Jablonski, 2011).....	29
Figura 12 Arquitectura de MySQL Server (Oracle, 2013).....	34
Figura 13 Vista Lógica de la arquitectura de MySQL Server (Schwartz & Zaitsev, 2012).....	35
Figura 14 Comparación de motores de almacenamiento MySQL Server (Oracle, 2013).....	47
Figura 15 Modelo del sistema MongoDB (Keller, 2012)	48
Figura 16 Ejemplo de un modelo relacional para una aplicación de blogging (MongoDB, 2015)	51
Figura 17 Los datos como documentos, simple para los desarrolladores y rápido para los usuarios (MongoDB, 2015).....	51
Figura 18 Conceptos comunes en cada sistema de base de datos	56
Figura 19 Comparación entre las formas de consulta de las dos bases de datos.....	57
Figura 20 Comparación de algunas funcionalidades más comunes entre las dos bases de datos	58
Figura 21 Esquema de base de datos relacional.....	68
Figura 22 Esquema de la base de datos en MongoDB	69
Figura 23 Creación de un índice en MongoDB para el campo “cod_artista”	74
Figura 24 Tiempo promedio de inserción de los dos gestores de bases de datos.....	76
Figura 25 Tiempo promedio de la primera consulta en los dos gestores de bases de datos.....	76
Figura 26 Tiempo promedio de la segunda consulta en los dos gestores de bases de datos.	77
Figura 27 Tiempo promedio de la tercera consulta en los dos gestores de bases de datos.	77
Figura 28 Uso de memoria de MongoDB.	78
Figura 28 Uso de memoria de MySQL.	78
Figura 29 Tiempo promedio de eliminación de registros.	79

RESUMEN

El presente trabajo de graduación realiza un análisis comparativo de dos gestores de bases de datos: MySQL y MongoDB, en lo referente al comportamiento del motor en tiempos de respuestas ante diferentes operaciones de consulta y manipulación; inicialmente se realiza un resumen de las nuevas tecnologías a las que se enfrentan las bases de datos relacionales y las bases de datos NoSQL.

Se da una perspectiva de los diferentes modelos de consultas, datos y arquitecturas de base de datos y se procede con la instalación e implementación de los dos gestores; paralelamente se desarrolla un esquema de base de datos, en el cual se realizan pruebas de rendimiento que incluyen: el tiempo de inserción, consulta y eliminación de n registros.

Los resultados se presentan en varios cuadros comparativos y gráficos que miden los tiempos de las diferentes pruebas en los dos gestores, y justifica el porqué el uso de una u otra base de datos, así también se subrayan las ventajas de usabilidad de una base de datos no-relacional frente a otra no relacional en el campo empresarial.


ABSTRACT

This graduation paper presents a comparative analysis of two database management systems: MySQL and MongoDB, with regard to the behavior of the engine in response times, to different query and handling operations. At the beginning the paper presents a summary of new technologies to which relational databases and NoSQL databases are faced with.

An overview of the different models of query, data and database architectures are given. Then, we proceed with the installation and implementation of the two management systems. At the same time, a database scheme, in which performance tests that include insertion time, consultation and removal of n records, is developed.

The results are presented in several comparative tables and chart that measure the times of the various tests on the two managers, and justify the reason for the use of either database. Also, the advantages of usability in the business field of non-relational database are emphasized against the non-relational one.




Translated by,
Lic. Lourdes Crespo

CAPÍTULO 1 - INTRODUCCIÓN

1 Sistematización del problema

1.1 Planteamiento del problema

Las bases de datos basadas en el estándar SQL han tenido una posición hegemónica en la mayoría de las organizaciones. Sin embargo, estas organizaciones están considerando incrementar las alternativas al legado de la infraestructura relacional; en algunos casos la motivación ha sido generalmente técnica, como la necesidad de escalar o ejecutar operaciones más allá de las capacidades de sus sistemas existentes, otra motivación es que las empresas están impulsadas por el deseo de identificar alternativas viables al costoso software propietario (Oracle, IBM DB2, Microsoft SQL Server, etc). Un estímulo final es la velocidad de desarrollo con la cual las compañías buscan adaptarse al mercado con mayor rapidez y adoptar metodologías de desarrollo ágil.

La comunidad de desarrolladores de software ha encontrado que el modelo relacional no está bien alineado con las necesidades de sus aplicaciones (MongoDB, 2015). Por esta razón, ellos consideran algunos aspectos de las *RDBMS (Relacional Database Management System)* limitan el rendimiento de sus aplicaciones:

Estructura.- La estructura de los datos en una base de datos relacional está predefinida por el diseño de las tablas y los nombres y tipos fijos de las columnas.

Nuevos tipos de datos.- Los desarrolladores están trabajando con nuevos tipos de datos estructurados, semiestructurados, no estructurados y datos polimórficos; que se operan generalmente en grandes volúmenes (Floyer, 2014).

Escalabilidad.- Los usuarios pueden escalar una base de datos relacional mediante la ejecución en un ordenador más potente y caro; y para escalar más allá de cierto punto, ésta debe ser distribuida a través de múltiples servidores. Las bases de datos relacionales no funcionan fácilmente en una forma distribuida, porque es difícil unir sus tablas a través de un sistema distribuido. Además, las bases de datos relacionales no están diseñadas para funcionar con la partición de datos, por lo que la distribución de su funcionalidad es una tarea, dijo Stephen O'Grady, analista de la firma de investigación de mercado RedMonk (Software Developer's Journal, 2012).

Complejidad.- Con las bases de datos relacionales, los usuarios deben convertir todos los datos en tablas. Cuando los datos no encajan fácilmente en una tabla, la estructura de la base de datos puede ser compleja, difícil y lenta para trabajar.

SQL.- Uso de SQL es conveniente con los datos estructurados (tales como un conjunto de cifras de ventas, que encajan bien en tablas organizadas) pero no es el más adecuado en el caso de datos no estructurados, como los que se encuentran en los documentos de procesamiento de textos (blogs) e imágenes. Además, las aplicaciones modernas con frecuencia se ocupan de los datos no estructurados como blogs, páginas web, transcripciones de voz, etc que no son esencialmente texto.

El desarrollo de aplicaciones móviles hace uso de datos no estructurados. La demanda y las expectativas para aplicaciones móviles han crecido significativamente debido a un creciente aumento de *tablets* y *smartphones*, por tal motivo se ha vuelto cada vez más importante agilizar el proceso de desarrollo de modo que sea más eficiente y menos estresante para los desarrolladores.

La industria de las aplicaciones móviles es una de las áreas donde un RDBMS no es una buena opción para las necesidades dinámicas de las aplicaciones móviles, los desarrolladores vienen con nuevas ideas y características para sus aplicaciones, y los cambios que se pretenden hacer se convierten en una tarea que consume tiempo, porque los cambios constantes tienen que ser hechas en el esquema de base de datos (Asay, 2013).

Por ejemplo, un desarrollador está creando una aplicación similar a "*Angry Birds*", donde diferentes tipos de personajes realizan diferentes acciones. Con una base de datos relacional, agregar otros tipos de personajes o acciones que puedan realizar, pueden requerir alterar completamente el esquema para acomodar el cambio. Dependiendo de la magnitud del cambio, esto podría llevar mucho tiempo y esfuerzo del lado del desarrollador.

Además, las bases de datos relacionales no están construidas para manejar todos los diferentes casos de uso que requieren las aplicaciones móviles, casos de uso que se pueden dividir en términos de tipo de dispositivo móvil, sistema operativo, el firmware del sistema operativo y la ubicación (Asay, 2013).

De manera general, las RDBMS se enfrentan al desafío del almacenamiento de datos de manera masiva, sin la necesidad de seguir estándares que intenten adaptarse a ellos. Estos tipos de datos estan compuestos por una cierta estructura, estructura que puede ser flexible para ser gestionados dentro de un sistema de almacenamiento distribuido.

1.2 Objetivos

1.2.1 Objetivo General

Evaluar el rendimiento de las bases de datos relacionales y no relacionales, aplicado a los gestores de bases de datos Mongo DB y MySQL, en tareas de consulta y manipulación de los datos.

1.2.2 Objetivos Específicos

- Documentar las características teóricas más relevantes de las bases relacionales y no relacionales.
- Instalar los gestores de bases de datos MySql y MongoDB.
- Analizar el rendimiento bajo diferentes criterios de evaluación.
- Presentar los resultados de las pruebas realizadas, mediante un cuadro comparativo.
- Analizar y comparar los resultados y planes de ejecución

1.3 Justificación

La decisión de evaluar el rendimiento de dos bases de datos de código abierto distintas, se debe a que en la última década el término “base de datos” se había convertido en sinónimo de SQL, y durante ese tiempo estaba cerca de ser una solución viable para el almacenamiento de datos. Compañías como Amazon Dynamo (DeCandia, 2013), Google BigTable (Google inc., 2013), LinkedIn Voldemort (LinkedIn, 2013), Twitter Flock (Twitter, 2013), Facebook Cassandra (Facebook, 2015), Yahoo PNUTS entre otros, han estado utilizando las tecnologías relacionales para adaptarla a sus necesidades, en los últimos años han intentado añadir más hardware o actualizar a un hardware más rápido. Cuando eso no funcionó intentaron simplificar su esquema de base de datos como desnormalizar el esquema, esto funcionó de manera parcial, pero no solucionó el problema, entonces se llegó a una conclusión final: éstas tecnologías no cumplían con sus requerimientos.

La tendencia actual de la *Big Data*(datos diversos, datos no estructurados, datos semiestructurados y datos cambiando rápidamente), *Big Users*(usuarios globales las 24 horas al día, 365 días al año) y el *Cloud Computing*(nuevas aplicaciones que usan la arquitectura de internet de tres niveles, corriendo en una nube pública o privada) hacen

que a las bases de datos relacionales se les haga más difícil lidiar con las nuevas tendencias, obligando a las organizaciones migrar hacia las bases de datos no relacionales (NoSQL es llamado popularmente como "No sólo SQL"), que proveen esquemas dinámicos, modelado de datos flexible, arquitectura escalable y almacenamiento eficiente de grandes datos (Zaki, 2013).

Hoy en día el uso de NoSQL se debe principalmente a sus características de escalabilidad y rendimiento. Hace sólo unos años, la escalabilidad y el rendimiento no eran un problema tan grande, pero la enorme cantidad de datos que se recopilan hoy es infinitamente mucho más que hace diez años.

1.4 Alcances y Limitaciones

1.4.1 Los Alcances

El desarrollo del trabajo alcanza una breve introducción a las bases de datos de manera general, para obtener información previa que servirá para analizar el modelo de bases de datos relacional, definiciones, propiedades y reglas de integridad; además, se hará una revisión de la arquitectura de un SGBD relacional, no-relacional y las características de las bases de datos No-SQL como estructura, esquema, modelo de datos, etc.

La investigación abarca únicamente el análisis de rendimiento de la base de datos MySQL y la base de datos documental MongoDB, los criterios de evaluación harán uso de grandes flujos de datos con los que se realizarán las siguientes operaciones: inserción de una gran cantidad de registros generados aleatoriamente(*insert*) de un usuario, lectura de los registros previamente insertados(*read*) de un usuario, inserción de una gran cantidad de registros generados aleatoriamente(*insert*) de dos o más usuarios concurrentes, lectura de los registros previamente insertados(*read*) de dos o más usuarios concurrentes.

Las pruebas de rendimiento que se implementarán se enfocan principalmente en transacciones, que intentan simular un entorno informático donde un usuario o un conjunto de usuarios ejecuta consultas a un sistema de gestión de base de datos. Esta investigación servirá como referencia para implementar un sistema capaz de responder a las necesidades actuales de las diferentes organizaciones.

1.4.2 Las Limitaciones

Al existir diversas maneras de probar el rendimiento de una base de datos haciendo uso de grandes volúmenes, la investigación se limitará solamente a consultas masivas a la

las dos bases de datos planeadas en la investigación. Una última limitación es la falta de investigaciones realizadas anteriormente o artículos relacionados con el tema que se está investigando.

CAPÍTULO 2 - MARCO TEÓRICO

2.1 Bases de datos relacionales

2.1.1 Introducción

En un artículo científico publicado por (Codd, 1970), mostraba que la manera de organizar y acceder a los datos almacenados en grandes bases de datos, podía hacerse sin necesidad de conocer cómo estaba estructurada la información o dónde residía la base de datos. Esta “idea revolucionaria” que Codd presentó, abrió nuevas puertas en el mundo de la independencia de los datos y dio lugar al nacimiento del lenguaje de consulta estructurado llamado SQL. (Codd,1985) publicó una lista de 12 reglas que definen una base de datos ideal; desde entonces, estas reglas se han utilizado como una guía para el diseño de todos los sistemas de bases de datos relacionales.

2.1.2 El modelo relacional

(Silberschatz & Korth, 2006) mencionan que un modelo de datos es un conjunto de herramientas conceptuales para la descripción de los datos, las relaciones entre ellos, su semántica y las restricciones de consistencia. Actualmente el modelo relacional es el modelo más empleado porque permite representar fácilmente la información de mundo real de una manera intuitiva y mantiene información sobre las propias características de la base de datos (metadatos), que facilitan las modificaciones, disminuyendo los problemas ocasionados en las aplicaciones ya desarrolladas.

Por otro lado, incorpora mecanismos de consulta muy potentes, totalmente independientes del RDBMS, e incluso de la organización física de los datos y el propio RDBMS es el encargado de optimizar estas preguntas en formato estándar, a sus características propias de almacenamiento (Salgado, 2007).

2.1.3 Estructura de las Bases de Datos Relacionales

La estructura fundamental del modelo relacional es la relación, es decir una tabla bidimensional constituida por filas (tuplas) y columnas (atributos). Las relaciones representan las entidades que se consideran interesantes en la base de datos. Cada instancia de la entidad encontrará sitio en una tupla de la relación, mientras que los atributos de la relación representan las propiedades de la entidad. Por ejemplo, si en la

base de datos se tienen que representar personas, podrá definirse una relación llamada "Personas", cuyos atributos describen las características de las personas. Cada tupla de la relación "Personas" representará una persona concreta (Quiroz, 2003).

Productos

código_producto	nombre_producto	precio_producto
UA048711	sellante	23,90
CB040271	desbrozadora	456,00
ON763478	llave allen	12,50
YH578500	pegamento	500,90

Figura 1 Tabla o Relación "Productos"

Una **tabla o relación** es una matriz rectangular que almacena líneas con una estructura concreta. Considérese la relación “Productos” tiene cuatro cabeceras de columna: “codigo_producto”, “nombre_producto” y “precio_producto”, según el modelo relacional, se puede hacer referencia a estas cabeceras como **atributos**, que describen las características particulares de la relación y son atómicos. En la relación “Productos”, el atributo “precio_producto” tendrá como cometido almacenar el precio de los diferentes productos con los que pretende comercializar. Para cada atributo hay un conjunto de valores permitidos, denominado **dominio** de ese atributo, por ejemplo para el atributo “nombre_producto”, el dominio de éste es el conjunto de todos los nombres de producto. Cada fila o línea (excepto la primera) recibe el nombre de **tupla**, almacenan ítems concretos para cada columna y deben ser diferentes entre sí. Para el modelo relacional el orden en que aparecen las tuplas y los atributos es irrelevante. El **grado** de la relación es el número de atributos que posee, en este caso de la relación “Productos” es de grado 3. La cardinalidad es el número de tuplas concretas que almacena: 4 (Salgado, 2007).

Una **clave** es un atributo o conjunto de atributos cuyo valor es único y diferente para cada tupla. Las **claves candidatas** son un conjunto de atributos de una tabla que identifican unívocamente cada tupla de una tabla, en este caso de la relación Productos, la clave candidata puede ser “codigo_producto” o “nombre_producto” y de una de éstas dos claves se elige la **clave primaria** cuando se define la tabla; mientras que las claves restantes pasan a llamarse claves_alternas (Sanchez, 2004).

2.1.4 Esquema de la base de datos

El esquema es el diseño lógico de la base de datos que identifica la relación y determina la información a ser guardada, es por ello que es importante dar nombres a los esquemas de las relaciones. Se usará “esquema_producto” para denotar el esquema de la relación “Producto”

```
esquema_producto=(codigo_producto, nombre_producto,precio_producto)
```

De manera general los esquemas de las relaciones consisten en una lista de los atributos y de sus dominios correspondientes (Silberschatz & Korth, 2006).

2.1.5 Restricciones de Integridad

Se trata de unas condiciones de obligado cumplimiento por los datos de la base de datos. Las restricciones de integridad son necesarias para que las tuplas de una relación se distingan entre sí y se expresan en términos de sus atributos (Sanchez, 2004).

Las hay de varios tipos.

a) Inherentes. Son aquellas que no son determinadas por los usuarios, sino que son definidas por el hecho de que la base de datos sea relacional. Por ejemplo:

- No puede haber dos tuplas iguales
- El orden de las tuplas no importa
- El orden de los atributos no importa
- Cada atributo sólo puede tomar un valor en el dominio en el que está inscrito

b) Semánticas. Son aquellas que el modelo relacional permite a los usuarios incorporar restricciones personales a los datos. Las principales son:

- **Clave primaria.** Hace que los atributos marcados como clave primaria no puedan repetir valores.
- **Unicidad.** Impide que los valores de los atributos marcados de esa forma, puedan repetirse.
- **Obligatoriedad.** Prohíbe que el atributo marcado de esta forma no tenga ningún valor
- **Integridad referencial.** Prohíbe colocar valores en una clave externa que no estén reflejados en la tabla donde ese atributo es clave primaria.
- **Regla de validación.** Condición que debe de cumplir un dato concreto para que sea actualizado.

2.2 Bases de datos no relacionales

2.2.1 Introducción

El término NoSQL fue usado por primera vez en 1998 para una base de datos relacional que omitía el uso de SQL. NoSQL significa no-sólo SQL que es otro tipo de almacenamiento de datos que se utiliza para almacenar grandes volúmenes de información, debido a las miles de aplicaciones derivadas principalmente del uso de la Web 2.0. (por ejemplo la cantidad de información de Facebook que tiene un enorme crecimiento diario). Esta base de datos interactúa generalmente con los sistemas operativos basados en UNIX y poseen un alto rendimiento del sistema en la manera de escalar horizontalmente; esto se debe a que las bases de datos NoSQL no organizan sus datos en tablas relacionadas (es decir los datos se almacenan de una manera desnormalizada). Además, las bases de datos NoSQL son *open source*, por lo tanto todo el mundo puede mirar su código libremente, actualizarlo y compilarlo según sus necesidades.

En los últimos años han sido organizados grandes eventos alrededor del mundo para fomentar el avance del movimiento NoSQL, eventos como: *NoSQL Matters* (Dublín, París, Barcelona), *Devcon TLV Summit* (Tel-Aviv, Israel), *NoSQL Now* (San José, Costa Rica), etc, la mayoría de ellos estrechamente relacionados con el *Internet of Things*, *Big Data*, y el *Cloud*.



Figura 2 Representación Simbólica de NoSQL

La Figura 2 representa que se puede ejecutar una consulta a la base de datos sin necesidad de ninguna interacción o interfaz del lenguaje SQL (*Structured Query Language*). Así que para acceder a estas bases de datos podemos utilizar algunos otros formatos como XML para almacenar y recuperar información de la base de datos. Hoy en día NoSQL se está volviendo tan popular debido a su gran almacenamiento y a sus propiedades que evitan las características básicas de SQL.

2.2.2 Por qué utilizar NoSQL?

“Con todos los sistemas de gestión de base de datos disponibles hoy en día, ¿por qué necesitamos otro?”. (Strozzi, 2010) quien fue el primero en usar una base de datos

relacional que omitió el uso de SQL, expone la razón principal del porqué utilizar NoSQL:

“Varias veces me he encontrado escribiendo aplicaciones que necesitan realizar tareas simples en una base de datos. La mayoría de productos de bases de datos son demasiado costosas y repletas de características para un uso casual. También hay un montón de buenas bases de datos gratuitas en todo, pero la mayoría de las veces éstas tienden a ofrecer mucho más de lo que se necesita.”

(Moniruzzaman & Hossain, 2013) explican que de los diferentes modelos de datos, el modelo relacional ha sido el dominante desde los años 80, con implementaciones como las bases de datos Oracle, MySQL y SQL Server de Microsoft conocidos como RDBMS. Sin embargo, en la mayoría de aplicaciones el uso de las bases de datos relacionales conduce a déficits y problemas en el modelado de datos, así como las limitaciones que lleva consigo la escalabilidad horizontal sobre múltiples servidores y grandes cantidades de datos; por consiguiente existen dos tendencias que acarrearán estos problemas:

1. El crecimiento exponencial del volumen de datos generados por los usuarios, sistemas y sensores, acelerados aún más por la concentración de gran parte de este volumen en grandes sistemas distribuidos como Amazon, Google y otros servicios en la nube.
2. La creciente interdependencia y complejidad de los datos, impulsados por el Web 2.0, redes sociales y el acceso abierto y estandarizado para fuentes de datos a partir de un gran número de sistemas.

Otra de las razones es el auge de la *Big Data*, en la mayoría de áreas que tienen que ver con la tecnología o los negocios, que involucra datos muy diversos, en constantes cambios y demasiado masivos para las tecnologías tradicionales.

(Connolly, 2012) define a la *Big Data* en una simple ecuación :

Big Data= Transacciones + Interacciones + Observaciones

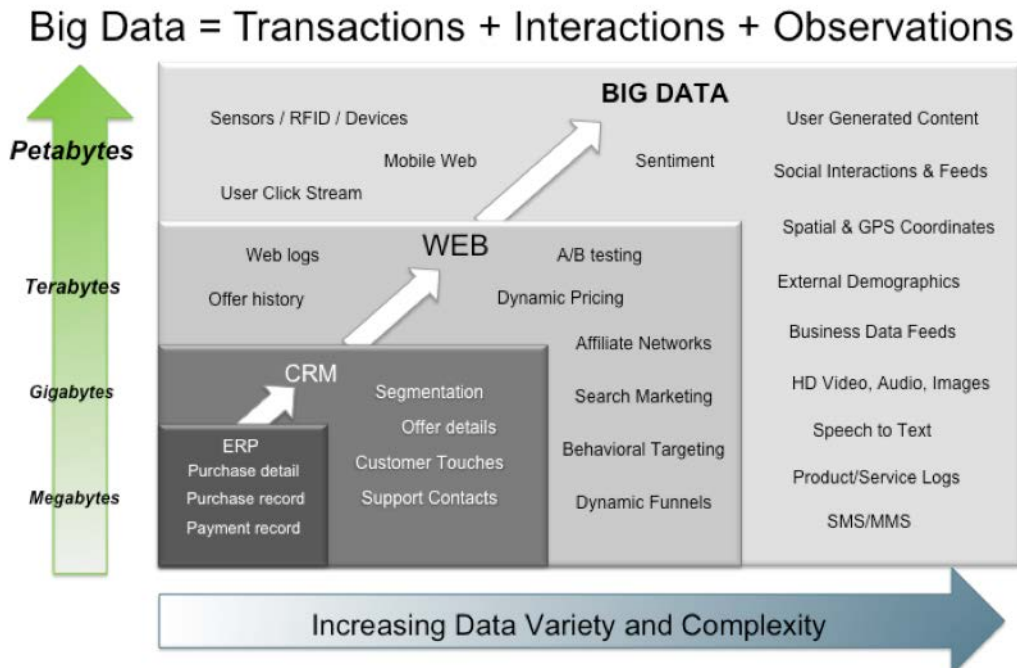


Figura 3 Transacciones Big Data con Interacciones y Observaciones (Zaki, 2013).

La Figura 3 ilustra el crecimiento del volumen de información, así como el incremento de la variedad de información y su complejidad; comienza por los sistemas ERP, SCM, CRM y aplicaciones Web transaccionales, cuyos datos altamente transaccionales están generalmente guardados en bases de datos relacionales. Las interacciones se refieren a acerca de cómo las personas y las cosas interactúan entre ellas o sus negocios. Los sitios que albergan contenido generado por el usuario, *Click Streams* de Usuarios, etc. son un ejemplo de lugares para encontrar información de interacción, su volumen supera los terabytes de almacenamiento. Mientras que los datos de observación provienen del “Internet de las cosas”, como sensores de calor, sensores de presión, dispositivos móviles, GPS y motores de automóviles que arrojan datos de observación.

2.2.3 Características

A continuación se describen algunos conceptos fundamentales, técnicas y patrones comunes entre las bases de datos NoSQL.

1. Libre de ACID (Atomicity-Consistency-Isolation-Durability)

ACID es un conjunto de propiedades que se aplican específicamente a las transacciones de bases de datos, que se define de la siguiente manera:

Atomicidad - Todo en una transacción debe suceder con éxito (*committed*) o ninguno de los cambios son ejecutados (*rolled-back*). Esto evita que una operación que esté cambiando múltiples bloques de datos, falle a mitad de camino y sólo haga algunos cambios.

Consistencia - Los datos sólo serán ejecutados (*committed*) si cumplen todas las reglas establecidas de la base de datos (es decir: tipos de datos, disparadores, restricciones, etc.).

Aislamiento – Las transacciones no afectarán otras transacciones cambiando datos que otra operación está procesando, y otros usuarios no verán los resultados parciales de una transacción en curso (en función del modo de aislamiento).

Durabilidad - Una vez que los datos son ejecutados (*committed*), éstos se almacenan de forma duradera y segura contra errores, accidentes o cualquier otro mal funcionamiento (software) dentro de la base de datos.

Estas cualidades parecen indispensables, y sin embargo, son incompatibles con el rendimiento en sistemas muy grandes. (StackExchange, 2012) indica que ACID se orienta más a Consistencia y Disponibilidad.

ACID es comúnmente proporcionada por la mayoría de las bases de datos relacionales clásicas como MySQL, Microsoft SQL Server, Oracle y otros. Estas bases de datos son conocidas para el almacenamiento de datos en tablas que tienen sus columnas y tipos de datos estrictamente definidos. Las tablas pueden tener relaciones entre sí y los datos se consultan con SQL (Structured Query Language) que es un lenguaje estandarizado para trabajar con bases de datos relacionales.

2. BASE

El acrónimo BASE fue definida por (Brewer, 2000) quien también formuló el teorema CAP, se utiliza para describir las propiedades de ciertas bases de datos, por lo general bases de datos NoSQL. Generalmente se refiere como lo contrario de ACID, debido a que renuncia a la Consistencia.

BA Básicamente Disponible indica que el sistema hace la garantía la disponibilidad, en términos del teorema CAP.

S Estado Flexible indica que el estado del sistema puede cambiar con el tiempo, incluso sin entradas. Esto es porque el modelo de consistencia eventual.

E Consistencia eventual indica que el sistema se mantendrá constante en el tiempo, siempre y cuando el sistema no recibe de entradas durante ese tiempo.

2. El Teorema CAP (Consistency-Availability-Partition Tolerance)

En un simposio presentado por (Brewer, 2000) titulado “Principios de Computación Distribuida”, expone el Teorema CAP, el cual establece tres requerimientos básicos en los sistemas distribuidos, que se adopta ampliamente por la comunidad NoSQL.

La **Consistencia** es el estado coherente de un sistema después de la ejecución de una operación. Un sistema distribuido es considerado ser consistente si después de una operación de actualización, todos los nodos pueden ver las actualizaciones en todo momento. Este estado tiene relación con la integridad de la información.

Disponibilidad y alta disponibilidad significa que un sistema es diseñado e implementado de una manera que permite su operación continua (si algún nodo cae, los demás pueden seguir permitiendo operaciones de lectura y escritura sin problemas).

Tolerancia al particionamiento entendida como la habilidad de un sistema para continuar sus operaciones en presencia de particiones de red, esto sucede cuando dos o más nodos no pueden conectarse (temporalmente o permanentemente) con los otros (Ippolito, 2009). También se define la tolerancia a particiones como la capacidad de un sistema para hacerle frente a la adición y eliminación dinámica de nodos, generalmente utilizados para propósitos de mantenimiento.

Siguiendo ese contexto, Brewer alega que es imposible satisfacer los tres requerimientos de forma simultánea, es por ello que se debe elegir dos de tres. Generalmente los sistemas NoSQL le dan mayor prioridad a la tolerancia y a veces a la disponibilidad, en cambio los sistemas RDBMS le dan prioridad a la consistencia y disponibilidad. En su discurso Brewer señala las características y ejemplos de las tres opciones diferentes que se pueden hacer de acuerdo a su teorema CAP.

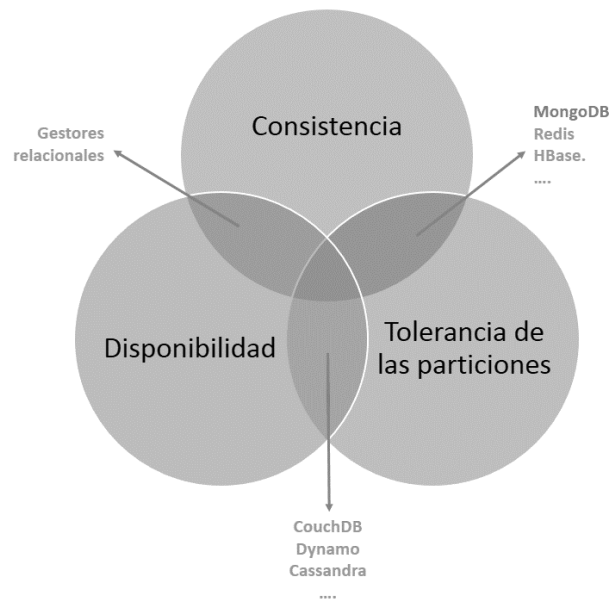


Figura 4 El teorema CAP, indica que se puede realizar dos operaciones a la vez (Alarcon, 2014).

2.2.4 Tipos de Bases de Datos NoSQL

Las bases de datos manejan muy bien el volumen, variedad y velocidad de gran cantidad de información. Pero cada una de ellas, manejan estas tres características dependiendo de su modelo de datos (Moniruzzaman & Hossain, 2013). Por esta razón, las bases de datos NoSQL son agrupadas de acuerdo a su modelo de datos :

1. Bases de Datos Clave- Valor

Este tipo de bases de datos tienen el modelo de bases de datos más simple entre las bases de datos NoSQL, porque los datos se almacenan como un par clave-valor; la clave es asociada con un solo elemento en la base de datos, lo cual permite la recuperación de la información de forma muy rápida. Este tipo de bases de datos han existido por muchos años como bases relacionales, pero las nuevas bases de datos clave-valor entran en la categoría de NoSQL porque son construidas para ser veloces y escalables a costa de sacrificar algunas funcionalidades como la consistencia y las consultas complejas (*joins* y operaciones de agregación); generalmente en el almacenamiento clave-valor no existen claves foráneas o claves alternas y tampoco existe un orden implícito y la simplicidad de estas bases de datos lo hacen ideales para la rápida recuperación de perfiles de usuario, sesiones o nombres de productos. Esta es la razón de porqué Amazon hace el uso extensivo de su propio sistema clave-valor llamado Dynamo, en su carrito de compras.

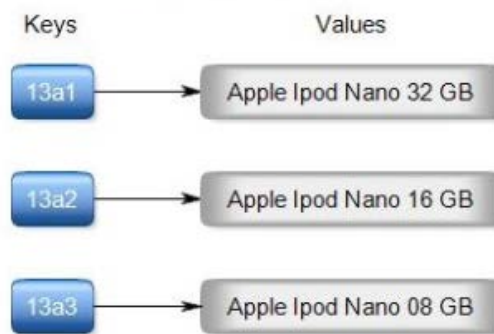


Figura 5 Almacenamiento Clave-Valor (Antiñanco, 2013)

Ejemplos: Dynamo (Amazon), Voldemort(Linked-In), Redis, BerkeleyDB; Riak.

2. Bases de Datos Documentales

Inspirado por Lotus Notes, las bases de datos documentales fueron diseñadas para gestionar y almacenar documentos; estos documentos son codificados en un formato estándar de intercambio de datos como XML, JSON o BSON y no tienen restricciones de esquema (similares al almacenamiento clave-valor). Cada documento contiene una clave especial “ID” , la cual también es única dentro de una colección de documentos y además, identifica al documento explícitamente. A diferencia del almacenamiento clave-valor, los valores (datos) no son invisibles y se puede consultarlos porque los pares clave-valor son encapsulados en documentos. Además, el almacenamiento de nuevos documentos que contienen cualquier tipo de atributos puede ser fácilmente realizado, al añadir nuevos atributos a los documentos existentes en tiempo de ejecución.

También, las estructuras de datos complejas como los objetos anidados pueden ser manejados más convenientemente, porque cuando se almacenan datos en documentos interpretables JSON, éstos tienen una ventaja adicional en el soporte de varios tipos de datos, lo cual hace que el almacenamiento sea amigable con el desarrollador. Los casos de uso populares son en análisis en tiempo real, registro, y el almacenamiento de capas de sitios web pequeños y flexibles.

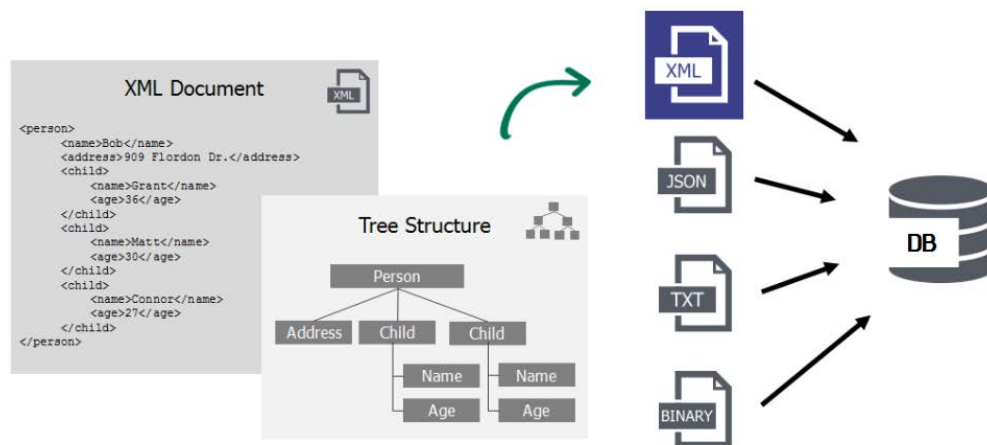


Figura 6 Colección de documentos complejos con formatos de datos arbitrarios, anidados y un formato de “registros variables” (MarkLogic, 2014)

Ejemplos: Apache CouchDB, MongoDB

3. Bases de Datos orientadas a Columnas

La familia de bases de datos orientadas a columnas, en teoría es similar a una tabla en una base de datos relacional, excepto que ésta puede escalar a un número indeterminado de filas y cada fila puede tener cualquier número de columnas; luego cada familia de columnas es asociada con una fila que consiste de un par clave-valor (una clave de la columna y un valor de la columna).

La familia de bases de datos orientadas a columnas, fueron conocidas después de que Google publicó su artículo científico, además, de ser impulsado por la popularidad de Cassandra y HBase y los usos comunes de estas bases de datos son para aplicaciones de monitoreo de eventos, sistemas de gestión de contenidos y plataformas de blogs. (MarkLogic, 2014)

Column Families					
Row Key	first_name	last_name	address	address2	age
mallen	Matt	Allen	909 Flordon Dr.	1012	30
gallen	Grant	Allen	gallen@hotmail.com		
callen	Connor	Allen	28		

Figura 7 La familia de bases de datos orientadas a columnas como Cassandra organiza los datos a través de un clave de fila que es asociada con cualquier número de columnas (MarkLogic, 2014).

Las bases orientadas a columnas más populares son: BigTable(Google), Cassandra.

4. Bases de Datos orientadas a Grafos

Inspiradas en Euler y la teoría de grafos, éstas bases de datos permiten contar con un modelo de negocio más complejo, con flexibilidad en las relaciones entre entidades; utilizan una estructura de grafo con nodos, aristas y propiedades para representar y almacenar datos. Por definición, una base de datos orientada a grafos es cualquier sistema de almacenamiento que provea libre indexado por adyacencia, esto significa que cada elemento contiene un puntero directo a su elemento adyacente y no requiere búsqueda por índices. Se distinguen las bases de datos generales orientadas a grafos que pueden almacenar cualquier tipo de grafo, de las especializadas tales como bases de datos de red y *triple-store* (Antiñanco, 2013).

Algunos ejemplos destacables de las bases de datos orientadas a grafos son: Neo4j, HyperGraphDB, AllegroGraph y VertexDB.

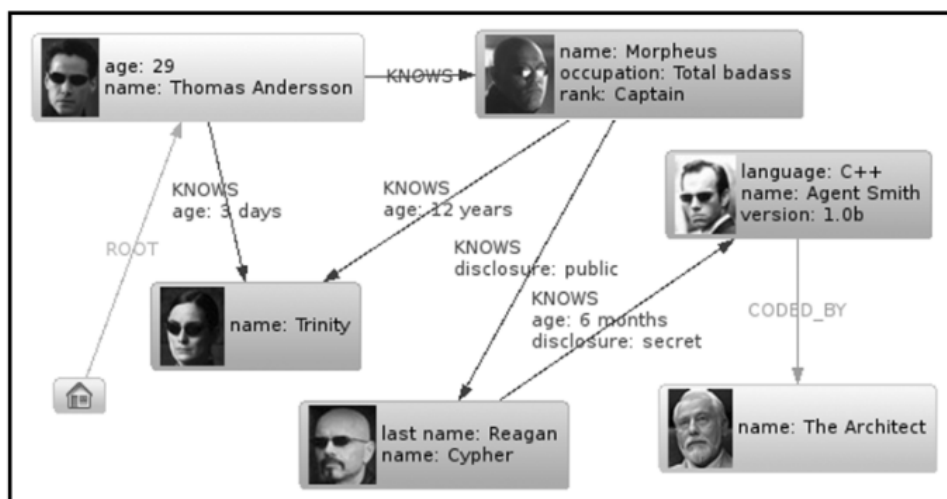


Figura 8 Ejemplo de una base de datos orientada a grafos (Moniruzzaman & Hossain, 2013).

NoSQL comenzó dentro del dominio del *open source* con pocos proveedores, pero continuó creciendo en datos y esto ha atraído a muchos competidores en el mercado. Actualmente, docenas de productos son autoidentificados como NOSQL, y cada uno tiene un diseño y arquitectura únicos e incluso el paradigma de almacenamiento de datos varía a través de las distintas implementaciones de NoSQL y existen repositorios de bases orientadas a columnas, clave-valor y documentales (ver Figura 9).

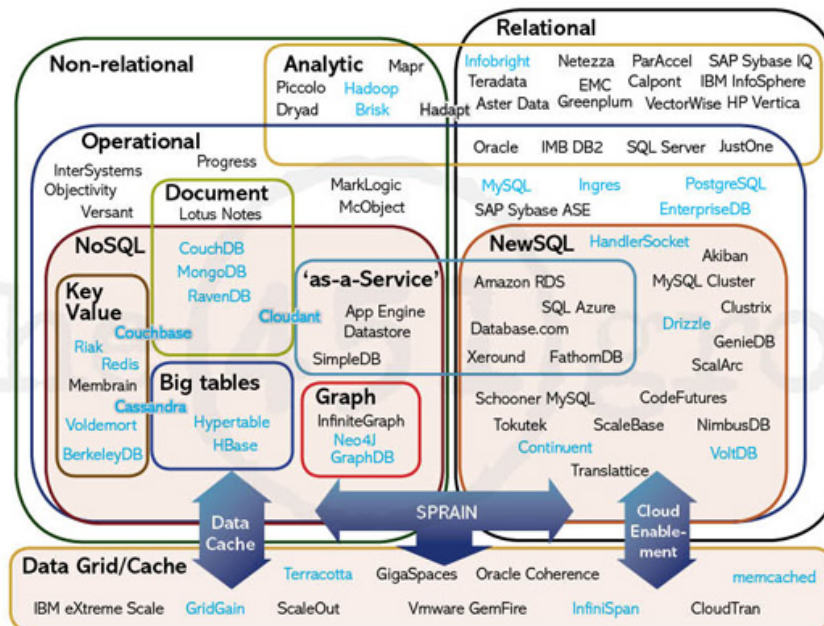


Figura 9 Estado actual de las bases de datos NoSQL (Moniruzzaman & Hossain, 2013)

2.2.5 Comparación entre Bases de Datos NoSQL

En esta sección, se proporciona la evaluación de algunas bases de datos NoSQL (las cuatro categorías descritas en el numeral anterior), en una tabla que contiene las siguientes características de evaluación : diseño, integridad, indexación y distribución de datos.

Attributes		NoSQL Databases								
Database model		Document-Stored		Wide-Column Stored			Key-Value Stored		Graph-orientated	
Design & Features	Features	MongoDB	CouchDB	DynamoDB	HBase	Cassandra	Accumulo	Redis	Riak	Neo4j
	Data storage	Volatile memory File System	Volatile memory File System	SSD	HDFS		Hadoop	Volatile memory File System	Bitcask LevelDB Volatile memory	File System Volatile memory
	Query language	Volatile memory File System	JavaScript Memcached-protocol	API calls	API calls REST XML Thrift	API calls CQL Thrift		API calls	HTTP JavaScript REST Erlang	API calls REST SparQL Cypher Tinkerpop Gremlin
	Protocol	Custom, binary (BSON)	HTTP, REST	-	HTTP/REST Thrift	Thrift & custom binary CQL3	Thrift	Telnet-like	HTTP, REST	HTTP/REST Embedding in Java
	Conditional entry updates	Yes	Yes	Yes	Yes	No	Yes	No	No	
	MapReduce	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No
	Unicode	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	TTL for Entries	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	
	Compression	Yes	Yes	-	Yes	Yes	Yes	Yes	Yes	
	Integrity model	BASE	MVCC	ASID	Log Replication	BASE	MVCC	-	BASE	ASID
Integrity	Atomicity	Conditional	Yes	Yes	Yes	Yes	Conditional	Yes	No	Yes
	Consistency	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
	Isolation	No	Yes	Yes	No	No	-	Yes	Yes	Yes
	Durability (data storage)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-	Yes
	Transactions	No	No	No	Yes	No	Yes	Yes	No	Yes
	Referential integrity	No	No	No	No	No	No	Yes	No	Yes
	Revision control	No	Yes	Yes	Yes	No	Yes	No	Yes	No
Indexing	Secondary Indexes	Yes	Yes	No	Yes	Yes	Yes	-	Yes	-
	Composite keys	Yes	Yes	Yes	Yes	Yes	Yes	-	Yes	-
	Full text search	No	No	No	No	No	Yes	No	Yes	Yes
	Geospatial Indexes	Yes	No	No	No	No	Yes	-	-	Yes
	Graph support	No	No	No	No	No	Yes	No	Yes	Yes
Distribution	Horizontal scalable	Yes	Yes	Yes	Yes	Yes	Yes		Yes	No
	Replication	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes
	Replication mode	Master-Slave-Replica Replication	Master-Slave Replication	-	Master-Slave Replication	Master-Slave Replication	-	Master-Slave Replication	Multi-master replication	-
	Sharding	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
	Shared nothing architecture	Yes	Yes	Yes	Yes	Yes	-	-	Yes	-
System	Value size max.	16MB	20MB	64KB	2TB	2GB	1EB	-	64MB	
	Operating system	Cross-platform	Ubuntu Red Hat Windows Mac OS X	Cross-platform	Cross-platform	Cross-platform	NIX 32 entries Operating system	Linux *NIX Mac OS X Windows	Cross-platform	Cross-platform
	Programming language	C++	Erlang C++ C Python	Java	Java	Java	Java	C C++	Erlang	Java

Figura 10 Comparación de cuatro categorías de bases de datos NoSQL en base a atributos como diseño, integridad, indexación, distribución, sistema (Moniruzzaman & Hossain, 2013).

2.2.6 Modelos de consultas

Puesto que los modelos de datos están estrechamente acoplados con las posibilidades de consulta, el análisis de consultas es un proceso muy importante que debería ser soportado por la base de datos, para encontrar un modelo de datos adecuado. Las bases de datos NoSQL no sólo difieren en su modelo de datos proporcionado, también difieren en la riqueza que ofrecen las funcionalidades de consulta. El estado actual que ofrece el panorama NoSQL puede ser comparado con el tiempo antes de la introducción del SQL de Codd, por ejemplo: en los últimos años ha surgido una gran cantidad de diferentes bases de datos, las cuales difieren en sus modelos de datos, lenguajes de consulta y *APIs*.

Por lo tanto, se realizan algunas investigaciones con el fin de lograr una mayor incorporación de sistemas NoSQL mediante el desarrollo de lenguajes de consulta normalizados (Meijer & Bierman, 2011). Hasta tanto, para algunos sistemas, los desarrolladores todavía deben hacer frente a las características específicas de cada base de datos NoSQL (Hecht & Jablonski, 2011).

Debido a su modelo de datos simple, los APIS de los sistemas clave-valor solamente proporcionan operaciones basadas en *put*, *get*, y *delete*. Debido a que, cualquier lenguaje de consulta sería una sobrecarga innecesaria para estos sistemas, porque se requieren funcionalidades de consulta adicionales, que tienen que ser implementadas en la capa de aplicación, lo que significaría mucha más complejidad del sistema y la penalización de su rendimiento.

Por lo tanto, los sistemas clave-valor no deberían ser utilizados si se requieren consultas más complejas o consultas sobre valores, y ahí surgen las interfaces REST (*Representational State Transfer*), muy útiles en el campo del desarrollo de aplicaciones web y las aplicaciones orientadas a servicios (SOA). Finalmente, los diferentes tipos de clientes pueden interactuar directamente con el sistema de una manera uniforme, mientras que las peticiones pueden ser balanceadas y los resultados representados a través de un proxy caché.

A diferencia de los sistemas clave-valor, las bases de datos orientadas a documentos ofrecen APIs mucho más ricos. Las búsquedas por rangos sobre valores (datos), índices secundarios, consultas de documentos anidados y operaciones como: “*and*”, “*or*” y “*between*” son características que pueden ser usadas a conveniencia y las consultas en

Riak y MongoDB pueden extenderse con expresiones regulares. Mientras MongoDB soporta operaciones adicionales como “count” y “distinct”, Riak ofrece funcionalidades para atravesar vínculos fácilmente entre documentos.

Las interfaces REST también son soportadas por sistemas documentales. Debido a sus potentes interfaces, un lenguaje de consultas incrementa la facilidad de uso, al ofrecer una capa de abstracción adicional, la cual sería útil para el almacenamiento de documentos. Dado que ninguna de estas distribuciones no ofrecen ningún lenguaje de consulta, el proyecto N1QL (Couchbase, 2015) está trabajando en un lenguaje de consulta común, que ofrece una sintaxis similar a SQL, para consultar sistemas documentales basados en JSON.

La familia de bases de datos orientadas a columnas solamente proveen algunas operaciones como “in”, “and/or” y expresiones regulares, que tienen como consecuencia la falta de cooperación para desarrollar un lenguaje de consulta común; también se debe a que

esos lenguajes están especializados en las características específicas de sus productos.

Como las bases de datos documentales y las bases orientadas a columnas son capaces de almacenar grandes cantidades de datos estructurados y las consultas pueden llegar a ser muy ineficientes si una sola máquina tiene que procesar los datos requeridos. Por lo tanto, los dos tipos de bases de datos proporcionan *MapReduce* que permiten cálculos paralelizados sobre enormes conjuntos de datos. Sin embargo, los trabajos de *MapReduce* son escritos en un muy bajo nivel de abstracción que conducen a producir a programas personalizados que son difíciles de mantener y reutilizar. (Hecht & Jablonski, 2011)

La mayoría de las bases de datos en grafo ofrecen interfaces REST, interfaces específicas de lenguajes de programación y lenguajes de consulta específicos para almacenamiento. A diferencia de otras bases de datos NoSQL, existen algunos lenguajes de consulta, los cuales son utilizados por más de una base de datos en grafo como SPARQL, que es un lenguaje declarativo de consulta con una sintaxis muy simple, también se usa *Gremlin* que es un lenguaje de programación imperativo usado para realizar recorridos en grafos basados en XPATH.

Las bases de datos NoSQL difieren fuertemente en sus funcionalidades de consulta ofrecidas. Además, de considerar el modelo de datos soportado y cómo éste influye en las consultas sobre atributos específicos, es necesario tener una mirada más cercana sobre las interfaces ofrecidas a fin de encontrar una base de datos adecuada para un caso de uso específico. Las interfaces REST pueden ser una solución adecuada especialmente para aplicaciones web, donde las consultas críticas de rendimiento pueden ser pasadas a un lenguaje específico que esté disponible para casi todos los desarrolladores, como por ejemplo Java (Hecht & Jablonski, 2011).

Los lenguajes de consulta ofrecen un mayor nivel de abstracción con el fin de reducir la complejidad y su uso es muy útil cuando deberían ser realizadas consultas complejas. Si se requieren consultas intensivas de cálculo sobre grandes conjuntos de datos, se deben utilizar frameworks *MapReduce* (Hecht & Jablonski, 2011).

Database		Query Possibilities			
		REST API	JAVA API	Query Language	Map Reduce Support
Key Value Stores	<i>Voldemort</i>	-	+	-	-
	<i>Redis</i>	-	+	-	-
	<i>Membase</i>	+	+	-	-
Document Stores	<i>Riak</i>	+	+	-	+
	<i>MongoDB</i>	+	+	-	+
	<i>CouchDB</i>	+	+	-	+
Column Family Stores	<i>Cassandra</i>	-	+	+	+
	<i>HBase</i>	+	+	+	+
	<i>Hypertable</i>	-	+	+	+
Graph Databases	<i>Sesame</i>	+	+	+	-
	<i>BigData</i>	+	+	+	-
	<i>Neo4J</i>	+	+	+	-
	<i>GraphDB</i>	+	+	+	-
	<i>FlockDB</i>	-	+	-	-

Figura 11 Posibilidades de Consulta (Hecht & Jablonski, 2011).

La Figura 11 muestra las diferentes posibilidades de consulta que ofrecen las bases de datos NoSQL, y se puede observar que los tipos Clave-Valor, Documentales, Orientadas a Columnas no poseen un lenguaje de consulta propio, aunque su mayoría soporta la API REST para su uso en aplicaciones web, en las cuales son utilizadas enormemente.

2.2.7 Seguridad

Las bases de datos NoSQL surgen con distintos problemas de seguridad, ya que el enfoque principal de bases de datos NoSQL es el manejo de nuevos conjuntos de datos, los cuales tienen poca prioridad en la seguridad, por ejemplo al satisfacer los requerimientos de la *Big Data* se pone poco énfasis en la seguridad, la cual es dada en la fase de diseño. Sumado a esto, las bases de datos NoSQL no proporcionan ninguna característica de seguridad embebida en propia base de datos y solamente proveen una capa muy pequeña de seguridad. Todo lo expuesto anteriormente conlleva a amenazas principales de seguridad, con las que tienen que enfrentarse las bases de datos NoSQL, amenazas que se enumeran a continuación:

1. Integridad Transaccional. Las bases de datos NoSQL fracasan al asegurar la integridad transaccional, debido a su naturaleza flexible. Las restricciones complejas de integridad no pueden ser añadidas a la arquitectura de NoSQL, porque se traduce en el incumplimiento del principal objetivo de la NoSQL de lograr un mejor rendimiento y escalabilidad.

2. Mecanismos de Autenticación. Las bases de datos NoSQL están expuestas a ataques de *replay*, ataques de fuerza bruta, ataques *CSRF*, ataques de inyección y ataques *man in the middle*, que tienen como consecuencia la fuga de información. La razón principal es porque las bases de datos NoSQL incorporan un mecanismo de autenticación débil y frágiles técnicas de almacenamiento de contraseñas. Aunque algunas bases de datos NoSQL aplican mecanismos de autenticación a nivel de nodo local, no aplican la autenticación a través de todos los servidores.

3. Susceptibilidad en ataques de inyección. Dado que NoSQL emplea protocolos de peso muy ligero y mecanismos de acoplamiento flexible en su arquitectura, permite a un atacante acceso al sistema de archivos a través de *back-doors* para actividades maliciosas.

4. Falta de Consistencia. Las bases de datos NoSQL no satisfacen simultáneamente las tres propiedades (consistencia, disponibilidad y tolerancia a particiones), establecidas por el teorema CAP.

5. Ataques desde adentro. Este tipo de bases, generalmente posee métodos de análisis de logueo deficientes, debido a esto, un ataque desde adentro del sistema podría tener acceso a datos críticos de otros usuarios. Esto se debe a su delgada capa de seguridad, que viene a ser difícil para los usuarios mantener el control de sus datos. De manera que el

uso de un mecanismo de aplicación de seguridad externa es esencial para las bases de datos NoSQL (Zaki, 2013).

2.3 Diferencias conceptuales entre RDBMS y NoSQL DBMS

(Luke, 2014) presenta algunas diferencias de alto nivel, entre los sistemas SQL y la amplia gama de tecnologías que ha generado que el movimiento NoSQL.

- Las bases de datos basados en SQL son llamadas principalmente Bases de Datos Relacionales (RDBMS); mientras que las bases de datos NoSQL son llamadas bases de datos no-relacionales o bases de datos distribuidas.
- Los sistemas SQL son bases de datos basadas en tablas mientras que las bases NoSQL son basadas en: documentos, pares clave-valor, en grafo o orientadas a columnas. Esto significa que las bases de datos SQL representan la información en forma de tablas, las cuales constan de n números de filas mientras que las bases de datos NoSQL (colecciones de clave-valor, documentales, en grafo y orientadas a columnas) no tienen un esquema estándar definido, el cual se agrega según el tipo de base de datos.
- Las bases de datos relacionales tienen un esquema predefinido mientras que las bases de datos NoSQL un esquema dinámico para datos no-estructurados.
- Las bases de datos relacionales son escalables verticalmente, mientras que las bases de datos no relacionales son escalables horizontalmente. Esto quiere decir que los sistemas SQL escalan incrementando el poder de procesamiento del hardware mientras que los sistemas NoSQL escalan incrementando los servidores de bases de datos en el *pool* de recursos para reducir la carga.
- Las bases de datos relacionales utilizan SQL para la definición y manipulación de datos, el cual es muy poderoso. En las bases de datos no-relacionales, las consultas son enfocadas en la colección de documentos, generalmente es llamado *UnQL* (Unstructured Query Language). La sintaxis de uso del UnQL varía de base de datos a base de datos.
- Para consultas complejas, las bases de datos relacionales se ajustan muy bien por el ambiente complejo de consultas que poseen, mientras que las bases de datos NoSQL no se ajustan correctamente porque no poseen interfaces estandarizados para realizar consultas complejas, y las mismas consultas no son lo suficientemente poderosas como el lenguaje SQL.
- Para el tipo de dato a ser almacenado, las bases de datos relacionales no son las más adecuadas para el almacenamiento de datos jerárquicos. Pero las bases de datos

NoSQL, se ajustan mejor para el almacenamiento de datos jerárquicos ya que sigue el modo de almacenamiento clave-valor y son altamente preferidas para grandes conjuntos de datos

- Para la escalabilidad, en la mayoría de situaciones típicas, las bases de datos relacionales escalan verticalmente; es decir que se puede incrementar el rendimiento, añadiendo más CPU, RAM, SSD, etc, en un solo servidor. Por otro lado, las bases de datos NoSQL son escalables horizontalmente; al agregar más servidores fácilmente en nuestra infraestructura de base de datos para manejar el alto tráfico.
- Para aplicaciones basadas en alta transaccionalidad, las bases de datos relacionales son mejor opción para aplicaciones de trabajo pesado de tipo transaccional, ya que es más estable y promete la atomicidad, así como la integridad de los datos.
- Para soporte, la mayoría de las bases de datos relacionales brindan un excelente servicio de soporte técnico para por parte de sus vendedores. Sin embargo, en algunas bases de datos NoSQL todavía se tiene que contar con el apoyo de la comunidad y solamente pocos expertos están disponibles para configurar y desplegar sistemas NoSQL a gran escala.
- Para las propiedades, las bases de datos relacionales hacen hincapié en las propiedades ACID (atomicidad, coherencia, aislamiento y durabilidad), mientras que las bases de datos NoSQL siguen el teorema CAP (consistencia, disponibilidad y tolerancia a particiones)
- Para los tipos de bases de datos, en un alto nivel se puede clasificar las bases de datos SQL, ya sea como *open-source* o *closed-source* de proveedores comerciales. Las bases de datos NoSQL se pueden clasificar en función de medio de almacenamiento de datos como bases de datos de gráficos, clave-valor, bases de datos documentales, bases de datos orientadas a columnas.

2.4 MySQL

MySQL, es el sistema de gestión de base de datos *SQL Open Source* más popular, desarrollado, distribuido y soportado por Oracle Corporation (MySQL, 2013). A continuación se describen algunas características más relevantes:

La base de datos MySQL es relacional. La parte SQL de "MySQL" es sinónimo de "Structured Query Language" o SQL que es el lenguaje estandarizado más común utilizado para acceder a bases de datos. Dependiendo de su entorno de programación, se

puede introducir SQL directamente (por ejemplo, para generar informes), sentencias SQL embebidas en código escrito en otro idioma, o utilizar una API específica que posee internamente la sintaxis SQL.

La base de datos MySQL es *open-source*._ Open Source significa que es posible para cualquier usuario usar y modificar el software. Cualquiera, puede descargar el software MySQL desde internet y usarlo sin costo; si lo desea, puede estudiar el código fuente y modificarlo para que se adapte a sus necesidades. El software MySQL utiliza la licencia GPL (GNU *General Public License*), para definir lo que puede y no hacer con el software en diferentes situaciones.

La base de datos MySQL es rápida, fiable, escalable y fácil de usar._ MySQL Server se puede ejecutar cómodamente en un escritorio o portátil, junto con otras aplicaciones, servidores web, etc. que requieren poca o ninguna atención. Si se le dedica una máquina entera a MySQL, se puede ajustar la configuración para aprovechar toda la memoria, potencia de CPU, y la capacidad de E / S disponibles. MySQL también puede escalar hasta grupos de máquinas, conectados en red entre sí.

La base de datos MySQL funciona en cliente / servidor o sistemas embebidos._ El software de base de datos MySQL es un sistema cliente / servidor que consiste en un servidor multi-hilo SQL que soporta diferentes *backends*, varios programas cliente diferentes, librerías, herramientas administrativas, y una amplia gama de interfaces de programación de aplicaciones *API* (MySQL, 2013).

2.4.1 Arquitectura del Sistema

La arquitectura de MySQL es muy diferente a la de otros servidores de bases de datos. Porque posee la característica más inusual e importante, que es su arquitectura de motores de almacenamiento, cuyo diseño separa el procesamiento de consultas y otras tareas del servidor, del almacenamiento de datos y su recuperación, incluso a partir de la versión 5.1 se puede cargar motores de almacenamiento como plugins en tiempo de ejecución.

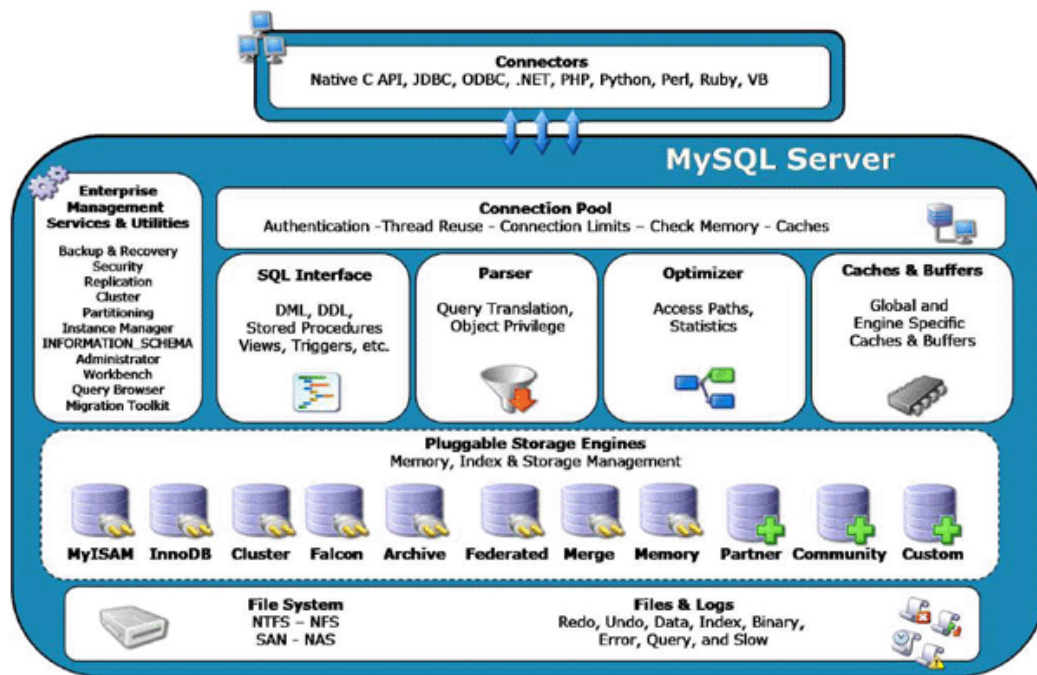


Figura 12 Arquitectura de MySQL Server (*Oracle, 2013*).

Sobre la capa del motor de almacenamiento, existe un pool de conectores disponibles a través de los cuales se acceden a los módulos del servidor que analizan las consultas, optimizan rutas de acceso, etc. El pool de conexión, proporciona autenticación, gestiona las amenazas, las conexiones, la memoria y las cachés.

Los motores de almacenamiento son los componentes del servidor de bases de datos que llevan a cabo acciones en los datos subyacentes que se mantienen en el nivel del servidor físico.

La base de datos MySQL dispone de algunos motores de almacenamiento internos, e InnoDB es el motor de almacenamiento por defecto, el más utilizado para la versión 5.6; y además, Oracle recomienda usarlo, salvo para aplicaciones especializadas (MySQL, 2013) . En cambio, los motores de almacenamiento externos están configurados para optimizar el rendimiento de productos y situaciones específicos, los suministran tanto desarrolladores de software independientes como la comunidad MySQL.

Y en el caso de contar con varios motores de almacenamiento, se tiene la posibilidad de aprovechar varias bases de datos. Por ejemplo: algunos motores de almacenamiento, como los utilizados para el archivado, son no transaccionales por naturaleza, pero permiten insertar y leer datos de forma muy eficiente. Otros motores de almacenamiento

están configurados para ser eficientes en las operaciones transaccionales, mientras que un tercer grupo proporciona alta disponibilidad mediante técnicas de *Clustering*.

La Figura 13 muestra una vista lógica de la arquitectura de la base de datos a la vez que ofrece una breve representación de cómo trabajan juntos los componentes de MySQL y a ayudar a entender el servidor.

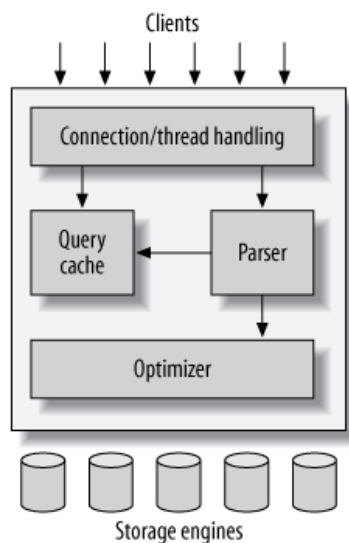


Figura 13 Vista Lógica de la arquitectura de MySQL Server (*Schwartz & Zaitsev, 2012*).

La capa más alta contiene los servicios que la mayoría de redes basadas en servidores o herramientas cliente/servidor necesitan: manejo de conexiones, autenticación, seguridad, etc.

La segunda capa se compone por muchos de los motores de MySQL, incluido el código para el analizador sintáctico, análisis y optimización de consultas, gestión de cache y buffer, además, de todas las funciones incorporadas (datos, reloj, matemáticas, encriptación) y cualquier funcionalidad proporcionada al otro lado de los motores de almacenamiento perdura en este nivel, por ejemplo procedimientos almacenados, triggers, vistas, etc.

La tercera capa contiene los motores de almacenamiento, y es la responsable de almacenar y recuperar todos los datos guardados en MySQL, mediante los motores de almacenamiento; por lo tanto, cada motor de almacenamiento tiene sus propios beneficios

y desventajas. Finalmente, el servidor se comunica con los motores de almacenamiento a través del *API storage engine*.

Esta interface oculta las diferencias de los motores de almacenamiento y los hace transparentes principalmente a la capa de consulta. Mientras tanto que, el API también contiene funciones de bajo nivel que ejecutan operaciones como por ejemplo “empezar una transacción” o “extraer filas que tienen su llave primaria”. Es importante mencionar que los motores de almacenamiento no analizan SQL o se comunican con otro motor, ellos simplemente responden a las peticiones del servidor (Schwartz & Zaitsev, 2012).

2.4.2 Catálogo del Sistema

El catálogo de datos denominado `INFORMATION_SCHEMA` proporciona acceso a los metadatos de la base de datos, tales como el nombre de la base de datos o tabla, el tipo de datos de una columna, o permisos de acceso (cada usuario MySQL tiene derecho a acceder a estas tablas, pero sólo a los registros que se corresponden a los objetos a los que tiene permisos de acceso). Además, almacena información acerca de todas las otras bases de datos que mantiene el servidor MySQL y está compuesto por varias vistas de sólo lectura, por lo que no existe ningún fichero asociado con ellas (Quizhpe, 2009).

.

2.4.3 Administración de Conexiones y Seguridad

Cada conexión de cliente obtiene su propio hilo dentro de el proceso del servidor, luego las consultas de la conexión se ejecutan dentro de un sólo hilo, que a su vez reside en un *core* y el servidor almacena en caché los hilos de conexión, por lo que no necesitan ser creados y destruidos por cada conexión nueva.

Cuando los clientes(aplicaciones) se conectan a MySQL Server, el servidor necesita autenticarlos. La autenticación está basada en: el nombre de usuario, host de origen y contraseña. El certificado X.509 puede ser usado también através de una conexión *SSL (Secure Socket Layer)* y una vez que el cliente se ha conectado, el servidor verifica si el cliente tiene privilegios para cada consulta que emite(por ejemplo, si el cliente se le esta permitido realizar una sentencia `SELECT`, la cual tiene acceso a la tabla “Ciudad” en la base de datos “Empresas”).

2.4.4 Ejecución y Optimización de Consultas

MySQL realiza un *parsing* ó análisis sintáctico de las consultas para crear una estructura interna (*the parse tree*) y entonces aplica una variedad de optimizaciones, que pueden incluir: la reescritura de la consulta, el determinar el orden en el cual se va a leer las tablas, elegir qué índices usar, y así sucesivamente. Al realizar la consulta, se pueden pasar consejos al optimizador a través de *keywords* ó palabras clave especiales, que afectan a su proceso de toma de decisiones. También se puede pedir al servidor explicar varios aspectos de optimización, esto permite saber qué decisiones está tomando el servidor y da un punto de referencia para la reelaboración de consultas, esquemas y ajustes para hacer que todo funcione lo más eficientemente posible.

Al optimizador no le importa qué tabla en particular usa el motor de almacenamiento, pero el motor incide en cómo el servidor optimiza las consultas. El optimizador pregunta al motor de almacenamiento acerca de su capacidad y el costo de ciertas operaciones. Por ejemplo algunos motores de almacenamiento soportan algunos tipos de indexación que pueden ser muy útiles para determinadas consultas.

Incluso antes de hacer un *parsing* de la consulta, el servidor consulta la caché de consultas, la cual puede almacenar solamente las sentencias `SELECT`, junto a sus conjuntos de resultados y si alguien emite una consulta que es idéntica a otra que ya está en la caché, el servidor no necesita analizar, optimizar, o ejecutar la consulta en absoluto, simplemente pasa otra vez el conjunto de resultados almacenado (Schwartz & Zaitsev, 2012).

2.4.5 Control de Concurrency

Cuando más de una consulta necesita cambiar los mismos datos al mismo tiempo, ahí surge el problema del control de concurrencia, MySQL generalmente realiza eso en dos niveles: el nivel de servidor y el nivel de motor de almacenamiento (Schwartz & Zaitsev, 2012).

Lo mencionado en el párrafo anterior se puede representar con un ejemplo: un clásico buzón de correo electrónico en un sistema Unix, en el cual el formato del archivo clásico *mbox* es muy simple, a tal punto que todos los mensajes en un buzón *mbox* se concatenan juntos, uno tras otro. Esto hace que sea muy fácil de leer y analizar los mensajes de correo y hace también que la entrega de correo sea fácil (sólo anexar un nuevo mensaje al final del archivo).

¿Pero qué sucede cuando dos procesos intentan entregar mensajes al mismo tiempo para el mismo buzón? Claramente podría corromper el buzón, dejando dos mensajes intercalados al final del archivo de buzón de correo; entonces para prevenir este bloqueo, los sistemas de entrega de correo utilizan un *locking* ó bloqueo para prevenir que ocurra la entrega simultánea. Si se intenta segunda entrega, mientras que el buzón está bloqueado se debe esperar para adquirir su propio locking antes de entregar el mensaje.

Este esquema funciona razonablemente bien en la práctica, pero provee una concurrencia bastante deficiente, ya que sólo un único programa puede realizar cualquier cambio en el buzón de correo en un momento dado, y en un buzón de alto volumen que recibe miles de mensajes por minuto esto se convierte en un problema. Por lo tanto, este bloqueo exclusivo hace que sea difícil no atrasarse durante la entrega de correo, si alguien intenta leer, responder y eliminar mensajes en ese mismo buzón (Schwartz & Zaitsev, 2012)

Bloqueos de Lectura/ Escritura

La lectura del buzón de correo no es tan problemática. No hay nada de malo con múltiples clientes leyendo el mismo buzón de correo simultáneamente porque no están realizando cambios. Pero, ¿qué pasa si alguien trata de eliminar el mensaje número 25, mientras que los programas están leyendo el buzón? Depende, pero un lector podría salir con una vista dañada o inconsistente del buzón. Así que, para estar seguro, incluso la lectura de un buzón requiere cuidados especiales.

Si se piensa en el buzón como una tabla de base de datos y cada mensaje de correo electrónico como una fila, es fácil ver que el problema es el mismo en este contexto. En muchos sentidos, un buzón es realmente simplemente una tabla de base de datos. La modificación de filas de una tabla de base de datos es muy similar, quitando o cambiando el contenido de los mensajes en un archivo de buzón de correo.

La solución a este problema clásico de control de concurrencia es bastante simple. Los sistemas que tratan con el acceso concurrente de lectura/escritura, normalmente implementan un sistema de bloqueo que se divide en dos tipos. Estos bloqueos son generalmente conocidos como *shared locks* y *exclusive locks*, ó *read locks* y *write locks*.

Bloqueo Granular

Una manera de mejorar la concurrencia de un recurso compartido, es ser más selectivo acerca de qué recurso bloquear; en lugar de bloquear todo el recurso, se bloquea sólo la

parte que contiene los datos que se necesita cambiar. Mejor aún, se bloquea solamente el pedazo exacto de los datos que va a cambiar.

El problema del bloqueo es que consume recursos, al realizar procedimientos como: el obtener un bloqueo, el comprobar si un bloqueo está libre, el liberar un bloqueo y así sucesivamente. Por lo tanto, el rendimiento del sistema puede verse afectado si el sistema pasa demasiado tiempo administrando bloqueos en lugar de almacenar y recuperar datos

Una estrategia de bloqueo representa una operación compleja que combina la sobrecarga de bloqueo y la seguridad de los datos, y eso afecta al rendimiento, por eso la mayoría de las bases de datos comerciales no ofrecen muchas opciones: lo que se obtiene se conoce como *row-level locking* ó bloqueo de filas en las tablas, con una variedad de maneras a menudo complejas para dar un buen rendimiento con muchos bloqueos.

MySQL por otro lado, ofrece algunas opciones, porque sus motores de almacenamiento pueden implementar sus propias políticas de bloqueo además del bloqueo granular. La gestión de bloqueo es una decisión muy importante en el diseño del motor de almacenamiento; la fijación de la granularidad en un cierto nivel puede ofrecer un mejor rendimiento para ciertos usos, sin embargo, puede hacer al motor menos adecuado para otros fines.

Debido a que MySQL ofrece múltiples motores de almacenamiento, que no requieren una única solución de propósito general. Vamos a echar un vistazo a las dos estrategias de bloqueo más importantes que son el bloqueo de tabla y bloqueo de fila:

Bloqueo de tabla

El bloqueo de tabla o *table-level lock* es la estrategia básica de bloqueo más disponible en MySQL y la que tiene la menor sobrecarga. Por ejemplo: cuando un cliente desea escribir en una tabla (insertar, eliminar, actualizar,etc), adquiere un bloqueo de escritura y esto mantiene todas las demás operaciones aparte. Cuando nadie está escribiendo, los lectores pueden obtener bloqueos de lectura, los cuales no entran en conflicto con otros bloqueos de lectura.

Los bloqueos de tabla tienen variaciones específicas para el buen desempeño en situaciones específicas. Por ejemplo, los bloqueos de tabla `READ LOCAL` permiten algunos tipos de operaciones de escritura concurrentes. Los bloqueos de escritura tienen mayor prioridad que los bloqueos de lectura, por lo que una solicitud de bloqueo de

escritura avanzara a la parte delantera de la cola, incluso si los lectores ya están en cola (los bloqueos de escritura pueden avanzar sobre los bloqueos de lectura en la cola, pero los bloqueos de lectura no pueden avanzar sobre los bloqueos de escritura).

Aunque los motores de almacenamiento pueden administrar sus propios bloqueos, MySQL por sí mismo también utiliza una variedad de bloqueos que son a nivel de tabla para diversos fines. Por ejemplo, el servidor utiliza el bloqueo de tabla para los estados como ALTER TABLE, independientemente del motor de almacenamiento.

Bloqueos de Fila

El estilo de bloqueo que ofrece mayor concurrencia y a la vez mayor sobrecarga, es el uso de bloqueos de fila ó *row-level locking*; está disponible en los motores de almacenamiento InnoDB, XtraDb, entre otros. Los bloqueos de fila son implementados en el motor de almacenamiento, no en el servidor, porque éste es completamente inconsciente de los bloqueos implementados en los motores de almacenamiento.

La mayoría de los motores de almacenamiento transaccionales de MySQL no utilizan un mecanismo sencillo de bloqueo de filas, en su lugar, utilizan el bloqueo de fila en conjunto con una técnica para aumentar la concurrencia conocido como control de concurrencia multiversión (MVCC en inglés). MVCC no es única para MySQL, también la utilizan Oracle, PostgreSQL, y algunos otros sistemas de bases de datos, aunque hay diferencias significativas, porque no existe un estándar de cómo debería funcionar MVCC.

Se puede pensar en MVCC como una mejora al bloqueo de fila. Evita la necesidad del bloqueo en la mayoría de los casos y puede tener mucha más baja sobrecarga. Dependiendo de cómo está implementada, puede permitir lecturas sin realizar el bloqueo mientras que se bloquea sólo las filas necesarias durante las operaciones de escritura.

MVCC trabaja manteniendo una instantánea de los datos, tal como existían en algún momento en el tiempo. Esto significa que las transacciones pueden tener una vista consistente de los datos, sin importar cuánto tiempo se ejecutan, es decir que diferentes transacciones pueden ver datos diferentes en las mismas tablas a la vez (porque cada motor de almacenamiento implementa un MVCC diferente) y algunas de las variaciones antes descritas incluyen el control de concurrencia optimista y pesimista.

InnoDB implementa MVCC almacenando en cada fila dos valores adicionales, ocultos que registran cuando se creó la fila y cuando fue eliminada. En lugar de almacenar los

tiempos reales en que se produjeron estos hechos, la fila almacena el número de versión del sistema en el momento en que ocurrió cada evento, éste número aumenta cada vez que se inicia una transacción, la cual mantiene su propio registro de la versión actual del sistema, a partir del momento en que se inició, por lo tanto, cada consulta tiene que comprobar los números de versión de cada fila contra la versión de la transacción (Schwartz & Zaitsev, 2012).

2.4.6 Transacciones

Una transacción es un grupo de consultas SQL que son tratadas atómicamente, como a una única unidad de trabajo; si el motor de base de datos se puede aplicar todo el grupo de consultas a una base de datos, lo hace, pero si alguno de ellos no se puede hacer debido a un choque u otro motivo, no se aplica ninguno de ellos. Además MySQL provee dos motores transaccionales de almacenamiento: InnoDB y NDB Cluster (Schwartz & Zaitsev, 2012).

AUTOCOMMIT

MYSQL opera en modo AUTOCOMMIT por defecto. Esto significa que, a menos que haya comenzado de forma explícita una transacción, automáticamente se ejecuta cada consulta en una transacción separada.

```
mysql> SHOW VARIABLES LIKE 'AUTOCOMMIT';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | ON    |
+-----+-----+
1 row in set (0.00 sec)
mysql> SET AUTOCOMMIT = 1;
```

El valor 1 y ON son equivalentes, como lo es 0 y OFF. Cuando se ejecuta AUTOCOMMIT=0 , se está siempre en una transacción, hasta que se ejecuta un *COMMIT* o *ROLLBACK*, entonces MySQL comienza una transacción inmediatamente.

Al cambiar el valor de AUTOCOMMIT, este cambio no tiene efecto en tablas no-transaccionales, como MyISAM, las cuales no tienen noción de los cambios por *COMMIT* o *ROLL-BACK*.

Con el AUTOCOMMIT desactivado habrá siempre una transacción abierta, que tendremos que terminar con las sentencias COMMIT o ROLLBACK. Estos son comandos típicos

DDL que realizan cambios significativos, como `ALTER TABLE`, sin embargo, `LOCK TABLES` y otras sentencias también tienen ese efecto.

MySQL deja establecer el nivel de aislamiento usando el comando `SET TRANSACTION ISOLATION LEVEL`, la cual hace efecto cuando comienza la siguiente transacción. Entonces, se puede establecer el nivel de aislamiento para todo el servidor en el archivo de configuración, ó sólo para la sesión actual.

```
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

MySQL reconoce los cuatro niveles de aislamiento estándar ANSI, e InnoDB soporta todos ellos (Schwartz & Zaitsev, 2012).

2.4.7 Motores de Almacenamiento

MySQL almacena cada base de datos (también llamada esquema) como un subdirectorio de su directorio de datos en un sistema de archivos subadyacente. Cuando se crea una tabla, MySQL almacena la definición de la tabla en un archivo *.frm* con el mismo nombre de la tabla (por ejemplo al crear una tabla llamada *miTabla*, MySQL almacena la definición de tabla en *miTabla.frm*); esto es debido a que MySQL utiliza el sistema de archivos para almacenar los nombres de bases de datos y definiciones de tablas, la diferencia entre mayúsculas y minúsculas (*sensitive case*) dependen de la plataforma. En una instancia de MySQL en Windows, las tablas y nombres de bases de datos no diferencia mayúsculas (*case insensitive*); en cambio los sistemas Unix son lo contrario (*case sensitive*).

Se puede usar el comando `TABLE STATUS` para mostrar la información acerca del motor de almacenamiento, para mostrar información acerca de una tabla. Por ejemplo, para examinar la tabla *user* en la base de datos *mysql*, ejecutar lo siguiente.

```
mysql> SHOW TABLE STATUS LIKE 'user' \G
***** 1. row *****
      Name: user
      Engine: MyISAM
    Row_format: Dynamic
          Rows: 6
    Avg_row_length: 59
     Data_length: 356
Max_data_length: 4294967295
   Index_length: 2048
      Data_free: 0
    Auto_increment: NULL
   Create_time: 2002-01-24 18:07:17
   Update_time: 2002-01-24 21:56:29
   Check_time: NULL
```

```
Collation: utf8_bin
Checksum: NULL
Create_options:
Comment: Users and global privileges
1 row in set (0.00 sec)
```

La salida muestra que es una tabla MyISAM. También se podría notar una gran cantidad de información adicional y estadísticas en la salida. Veamos brevemente el significado algunas líneas importantes:

Name El nombre de la tabla

Motor El motor de almacenamiento de la tabla.

Row_format El formato de fila. Para las tablas MyISAM, puede ser Dinámico, Fijo, o Comprimido. Las filas dinámicas varían en longitud, ya que contienen campos de longitud variable como: VARCHAR o BLOB. El formato fijo, es aquel que siempre tiene el mismo tamaño y está hecho para campos que no varían en longitud como CHAR e INTEGER y las filas comprimidas existen solamente en las tablas comprimidas.

Rows Es el número de filas que contiene una tabla (para MyISAM y la mayoría de motores, este número es siempre exacto, para InnoDB, es una estimación).

Avg_row_length Indica el promedio de bytes que contiene una fila promedio.

Data_length Cuantos datos (en bytes) contiene la tabla entera.

El almacenamiento e indexación de los datos es controlado por la capa de motores de almacenamiento y este motor o *storage engine* es quien almacenará, manejará y recuperará información de una tabla en particular. MySQL soporta varios motores de almacenamiento que tratan con distintos tipos de tabla. Pero, MyISAM e InnoDB son los dos motores de almacenamiento más comunes en MySQL (Quizhpe, 2009).

InnoDB

Según (Schwartz & Zaitsev, 2012), InnoDB es el motor de almacenamiento transaccional por defecto y ampliamente el más utilizado. Por lo tanto, la base de datos fue diseñada para procesar muchas transacciones de corta duración, que suelen ser completas en lugar de ser revertidas, y su rendimiento y recuperación automática de fallos hacen que también sea popular para las necesidades de almacenamiento no transaccionales.

InnoDB tiene una historia de liberación compleja, pero es muy útil para entenderlo. En 2008, el *plugin* denominado InnoDB fue lanzado para MySQL 5.1. Esta fue la siguiente generación de InnoDB creado por Oracle, que en ese momento tenía la propiedad de InnoDB, pero no MySQL. Finalmente, Oracle adquirió Sun Microsystems y por lo tanto MySQL, y se retira el viejo código base, sustituyéndolo por el "plugin" por defecto en MySQL 5.5.

En la versión moderna de InnoDB, introducida como el plugin InnoDB en MySQL 5.1, se agregan nuevas características tales como la construcción de índices de selección, la capacidad de quitar, la agregación de índices sin reconstruir toda la tabla, un nuevo formato de almacenamiento que ofrece compresión y una nueva forma para almacenar valores grandes (tales como columnas BLOB, y administración de archivos). En general, el desarrollo de InnoDB se ha acelerado considerablemente en los últimos años, con importantes mejoras en la instrumentación, escalabilidad, capacidad de configuración, rendimiento, características y soporte para Windows, entre otros artículos notables (Schwartz & Zaitsev, 2012).

Funcionamiento._ InnoDB almacena sus datos en una serie de uno o más archivos de datos que se conocen colectivamente como un espacio de tabla ó *tablespace*, que es esencialmente una caja negra que InnoDB gestiona todo por sí mismo. También InnoDB puede utilizar particiones de disco en bruto para la construcción de su *tablespace*, pero los modernos sistemas de archivos hacen esto innecesario.

InnoDB utiliza MVCC para lograr alta concurrencia, e implementa cuatro niveles de aislamiento, por defecto el nivel de aislamiento *REPEATABLE READ* y tiene una estrategia *next-key locking* (un tipo de bloqueo a nivel de registro) que previene lecturas fantasma en este nivel de aislamiento: en su lugar bloquea solamente las filas que se han tocado en la consulta, e InnoDB bloquea los vacíos en la estructura del índice.

Las tablas de InnoDB están construidas en un *clustered index*. Las estructuras de índices de InnoDB son muy diferentes de las de la mayoría de los otros motores de almacenamiento de MySQL, como resultado proporciona búsquedas de clave primaria muy rápidos. Sin embargo, los índices secundarios (índices que no son la clave primaria) contienen las columnas de clave primaria, por lo que si su clave primaria es grande, otros índices también será grandes. El formato de almacenamiento es independiente de la

plataforma, lo que significa que puede copiar los datos y archivos de índice de un servidor basado en *Intel* a un *PowerPC* o *Sun SPARC* sin ningún problema.

InnoDB tiene una variedad de optimizaciones internas, estos incluyen una lectura anticipada para la obtención previa de datos desde el disco, un índice de *hash* adaptativo que crea automáticamente índices *hash* en memoria para búsquedas muy rápidas, y un buffer de inserciones para acelerar las inserciones.

Como motor de almacenamiento transaccional, InnoDB soporta copias de seguridad en línea ó llamadas también *hot backups* a través de una variedad de mecanismos, incluyendo MySQL Enterprise de Oracle y la fuente abierta Percona XtraBackup.

Otros motores de almacenamiento de MySQL no pueden realizar copias de seguridad en “caliente” para obtener una copia de seguridad coherente, se tiene que detener todas las escrituras en la tabla, que en una carga de trabajo de lectura/ escritura por lo general se tienen que parar las operaciones.

MyISAM

MyISAM ofrece una larga lista de características, tales como la indexación de texto completo, compresión y funciones espaciales (*SIG Sistemas de Información Geográfica*). Sin embargo, MyISAM no soporta transacciones o bloqueos a nivel de fila; es por ello que MySQL todavía tiene la reputación de ser un sistema de gestión de base de datos no transaccional.

MyISAM normalmente almacena cada tabla en dos archivos: un archivo de datos y un archivo de índice, estos dos archivos tienen extensiones *.MYD* y *.MYI* respectivamente. Las tablas MyISAM pueden contener filas ya sea filas dinámicas o estáticas (de longitud fija), por lo tanto MySQL decide qué formato utilizar en función de la definición de la tabla. Adicionalmente, es importante mencionar que el número de filas que una tabla MyISAM puede contener, está limitada principalmente por el espacio de disco disponible en el servidor de base de datos y el archivo más grande que el sistema operativo anfitrión permita crear.

Las principales diferencias entre InnoDB y MyISAM (con respecto al diseño de una tabla o base de datos) tienen relación con la integridad referencial y transacciones. Si se necesita la base de datos para aplicar *foreign key constraints* ó restricciones de clave externa, ó si necesita la base de datos para soportar transacciones (es decir, cambios

realizados por dos o más operaciones DML manipuladas como sola unidad de trabajo, con todos los cambios aplicados, o todos los cambios revertidos) entonces se debe elegir el motor InnoDB, ya que estas características están ausentes en el motor MyISAM.

Las tablas MyISAM creadas en MySQL 5.0 con filas de longitud variable están configuradas por defecto para manejar 256 TB de datos, usando punteros 6 bytes de los registros de datos; sin embargo, todas las versiones de MySQL pueden manejar un tamaño de puntero de hasta 8 bytes. Como uno de los motores de almacenamiento más antiguos incluidas en MySQL, MyISAM tiene muchas características que se han desarrollado durante años de uso, como son:

Bloqueo y concurrencia._ MyISAM bloquea tablas enteras, no filas y para la lectura se puede obtener bloqueos compartidos en todas las tablas que necesitan para leer, mientras que para la escritura se obtienen bloqueos exclusivos de escritura. Sin embargo, se puede insertar nuevas filas en la tabla, mientras las consultas *select* se están ejecutando contra ella (son conocidas como inserciones concurrentes).

Reparación._ MySQL soporta la comprobación manual y automática, además, de la reparación de tablas MyISAM, pero no se debe confundir esto con transacciones o recuperación de fallos. Después de la reparación de una tabla lo más probable es encontrar que algunos datos simplemente han desaparecido; la reparación es demasiado lenta. Se pueden usar los comandos `CHECK TABLE mytable` and `REPAIR TABLE mytable` para comprobar una tabla con errores y repararlos, también se puede usar el comando en línea *myisamchk* para comprobación y recuperación de tablas cuando el servidor está *offline*.

Características de indexación._ Se puede crear índices en los primeros 500 caracteres de columnas *BLOB* y *TEXT* en tablas MyISAM, también soporta indexación de texto completo, es decir indexar palabras individuales para operaciones de búsqueda complejas.

Escrituras de claves retrasadas._ Las tablas MyISAM marcadas con `DELAY_KEY_WRITE` crean la opción de no escribir datos modificados del índice en el disco al final de una consulta. Adicional a eso, MyISAM amortigua los cambios con *in-memory key buffer*. Esto puede aumentar el rendimiento, pero después de una caída del servidor o sistema, los índices podrían dañarse definitivamente y necesitar reparación.

Diferencias InnoDB y MyISAM

InnoDB implementa *row-level lock* o bloqueo de registro (esto sucede cuando un usuario esta realizando un *update* a un registro y al mismo tiempo otro usuario ingresa a la misma tabla a actualizar otro registro sin problema), mientras que MyISAM solamente implementa *table-level lock* o bloqueo de tabla (es decir cuando un usuario actualiza una tabla, y ésta se mantiene bloqueada hasta que el cambio se haya ejecutado o revertido) (StackExchange, 2011).

En la Figura 14 se muestra la comparación de algunas características más relevantes, descritas anteriormente como: soporte de transacciones, control de concurrencia, granularidad de bloqueo, etc.

Características de los Motores de Almacenamiento						
Característica	MyISAM	BDB	Memory	InnoDB	Archive	NDB
Soporte de Transacciones	No	Si	No	Si	No	Si
Granularidad de Bloqueo	Tabla	Página	Tabla	Registro	Registro	Registro
Control de Concurrencia	No	No	No	Si	Si	Si
Limites de Almacenamiento	Ilimitado	Ilimitado	Limitado	64TB	Ilimitado	Limitado
Indexación	B TREE Full Text	B TREE Hash	B TREE	B TREE Hash Clustered	No soporta	B TREE Hash
Costo de Almacenamiento	Bajo	Bajo	-	Alto	Muy Bajo	Alto
Costo de Memoria	Bajo	Bajo	Medio	Alto	Bajo	Alto
Velocidad de Inserción	Alto	Alto	Alto	Bajo	Muy Alto	Alto
Compresión de Datos	Si	No	No	No	Si	No
Encriptación de Datos	Si	Si	Si	Si	Si	Si

Figura 14 Comparación de motores de almacenamiento MySQL Server (Oracle, 2013).

2.5 MongoDB

MongoDB es una base de datos open-source desarrollada por 10gen en C++, fué lanzada en el año 2009 y pertenece a la categoría de las bases de datos documentales, esta base de datos surge como una nueva tendencia en el desarrollo de bases de datos y se refieren en general, a las bases de datos sin un esquema fijo. Además, suelen tener una seguridad de las transacciones a un nivel más bajo, pero son más rápidos en el acceso a los datos y escalan mejor que las bases de datos relacionales (Keller, 2012).

La base de datos MongoDB consiste en un conjunto de bases de datos en la que cada base de datos contiene varias colecciones y cada colección puede contener diferentes tipos de objetos y cada objeto también llamado documento se representa como una estructura JSON que es una lista de pares de clave-valor (esto se debe a que MongoDB trabaja con esquemas dinámicos). El valor puede ser de tres tipos: un valor primitivo, un array de documentos, o de nuevo una lista de pares de clave-valor o documentos como se muestra en la Figura 15 (Keller, 2012).

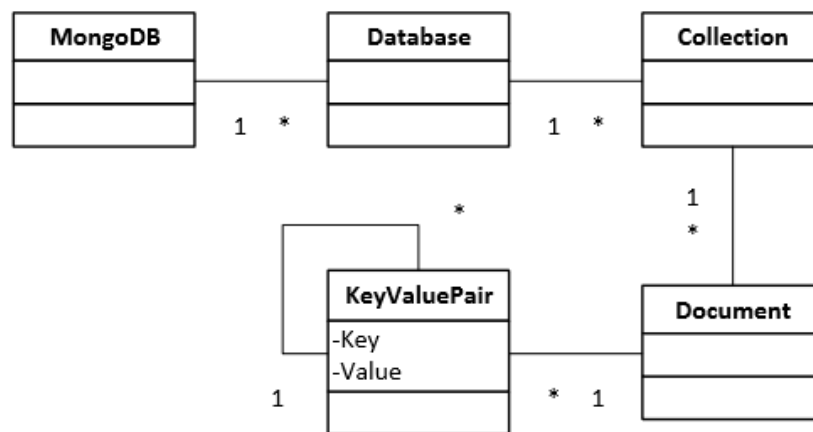


Figura 15 Modelo del sistema MongoDB (Keller, 2012)

A continuación se describen algunas características de MongoDB que sustentan el análisis de rendimiento que se realizará en el siguiente capítulo:

Un modelo de datos enriquecido. La idea básica es reemplazar el concepto de "fila" con un modelo más flexible, el "documento" al permitir documentos y arreglos embebidos, el enfoque orientado a documentos permite representar relaciones jerárquicas complejas con un solo registro. Por lo tanto, esto encaja muy naturalmente en la forma en que los desarrolladores modernos de lenguajes orientados a objetos piensan acerca de sus datos.

MongoDB es también libre de un esquema: las claves de un documento no están predefinidas o fijadas de ninguna manera; entonces al no tener un esquema para cambiar, las migraciones masivas de datos suelen ser innecesarias, por consiguiente, las claves nuevas o las que faltan pueden ser tratados a nivel de aplicación, en lugar de obligar a todos los datos a tener la misma forma; esto proporciona a los desarrolladores una gran flexibilidad en la forma en que trabajan con la evolución de los modelos de datos (Chorodow, 2010).

Escalado Fácil._ Los tamaños de los conjuntos de datos para las aplicaciones están creciendo a un ritmo increíble. Los avances en la tecnología de sensores, el aumento de ancho de banda disponible y la popularidad de los dispositivos portátiles que se pueden conectar a Internet han creado un ambiente donde incluso aplicaciones de pequeña escala necesitan almacenar más datos que muchas bases de datos estaban destinados a manejar. Un terabyte de datos, que alguna vez fue una cantidad inaudita de información, es ahora un lugar común (Chorodow, 2010).

A medida que crece la cantidad de datos que los desarrolladores necesitan almacenar, ellos se enfrentan a una decisión difícil: ¿cómo deben escalar sus bases de datos? Entonces el escalamiento una base de datos se reduce a la elección entre la ampliación (conseguir una máquina más potente) o escalar (particionar datos a través de más máquinas). La ampliación es a menudo el camino de menor resistencia, pero tiene inconvenientes: las máquinas potentes son a menudo muy caras y finalmente se llega a un límite físico.

MongoDB fue diseñado desde el principio para escalar porque su modelo de datos orientado a documentos le permite dividir automáticamente los datos a través de múltiples servidores. Se puede balancear los datos y la carga a través de un clúster, redistribuyendo los documentos automáticamente, esto permite a los desarrolladores centrarse en la programación de la aplicación y no escalarla. Cuando necesitan más capacidad, sólo añaden nuevas máquinas al clúster y dejan que la base de datos encuentre la manera de organizar todo (Chorodow, 2010). A continuación se enumeran algunas características que favorecen el escalamiento en MongoDB:

Indexado._ MongoDB soporta índices secundarios genéricos, lo que permite una variedad de consultas rápidas, y proporciona indexado único, compuesto y también disponible para capacidades geoespaciales.

Almacenamiento Javascript._ En lugar de procedimientos almacenados, los desarrolladores pueden almacenar y utilizar las funciones y valores JavaScript en el lado del servidor.

Agregación._ MongoDB soporta MapReduce y otras herramientas de agregación.

Colecciones de tamaño fijo._ Son colecciones encapsuladas que son fijadas en tamaño y son útiles para ciertos tipos de datos, tales como *logs*.

Almacenamiento de Archivos._ MongoDB soporta un protocolo fácil de usar para almacenar grandes archivos y archivos de metadatos.

MongoDB utiliza un protocolo binario como el principal modo de interacción con el servidor (opuesto al protocolo con más sobrecarga, como *HTTP / REST*), añade relleno dinámico a los documentos y archivos de datos para preasignar uso de espacio extra para un rendimiento consistente y utiliza archivos asignados en memoria en el motor de almacenamiento por defecto, lo que empuja a la responsabilidad de la gestión de memoria para el sistema operativo. También cuenta con un optimizador dinámico de consulta que "recuerda" la forma más rápida para realizar una consulta.

Aunque MongoDB es de gran alcance, ésta no tiene la intención de hacer todo lo que hace una base de datos relacional. Siempre que sea posible, el servidor de base de datos descarga el procesamiento y la lógica para el lado del cliente (ya sea manejado por los *drivers* o por código de la aplicación de un usuario). Por consiguiente, el mantenimiento de este diseño flexible es una de las razones por las que MongoDB puede lograr un alto rendimiento (Chorodow, 2010).

2.5.1 Modelo de Datos

Datos como Documentos

MongoDB almacena los datos como documentos en una representación binaria llamada *BSON (Binary JSON)*. La codificación *BSON* extiende la representación popular *JSON (JavaScript Object Notation)* para incluir otros tipos como *int*, *long*, y *float*. Los documentos *BSON* contienen uno o más campos, y cada campo contiene un valor de un tipo de dato específico, incluyendo arreglos, datos binarios y sub-documentos (MongoDB, 2015).

Los documentos que tienden a compartir una estructura similar, se organizan como colecciones. Por lo tanto, puede ser útil pensar en una colección como análoga a una tabla en una base de datos relacional: los documentos son similares a las filas, y los campos son similares a las columnas.

Por ejemplo, se puede considerar el modelo de datos para una aplicación de *blogging*. En una base de datos relacional el modelo de datos comprendería varias tablas. Por ejemplo en la Figura 16 se asume que hay tablas para “Categorías”, “Etiquetas”, “Usuarios”, “Comentarios” y “Artículos”.

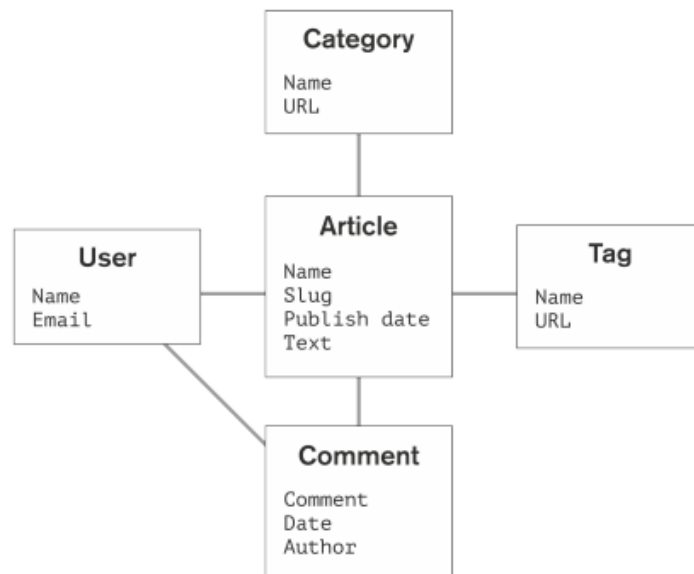


Figura 16 Ejemplo de un modelo relacional para una aplicación de blogging (MongoDB, 2015)

En MongoDB los datos podrían ser modelados como dos colecciones, una para usuarios y otra para artículos. En cada documento de blog que puede haber varios comentarios, múltiples etiquetas y múltiples categorías, cada una se expresa como un conjunto integrado.

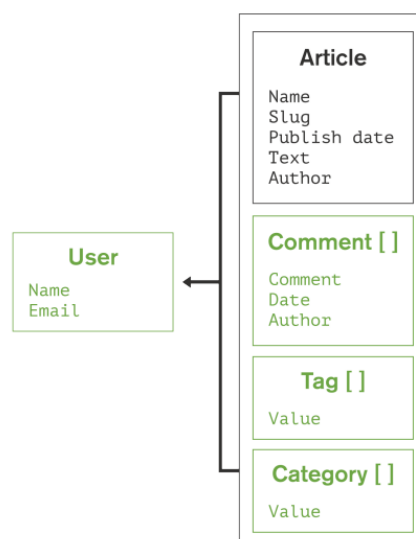


Figura 17 Los datos como documentos, simple para los desarrolladores y rápido para los usuarios (MongoDB, 2015)

Como se ilustra en este ejemplo, los documentos de MongoDB tienden a tener todos los datos de un registro dado en un solo documento, mientras que la información de una base de datos relacional para un registro se suele propagarse a través de muchas tablas. Con el modelo documental de MongoDB, los datos son más localizables, lo que reduce significativamente la necesidad de *JOINS* para tablas separadas. El resultado muestra un alto rendimiento y escalabilidad porque con una sola lectura a la base de datos se puede recuperar todo el documento que contiene todos los datos relacionados.

Además, los documentos BSON de MongoDB, están más estrechamente alineados con la estructura de los objetos en el lenguaje de programación. Esto hace que sea más sencillo y más rápido para los desarrolladores el modelar cómo los datos de la aplicación se asignarán a los datos almacenados en la base de datos.

2.5.2 Modelo de Consultas

MongoDB proporciona controladores nativos para todos los lenguajes de programación y *frameworks* para un desarrollo natural. Los *drivers* compatibles incluyen Java, .NET, Ruby, PHP, JavaScript, Node.js, Python, Perl, PHP, Scala y otros, mientras que los drivers de MongoDB están diseñados para ser compatibles para cualquier lenguaje dado (MongoDB, 2015).

Una diferencia fundamental en comparación con bases de datos relacionales es que el modelo de consulta MongoDB se implementa como métodos o funciones dentro de la *API* de un lenguaje de programación específico, en contraposición a un lenguaje completamente separado como SQL. Esto, junto con la afinidad entre el modelo de documento JSON de MongoDB y las estructuras de datos utilizadas en la programación orientada a objetos, hace simple la integración con las aplicaciones (Chorodow, 2010).

Intérprete de Mongo

Todas las distribuciones de Mongo poseen un intérprete interactivo JavaScript. Casi todos los comandos soportados por MongoDB se pueden emitir a través del intérprete, incluidas las operaciones administrativas.

Tipos de Consultas

A diferencia de otras bases NoSQL, MongoDB no está limitada a las operaciones clave-valor, porque los desarrolladores pueden construir aplicaciones poderosas usando

consultas complejas e índices secundarios que dividen la información en datos estructurados y semiestructurados.

Un elemento clave de la flexibilidad de MongoDB es el soporte para muchos tipos de consultas. Una consulta puede devolver un documento, un subconjunto de campos específicos dentro del documento o agregaciones complejas de muchos documentos, por ejemplo:

- Las consultas de clave-valor devuelven resultados basados en cualquier campo del documento, generalmente la llave primaria.
- Las consultas de rango devuelven resultados basados en valores como las desigualdades (por ejemplo mayor que, menor que, igual que, entre).
- Las consultas geoespaciales devuelven resultados basados en criterios de proximidad, intersección e inclusión que pueden ser especificados por un punto, línea, círculo o polígono.
- Las consultas de búsqueda de texto revuelven resultados en orden de relevancia basados en argumentos de texto usando operadores booleanos (AND, OR, NOT).
- Las consultas de agregación devuelven agregaciones (count, min, max, average y similares a GROUP BY de SQL).
- Las consultas Map Reduce ejecutan procesamiento complejo de datos que es expresado en Javascript.

2.5.3 Indexación

Las indexaciones son un mecanismo crucial para optimizar el rendimiento del sistema y la escalabilidad, mientras proporcionan un acceso flexible a los datos. Sin embargo, cuando los índices mejoran el rendimiento de algunas operaciones en órdenes de magnitud, incurren en gastos indirectos asociados en operaciones de escritura, uso de disco, y el consumo de memoria, algo común que sucede en la mayoría de los sistemas de gestión de base de datos; para evitar eso, el motor de almacenamiento *WiredTiger* de MongoDB comprime índices en la memoria RAM, liberando más carga de trabajo establecida para los documentos (MongoDB, 2015).

MongoDB incluye soporte para muchos tipos de índices secundarios que se pueden declarar en cualquier campo en el documento, incluidos los campos dentro de arreglos. Se detallan a continuación algunos tipos de índices:

Índices Únicos._ Al especificar un índice como único, MongoDB rechazará inserciones de nuevos documentos o la actualización de un documento con un valor existente para el campo, para el que se ha creado el índice único. Por defecto, todos los índices no se establecen como únicos y si se especifica un índice compuesto como único, la combinación de valores debe ser único.

Índices Compuestos._ Puede ser útil crear índices compuestos para las consultas que especifican múltiples predicados. Por ejemplo, una aplicación que almacena datos sobre los clientes, necesita encontrar clientes basados en el nombre, apellido y ciudad de residencia; al definir un índice compuesto en el nombre, apellido y ciudad de residencia, se puede especificar los tres valores en las consultas, y éstas podrían localizar personas de manera eficiente. Un beneficio adicional de un índice compuesto es que se puede utilizar cualquier campo en el índice, por lo tanto pueden ser necesarios menos índices en campos individuales. Cabe mencionar que este índice compuesto también optimizaría las consultas en busca de clientes por apellido.

Índices de arreglos._ Para los campos que contienen un arreglo, cada valor del arreglo se almacena como una entrada de índice separada. Por ejemplo, los documentos que describen “productos” podrían incluir un campo para los “componentes”. Si hay un índice en el campo de componentes, cada componente está indexado y las consultas en el campo componente pueden ser optimizadas por este índice. No hay ninguna sintaxis especial necesaria para la creación de índices de arreglos, si el campo contiene un arreglo, éste se indexará como un índice de arreglo.

Índices TTL (time to live)._ Los índices *time to live* son utilizados porque permiten al usuario especificar un período de tiempo tras el cual los datos se eliminarán automáticamente de la base de datos y son utilizados cuando los datos deben expirar del sistema de forma automática. Los índices TTL son muy utilizados en aplicaciones que mantienen un historial de las acciones del usuario como los *clicks*.

Índices Geoespaciales._ MongoDB proporciona índices geoespaciales para optimizar las consultas a la ubicación dentro de un espacio bidimensional, tales como sistemas de proyección de la Tierra.

Índices Dispersos._ Estos índices sólo contienen entradas para los documentos que contienen el campo especificado, porque el modelo de datos documental de MongoDB permite flexibilidad en el modelo de datos de documento a documento.

Índices de búsqueda de texto._ MongoDB proporciona un índice especializado para la búsqueda de texto que utiliza reglas lingüísticas avanzadas del idioma. Las consultas que utilizan el índice de búsqueda de texto devolverán documentos en orden de relevancia, en consecuencia uno o más campos pueden ser incluidos en el índice de texto

2.5.4 Motores de Almacenamiento

El motor de almacenamiento por defecto para MongoDB es un motor asignado en memoria ó *memory mapped*. Cuando el servidor arranca, todos sus archivos de datos son asignados en memoria. Es entonces la responsabilidad del sistema operativo es gestionar el copiado de datos de los datos a disco y de paginación. Por lo tanto, este motor de almacenamiento tiene varias propiedades importantes:

- El código de MongoDB para la gestión de memoria es pequeño y limpio, porque la mayor parte de ese trabajo se empuja al sistema operativo.
- El tamaño virtual de un proceso de servidor MongoDB es a menudo muy grande, superando el tamaño de todo el conjunto de datos. Esto está bien, ya que el sistema operativo se encargará de mantener la cantidad de datos residentes en la memoria contenida.
- MongoDB no puede controlar el orden en que los datos se escriben en el disco, lo que hace imposible usar un *write-ahead log* para proporcionar durabilidad de un solo servidor (se está trabajando en un motor de almacenamiento alternativo para MongoDB para proporcionar durabilidad de un solo servidor).
- Los servidores MongoDB de 32 bits están limitados a un total de alrededor de 2 GB de datos por *mongod*. Esto se debe a que todos los datos debe ser direccionables utilizando sólo 32 bits (Chorodow, 2010).

2.5.5 BSON

Los documentos en MongoDB son un concepto abstracto, la representación concreta de un documento varía según el controlador/lenguaje que se utilice. Debido a que los documentos se utilizan ampliamente para la comunicaciones en MongoDB, también es necesario que haya una representación de los documentos que es compartido por todos los *drivers*, las herramientas y procesos del ambiente MongoDB y esa representación se llama Binario JSON (BSON). Además, BSON es un formato binario ligero capaz de representar e interpretar cualquier documento MongoDB como una cadena de bytes. Por lo tanto, los documentos se guardan en el disco bajo este formato.

Cuando un *driver* recibe un documento para insertar o utilizarlo como una consulta, se codificará ese documento a BSON antes de enviarlo al servidor. Del mismo modo, los documentos que se devuelven al cliente desde el servidor se envían como cadenas BSON y estos datos BSON se decodifica por el conductor para su representación documento nativo antes de ser devuelto al cliente (Chorodow, 2010).

El formato BSON tiene tres objetivos principales:

- a) *Eficiencia.*_ BSON está diseñado para representar datos de manera eficiente, sin necesidad de utilizar mucho espacio adicional. En el peor de los casos, BSON es ligeramente menos eficiente que JSON y en el mejor de los casos (por ejemplo, cuando se almacena datos binarios o datos numéricos grandes), es mucho más eficiente.
- b) *Transitabilidad.*_ En algunos casos, BSON hace un sacrificio de eficiencia de espacio para hacer que el formato sea más fácil de insertar. Por ejemplo, los valores de cadena son prefijados con una longitud en vez de depender de un terminador para indicar el final de una cadena. Esta transitabilidad es útil cuando el servidor MongoDB necesita hacer una introspección de documentos.
- c) *Rendimiento* Por último, BSON está diseñado para ser rápido para codificar y decodificar. Se utiliza representaciones de estilo C para los tipos, que son rápidos para trabajar con en la mayoría de los lenguajes de programación.

2.6 Comparación entre MongoDB y MySQL

2.6.1 Terminología y conceptos

Muchos conceptos en MySQL son análogos en MongoDB. En la siguiente tabla se describen algunos de los conceptos comunes en cada sistema.

MYSQL	MongoDB
Tabla	Colección
Fila	Documentos
Columna	Campo
Joins	Documentos embebidos, <i>linking</i>

Figura 18 Conceptos comunes en cada sistema de base de datos

2.6.2 Lenguajes de Consulta

Tanto MySQL como MongoDB tienen un rico lenguaje de consulta. El SQL en MySQL significa *Structured Query Language*. Eso es porque se tiene que poner una cadena en este lenguaje de consulta que es analizado por el sistema de base de datos y esto es lo que hace posibles los ataques de inyección SQL (MongoDB, 2015).

MongoDB utiliza objetos de consulta. Es decir, se pasa de un documento para explicar lo que está consultando, por lo que no hay ningún lenguaje que analizar.

MySQL	MongoDB
<pre>INSERT INTO usuarios(id_usuario,edad, estado) VALUES ("bcd001", 45, "APROBADO")</pre>	<pre>db.usuarios.insert({ id_usuario: "bcd001", edad: 45, estado: "APROBADO" })</pre>
<pre>SELECT * FROM usuarios WHERE id_usuario= '007' AND edad= 50</pre>	<pre>db.usuarios.find ({ "id_usuario": "007", "edad": 50 }).pretty()</pre>
<pre>UPDATE usuarios SET status = "C" WHERE age > 25</pre>	<pre>db.usuarios.update({ edad: { \$gt: 25 } }, { \$set: { estado: "C" } }, { multi: true })</pre>
<pre>DELETE FROM usuarios where estado = 'APROBADO'</pre>	<pre>db.usuario.remove({ estado : "APROBADO" })</pre>

Figura 19 Comparación entre las formas de consulta de las dos bases de datos.

2.6.3 Relaciones

La sentencia *JOIN* es una de las mejores funciones de MySQL y de las bases de datos relacionales en general, ya que permite realizar consultas a través de múltiples tablas (Crews, 2013).

MongoDB no soporta *JOINS*, pero lo hace con tipos de datos multidimensionales como arreglos e incluso otros documentos. La colocación de un documento dentro de otro se conoce como incrustación ó *embedding*. Por ejemplo, si se fuera a crear un blog a través

de MySQL, se tendría una tabla de publicaciones y una tabla para comentarios. En MongoDB es posible que tenga una única colección de “publicaciones”, y un *array* de comentarios en cada publicación.

2.6.4 Conjunto de Características

Al igual que MySQL, MongoDB ofrece un conjunto rico de características y funcionalidades más allá de los ofrecidos en sistemas de clave-valor (MongoDB, 2015).

	MySQL	MongoDB
Modelo rico de datos	NO	SI
Esquema Dinámico	NO	SI
Datos de Localización	NO	SI
Actualizaciones de Campos	SI	SI
Fácil para programadores	NO	SI
Transacciones cmplejas	SI	NO
Auditoría	SI	SI
<i>Auto-Sharding</i>	NO	SI

Figura 20 Comparación de algunas funcionalidades más comunes entre las dos bases de datos

2.6.5 Definición de Esquema

Para almacenar información en MySQL, se requiere definir primero las tablas y columnas; mientras tanto, que en MongoDB no se define el esquema, sólo se ingresa la información en los documentos y el propio gestor de bases de datos adecúa la estructura según los requerimientos (Crews, 2013).

CAPÍTULO 3 - INSTALACIÓN

3.1 Análisis de Requisitos

La versión de MySQL a instalarse será MySQL Server 5.6.17 liberada en Marzo del 2014, mientras que para MongoDB se usará la versión 2.6.1 liberada en Mayo del 2014. Los dos sistemas de gestión de base de datos serán instalados en el sistema operativo Windows

Los requisitos previos para MySQL son:

- Visual Studio Tools for Office 2012 Runtime
- Microsoft .NET Framework 4 Client Profile
- Microsoft .NET Framework 4 Client Profile
- Microsoft Visual C++ 2010 32-bit runtime
- Microsoft .NET Framework 4 Client Profile

3.2 Instalación de MySQL

El método más simple y recomendado es descargar *MySQL Installer* (para Windows) el cual instala y configura todos los productos de MySQL en el sistema.

3.2.1 Diseño de Instalación por defecto en Windows

Para MySQL 5.6 en Windows, el directorio de instalación por defecto es C:\Archivos de programa\MySQL\MySQL Server 5.6. Algunos usuarios de Windows prefieren instalar en C:\mysql. Sin embargo, el diseño de los subdirectorios sigue siendo el mismo (MySQL, 2014).

3.2.2 Determinación de la versión a instalar

Para MySQL 5.6, hay múltiples paquetes de instalación para escoger cuando se instala MySQL en Windows.

MySQL Installer Este paquete tiene un nombre similar a *mysql-installer-community-5.6.27.0.msi* ó *mysql-installer-commercial-5.6.27.0.msi* y utiliza *MSI's* para instalar automáticamente MySQL Server y otros productos. Se descarga y aplica cambios a sí mismo para cada uno de los productos instalados. También configura productos adicionales del servidor (MySQL, 2014). Los productos instalados son configurables, y esto incluye: documentación con muestras, ejemplos y conectores (como C, C++, J, NET, and ODBC), MySQL Workbench, MySQL Notifier, MySQL para Excel y MySQL Server con sus componentes.

3.2.3 Primeros pasos

Conectarse y desconectarse del servidor

Para conectar con el servidor, lo normal es que tenga que proporcionar un nombre de usuario de MySQL cuando invoque `mysql` y una contraseña.

```
shell> mysql -h host -u user -p
Enter password: *****
```

`host` y `user` representan el *hostname* donde el servidor MySQL está corriendo y el nombre de usuario de la cuenta de MySQL todo esto antes especificado durante la instalación de MySQL. Si todo funciona debería observarse lo siguiente:

```
shell> mysql -h host -u user -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 5.6.26-
standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Ingresar Consultas

Éste es un comando simple que pide al servidor que le diga su número de versión y la fecha actual.

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----+-----+
| VERSION() | CURRENT_DATE |
+-----+-----+
| 5.6.1-m4-log | 2010-08-06 |
+-----+-----+
1 row in set (0.01 sec)
mysql>
```

La sentencia `SHOW` muestra las bases de datos que actualmente existen en el servidor.

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| mysql    |
| test     |
| tmp      |
+-----+
```

La base de datos `mysql` describe los privilegios de acceso de usuario. La base de datos `test` a menudo está disponible como un espacio de trabajo para que los usuarios realicen algunas pruebas.

Creación y Uso de una base de datos

```
mysql> CREATE DATABASE mibase;
```

Para usar `mibase` como la base actual, se realiza:

```
mysql> USE mibase
Database changed
```

Una base de datos se debe crear una sola vez, pero hay que seleccionarla para su uso cada vez que se comienza una sesión `mysql`.

```
shell> mysql -h host -u user -p mibase
Enter password: *****
```

Creación de tablas

Antes de crear una tabla, es importante decidir la estructura de la base de datos (qué tablas se necesita y qué columnas irán en cada una de ellas). Para crear una tabla, se usa la sentencia `CREATE TABLE`:

```
shell> CREATE TABLE miTabla (name VARCHAR(20), sex CHAR(1),
-> species VARCHAR(20), birth DATE, death DATE);
```

Una vez que se ha creado la tabla, la sentencia `SHOW TABLES`, que muestra las tablas disponibles dentro de la base de datos.

```
mysql> SHOW TABLES;
+-----+
| Tables in miBase |
+-----+
| miTabla          |
+-----+
```

Para verificar que la tabla fué creada, se usa la sentencia DESCRIBE:

```
mysql> DESCRIBE miTabla;
```

Field	Type	Null	Key	Default	Extra
name	varchar(20)	YES		NULL	
species	varchar(20)	YES		NULL	
sex	char(1)	YES		NULL	
birth	date	YES		NULL	
death	date	YES		NULL	

Carga de datos en una tabla

Para cargar información en una tabla se utiliza la sentencia INSERT que agrega nuevos registros de uno en uno.

```
mysql> INSERT INTO pet  
-> VALUES ('Tony','hamster','f','1999-03-30',NULL);
```

Recuperación de datos de una tabla

La sentencia SELECT es usada para recuperar información de una tabla. La forma general de la sentencia es:

```
SELECT qué_seleccionar  
FROM cuál_tabla  
WHERE condiciones_a_satisfacer;
```

3.3 Instalación de MongoDB

Instalación de MongoDB es un proceso simple en la mayoría de plataformas. Los binarios precompilados están disponibles para Linux, Mac OS X, Windows y Solaris. Para instalar MongoDB en Windows, hay que descargar el zip de Windows desde la página de descargas MongoDB, existen versiones de 32 bits y de 64 bits para Windows. Entonces se procede a descomprimir el archivo, por ejemplo en: c:\mongodb\

Configurar el ambiente de MongoDB

La carpeta (directorio de datos) que MongoDB utiliza por defecto para almacenar los datos es C:\data\db Esta carpeta hay que crearla manualmente antes de iniciar el servidor. De igual manera se crea la carpeta C:\data\log

Crear el archivo de configuración

Durante la instalación se creó el archivo en `C:\mongodb\mongod.cfg` que especifica:

```
systemLog:
  destination: file
  path: c:\data\log\mongod.log
storage:
  dbPath: c:\data\db
```

Crear el servicio MongoDB

Los servicios de Microsoft Windows, permiten crear aplicaciones ejecutables de larga duración, que se ejecutan en sus propias sesiones de Windows. Estos servicios pueden iniciarse automáticamente cuando el equipo arranca, se pueden pausar y reiniciar, y no muestran ninguna interfaz de usuario. Estas características hacen que los servicios resulten perfectos para ejecutarse en un servidor o donde se necesite una funcionalidad de ejecución larga, que no interfiera con los demás usuarios que trabajen en el mismo equipo.

3.3.1 Primeros pasos

Arrancar el servidor MongoDB

Para iniciar MongoDB, se debe arrancar *mongod.exe* desde el símbolo del sistema de Windows.

```
C:\mongodb\bin\mongod.exe
```

Esto inicia el proceso principal de la base de datos. El mensaje *waiting for connections* en la consola de salida indica que el proceso *mongod.exe* esta corriendo exitosamente.

Dependiendo del nivel de seguridad de Windows, puede aparecer una ventana emergente sobre el bloqueo de “algunas características” de *C:\mongodb\bin\mongod.exe* de comunicación en redes. Todos los usuarios deberían seleccionar *Redes Privadas como mi casa o red de trabajo* y permitir el acceso.

Conectarse al servidor de MongoDB

Para conectarse al servidor, se usa mongo.exe.

```
c:\mongodb\bin> mongo.exe
MongoDB shell version: 2.2.3
connecting to: test
> //mongodb shell
```

Creación de una base de datos

Para crear una base de datos con el nombre **mibase** , se introduce la sentencia `use DATABASE:`

```
> use mibase
switched to db mibase
```

Para comprobar la base de datos seleccionada actualmente utiliza el comando `db`

```
> db
mibase
```

Si desea comprobar la lista de bases de datos, a continuación, utilice el comando `show dbs`.

```
> show dbs
local      0.78125GB
test       0.23012GB
```

Creación de una colección

La sentencia `db.createCollection(name, options)` es usada para crear una colección. En donde *name* es el nombre de la colección a ser creada y *options* es un documento que es usado para especificar la configuración de la colección.

```
>use mibase
switched to db mibase
>db.createCollection("micoleccion")
{ "ok" : 1 }
>
```


Se puede revisar la colección creada.

```
>show collections
micoleccion
system.indexes
```

Sin embargo, en MongoDB no se necesita crear colecciones, porque éstas son creadas automáticamente cuando se inserta un documento.

```
>db.mibase.insert({"name" : "coleccion2"})
>show collections
mycol
mycollection
system.indexes
coleccion2
>
```

Inserción de documentos

Se puede utilizar el método *insert()* para agregar documentos a una colección en MongoDB, y si intenta agregar documentos a una colección que no existe, MongoDB creará la colección para ello. A continuación se inserta un documento en una colección llamada *restaurantes*.

```
db.restaurantes.insert(
  {
    "direccion" : {
      "calle" : "2 Avenue",
      "zip" : "10075",
      "edificio" : "1480",
      "coord" : [ -73.9557413, 40.7720266 ],
    },
    "ciudad" : "Manhattan",
    "cocina" : "Italian",
    "nombre" : "Vella",
    "restaurant_id" : "41704620"
  }
)
```

Consultar Datos

Se puede usar el método *find()* para emitir una consulta que recupera datos de una colección en MongoDB (todas las consultas tienen el alcance de una sola colección). Éste método puede devolver todos los documentos en una colección o sólo los documentos

que coinciden con un filtro o criterio especificado. Además, se puede especificar el filtro o criterios en un documento y pasar como un parámetro para el método *find()*.

El método *find()* devuelve resultados de la consulta en un cursor, que es un objeto iterable que produce documentos. Por ejemplo, la siguiente operación consulta para todos los documentos de la colección “restaurantes”.

```
db.restaurantes.find( { ciudad: 'Manhattan', cocina:
'Italiana' })
```

Actualizar Datos

Se puede utilizar el método *update()* para actualizar los documentos de una colección. El método acepta como sus parámetros:

- un documento de filtro para que coincida con los documentos a actualizar,
- un documento de actualización para especificar la modificación a hacerse, y
- un parámetro de opciones (opcional).

Por defecto, el método *update()* actualiza un documento único. Se usa la opción *multiple* para actualizar todos los documentos que coinciden con un criterio.

```
db.restaurantes.update( { 'cocina': 'Italiana' }, { $set: { 'cocina': '
Peruana' } })
```

Eliminar datos

Se puede utilizar el método *remove()* para eliminar documentos de una colección, el cual toma un documento de condiciones que determina los documentos de eliminar. Para especificar una eliminación con una condición, se debe utilizar la misma estructura y sintaxis que las condiciones de la consulta. A continuación la siguiente operación elimina todos los documentos que coinciden con la condición especificada.

```
db.restaurants.remove( { "ciudad": "Manhattan" } )
```

CAPÍTULO 4 - ANÁLISIS DE RENDIMIENTO

4.1 Descripción del problema

El desempeño de un sistema de base de datos debe ser óptimo para satisfacer las necesidades de acceso a la información de los usuarios. Por lo tanto, la comparación a realizar entre las dos bases de datos (MongoDB y MySQL) es importante, ya que nos permite obtener conclusiones en cuanto a su capacidad de procesamiento de datos y cómo éstos gestores manejan grandes cantidades de transacciones y datos (conocidos como *Big Data*).

Las bases de datos juegan un rol importante en las aplicaciones y una decisión equivocada tomada al principio puede tener efectos graves, es por ello que hay que tener en consideración que el rendimiento y la escalabilidad de las bases de datos son los factores más importantes, además de la confiabilidad. Puede ser difícil el comparar las diferentes opciones que trae cada base de datos, debido a los diferentes diseños, configuraciones y métodos de acceso a los datos; por lo cual, en esta tesis se intenta tratar de encontrar un punto medio donde sus implementaciones estén lo más cerca como sea posible y las pruebas realizadas no favorezcan a un sistema de base de datos sobre el otro.

4.2 Plataforma Hardware

El equipo usado en las pruebas de rendimiento, posee las siguientes características

Tipo:	Sony Vaio VPCEH14FM
Procesador:	Intel® Core™ i3-2310M + (4 threads)
Velocidad	2.10 GHz
RAM	4GB
Disco Duro	500GB

4.3 Plataforma Software

Sistema Operativo	Windows 7 Ultimate Edition 64bits
Gestor de Base de Datos	MySQL 5.6.25 Community Server 64bits
Gestor de Base de Datos	MongoDB 2.6.10 64bits
Lenguaje de Programación	PHP 5.6
Driver PHP para MongoDB	php_mongo-1.5.5-5.6 -x86
Driver PHP para MySQL	MySQL trae por defecto el driver para PHP

4.4 Esquema de la base de datos

El esquema de base de datos usado fue modelado en base a una aplicación de música en internet la cual utiliza diferentes algoritmos para sugerir canciones a los usuarios de acuerdo a sus gustos y sugerir a otros usuarios quienes tienen gustos similares. El esquema de base de datos mostrada en la Figura 21, fue diseñada para la implementación en MySQL, ésta ha sido normalizada y se ha eliminado cualquier duplicación de datos entre tablas.

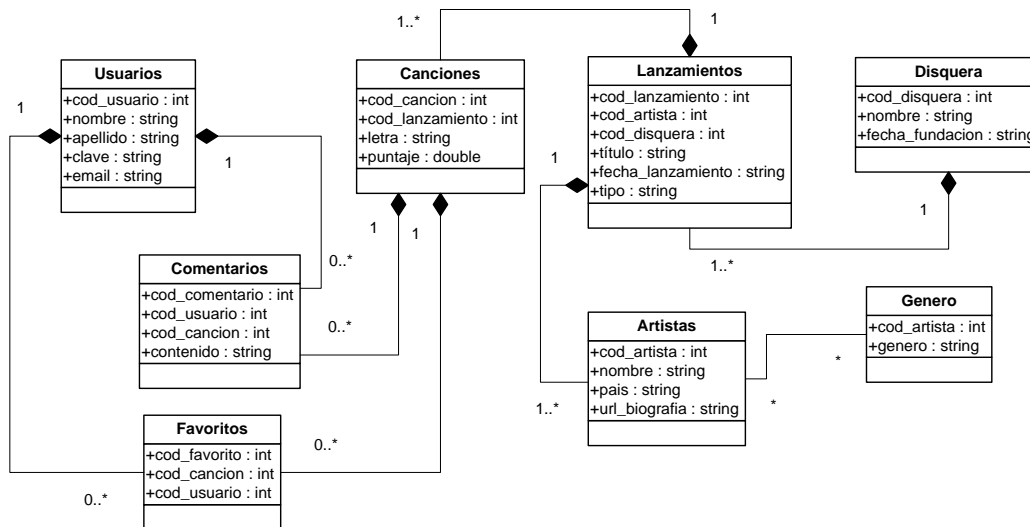


Figura 21 Esquema de base de datos relacional

Aunque el mismo esquema informalmente puede ser usado en MongoDB, se han hecho algunos cambios para compensar el hecho de que MongoDB no soporta operaciones complicadas tales como *JOINS* que son típicos de los RDBMS. En cambio, el esquema de base de datos MongoDB combina algunas tablas para tomar ventaja de los subdocumentos.

El proceso de combinar tablas es desnormalizar aunque en este caso es solamente parcial (la desnormalización completa significaría la combinación de todas las tablas en una sola tabla). Adicionalmente, el diseño del esquema de MongoDB permite una comparación más directa con MySQL para tareas similares como *JOINS* o *SELECTs*. En la Figura 22 el esquema para MongoDB incluye subdocumentos los cuales fueron adaptados del esquema relacional. La colección “Lanzamiento” ahora incluye los atributos de la tabla “Disquera”, porque cada registro tiene solamente una disquera; también incluye un subdocumento “Canciones”, el cual reemplaza la tabla “Canciones”. La justificación para

utilizar un subdocumento para las canciones en cada lanzamiento, es porque generalmente el número de canciones es relativamente pequeño (10 a 20 canciones) , y éstas canciones no cambian despues de que un lanzamiento esta hecho. Además, MongoDB permite transacciones rápidas cuando se manejan subdocumentos, comparado con encontrar datos en otras colecciones debido al formato binario que MongoDB usa para su representación interna.

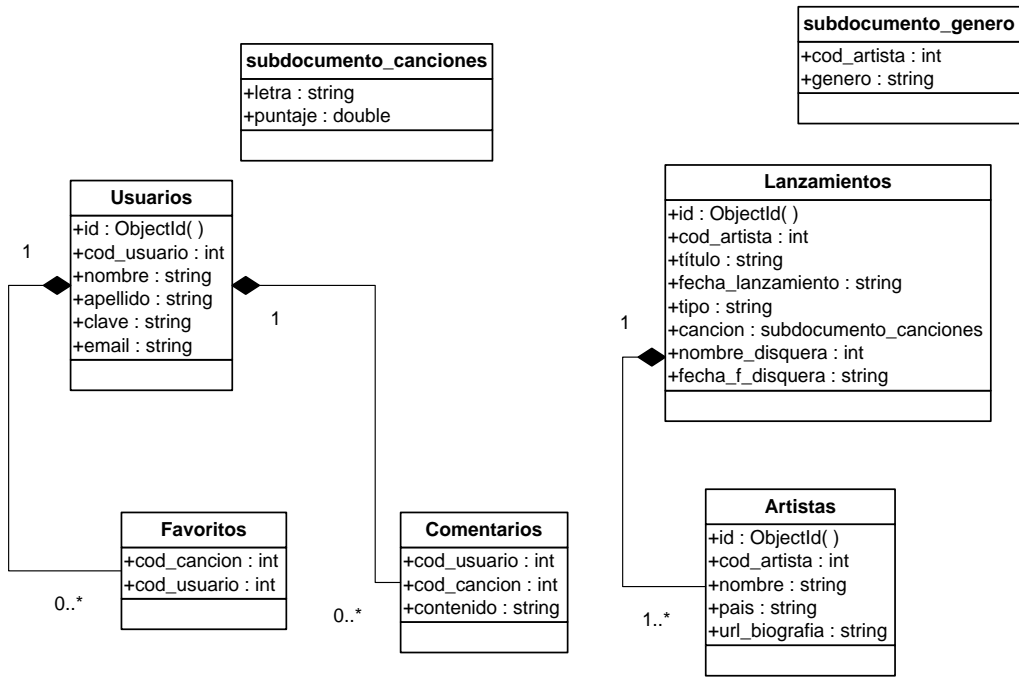


Figura 22 Esquema de la base de datos en MongoDB

4.5 Consultas a los gestores de bases de datos

Test No 1			
Operación	<i>INSERT</i>		
Objetivo	Obtener el tiempo total y el tiempo promedio de inserción de 5000,50000,500000 registros en dos los sistemas gestores de bases de datos.		
Proceso	<p>Se procede a utilizar un script realizado en PHP, que realiza la inserción de miles de registros en una tabla/colección que poseen un modelo de datos similar. El tiempo de ejecución se mide con la funcion <i>microtime()</i>.</p> <p>Este no es un caso de carga masiva, para lo cual la base datos debería configurarse de diferente manera y utilizar herramientas propias como <i>mysqldump</i> para mysql y la función análoga en MongoDB llamada <i>insert()</i>.</p> <p>Sin embargo, esta prueba de inserción puede ser aplicable al trabajar con datos de <i>clickstreams</i> o flujos de datos financieros.</p>		
Variables	Rendimiento	Indicadores	

Descripción	Se realizará 5 inserciones por cada proceso.
Herramientas	PHP microtime()
Procedimiento	
1. Crear el script en PHP para insertar los miles de registros en la tabla y colección de los gestores de base de datos. 2. Agregar la función <i>microtime()</i> al script ya creado. 3. Iniciar el servicio mysql 4. Ejecutar el script 5. Obtener el tiempo utilizado para la tarea. 6. Detener el servicio mysql 7. Iniciar el servicio mongod 8. Ejecutar el script 9. Obtener el tiempo utilizado para la tarea.	
<pre>-- SQL INSERT INTO ms_artista(codigo, nombre, pais,url_biografia) VALUES ('45059', 'Jose Jose', 'Argentina', 'ww.rior.com') -- MONGODB > musica.artistas.insert({ _id: ObjectId(), cod_cancion: '64765', cod_artista: '45059', nombre: 'Jose Jose', pais: 'Argentina', url_biografia: 'ww.rior.com' })</pre>	

Test No 2			
Operación	SELECT - FIND		
Objetivo	Obtener el tiempo total y el tiempo promedio de consulta de los diferentes consultas select.		
Proceso	Se intentará medir el tiempo de respuesta de las diferentes operaciones <i>select</i> , entre las cuales constan: 1. Selects a una tabla , con alguna condición a una columna indexada con diferentes cláusulas <i>limit</i> . 2. Selects a una tabla, con alguna condición a una columna no indexada, con diferentes cláusulas <i>limit</i> . 3. Selects a una tabla, con alguna condición a una columna indexada y con ordenamiento.		
Variables	Rendimiento	Indicadores	
Descripción	Se realizará 3 <i>selects</i> por cada operación.		
Herramientas	PHP microtime()		
Procedimiento			

1. Crear el script en PHP para seleccionar los registros en la tabla con la cláusula *limit* para los dos gestores de base de datos.
2. Agregar la función *microtime()* al script ya creado.
3. Iniciar el servicio mysql
4. Ejecutar el script
5. Obtener el tiempo utilizado para la tarea.
6. Detener el servicio mysql
7. Iniciar el servicio mongod
8. Ejecutar el script
9. Obtener el tiempo utilizado para la tarea.

-- SQL

```
SELECT * FROM musica.ms_artista
where codigo=81 and cod_artista=534789
and nombre='Antonio Gardner' and pais='Poland'
and url_biografia= 'squidoo.com'
```

-- MONGODB

```
db.artistas.find( {_id:
ObjectId("55da5cc009d046700700263d"),cod_artista: 76,
cod_cancion:915, nombre: "Marilyn Myers",pais:"China"})
```

Test No 3			
Operación	DELETE - REMOVE		
Objetivo	Obtener el tiempo total y el tiempo promedio al borrar un número determinado de filas en los dos gestores de bases de datos.		
Proceso	Luego de realizar la inserción masiva de registros (actualmente la tabla y colección “artistas” cuenta con 55000 registros), se procederá a medir el tiempo que los dos gestores tardan en la eliminación de registros bajo los siguientes criterios: 1. <i>Deletes</i> a una tabla que satisfagan una condición dada para dos columnas no indexadas. 2. <i>Deletes</i> a una tabla que satisfagan una condición dada para todas las columnas existentes.		
Variables	Rendimiento	Indicadores	
Descripción	Se realizará tres <i>deletes</i> por cada proceso.		
Herramientas	PHP microtime()		
Procedimiento			
1. Crear el script en PHP para insertar los miles de registros en la tabla y colección de los gestores de base de datos.			
2. Agregar la función <i>microtime()</i> al script ya creado.			
3. Iniciar el servicio mysql			

4. Ejecutar el script
5. Obtener el tiempo utilizado para la tarea.
6. Detener el servicio mysql
7. Iniciar el servicio mongod
8. Ejecutar el script
9. Obtener el tiempo utilizado para la tarea.

-- SQL

```
DELETE FROM musica.ms_artista
where codigo=81 and cod_artista=534789
and nombre='Antonio Gardner' and pais='Poland'
```

-- MONGODB

```
db.artistas.remove( {_id:
ObjectId("55da5ca009d046700700263d"),cod_artista: 6,
cod_cancion:4253, nombre: "Stephen Freeman",pais:"Indonesia"})
```

4.6 Métricas

Para medir el rendimiento de las bases de datos en lo que se refiere a tareas de consulta y manipulación de datos, es necesario una métrica común. El factor más importante para una aplicación es el tiempo requerido para completar una tarea, y en el caso de las bases de datos es el tiempo requerido para completar una transacción.

Los scripts creados en PHP miden el tiempo requerido para completar un número determinado de transacciones, ya que cada transacción en sí misma es insignificante. También, se obtiene el promedio del tiempo de ejecución de cada *benchmark* realizado, con el fin de tener una idea global de cómo se comportan las bases de datos frente a factores externos como: un proceso del sistema operativo utilizando temporalmente el CPU o una operación de I/O.

La función *microtime()* que provee PHP obtiene el tiempo inicial en microsegundos, luego se ejecutan los scripts, a los cuales se calculará el tiempo de procesamiento; de nuevo se obtiene el tiempo que es el tiempo final, y a éste se le resta del tiempo inicial para obtener el tiempo que toma el procesamiento cada script.

Este procedimiento es el mismo para los diferentes scripts de las dos bases de datos que se realizarán las pruebas. El tiempo a medir va desde la conexión de la base de datos, hasta la ejecución de la consulta.

4.7 Pruebas de Rendimiento

A continuación se muestran los resultados simplificados de las pruebas de rendimiento, el código fuente utilizado realizar las pruebas se encuentra en el ANEXO I – “SCRIPTS UTILIZADOS PARA LAS PRUEBAS DE RENDIMIENTO”.

4.7.1 Operación: Inserción masiva de registros

La inserción masiva de registros en los dos gestores de base de datos se realiza desde un cliente (driver PHP). Hay que tener en cuenta que esta inserción no es una carga masiva (*bulk-loading*). El objetivo de esta prueba es observar el comportamiento de la base de datos, según el número de registros insertados.

La siguiente tabla, muestra el tiempo promedio de inserción para diferentes números de registros en MongoDB y MySQL.

Número de registros insertados	Tiempo de inserción (segundos)	
	MySQL	MongoDB
1000	0,82	1,23
10000	24,09	2,30
100000	322,63	13,50

Durante la primera fase de las pruebas, que es la fase de carga de registros, se han realizado se han insertado en total 555000 registros de 1Kb cada uno.

4.7.2 Operación: Consulta masiva

Se realizarán tres tipos de consultas con diferentes grados de complejidad, para obtener resultados más confiables del comportamiento de los dos gestores frente a cada tipo de consulta. El tiempo de respuesta se medirá en milisegundos debido a que los resultados obtenidos en las pruebas iniciales son menores a un segundo, cabe mencionar que para el *select* de 10 millones de registros se bajó la memoria RAM a 2Gb para observar el comportamiento de los gestores en el disco duro. En MongoDB, para proceder con la consulta se creó previamente un índice, en el campo “cod_artista” como se muestra en la siguiente figura:

```

> db.artistas.createIndex(<<"cod_artista":1>>)
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> _

```

Figura 23 Creación de un índice en MongoDB para el campo “cod_artista”

Las siguientes tablas muestran el tiempo promedio de ejecución de las consultas en los dos gestores de bases de datos:

- a) Consultas *select* a una tabla, con una condición a una columna indexada con diferentes cláusulas *limit*.

Número de registros recuperados	Tiempo de consulta (milisegundos)	
	MySQL	MongoDB
1000	10,40	7,90
10000	36,40	48,87
100000	338,0	103,01
10000000	60660,55	3572,41

- b) Selects a una tabla, con alguna condición a una columna no indexada, con diferentes cláusulas *limit*.

Número de registros recuperados	Tiempo de consulta (milisegundos)	
	MySQL	MongoDB
1000	15,60	6,24
10000	47,20	36,19
100000	348,40	78,80
10000000	79627,74	1128,40

- c) Selects a una tabla, con alguna condición a una columna indexada y con ordenamiento.

Número de registros recuperados	Tiempo de consulta (milisegundos)	
	MySQL	MongoDB
1000	46,80	15,71
10000	41,60	70,39
100000	343,20	73,76
10000000	91395,36	1071,20

4.7.3 Operación: Eliminación Masiva

Se realizarán dos tipos de eliminación de registros, a las cuales se les llamará “Primera Operación” que borrará registros que cumplan con una condición para dos columnas no indexadas y “Segunda Operación ” que cumplan alguna condición para todas las columnas.

Operaciones	Tiempo de consulta (milisegundos)	
	MySQL	MongoDB
Primera Operación	31,20	3,16
Segunda Operación	691,60	99,89

4.8 Análisis de Resultados

Durante la fase de carga, se comenzó con la inserción de 1000 registros. Las claves primarias fueron generadas automáticamente para ambos gestores. Luego de ejecutar el primer script se arrojaron resultados similares para los dos gestores, 0.8 segundos para MongoDB y 1.2 segundos para MySQL. Sin embargo, al aumentar el número de registros a 10000, el tiempo de respuesta de MySQL cambió de manera drástica a 24.1 segundos, mientras que para MongoDB sólo se incrementó el tiempo a sólo 2.3 segundos.

En la última fase de inserción de 100000 registros existe una marcada diferencia entre los dos gestores de casi 310 segundos. A continuación, se muestra un gráfico del tiempo promedio de inserción de los dos gestores:

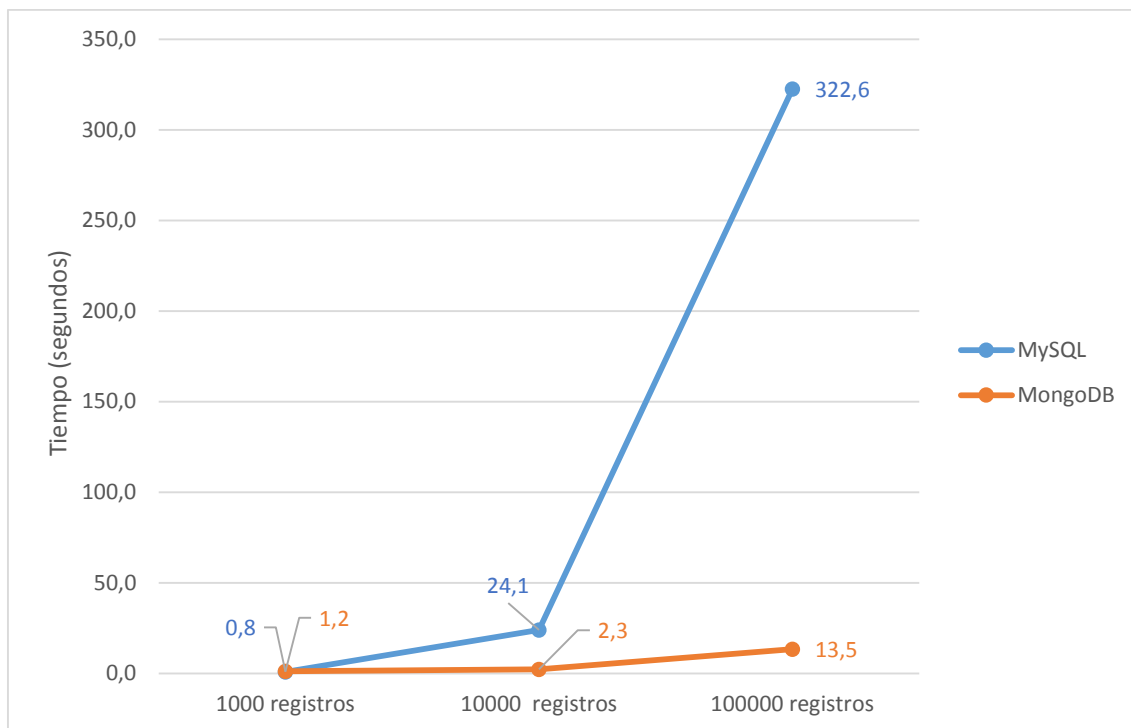


Figura 244 Tiempo promedio de inserción de los dos gestores de bases de datos.

Para la segunda fase de pruebas, el rendimiento de los dos gestores se medirá en milisegundos, esto se debe a que los tiempos de respuesta no sobrepasan las unidades en segundos. En esta fase de consulta de registros, se puede observar que MongoDB mantiene la ventaja sobre MySQL a medida que el número de registros recuperados se incrementa.

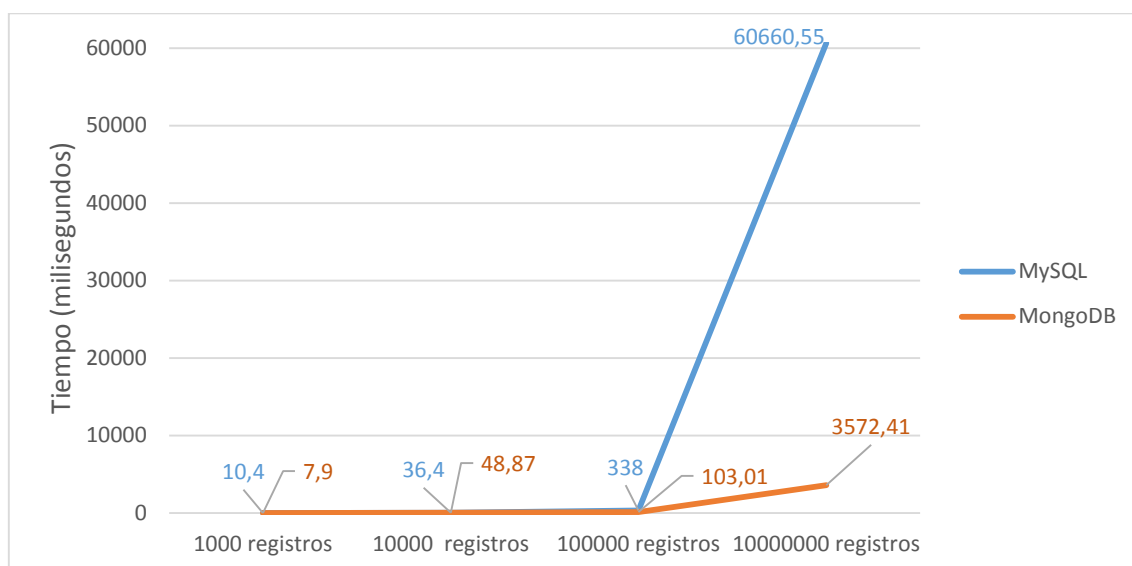


Figura 255 Tiempo promedio de la primera consulta en los dos gestores de bases de datos.

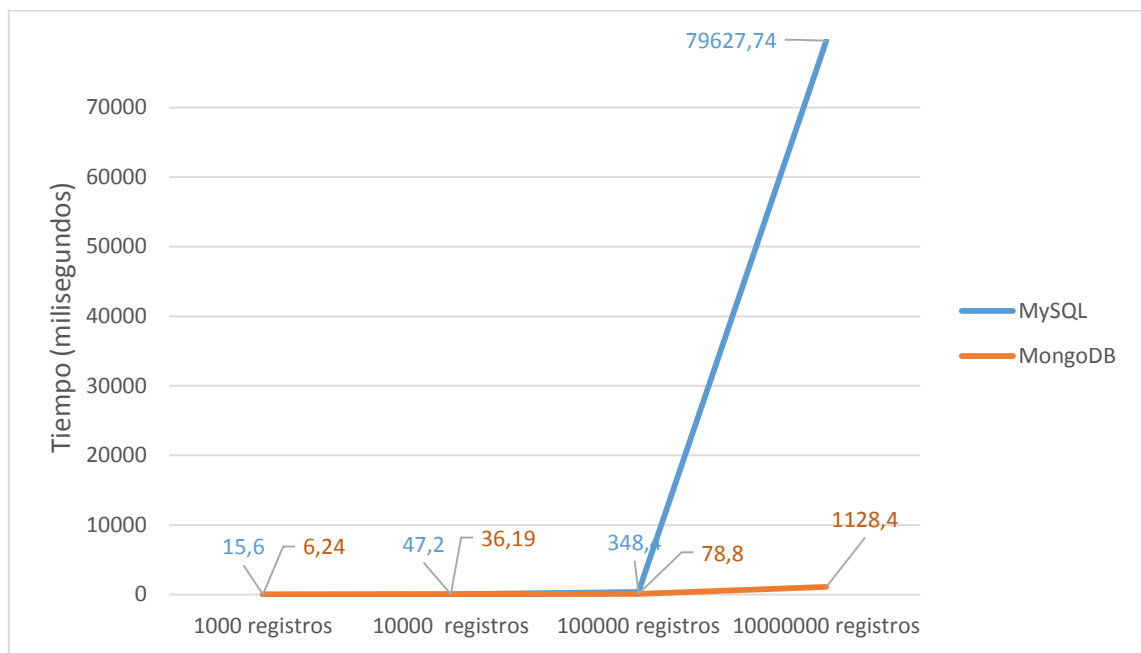


Figura 266 Tiempo promedio de la segunda consulta en los dos gestores de bases de datos.

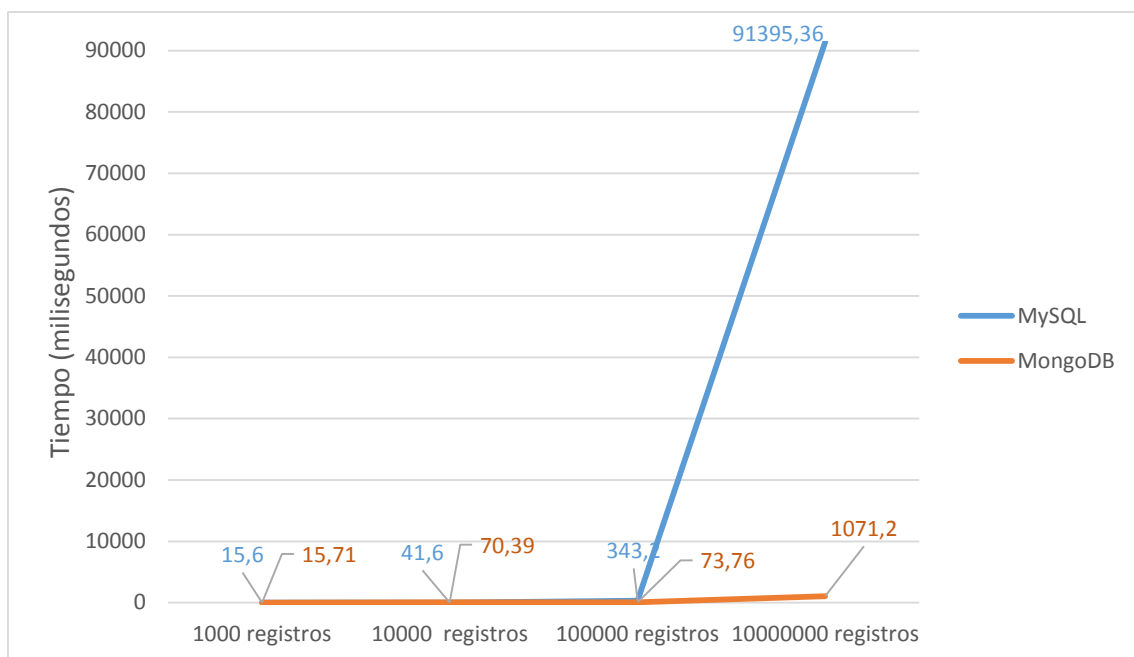


Figura 277 Tiempo promedio de la tercera consulta en los dos gestores de bases de datos.

Es importante mencionar que luego de remover del equipo 2Gb de RAM, para observar el comportamiento del uso de memoria, MySQL sobrepasa en más de 600Mb a MongoDB como se puede observar a continuación:

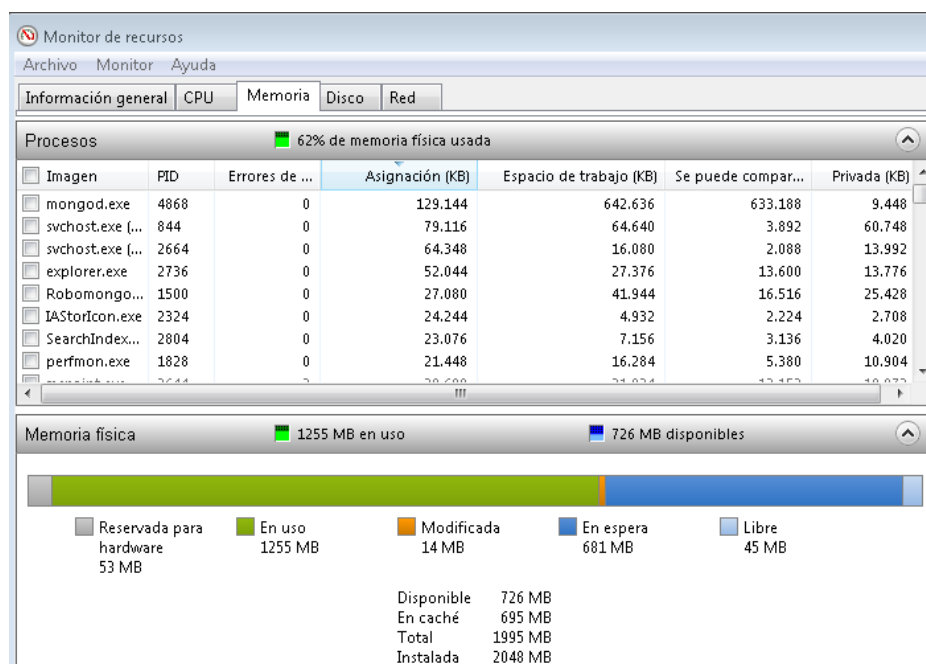


Figura 288 Uso de memoria de MongoDB.

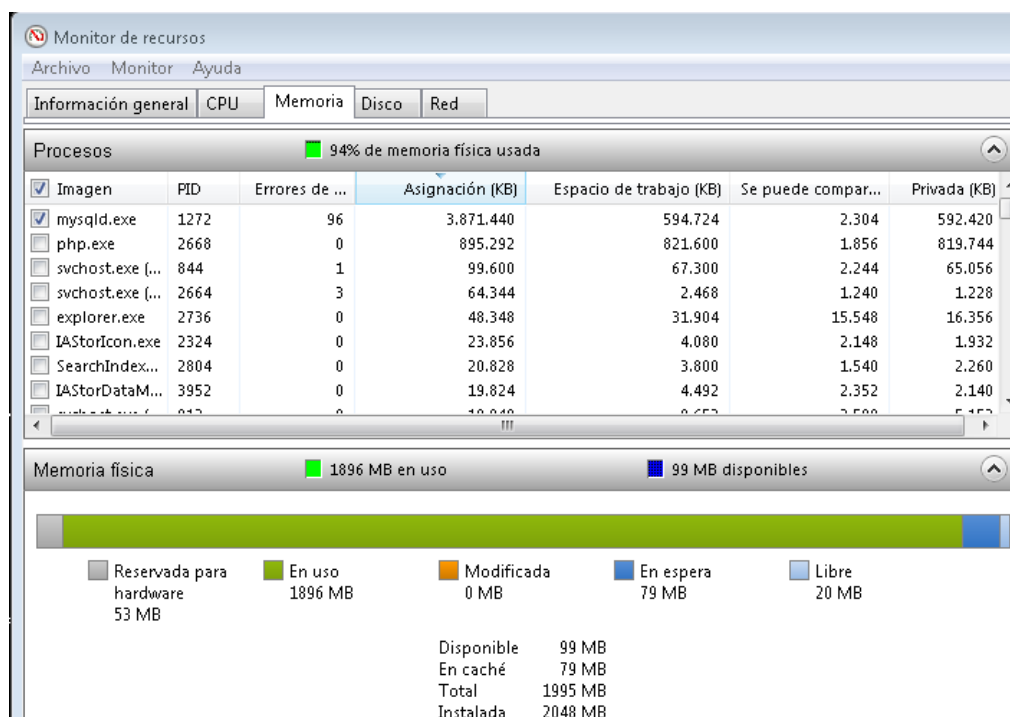


Figura 299 Uso de memoria de MySQL.

En la tercera fase, de igual manera se siguieron las métricas aplicadas para las pruebas anteriores y se ejecutaron dos operaciones de eliminación de registros. Se puede observar que MongoDB muestra una clara ventaja en la segunda operación de eliminación.

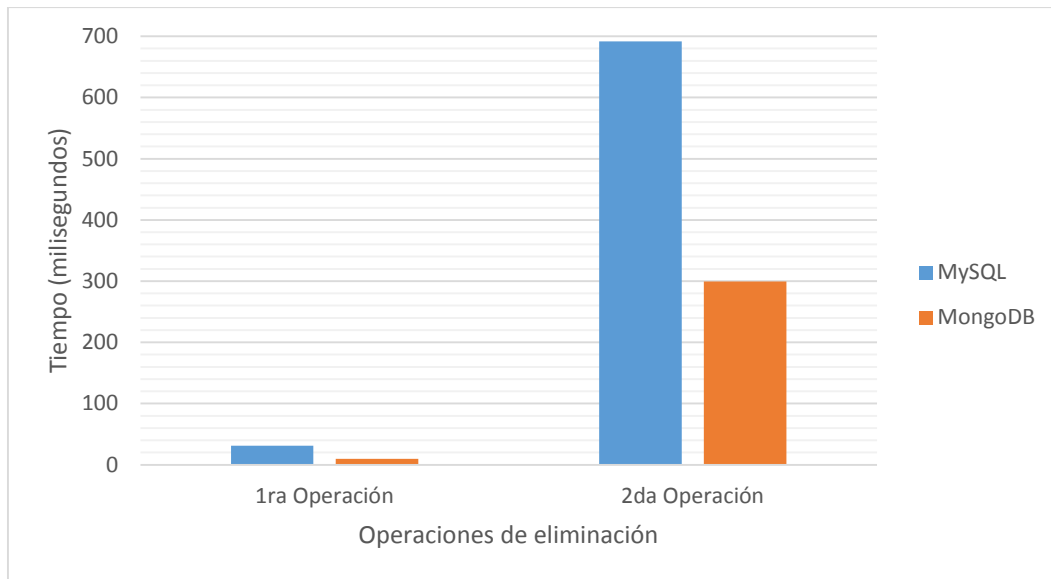


Figura 30 Tiempo promedio de eliminación de registros.

4.9 Tendencia a futuro de los sistemas gestores de bases de datos.

Gartner, Inc es una compañía de investigación y asesoramiento, en el campo de las tecnologías de información más importante del mundo y anualmente presenta el Cuadrante Mágico de Gartner que tiene como objetivo proporcionar un análisis cualitativo en un mercado, su dirección, madurez, innovaciones y participantes.

Gartner publicó su cuadrante mágico en el año 2014, que presentaba a MongoDB como un proveedor de tecnología desafiante (*challenger*), lo cual significa que este proveedor domina un segmento largo de mercado pero que no demostraba un entendimiento de la dirección del mercado al cual apuntaba.



Figura 30 Comparación del Cuadrante Mágico para Sistemas de Gestión de Bases de Datos Operacionales para el año 2014 y 2015 (Datastax, 2015).

Es importante mencionar que MongoDB además de ser NoSQL, entra en la categoría de base de datos operacional porque permite administrar datos dinámicos en tiempo real y se puede observar que MongoDB se posiciona en el cuadrante del año 2015, dentro de la categoría de *leaders*, que significa que ya está posicionado en el mercado a largo plazo (MySQL no se menciona directamente en el reporte, porque forma parte de los productos de Oracle).

De igual manera, en el reporte realizado por Forrester Research, otra empresa de investigación de mercados, se hizo la evaluación bajo los criterios: rendimiento, escalabilidad, integración y alta disponibilidad, en la cual MongoDB comparte el liderazgo con MarkLogic, como se indica a continuación:

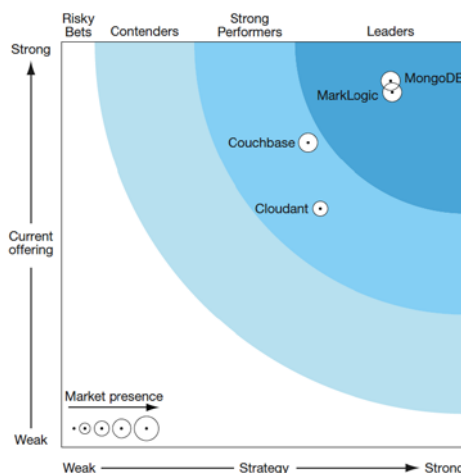


Figura 31 Reporte *Forrester Wave* para bases de datos NoSQL Orientadas a Documentos (Forrester Inc, 2015).

CONCLUSIONES

El cambio desde un ambiente SQL a NoSQL, es un desafío para los desarrolladores de sistemas. Esto se debe a que la historia y la evolución de la gestión de los sistemas de almacenamiento de datos, han estado estrechamente vinculados con las bases de datos relacionales y el lenguaje de consulta SQL, que es el lenguaje estándar para las bases relacionales.

Este tipo de bases han desempeñado su trabajo sorprendentemente bien, hasta que en los últimos años comenzaron a desarrollarse sistemas de software que demandaban altos volúmenes de transacciones y datos de distinto género, los cuales, afectaban la escalabilidad de todo el sistema porque difícilmente se podía modelar y manejar altos volúmenes de datos con el uso de las bases de datos relacionales. Tradicionalmente, la manera de resolver este problema fue el aumentar las capacidades del hardware (llamada también escalabilidad vertical), sin embargo, esta solución llegó a un punto donde el costo económico se volvió demasiado alto y la gestión del sistema demasiado compleja. Por lo tanto, el movimiento NoSQL ha presentado una solución a los desafíos recientes que enfrentan las bases de datos relacionales, porque proveen esquemas dinámicos, modelado de datos flexible, arquitectura escalable y almacenamiento eficiente de grandes datos, características que aumentan el rendimiento y escalabilidad.

De acuerdo a ese panorama, se realizó la comparación de rendimiento de los dos gestores de código abierto, con el uso del lenguaje PHP como cliente y la aplicación de música en internet para las pruebas de carga, lectura y eliminación.

Al realizar las pruebas de inserción, se ha demostrado que en términos de tiempos de ejecución, MongoDB supera a MySQL, esto se debe a que MongoDB no impone un esquema a los documentos que son almacenados en la colección. Debido a que, cada documento puede tener su propio conjunto definido de campos, sin tener la necesidad de alterar la estructura o crear otra colección; por ejemplo, en la aplicación de música desarrollada para las pruebas, se tiene la colección de artistas a la cual se desea agregar el campo lugar_nacimiento, para ello sólo se inserta un documento con el nuevo campo sin problemas, esto incrementa significativamente la velocidad; aunque si obliga especificar un tipo de datos, característica que comparte con MySQL.

En la etapa de recuperación de registros, los índices son muy importantes, porque con ellos se pueden obtener los registros más rápidamente. En el caso de MongoDB todos los documentos tienen una clave primaria embebida llamada *_id*, este campo es asignado automáticamente al insertar un documento, aunque es modificable; estas características dan una cierta ventaja sobre MySQL a medida que el número de registros seleccionados se incrementa.

En los tres tipos de pruebas realizadas, MongoDB fue superior al lograr ejecutar las tareas en menor tiempo, lo cual es muy importante cuando una aplicación debe soportar un uso intensivo de manipulación de datos, siempre y cuando no sean operaciones complejas. No se pretende eliminar el uso de las bases de datos relacionales, porque contienen características únicas como el mantenimiento de la integridad de datos, con el uso de las *foreign keys* y *joins*.

Se concluye que no existe la solución perfecta para todas las aplicaciones que requieren un gestor de almacenamiento de datos; si los datos son críticos para una empresa, MySQL es la mejor opción debido a sus propiedades de integridad de los datos, pero si se manejan sistemas de administración de contenido, administración de logs, foros y blogs, los cuales son generados de forma masiva, MongoDB es la solución.

ÍNDICE DE ABREVIATURAS

RDBMS *Relational Database Management System*. Un tipo de sistema de gestión de base de datos (DBMS) que almacena los datos en forma de tablas relacionadas.

XML *Extensible Markup Language*. Es una especificación desarrollada por el W3C., diseñado especialmente para los documentos Web.

JSON *JavaScript Object Notation*. Es un formato de intercambio de datos ligera que es fácil para los seres humanos a leer y escribir, y para las máquinas el analizar y generar. JSON se basa en la notación objeto del lenguaje JavaScript.

BSON *Binary JSON*. BSON es un formato de intercambio de datos usado principalmente para su almacenamiento y transferencia en la base de datos MongoDB y es una representación binaria de estructuras de datos y mapas.

ACID *Atomicity Consistency Isolation Durability*. El término ACID expresa la función que las transacciones desarrollan en aplicaciones críticas cuando éstas son ejecutadas y fue acuñado por los pioneros en el procesamiento de transacciones, el acrónimo ACID responde a los términos atomicidad, coherencia, aislamiento y permanencia.

API *Application Program Interface*. Es un conjunto de rutinas, protocolos y herramientas para la construcción de aplicaciones de software, los APIs se utilizan en la programación de componentes de la interfaz gráfica de usuario (GUI) y especifican cómo deben interactuar los componentes de software,

SPARQL *Simple Protocol and RDF Query Language*. SPARQL define un protocolo de lenguaje de consulta y acceso de datos estándar para su uso con el modelo de datos *Resource Description Framework* (RDF) y funciona para cualquier fuente de datos que se pueden asignar a RDF.

XPATH *XPath* se utiliza para navegar por los elementos y atributos en un documento XML.

REST *REpresentational State Transfer*. Es una arquitectura simple sin estado que generalmente se ejecuta a través de HTTP.

SQL *Structured Query Language*. SQL es un lenguaje de consulta estándar para solicitar información a una base de datos.

MVCC *Multi-Version Concurrency Control*. Es una técnica avanzada para mejorar el rendimiento de base de datos multiusuario. Además se utiliza en la base de datos PostgreSQL, que es software libre.

SIG *Sistemas de Información Geográfica* es un sistema de computadoras para capturar, almacenar, verificar y mostrar información relacionada con las posiciones de la superficie de la Tierra.

CSRF *Cross-Site Request Forgery* es un ataque que obliga a un usuario final para ejecutar acciones no deseadas en una aplicación web en la que actualmente están autenticados.

GLOSARIO

BIG DATA es una palabra de moda, o eslogan, utilizado para describir un volumen masivo de datos tanto estructurados como no estructurados que es tan grande que es difícil de procesar utilizando técnicas de bases de datos y software tradicionales. En la mayoría de los escenarios de la empresa, el volumen de datos es demasiado grande o se mueve demasiado rápido o se supera la capacidad de procesamiento actual.

CLICK STREAMS es una pista virtual que un usuario deja tras de sí mientras se navega por Internet. Un *clickstream* es un registro de la actividad de un usuario en Internet, incluyendo todos los sitios web y todas las páginas de cada sitio web que el usuario visita, el tiempo que el usuario estaba en una página o sitio, en qué orden se visitaron las páginas, los grupos de noticias en donde el usuario participa e incluso las direcciones de correo electrónico de correo electrónico que el usuario envía y recibe.

PROXY CACHE se le conoce como una *Web proxy-cache*, es una función de un servidor proxy que almacena en caché las páginas Web recuperados en el disco duro del servidor para que la página pueda ser recuperada rápidamente por el mismo usuario o otro diferente la próxima vez que se solicite esa página. El caché de proxy facilita requisitos de ancho de banda y reduce los retrasos que son inherentes a una red conectada a Internet con mucho tráfico.

CLUSTERING es la conexión de dos o más ordenadores, de una manera tal que se comporten como un solo equipo. El *clustering* se utiliza para el procesamiento paralelo, balanceo de carga y tolerancia a fallos.

MAP REDUCE es un paradigma de procesamiento de datos para condensar grandes volúmenes de datos en resultados globales útiles.

POOLING es un término informático utilizado en entornos de cloud computing para describir una situación en la que los proveedores sirven a varios clientes o "inquilinos" con servicios provisionales y escalables.

SENSITIVE CASE describe la capacidad de un programa para distinguir entre letras mayúsculas y letras minúsculas.

WRITE-AHEAD LOGGING es una familia de técnicas para proporcionar atomicidad y durabilidad (dos de las propiedades ACID) en los sistemas de bases de datos.

MONGOD es el demonio primario para el sistema de MongoDB. Se ocupa de las solicitudes de datos, gestiona el formato de datos, y realiza operaciones de gestión de *background*.

DRIVER es un programa que controla un dispositivo. Cada dispositivo, ya sea una impresora, unidad de disco, o el teclado, debe tener un programa controlador y muchos *drivers*, como los de teclado, vienen con el sistema operativo.

WIRE PROTOCOL Un protocolo de conexión o *wire protocol* se refiere a una forma de obtener los datos de punto a punto. Un protocolo de conexión es necesario si tiene que interoperar más de una aplicación. En contraste con los protocolos de transporte como el nivel de transporte (como TCP o UDP), el término "protocolo de conexión" se utiliza para describir una forma común para representar la información en el nivel de aplicación.

SHARDING es la frase que se usa para describir una partición horizontal en una base de datos o motor de búsqueda. La idea detrás de sharding es dividir los datos entre múltiples máquinas al tiempo que garantiza que los datos serán siempre accedidos desde el lugar correcto.

MBOX es el formato más común para almacenar mensajes de correo electrónico en un disco duro. Todos los mensajes para cada buzón de correo son almacenados como un archivo de texto único, largo de mensajes e-mail, comenzando con la cabecera *From* del mensaje.

CLUSTERED INDEX cada tabla InnoDB tiene un *clustered index* que es sinónimo de una primary key de una tabla. Es un índice especial donde es insertada la información para las filas.

BIBLIOGRAFÍA

- Antiñanco, J. (2013). *Bases de Datos NoSQL: Escalabilidad y alta disponibilidad a través de patrones de diseño*. Recuperado el 10 de 06 de 2015, de http://sedici.unlp.edu.ar/bitstream/handle/10915/36338/Documento_completo.pdf?sequence=5
- Asay, M. (2013). *Why NoSQL Trumps Relational Databases for Mobile Applications*. Recuperado el 18 de Marzo de 2015, de <http://www.techopedia.com/2/29256/development/mobile-development/why-nosql-trumps-relational-databases-for-mobile-applications>
- Brewer, E. (2000). *Towards Robust Distributed Systems*. Recuperado el 31 de 05 de 2015, de <http://www.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf>
- Chorodow, K. (2010). *MongoDB The Definitive Guide*. Recuperado el 15 de 06 de 2015, de <https://www.safaribooksonline.com/library/view/mongodb-the-definitive/9781449344795/>
- Connolly, S. (2012). *7 Key Drivers for the Big Data Market*. Recuperado el 09 de 06 de 2015, de 7 Key Drivers for the Big Data Market: <http://hortonworks.com/blog/7-key-drivers-for-the-big-data-market/>
- Couchbase. (2015). *Oracle, MySQL, Cassandra and MongoDB Among Developers to Benefit from Breakthrough Query Language*. Recuperado el 11 de 06 de 2015, de <http://www.couchbase.com/press-releases/couchbase-introduces-n1ql-breakthrough-query-language>
- Crews, K. (2013). *MySQL vs. MongoDB: Looking At Relational and Non-Relational Databases*. Recuperado el 29 de 06 de 2015, de <http://www.neonrain.com/blog/mysql-vs-mongodb-relational-and-non-relational-databases>
- DeCandia, G. (Diciembre de 2013). *Dynamo: Amazon's Highly Available Key-value Store*. Recuperado el 20 de Febrero de 2015, de <http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf>
- Facebook. (2015). *A Decentralized Structured Storage System*. Recuperado el 24 de 02 de 2015, de <https://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf>
- Floyer, D. (2014). *The Growth and Management of Unstructured Data*. Recuperado el 24 de 02 de 2015, de http://wikibon.org/wiki/v/The_Growth_and_Management_of_Unstructured_Data
- Google inc. (Diciembre de 2013). *Bigtable: A Distributed Storage System for Structured Data*. Recuperado el 20 de Febrero de 2015, de <http://static.googleusercontent.com/media/research.google.com/es//archive/bigtable-osdi06.pdf>

- Hecht, R., & Jablonski, S. (2011). *NoSQL Evaluation A Use Case Oriented Survey*. Recuperado el 11 de 06 de 2015, de <http://rogerking.me/wp-content/uploads/2012/03/DatabaseSystemsPaper.pdf>
- Ippolito. (2009). *Drop ACID and think about Data*. Recuperado el 31 de 05 de 2015, de <http://highscalability.com/drop-acid-and-think-about-data>
- Keller, S. (2012). *MongoDB An introduction and performance analysis*. Recuperado el 22 de 06 de 2015, de <http://wiki.hsr.ch/Datenbanken/files/MongoDB.pdf>
- LinkedIn. (2013). *LinkedIn Voldemort*. Recuperado el 02 de 24 de 2015, de <http://www.project-voldemort.com/voldemort/>
- Luke, I. (2014). *SQL vs NoSQL Database Differences Explained with few Example DB*. Recuperado el 16 de 06 de 2015, de <http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/>
- MarkLogic. (2014). *The NoSQL Generation : Embracing the Document Model*. Recuperado el 09 de 06 de 2015, de <http://www.marklogic.com/wp-content/uploads/2014/12/nosql-generation-embracing-document-model.pdf>
- Meijer, E., & Bierman, G. (2011). *A co-Relational Model of Data for Large Shared Data Banks*. Recuperado el 10 de 06 de 2015, de <http://algomagic.org/p30-meijer.pdf>
- MongoDB. (2015). *MongoDB and MySQL Compared*. Recuperado el 29 de 06 de 2015, de <https://www.mongodb.com/mongodb-and-mysql-compared>
- MongoDB. (2015). *MongoDB Architecture Guide*. Recuperado el 29 de 06 de 2015, de <https://www.mongodb.com/mongodb-architecture>
- MongoDB. (Febrero de 2015). *Top 5 Considerations When Evaluating NoSQL Databases*. Recuperado el 20 de Febrero de 2015, de http://info.mongodb.com/rs/mongodb/images/10gen_Top_5_NoSQL_Considerations.pdf?_ga=1.152315833.1641336935.1421449327
- Moniruzzaman, A. B., & Hossain, S. A. (2013). *NoSQL Database: New Era of Databases for Big data Analytics - Classification , Characteristics and Comparison*. Recuperado el 09 de 06 de 2015, de <http://arxiv.org/ftp/arxiv/papers/1307/1307.0191.pdf>
- MySQL. (2013). *What is MySQL*. Obtenido de <https://dev.mysql.com/doc/refman/5.6/en/what-is-mysql.html>
- MySQL. (2013). *Overview of MySQL Storage Engine Architecture*. Recuperado el 18 de 06 de 2015, de <http://dev.mysql.com/doc/refman/5.6/en/storage-engines.html>
- MySQL. (2014). *Choosing An Installation Package*. Obtenido de <http://dev.mysql.com/doc/refman/5.6/en/windows-choosing-package.html>
- MySQL. (2014). <http://dev.mysql.com/doc/refman/5.6/en/windows-installation-layout.html>. Obtenido de <http://dev.mysql.com/doc/refman/5.6/en/windows-installation-layout.html>
- Quiroz, J. (2003). *El modelo relacional de base de datos*. Obtenido de http://umad.zapatolibre.com/moodle_files/Articulo%20-%20El%20modelo%20relacional%20de%20bases%20de%20datos%20-%20Codd.pdf

- Quizhpe, P. (2009). *Diseño e Implementación de la Arquitectura de Datos Basada en Comparativas de Rendimiento entre SGBD*. Recuperado el 18 de 06 de 2015, de <http://dspace.esPOCH.edu.ec/bitstream/123456789/97/1/18T00373.pdf>
- Salgado, F. (2007). *Investigación, Bases de Datos e Investigación Web*. Obtenido de <http://www.uazuay.edu.ec/analisis/El%20modelo%20relacional.pdf>
- Sanchez, J. (2004). *Bases de datos relacionales*. Recuperado el 28 de Mayo de 2015, de <http://cursa.ihmc.us>
- Schwartz, B., & Zaitsev, P. (2012). *High Performance MySQL*. Beijing : O'Reilly Media. Recuperado el 18 de 06 de 2015
- Silberschatz, A., & Korth, H. F. (2006). *Fundamentos de base de datos*. Recuperado el 28 de Abril de 2015
- Software Developer's Journal. (15 de 04 de 2012). *Software Developer's Journal*, 14. Recuperado el 02 de 10 de 2014, de <http://sdjournal.org>: http://sdjournal.org/wp-content/uploads/downloads/2012/05/SDJTeaser_5_12.pdf
- StackExchange. (2011). *What are the main differences between InnoDB and MyISAM?* Recuperado el 18 de 06 de 2015, de <http://dba.stackexchange.com/questions/1/what-are-the-main-differences-between-innodb-and-mysam>
- StackExchange. (2012). *CAP Theorem vs. BASE (NoSQL)*. Recuperado el 07 de 06 de 2015, de <http://dba.stackexchange.com/questions/18435/cap-theorem-vs-base-nosql>
- Strozzi, C. (2010). *Philosophy of NoSQL*. Recuperado el 30 de 05 de 2015, de Philosophy of NoSQL: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/Philosophy%20of%20NoSQL
- Twitter. (2013). *Twitter Flock DB*. Recuperado el 24 de 02 de 2015, de <http://engineering.twitter.com/2010/05/introducing-flockdb.html>.
- Zaki, A. K. (2013). *NoSQL DATABASES: NEW MILLENNIUM DATABASE FOR BIG DATA, BIG USERS, CLOUD COMPUTING AND ITS SECURITY CHALLENGES*. Recuperado el 2015 de 03 de 03, de <http://esatjournals.org/Volumes/IJRET/2014V03/I15/IJRET20140315080.pdf>

ANEXO 1 – SCRIPTS UTILIZADOS PARA LAS PRUEBAS DE RENDIMIENTO

MySQL

Script utilizado para realizar *inserts* masivos de registros en MySQL.

```
<?php
//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$starttime = $mtime;
//=====TIMER=====

$servername = "";
$username = "root";
$password = "sopmvop1";
$hostname = "localhost";

// Crear conexion
$conn = mysql_connect($hostname, $username, $password);
mysql_select_db("musica", $conn);
for ($x = 0; $x < 100000; $x++) {
    $fp = fopen("1000.csv", "r");
    while(!feof($fp)) {
        $linea = fgets($fp);
        list($cod_cancion, $cod_artista, $nombre, $pais, $url)
= explode(";", $linea);
        $sql = "INSERT INTO ms_artista(codigo, nombre,
pais,url_biografia) VALUES ($cod_artista, '$nombre', '$pais',
'$url')";
        mysql_query($sql,$conn) or die(mysql_error());
    }
}

fclose($fp);

mysql_close($conn);
//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$endtime = $mtime;
$totaltime = ($endtime - $starttime);
//=====TIMER=====
//Imprimir tiempo en segundos
print "Tiempo de insercion";
echo "\r\n";
print $totaltime;
?>
```

Script utilizado para realizar *selects* a una tabla , con alguna condición a una columna indexada con diferentes cláusulas *limit*.

```
<?php
//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$starttime = $mtime;
//=====TIMER=====
    $servername = "";
    $username = "root";
    $password = "sopmvop1";
    $hostname = "localhost";

    // Crear conexion
    $conn = mysql_connect($hostname, $username, $password);

    $sql = 'SELECT cod_artista, codigo, nombre,pais,
url_biografia
        FROM ms_artista
        WHERE cod_artista>100
        LIMIT 1000';

    mysql_select_db("musica", $conn);

    $consulta = mysql_query( $sql, $conn );
    $num_rows = mysql_num_rows($consulta);
    if (!$consulta) {
        die('Invalid query: ' . mysql_error());
    }

    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $endtime = $mtime;
    $totaltime = ($endtime - $starttime);
    //=====TIMER=====

    //mostrar registros ingresados
    while($row = mysql_fetch_array($consulta))
    {
        echo "cod_artista :{$row['cod_artista']} ";
        echo "Codigo: {$row['codigo']} ";
        echo "Nombre: {$row['nombre']} ";
        echo "Pais: {$row['pais']} ";
        echo "\r\n";
    }

    mysql_close($conn);
```

```

    echo number_format($totaltime,3);
//Imprimir tiempo de consulta
    print "$num_rows";
    echo "\r\n";
    print "$totaltime";

    //$milliseconds = round(microtime(true) * 1000);
    //$milliseconds = round(microtime(true) * 1000);
?>

```

Script utilizado para realizar *selects* a una tabla , con alguna condición a una columna no indexada con diferentes cláusulas *limit*.

```

<?php
//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$starttime = $mtime;
//=====TIMER=====

    $servername = "";
    $username = "root";
    $password = "sopmvop1";
    $hostname = "localhost";

    // Crear conexion
    $conn = mysql_connect($hostname, $username, $password);

    $sql = 'SELECT cod_artista, codigo, nombre,pais,
url_biografia
        FROM ms_artista
        WHERE codigo>100
        LIMIT 100000';

    mysql_select_db("musica",$conn);

    $consulta = mysql_query( $sql, $conn );
    $num_rows = mysql_num_rows($consulta);
    if (!$consulta) {
        die('Invalid query: ' . mysql_error());
    }

//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$endtime = $mtime;
$totaltime = ($endtime - $starttime);
//=====TIMER=====

//mostrar registros ingresados
while($row = mysql_fetch_array($consulta))
{
    echo "cod_artista :{$row['cod_artista']} ";
    echo "Codigo: {$row['codigo']} ";
}

```

```

        echo "Nombre: {$row['nombre']} ";
        echo "Pais: {$row['pais']} ";
        echo "\r\n";
    }
    mysql_close($conn);
//Imprimir tiempo de consulta
    print "$num_rows";
    echo "\r\n";
    print "$totaltime";

?>

```

Script utilizado para realizar *selects* a una tabla , con alguna condición a una columna indexada y con ordenamiento.

```

<?php
//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$starttime = $mtime;
//=====TIMER=====
    $servername = "";
    $username = "root";
    $password = "sopmvop1";
    $hostname = "localhost";

    // Crear conexion
    $conn = mysql_connect($hostname, $username, $password);

    $sql = 'SELECT cod_artista, codigo, nombre,pais,
url_biografia
        FROM ms_artista
        WHERE codigo>100
        ORDER BY  cod_artista
        LIMIT 100000';

    mysql_select_db("musica", $conn);

    $consulta = mysql_query( $sql, $conn );
    $num_rows = mysql_num_rows($consulta);
    if (!$consulta) {
        die('Invalid query: ' . mysql_error());
    }

    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $endtime = $mtime;
    $totaltime = ($endtime - $starttime);
    //=====TIMER=====

//mostrar registros ingresados
while($row = mysql_fetch_array($consulta))

```

```

{
echo "cod_artista :{$row['cod_artista']} ";
echo "Codigo: {$row['codigo']} ";
echo "Nombre: {$row['nombre']} ";
echo "Pais: {$row['pais']} ";
echo "\r\n";
}
mysql_close($conn);

//Imprimir tiempo de consulta
print "$num_rows";
echo "\r\n";
print "$totaltime";

?>

```

Script utilizado para eliminar de forma masiva registros de una tabla, que satisfacen una condición dada para dos columnas no indexadas.

```

<?php
//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$starttime = $mtime;
//=====TIMER=====
$servername = "";
$username = "root";
$password = "sopmvop1";
$hostname = "localhost";

// Crear conexion
$conn = mysql_connect($hostname, $username, $password);

$sql = "DELETE
        FROM ms_artista
        WHERE codigo=87
        AND nombre='Sara Mendoza'";

mysql_select_db("musica", $conn);

$consulta = mysql_query( $sql, $conn );

// Muestra el número de registros eliminados
printf("Registros borrados: %d\n", mysql_affected_rows());
//valida consulta
$num_rows = mysql_num_rows($consulta);
if (!$consulta) {
die('Invalid query: ' . mysql_error());
}

//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];

```

```

        $endtime = $mtime;
        $totaltime = ($endtime - $starttime);
//=====TIMER=====

mysql_close($conn);
?>

```

Script utilizado para eliminar de forma masiva registros de una tabla, que satisfacen una condición dada para todas las columnas existentes.

```

<?php
//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$starttime = $mtime;
//=====TIMER=====
    $servername = "";
    $username = "root";
    $password = "sopmvop1";
    $hostname = "localhost";

    // Crear conexion
    $conn = mysql_connect($hostname, $username, $password);

    $sql = "DELETE
            FROM ms_artista
            WHERE cod_artista=127899
            AND codigo=42
            AND nombre='Gerald Dunn'
            AND pais='China' ";

    mysql_select_db("musica", $conn);

    $consulta = mysql_query( $sql, $conn );
    printf("Registros borrados: %d\n", mysql_affected_rows());
    // $num_rows = mysql_num_rows($consulta);
    // if (!$consulta) {
    //     die('Invalid query: ' . mysql_error());
    // }

    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $endtime = $mtime;
    $totaltime = ($endtime - $starttime);
//=====TIMER=====

    mysql_close($conn);

?>

```

MongoDB

Script utilizado para realizar las inserciones masivas de registros en una colección de MongoDB.

```
<?php

try {
    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $starttime = $mtime;
    //=====TIMER=====

    //Abrir la bd
    $conexion = new Mongo();
    $db = $conexion->selectDB('musica');
    $coll = $db->selectCollection('artistas');
    for ($x = 0; $x < 1000; $x++) {
        $fp = fopen("1000.csv", "r");

        while(!feof($fp)) {
            $linea = fgets($fp);
            $campos = explode(";", $linea);

            $doc = array(
                'cod_cancion' => (int)$campos[0],
                'cod_artista' => (int)$campos[1],
                'nombre' => $campos[2],
                'pais' => $campos[3],
                'url_biografia' => $campos[4]);

            $coll->insert($doc);
        }
    }

    fclose($fp);

    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $endtime = $mtime;
    $totaltime = ($endtime - $starttime);
    //=====TIMER=====
    //Obtener el tiempo de inserción
    print "Tiempo de insercion";
    echo "\r\n";
    print $totaltime;
} catch(MongoConnectionException $e) {
    die("No es posible conectarnos a la base de datos:".$e->getMessage());
}
```

```

catch(MongoException $e) {
    die('No es posible almacenar la informacion: '.$e-
    >getMessage());
}
?>

```

Script utilizado para realizar consultas *find* a una colección , con alguna condición a una columna indexada con diferentes cláusulas *limit*.

```

<?php
//try {

    // Configuracion
    $conexion = new Mongo();
    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $starttime = $mtime;
    //=====TIMER=====

    // Seleccionar bd
    $database = $conexion->selectDB('musica');

    // Obtener la coleccion artistas
    //$coleccion= $database->artistas;

    // Encontrar los artistas con codigo mayor a 50000
    $result = $database->artistas->find(array('cod_artista'
=> array( '$gt' => 50000)))->limit(1000);

    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $endtime = $mtime;
    $totaltime = ($endtime - $starttime);
    //=====TIMER=====

    //print "Tiempo de recuperacion de consulta";
    //echo "\r\n";
    //print $nResults;
    //echo "\r\n";
    print $starttime;
    echo "\r\n";
    print $endtime;
    echo "\r\n";
    print $totaltime;
    echo "\r\n";
    echo "Time: " . number_format(( microtime(true) -
$starttime), 4) . " Seconds\n";

    //} catch(MongoConnectionException $e) {

```



```
//die("No es posible conectarnos a la base de datos:".$$e-
>getMessage());
//}
//catch(MongoException $e) {
//die('No es posible -error'.'.$e->getMessage());
//}
?>
```

Script utilizado para realizar consultas *find* a una colección , con alguna condición a una columna no indexada con diferentes cláusulas *limit*.

```
<?php
try {
    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $starttime = $mtime;
    //=====TIMER=====

    // Configuracion
    $conexion = new Mongo();

    // Seleccionar bd
    $database = $conexion->selectDB('musica');

    // Obtener la coleccion artistas
    $coleccion= $database->artistas;

    // Encontrar los artistas con codigo mayo a 50000
    $query = array(
    'cod_cancion' => array( '$gt' => 100)
    );

    $result = $database->artistas->find($query);
    $result -> limit(100000);
    //foreach ($result as $doc) {
    //var_dump($doc);
    //}

    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $endtime = $mtime;
    $totaltime = ($endtime - $starttime);
    //=====TIMER=====

    print "Tiempo de recuperacion de consulta";
    echo "\r\n";
    print $totaltime;

    } catch(MongoConnectionException $e) {
    die("No es posible conectarnos a la base de datos:".$$e-
    >getMessage());
```

```

}
catch(MongoException $e) {
die('No es posible -error' . $e->getMessage());
}
?>

```

Script utilizado para realizar consultas *find* a una colección, con alguna condición a una columna indexada y con ordenamiento.

```

<?php
try {
    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $starttime = $mtime;
    //=====TIMER=====

    // Configuración
    $conexion = new Mongo();

    // Seleccionar bd
    $database = $conexion->selectDB('musica');

    // Obtener la colección artistas
    $coleccion = $database->artistas;

    // Encontrar los artistas con código mayor a 50000
    $query = array(
        'cod_artista' => array( '$gt' => 100)
    );

    $result = $database->artistas->find($query);
    $result -> limit(100000);
    $result -> sort(array('cod_artista' => 1));

    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $endtime = $mtime;
    $totaltime = ($endtime - $starttime);
    //=====TIMER=====

    print "Tiempo de recuperación de consulta";
    echo "\r\n";
    print $totaltime;

    } catch(MongoConnectionException $e) {
die("No es posible conectarnos a la base de datos:". $e->getMessage());
    }
    catch(MongoException $e) {
die('No es posible -error' . $e->getMessage());
    }
}

```

```
?>
```

Script utilizado para eliminar registros de forma masiva en una colección, que satisfagan una condición dada para dos campos no indexados.

```
<?php
try {
    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $starttime = $mtime;
    //=====TIMER=====

    $conexion = new Mongo();
    // Seleccionar bd
    $database = $conexion->selectDB('musica');

    //Eliminar registro
    $result = $database->artistas-
>remove(array('cod_artista'=>87, 'nombre'=>'Sara Mendoza'));

    //=====TIMER=====
    $mtime = microtime();
    $mtime = explode(" ", $mtime);
    $mtime = $mtime[1] + $mtime[0];
    $endtime = $mtime;
    $totaltime = ($endtime - $starttime);
    //=====TIMER=====

    print "Tiempo de recuperacion de consulta";
    echo "\r\n";
    print $totaltime;

    } catch(MongoConnectionException $e) {
    die("No es posible conectarnos a la base de datos:".$e-
>getMessage());
    }
    catch(MongoException $e) {
    die('No es posible -error'.$e->getMessage());
    }
?>
```

Script utilizado para eliminar registros de forma masiva en una colección, que satisfacen una condición dada para todos los campos no indexados.

```
<?php
try {

    //=====TIMER=====
    $mtime = microtime();
```

```

$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$starttime = $mtime;
//=====TIMER=====

$conexion = new Mongo();
// Seleccionar bd
$dbase = $conexion->selectDB('musica');

$result = $dbase->artistas-
>remove(array('cod_artista'=>6, 'cod_cancion'=>261,
'nombre'=>'Karen Carpenter', 'pais'=>'China'));

//=====TIMER=====
$mtime = microtime();
$mtime = explode(" ", $mtime);
$mtime = $mtime[1] + $mtime[0];
$endtime = $mtime;
$totaltime = ($endtime - $starttime);
//=====TIMER=====

print "Tiempo de recuperacion de consulta";
echo "\r\n";

} catch(MongoConnectionException $e) {
die("No es posible conectarnos a la base de datos:".$e-
>getMessage());
}
catch(MongoException $e) {
die('No es posible -error'.$e->getMessage());
}
?>

```

ANEXO 2 – RESULTADOS DE LAS PRUEBAS DE RENDIMIENTO

MySQL - Inserción

	Número de registros insertados		
	1000	10000	100000
1ra Iteración	0,58	23,51	320,89
2da Iteración	0,64	21,04	319,93
3ra Iteración	1,39	29,56	318,05
4ta Iteración	0,70	21,20	313,05
5ta Iteración	0,81	25,13	341,22
Tiempo Promedio (segundos)	0,82	24,09	322,63
Tiempo Total (segundos)	4,12	120,45	1613,14

```
mysql> select count(*) from musica.ms_artista;
+-----+
| count(*) |
+-----+
|      1000 |
+-----+
1 row in set (0.02 sec)

mysql> _
```

Figura 33 Consulta que muestra los mil registros ingresados

```
mysql> SELECT * FROM musica.ms_artista limit 100;
+-----+-----+-----+-----+-----+
| cod_artista | codigo | nombre          | pais    | url_biografia |
+-----+-----+-----+-----+-----+
| 123001      | 88995  | Cynthia Griffin | Chile   | biglobe.net    |
| 123002      | 5086   | Billy Nguyen    | China   | geocities.com  |
| 123003      | 87     | Sara Mendoza    | China   | 360.cn          |
| 123004      | 9      | Walter Murray   | Argentina | ycombinator.com |
| 123005      | 8      | Philip Hicks    | Russia  | usgs.gov        |
| 123006      | 1      | Tammy Willis    | Poland  | istockphoto.com |
| 123007      | 17     | Phillip Ryan    | China   | over-blog.com   |
+-----+-----+-----+-----+-----+
```

Figura 34 Consulta de ejemplo que muestra de los registros ingresados

```
mysql> select count(*) from musica.ms_artista;
+-----+
| count(*) |
+-----+
| 150000   |
+-----+
1 row in set (0.06 sec)

mysql> _
```

Figura 35 Consulta que muestra los ciento cincuenta mil registros ingresados, listos para la siguiente prueba.

MongoDB - Inserción

	Número de registros insertados		
	1000	10000	100000
1ra Iteración	1,23	2,33	13,53
2da Iteración	1,26	2,28	13,15
3ra Iteración	1,23	2,31	13,06
4ta Iteración	1,20	2,33	13,70
5ta Iteración	1,22	2,27	14,04
Tiempo Promedio (segundos)	1,23	2,30	13,50

Tiempo Total (segundos)	6,15	11,52	67,48
-------------------------	------	-------	-------

```
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Cesar>cd C:\php

C:\php>php imongo_v1.php
Tiempo de insercion
1.2324020862579
C:\php>_
```

Figura 36. Consulta que muestra el tiempo de inserción

```
2015-08-15T22:33:06.549-0500 Hotfix KB2731284 or later update is not
will zero-out data files
MongoDB shell version: 2.6.10
connecting to: test
> use musica
switched to db musica
> db.artistas.count()
1000
> _
```

Figura 37. Consulta que obtiene el número de documentos de la colección en MongoDB

```
2015-08-15T22:56:28.443-0500 Hotfix KB2731284 or later update is not install
will zero-out data files
MongoDB shell version: 2.6.10
connecting to: test
> use musica
switched to db musica
> db.artistas.find().pretty()
{
  "_id" : ObjectId("55d007bb09d0461c0900e29c"),
  "cod_cancion" : "364",
  "cod_artista" : "65",
  "nombre" : "\"Johnny Clark\"",
  "pais" : "\"Indonesia\"",
  "url_biografia" : "\"ask.com\\r\\n\""
}
{
  "_id" : ObjectId("55d007bb09d0461c0900e29d"),
  "cod_cancion" : "2",
  "cod_artista" : "9",
  "nombre" : "\"Lawrence Green\"",
  "pais" : "\"Portugal\"",
  "url_biografia" : "\"usda.gov\\r\\n\""
}
```

Figura 38. Consulta que muestra algunos registros ingresados en la colección *artistas*.

MySQL – Consulta A

	Número registros recuperados			
	1000	10000	100000	10000000
1ra Iteración	15,60	31,20	343,20	62229,56
2da Iteración	0,00	31,20	327,60	58287,98
3ra Iteración	15,60	46,80	343,20	61464,11
Tiempo Total (ms)	31,20	109,20	1014,00	181981,65
Tiempo Promedio (ms)	10,40	36,40	338,00	60660,55

MongoDB – Consulta A

	Número registros recuperados			
	1000	10000	100000	10000000
1ra Iteración	8,10	62,80	112,02	3619,21
2da Iteración	7,01	41,60	99,60	3556,81
3ra Iteración	8,60	42,20	97,40	3541,21
Tiempo Total (ms)	23,71	146,61	309,02	10717,22
Tiempo Promedio (ms)	7,90	48,87	103,01	3572,41

MySQL – Consulta B

	Número registros recuperados			
	1000	10000	100000	10000000
1ra Iteración	31,20	46,80	343,20	64240,91
2da Iteración	0,00	46,80	343,20	83803,35
3ra Iteración	15,60	48,00	358,80	90838,96
Tiempo Total (ms)	46,80	141,60	1045,20	238883,22
Tiempo Promedio (ms)	15,60	47,20	348,40	79627,74

MongoDB – Consulta B

	Número registros recuperados			
	1000	10000	100000	10000000
1ra Iteración	6,57	34,78	76,35	1263,60
2da Iteración	5,56	34,78	77,81	1045,20
3ra Iteración	6,57	39,03	82,24	1076,40
Tiempo Total (ms)	18,71	108,58	236,40	3385,21
Tiempo Promedio (ms)	6,24	36,19	78,80	1128,40

MySQL – Consulta C

	Número registros recuperados			
	1000	10000	100000	10000000
1ra Iteración	15,60	46,80	343,20	80059,34
2da Iteración	15,60	31,20	343,20	89450,56
3ra Iteración	15,60	46,80	343,20	104676,18
Tiempo Total (ms)	46,80	124,80	1029,60	274186,08
Tiempo Promedio (ms)	15,60	41,60	343,20	91395,36

MongoDB – Consulta C

	Número registros recuperados			
	1000	10000	100000	10000000
1ra Iteración	18,06	82,06	19,06	1107,60
2da Iteración	15,01	65,06	16,20	1060,80
3ra Iteración	14,06	64,06	186,02	1045,20
Tiempo Total (ms)	47,12	211,18	221,28	3213,61
Tiempo Promedio (ms)	15,71	70,39	73,76	1071,20

MySQL – Eliminación

	Operaciones de eliminación de filas	
	1ra Operación	2da Operación
1ra Iteración	46,80	795,60
2da Iteración	15,60	655,20
3ra Iteración	31,20	624,00
Tiempo Total (ms)	93,60	2074,80
Tiempo Promedio (ms)	31,20	691,60

MongoDB – Eliminación

	Operaciones de eliminación de filas	
	1ra Operación	2da Operación
1ra Iteración	4,35	154,40
2da Iteración	3,46	47,02
3ra Iteración	1,67	98,23
Tiempo Total (ms)	9,48	299,66
Tiempo Promedio (ms)	3,16	99,89

**GUÍA PARA LA ELABORACIÓN Y PRESENTACIÓN DE LA
DENUNCIA/PROTOCOLO DE TRABAJO DE TITULACIÓN**

1. DATOS GENERALES

1.1 Nombre del estudiante: Vele Zhingri Cesar Augusto

1.1.1 Código: 48711

1.1.2 Contactos: teléfono: 0983879508 augustocesar878@hotmail.com

1.2 Director sugerido: Marcos Orellana Cordero

1.2.1 Contactos: marore@uazuay.edu.ec

1.3 Tribunal designado:

1.4 Aprobación:

1.5 Línea de investigación de la carrera:

1.5.1 Línea: [1203] Informática de Computadores

1.5.2 Código UNESCO: [1203.17] Informática

1.5.2 Tipo de trabajo: Investigación Formativa

1.6 Área de estudio:

Bases de datos

1.7 Título propuesto:

Análisis de rendimiento entre la base de datos relacional: MySQL y una base de
datos no relacional: MongoDB

1.8. Subtítulo

1.9 Estado del proyecto

La propuesta del proyecto es una idea nueva, que mostrará las diferencias entre
un sistema de gestión de base de datos SQL: MYSQL y un sistema de gestión de
bases de datos orientado a documentos NoSQL: MongoDB; para ello se crearán
dos bases de datos, para finalmente determinar criterios de comparación que
incluyen diferencias conceptuales, requerimientos del sistema, integridad,
rendimiento, arquitectura, consultas y tiempos de inserción.



2. CONTENIDO

2.1 Motivación de la Investigación

La motivación que lleva a realizar esta investigación se debe al desarrollo de Internet y el Cloud Computing, que actualmente las organizaciones hacen uso y además operan con una gran cantidad de datos, por lo tanto las bases de datos tradicionales se enfrentan al desafío del alto rendimiento en la lectura y escritura en aplicaciones de alta concurrencia.

Las bases de datos NoSQL adoptan enfoques diferentes, con la principal ventaja de que manejan una gran cantidad de datos no estructurados, como archivos de procesamiento de texto (blogs), correo electrónico, multimedia y redes sociales de manera eficiente.

2.2 Problemática

Muchas organizaciones guardan grandes cantidades de datos (relacionadas con clientes, datos científicos, reportes financieros, etc.) para la gestión transaccional tradicional, análisis en tiempo real o con proyecciones a futuro. Tradicionalmente, la mayoría de estas organizaciones han estado almacenando datos estructurados en bases de datos relacionales. Sin embargo, un creciente número de desarrolladores y usuarios han empezado a recurrir a varios tipos de NoSQL, ya sean éstas orientadas a documentos, orientadas a columnas, de clave-valor o en grafo (dependiendo del tipo de aplicación); con algo en común: no son relacionales, manipulan grandes volúmenes de datos, poseen altas velocidades en el almacenamiento y recuperación de grandes cantidades de información y están ganando terreno en el mercado. Sin embargo, NoSQL no remplazará a las bases de datos relacionales; pero por su gran capacidad de procesamiento ¿serán una mejor opción para algunos tipos de proyectos?

2.3 Resumen

En este mundo contemporáneo donde la información fluye tan rápidamente, las organizaciones necesitan una base de datos sostenible, duradera, íntegra y segura que pueda crecer con el tiempo, permitir un desarrollo más rápido y una

implementación dinámica; estas organizaciones están recurriendo cada vez más a las bases de datos no relacionales, ahora llamados bases de datos NoSQL.

El presente proyecto tiene como objetivo obtener las características más relevantes y evaluar el rendimiento de las bases de datos Mongo DB y MySQL; para la etapa de implementación, constan la creación de dos bases de datos, una utilizando el estándar SQL y otra utilizando NoSQL, la evaluación de rendimiento se hace bajo criterios de comparación que incluyen definiciones conceptuales, requerimientos del sistema, integridad, arquitectura, consultas, rendimiento y tiempos de inserción.

En la etapa final de proyecto se analizan los resultados de las pruebas realizadas y un cuadro de ventajas y desventajas de esta solución al problema planteado.

2.4 Indagación exploratoria y base conceptual

Desde su creación, las bases de datos han sido un soporte para la organización de la información dentro de los diferentes tipos de entidades, debido a que "las bases de datos comenzaron a aparecer a finales de 1950 y comienzos de 1960, impulsadas por dos factores tecnológicos: el incremento de la fiabilidad de los procesadores de los ordenadores y la expansión de la capacidad de almacenamiento secundario en cintas y unidades de disco" [1]

En 1970 se propusieron por primera vez las bases de datos relacionales y las teorías subyacentes [3], entre las que se destaca el modelo de base de datos relacional, que implicó un cambio radical en el manejo de la información apoyándose en operaciones de conjuntos que combinan tablas de datos separadas (o relaciones) para producir un conjunto de respuestas. Las consultas se especifican utilizando el lenguaje de consulta estructurado SQL (por las siglas en inglés de Structured Query Language), soportado en el álgebra relacional, y que permite a un usuario expresar su consulta en forma declarativa, sin ningún tipo de instrucciones detalladas de programación.

Limitaciones de las bases de datos relacionales

La estructura de los datos en una base de datos relacional está predefinido por el diseño de las tablas y los nombres y tipos fijos de las columnas.

Escalabilidad. - Los usuarios pueden escalar una base de datos relacional mediante la ejecución en un ordenador más potente y caro; por lo tanto para escalar más allá



de cierto punto, ésta debe ser distribuida a través de múltiples servidores. Las Bases de datos relacionales no funcionan fácilmente en una forma distribuida porque es difícil unir sus tablas a través de un sistema distribuido. Además, las bases de datos relacionales no están diseñados para funcionar con la partición de datos, por lo que la distribución de su funcionalidad es una tarea, dijo Stephen O'Grady, analista de la firma de investigación de mercado RedMonk. [6]

Complejidad.- Con las bases de datos relacionales, los usuarios deben convertir todos los datos en tablas. Cuando los datos no encajan fácilmente en una tabla, la estructura de la base de datos puede ser compleja, difícil y lenta para trabajar.

SQL.- Uso de SQL es conveniente con los datos estructurados (tales como un conjunto de cifras de ventas, que encajan bien en tablas organizadas) y no el más adecuado con el caso con datos no estructurados, como los que se encuentran en los documentos de procesamiento de textos (blogs) e imágenes.[7]

El futuro de las bases de datos

Estas son algunas de las predicciones de lo que ocurrirá con los sistemas de bases de datos [2], en los próximos años.

- Muchos desarrolladores estarán dispuestos a abandonar a nivel mundial - transacciones ACID (Atomicity, Consistency, Isolation and Durability) para ganar escalabilidad, disponibilidad, y otras ventajas. La popularidad de los sistemas NoSQL basados en BASE (Basically Available, Soft State, Eventually Consistent) ya ha demostrado esto.
- Los almacenes de datos NoSQL no serán una "moda pasajera". La simplicidad, la flexibilidad y escalabilidad de estos sistemas viene a llenar un nicho de mercado, por ejemplo, para los sitios web con millones de lecturas/escrituras de los usuarios y con esquemas de datos relativamente simples. Incluso con una mejor escalabilidad relacional, los sistemas NoSQL mantienen muchas ventajas para algunas aplicaciones.
- Respecto a los DBMS relacionales, también tendrán una participación significativa en el mercado de almacenamiento de datos. Si las transacciones y consultas se limitan generalmente a nodos individuales, estos sistemas deberían poder escalar (R.Cattell). Cuando se desean transacciones importantes para SQL o ACID, estos sistemas serán la opción preferida.

- Uno o dos los sistemas probablemente se convertirán en los líderes en cada una de las categorías. El capital de riesgo y el apoyo de los principales actores serán probablemente un factor en esta consolidación. Por ejemplo, para el almacenamiento documental, MongoDB ha recibido importantes inversiones.

2.5 Objetivo General

- Evaluar el rendimiento de las bases de datos relacionales y no relacionales, aplicado a los gestores de bases de datos Mongo DB y MySQL, en tareas de consulta y manipulación de los datos.

2.6 Objetivos Específicos

- Documentar las características teóricas más relevantes de las bases relacionales y no relacionales.
- Instalar los gestores de bases de datos MySql y MongoDB.
- Analizar el rendimiento bajo diferentes criterios de evaluación.
- Presentar los resultados de las pruebas realizadas, mediante un cuadro comparativo.
- Analizar y comparar los planes de ejecución.

2.7 Metodología

La investigación se va a desarrollar con la recopilación de información procedente de referencias científicas en los siguientes portales web: Scielo, Scopus, Google Académico y la Biblioteca Científica de la Universidad del Azuay; para sintetizar las características y funcionalidades más relevantes de los dos gestores.

Se procederá a la instalación de la base de datos MySQL y de la base de datos MongoDB; posteriormente se definirán las pruebas de rendimiento que incluyen: el tiempo de acceso a lectura/escritura/borrado de una sola fila, el tiempo de acceso a consultas comunes que devuelven varias filas, el tiempo de acceso para consultas que incluyen varias tablas.

Los resultados esperados serán plasmados con la generación de un cuadro comparativo que mostrará los tiempos de las diferentes pruebas en los dos gestores y al final se sacarán las respectivas conclusiones y se mostrarán las recomendaciones respectivas.

2.8 Alcances y resultados esperados

El alcance de este trabajo es demostrar mediante un cuadro comparativo, cuál sistema de gestión de bases de datos es el mejor, bajo diferentes criterios de rendimiento; para que esta investigación sirva como referencia para implementar un sistema capaz de responder a las necesidades actuales de las diferentes organizaciones.

2.9 Supuestos y Riesgos

Supuestos	Riesgos	Posible Solución
La investigación se muestra sólida en su viabilidad, pues se cuenta con las referencias en la web y bibliográficas para apoyar su desarrollo conceptual y metodológico.	No existe la suficiente información para avalar la investigación.	Incorporación de referencias científicas en inglés.
En cada uno de los capítulos de la investigación, ésta cumple con los plazos con los plazos acordados	Se producen atrasos durante la investigación que no cumplen con los plazos acordados.	Posibilidad de solicitar un tiempo de gracia para terminar la investigación.
Se procede con normalidad al análisis de rendimiento en los dos gestores de bases de datos.	No existen las herramientas adecuadas para el análisis de rendimiento en uno de los gestores de bases de datos.	Creación de un script en el lenguaje PHP que obtenga los tiempos de respuesta en las diferentes pruebas.



3.1 Análisis y Planificación de Requisitos

3.2 Instalación MySQL

3.3 Instalación MongoDB

3.4 Pruebas de funcionalidad

4. ANALISIS DE RENDIMIENTO

4.1 Pruebas de rendimiento a MongoDB

4.2 Pruebas de rendimiento a MySQL

4.3 Reportes

4.4 Cuadro comparativo

5. CONCLUSIONES

2.13 Cronograma

Objetivo Específico	Actividad	Resultado esperado	Tiempo (semanas)
Documentar las características más relevantes de las bases relacionales y no relacionales	Investigación teórica basada en artículos científicos, en las diferentes bibliotecas virtuales.	Disponer de buenas referencias teóricas que avalen la propuesta para el proyecto.	4 semanas
Instalar los gestores de bases de datos MySql y MongoDB	Selección de las últimas versiones de los dos diferentes gestores de bases de datos y el S.O anfitrión.	Dos gestores de bases de datos instalados, para posteriormente proceder al análisis.	4 semanas
Analizar el rendimiento bajo diferentes criterios de evaluación.	Selección de los parámetros a evaluar a los dos gestores diferentes.	Obtener los tiempos de respuesta de los dos gestores de bases de datos de las operaciones de DML.	4 semanas

Presentar los resultados de las pruebas realizadas, mediante un cuadro comparativo	Creación de un cuadro comparativo.	Obtener un cuadro re resultados, basado en pruebas de funcionamiento.	4 semanas
------------------------------------------------------------------------------------	------------------------------------	-----------------------------------------------------------------------	-----------

2.14 Referencias

- [1] Berndt, D. <http://grandon.com>. 04 de 01 de 2012. 20 de 06 de 2014. <http://grandon.com/publications/SiteWit_NoSQL.pdf>.
- [2] Catell, Rick. «Scalable SQL and NoSQL Data Stores.» (2010): 16. Documento. 03 de Octubre de 2014. <<http://www.cattell.net/datastores/Datastores.pdf>>.
- [3] Codd, Edgar. «A relational model of data for large.» Codd, Edgar. *A relational model of data for large*. 1970. 377–387.
- [4] Drobi, S. <http://www.infoq.com/>. 3 de 08 de 2012. 20 de 06 de 2014. <<http://www.infoq.com/interviews/rich-Hickey-and-justin-sheehy-about-datastores,-nosql-and-cap>>.
- [5] R.Cattell, M. Stonebraker. «Ten Rules for ScalablePerfor mance in "Simple Operation" Datastores.» (2010). Documento. 03 de 10 de 2014. <<http://www.cattell.net/datastores/CACM-Paper.pdf>>.
- [6] «Software Developer's Journal.» *Software Developer's Journal* (2012): 14. Documento. 02 de 10 de 2014. <http://sdjournal.org/wp-content/uploads/downloads/2012/05/SDJTeaser_5_12.pdf>.
- [7] SONG Jia, CHEN Rui. «<http://www.paper.edu.cn>.» 11 de 05 de 2011. <http://www.paper.edu.cn>. Documento. 014 de 02 de 2014. <http://www.abaige.com/lwcs/so_article.asp?id=11032280598>.
- [8] Tiwari, Shashank. «Professional NoSQL.» Tiwari, Shashank. *Professional NoSQL*. John Wiley & Sons, 2011.

