FACULTAD REGIONAL¨ MARTIRES DE ARTEMISA ¨

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

TÍTULO: Propuesta de modelo dimensional y procesos de optimización en bases de datos NoSQL.

AUTOR: Ismael Gómez González

TUTOR: Ing. Ricardo Jorge Hera

CO-TUTOR: Ing. Dennys Lázaro Hernández Quintana

Artemisa, Junio 2012

"Año del 53 Aniversario del Triunfo de la Revolución Cubana"

"El país vivirá en el futuro de sus producciones intelectuales..."

"El futuro de nuestra patria tiene que ser necesariamente, un futuro de hombres de ciencia, de hombres de pensamiento".

"Ciencia y técnica significa reparar un país, crear un país" (1)

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas (UCI) y a la Facultad Regional de Artemisa los derechos patrimoniales del mismo, con carácter exclusivo.

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

_____ _____

Firma del Autor Firma del Tutor

AGRADECIMIENTOS Y DEDICATORIA

Agradezco a toda mi familia y de forma muy especial a mis padres por haberme apoyado siempre que los necesité y por ser mi fuente de inspiración y confianza. También les agradezco a mis colegas que han pasado conmigo estos 5 años de dura lucha apoyándonos en las buenas y en las malas. A mis tutores que me guiaron durante el desarrollo de este trabajo y me han enseñado a ser un mejor profesional y de forma general a todos mis profesores que durante estos 5 años me han enseñado mucho.

De manera especial quisiera dedicar este trabajo a dos personas muy importantes en mi vida que lamentablemente ya no están a mi lado: mi abuela Esther y mi tía Vivian; sus vidas no les alcanzaron para haber vistos el resultado de estos 5 años llenos de sacrificio y dedicación pero sé que donde quiera que estén van a sentirse muy felices y orgullosas de mi.

RESUMEN

Las Ciencias Informáticas, cumplen un papel esencial en la sociedad que vivimos hoy en día a la cual se le denomina "Sociedad de la Información". Rol protagónico dentro de estas ciencias lo ocupan las bases de datos, las cuales han ido desplazando a otras formas de almacenamiento de información debido a las grandes facilidades que brindan. Particularmente en Cuba, la Universidad de las Ciencias informáticas es una de las instituciones que se encuentran a la vanguardia en el desarrollo de aplicaciones utilizando estas herramientas. En dicho centro se hace uso de bases de datos relacionales para la gestión de la información las cuales resultan poco eficientes cuando el volumen de información almacenado es alto.

Con el desarrollo del presente trabajo de diploma se ofrece una base teórica bien fundamentada que ayudará a mejorar la rapidez de acceso a grandes volúmenes de datos en las aplicaciones para la gestión y almacenamiento de la información en la universidad y contribuirá al aumento considerable de la disponibilidad de dicha información. Dado un estudio realizado en el cual se detectaron las principales limitaciones de las herramientas de bases de datos que se utilizan actualmente en la universidad, el presente trabajo centra su investigación en la búsqueda de una herramienta de trabajo más óptima que permita mayor tolerancia a fallos y que disminuya los tiempos de respuesta de las peticiones, principalmente cuando el volumen de información es alto.

Las validaciones de la propuesta de solución realizadas fueron satisfactorias por lo que se puede afirmar que se logró realizar una propuesta que cumple con los objetivos definidos en el trabajo.

ÍNDICE

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El tránsito de la sociedad industrial a la sociedad de la información y el conocimiento constituye una de las principales transformaciones que ocurren en el mundo contemporáneo de hoy, en el cual se observa un poderoso auge de las Tecnologías de la Información y las Comunicaciones (TIC) lo que ha motivado un cambio en los paradigmas y estrategias establecidas durante muchos años. El desarrollo de la informática como parte de estas tecnologías, alcanza una posición relevante, al facilitar el control y la accesibilidad a los adelantos electrónicos.

La informática ha posibilitado el surgimiento de metodologías y productos capaces de ayudar al hombre a realizar sus tareas cotidianas, contribuyendo a una adecuada organización de su trabajo, debido a que permiten adquirir un mejor control de sus recursos e información. Traza nuevas metas a las ciencias modernas y de forma particular a las de la información, las cuales son un instrumento imprescindible del desarrollo social, político y económico de los países.

En el mundo informatizado de hoy, se observa un alto nivel de desarrollo científico-técnico, donde simples datos estadísticos constituyen una balanza de fortuna para las grandes compañías, ya que el análisis e interpretación que se les den a estos datos determina en un gran porciento el éxito de sus proyectos. Por lo que es necesario desarrollar búsquedas lo más exactas posibles dentro del enorme caudal de información que existe, para lo que deben ser empleados mejores métodos y herramientas para el almacenamiento y gestión de los datos.

Los datos fueron registrados por el hombre desde la antigüedad en soportes como la piedra, la madera y posteriormente el papel. Con el transcurso de los años y gracias al desarrollo del conocimiento, surgieron nuevas necesidades que conllevaron a que surgieran nuevas técnicas de almacenamiento de datos.

Las primeras muestras que evidenciaron el desarrollo de estas técnicas aparecen en los años sesenta del siglo XX, donde las aplicaciones informáticas eran escasas y las existentes acostumbraban a darse totalmente por lotes y estaban destinadas a resolver una tarea muy específica. Cada aplicación utilizaba ficheros tanto para actualizar como para consultar los datos, al inicio estos ficheros fueron almacenados en cintas magnéticas.

INTRODUCCIÓN

A medida que se fueron integrando las aplicaciones gracias al surgimiento de las líneas de comunicación, los terminales y los discos, fue necesario que se interrelacionaran sus ficheros y así eliminar la redundancia de los datos. El acceso en línea y la utilización eficiente de las interrelaciones exigían estructuras físicas que permitieran un acceso rápido a la información. La solución a estas exigencias llegó con el surgimiento de los Bancos de Datos, que posteriormente a inicios de los años setenta, tomaron el nombre de Bases de Datos.

El uso de las bases de datos rápidamente trajo consigo que surgiera la necesidad de contar con un sistema de administración para controlar tanto sus datos como los usuarios que se conectan a ella. La administración de bases de datos se realiza mediante los Sistemas Gestores de Bases de Datos (SGBD). Ellos permiten definir los datos a distintos niveles de abstracción y manipular dichos datos, garantizando la seguridad e integridad de los mismos.

Las bases de datos ocupan un lugar determinante en cualquier área del quehacer humano, comercial, y tecnológico; ya que se han convertido en un producto estratégico de primer orden, al constituir el fundamento de los sistemas de información y el soporte para la gestión de los datos y la toma de decisiones. Ellas facilitan: el almacenamiento de grandes cantidades de datos, la organización y reorganización de la información y su impresión y distribución de varias formas.

Actualmente se ha empezado a extender un tipo de aplicación de las Bases de Datos denominado Almacén de Datos, la cual es desplegada comúnmente en sistemas empresariales. Estos almacenes de datos se utilizan exclusivamente para hacer consultas de forma que se lleven a cabo estudios por parte de los analistas financieros y de mercado, con el objetivo de contribuir al éxito financiero en la empresa.

Para Cuba de forma particular el uso de las TIC abre un universo de posibilidades, permitiendo el desarrollo de las comunicaciones satelitales, la telefonía inalámbrica, internet, la televisión y la computación, a pesar del bloqueo económico comercial y financiero con el que Estados Unidos, le impide a la isla el acceso a su mercado lo que motiva a que el estado invierta más recursos al tener que recurrir a mercados muy distantes.

INTRODUCCIÓN

Cuba ha logrado dar pasos firmes en el proceso de inversión a favor de los avances de las nuevas tecnologías y se ha propuesto aumentar su desarrollo científico-técnico en pos de mejorar el nivel de vida de sus ciudadanos en el ámbito económico, social y cultural, reconociendo la influencia global de las técnicas informatizadas de trabajo usadas en la actualidad.

Con vistas a cumplir ese reto surge la Universidad de las Ciencias Informáticas (UCI), sede de altos estudios que inició su funcionamiento en el curso 2002– 2003. Es una universidad con un novedoso modelo de formación que combina la docencia, la producción y la investigación. La UCI es una universidad productiva, cuya misión es producir software y servicios informáticos a partir de la vinculación estudio – trabajo como modelo de formación.

Este centro tiene entre sus objetivos fundamentales: informatizar el país y desarrollar la industria del software para contribuir al desarrollo económico del mismo. Muchos de sus productos, están destinados a resolver las insuficiencias generadas por el análisis manual y el modo de almacenamiento obsoleto de los datos en organizaciones y empresas clientes. Perteneciente a esta institución, la Facultad Regional de Artemisa es una de las que lleva con mayor intensidad el trabajo con vistas a lograr estos objetivos, para lo cual sus internos aplican tecnologías de desarrollo modernas, tales como los anteriormente mencionados Almacenes de Datos, además utilizan como principal sistema para la gestión de datos el relacional, que es eficiente pero que no está exento de limitaciones y problemas con su empleo.

Estos sistemas presentan como principales dificultades, problemas para mapear datos relativamente sencillos y con un gran volumen de tamaño, lo cual puede generar los llamados cuellos de botella en la información, que consisten en la generación de demoras en el acceso a los datos e incluso que no esté a disposición del cliente cuando el servidor está saturado de pedidos. También suelen ser vulnerables a fallos, entiéndase que si deja de funcionar por algún motivo el servidor de la base de datos, se crearía un caos, ya que los clientes que en ese momento soliciten información de él, simplemente no la tendrían a su disposición.

Estas limitaciones vienen dadas a que debido a su arquitectura intrínseca las bases

de datos relacionales no se pueden distribuir de forma sencilla sobre varias máquinas; esto hace que toda la información esté centralizada en una sola y que para aumentar su rendimiento tengan que ser agregados nuevos dispositivos de hardware de almacenamiento y memoria. A este fenómeno de incremento de la capacidad de rendimiento del sistema es a lo que se le denomina escalabilidad.

Teniendo presente la situación descrita, el **problema de la investigación** radica en: ¿Cómo mejorar la velocidad de acceso y la tolerancia a fallos en las aplicaciones de gestión de datos que manejen grandes volúmenes de información?

Para lo cual se tiene como **objeto de estudio:** las bases de datos NoSQL y como **campo de acción** los modelos dimensionales y procesos para la optimización de rendimiento atendiendo a velocidad de acceso y la tolerancia a fallos en bases de datos NoSQL**.**

Con el objetivo de dar cumplimiento a lo planteado anteriormente, se tiene como **objetivo general** de esta investigación: Desarrollar una propuesta de modelo dimensional y procesos de optimización que garanticen un aumento en la velocidad de acceso y tolerancia a fallos en aplicaciones de gestión de datos que manejen grandes volúmenes de información haciendo uso de bases de datos NoSQL.

Para dar cumplimiento al objetivo anteriormente planteado se definen las siguientes **tareas** a desarrollar:

1. Establecimiento de los fundamentos teóricos para la realización de un Modelo Dimensional y procesos de optimización en Bases de datos NoSQL.

2. Caracterización de las principales soluciones NoSQL atendiendo a su modelo de datos y fines de uso.

3. Desarrollo de la propuesta de modelo dimensional y procesos de optimización en bases de datos NoSQL

4. Validación de la propuesta mediante criterios de especialistas y pruebas de rendimiento.

Se define la siguiente **idea a defender**: Elaborando las aplicaciones de consulta a grandes volúmenes de datos con sistemas de bases de datos NoSQL, se obtiene

un mejor rendimiento en la velocidad de acceso y la tolerancia a fallos de la información que en ellas se registra.

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

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

Histórico/Lógico: Este método permite establecer la necesaria correspondencia entre los elementos lógicos e históricos, con el fin de analizar la evolución histórica de los conceptos trabajados en la investigación. Se empleó debido a que se hizo una profunda investigación para analizar la trayectoria histórica de las bases de datos NoSQL, elaborando el estudio del estado del arte de este tema. Obteniendo una tendencia de cómo se comporta en la actualidad.

Analítico/Sintético: Este método permite analizar, comparar y confrontar las diferentes fuentes bibliográficas consultadas, en cuanto a tendencias actuales para el diseño e implementación de modelos dimensionales en base de datos NoSQL, lo que permite determinar las regularidades y así tomar decisiones. Además, analizar donde se emplearía la solución de la propuesta en cuestión**.**

Los **métodos empíricos** utilizados fueron**:**

Entrevista: Este método fue de gran utilidad para recopilar información cualitativa. Con dicha información se pudo conformar un marco teórico y la idea a defender del problema. Para el efecto se elaboraron cuestionarios no estructurados, es decir conformados por preguntas abiertas, las cuales fueron destinadas a profesionales especialistas en el área de investigación.

Estudio Documental: Aplicando éste método se revisó todo tipo de información al alcance que se relacione con el tema. Así mismo se chequeó la existencia de documentos o bibliografía iguales a la solución propuesta. Todo esto ayudó en la construcción teórica, la fundamentación del problema y motivó a la elaboración de

la solución propuesta.

Los **métodos matemáticos estadísticos** utilizados fueron**:**

Estadístico descriptivo: Para el análisis y procesamiento de la información, mediante de tablas y gráficos.

Análisis porcentual: Para procesar los datos de los instrumentos aplicados para la validación de la propuesta de solución búsqueda de información.

La **población** estudiada en esta investigación son los: Departamentos de desarrollo de proyectos especializados en la administración de datos en la Universidad de las Ciencias Informáticas. Teniendo como **muestra** el Departamento de Inteligencia del Negocio en la Facultad Regional “Mártires de Artemisa”.

Como variables de la investigación se identificaron las que se presentan a continuación:

Variables dependientes: Rendimiento en la velocidad de acceso a la información y la tolerancia a fallos.

Variable independiente: Modelo Dimensional y procesos de optimización en bases de datos NoSQL.

Estructura del Documento: El presente trabajo de diploma consta de 3 capítulos.

Capítulo 1: Fundamentación Teórica: En este capítulo se realiza un estudio del estado del arte de los sistemas dedicados a la gestión de la información, y las principales definiciones, metodologías y herramientas que se utilizarán.

Capítulo 2: Propuesta de la solución: En este capítulo se define el estilo arquitectónico y el Modelo de Diseño a utilizar, además se realiza la implementación de la propuesta de solución.

Capítulo 3: Validación de la solución: En este capítulo se valida la propuesta a través de criterios de especialistas y pruebas de rendimientos con el objetivo de demostrar las potencialidades que presenta en comparación con los sistemas relacionales.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 INTRODUCCIÓN

El fundamento teórico es una de las partes más importantes en el desarrollo de un trabajo de diploma pues provee un marco de referencias para la adecuada interpretación de los resultados que se recogen. En el presente capítulo se analizan un conjunto de enfoques teóricos e investigativos que han sido objeto de publicación en el área del conocimiento donde se ubica la investigación de este trabajo. Además se detallan con exactitud las características y conceptos fundamentales relacionados con el tema de investigación en cuestión y se argumenta todo lo relacionado con las tecnologías y herramientas definidas para el desarrollo de la aplicación demostrativa.

1.2 CONCEPTOS BÁSICOS

1.2.1 ESCALABILIDAD

La escalabilidad es la propiedad anhelada de una red, sistema, o proceso, que muestra su destreza para operar el incremento continuo de trabajo con fluidez, o muestra la preparación que tiene para crecer manteniendo su calidad en todos los servicios. Generalmente se puede definir la escalabilidad como la capacidad que tiene un sistema informático de modificar su configuración o su tamaño, para ajustarse a los cambios (Manuel L Jiménez, 2009) **(2)**

1.2.1.1 TIPOS DE ESCALABILIDAD

Existen dos tipos de escalabilidad:

1.2.1.1.1 Escalabilidad Vertical: El escalar verticalmente o escalar hacia arriba, significa el añadir más recursos a un solo nodo en particular dentro de un sistema, como añadir memoria o un disco duro más rápido a una computadora

1.2.1.1.2 Escalabilidad Horizontal: La escala horizontalmente significa agregar más nodos a un sistema.

1.2.2 BASES DE DATOS

Se define una base de datos como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular. **(3)** (Pérez Damián, 2008)

Se le llama base de datos a los bancos de información que contienen datos relativos a diversas temáticas y categorizados de distinta manera, pero que comparten entre sí algún tipo de vínculo o relación que busca ordenarlos y clasificarlos en conjunto.

Una base de datos puede ser de diversos tipos, desde un pequeño fichero casero para ordenar libros y revistas por clasificación alfabética hasta una compleja base que contenga datos de índole gubernamental en un Estado u organismo internacional. Recientemente, el término base de datos comenzó a utilizarse casi exclusivamente en referencia a bases construidas a partir de software informático, que permiten una más fácil y rápida organización de los datos. Las bases de datos informáticas pueden crearse a partir de software o incluso de forma online usando Internet. En cualquier caso, las funcionalidades disponibles son prácticamente ilimitadas. **(4)**(Victoria, 2008)

1.2.2.1 PROPIEDADES ACID

La mayoría de las bases de datos cumplen con las propiedades ACID (del inglés: atomicity, consistency, isolation, durability). Estas propiedades garantizan un buen comportamiento de la base de datos **(5)**.

1.2.2.1.1 Atomicidad (Atomicity)

Garantiza que las transacciones (sean una consulta, o grupos de sentencias SQL) no se puedan subdividir, es decir, se ejecutarán enteramente, o no se ejecutarán. Esto implica que en caso de fallo de hardware, fallo de base de datos, o fallos de la aplicación, se actualizarán todos lo datos o ninguno. Esto impide que la base de datos se corrompa o pierda el sincronismo lógico entre los datos.

1.2.2.1.2 Consistencia (Consistency)

Garantiza que la base de datos siempre estará en un estado consistente. De hecho, garantiza que cada transacción lleve a la base de datos de un estado consistente a otro estado consistente. En este caso, consistencia se refiere a la consistencia interna de relación entre tablas, y la consistencia en los datos almacenados. La propiedad de consistencia no permitiría guardar un entero en un campo float, o no permitiría borrar una fila que es referenciada por otra. Esta última forma de consistencia se le denomina integridad referencial.

➢ *1.2.2.1.3 Aislamiento (Isolation)*

Garantiza que los datos de una operación no puedan afectar a otra. Cuando se ejecutan dos transacciones sobre los mismos datos, estas son independientes, de esta manera no se generan errores en ninguna de las dos transacciones. Esto hace que los datos que manejan cada una de las transacciones no estén disponibles hasta que la transacción haya finalizado.

➢ *1.2.2.1.4 Durabilidad*

Garantiza que una vez que la transacción se haya completado, siempre se podrá recuperar independientemente de cualquier fallo de hardware o software. Una vez la base de datos manda la señal de que la transacción ha sido ejecutada correctamente, se puede tener la certeza de que esa transacción está aplicada correctamente a los datos y se va a poder recuperar.

1.2.3 BASES DE DATOS RELACIONALES

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

El modelo relacional resulta en la actualidad el más utilizado para modelar problemas reales y administrar datos dinámicamente. Su idea fundamental es el uso de relaciones. Estas relaciones podrían considerarse en forma lógica como un

conjunto de datos llamados "tuplas". Tratando cada relación como si fuese una tabla que está compuesta por registros, que representarían las tuplas, y campos.

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia. La información puede ser recuperada o almacenada mediante consultas que ofrecen una amplia flexibilidad y poder para administrar la información.

El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL (del ingles, 'Structured Query Language'), un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

1.2.4 BASES DE DATOS NO RELACIONALES

Las bases de datos no relacionales son comúnmente llamadas bases de datos NoSQL (término acuñado en 1998 por Eric Evans) ya que la gran mayoría de ellas comparte el hecho de no utilizar el lenguaje SQL para realizar las consultas a sus datos.

Se trata de una filosofía de sistemas de gestión de bases de datos que modifican por completo el modelo de las bases de datos relacionales. Esta nueva forma de trabajar responde a otra forma de organización de los datos que permiten una menor rigidez de los mismos y formas novedosas de trabajo.

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

1.2.4.1 VENTAJAS DE LAS BASES DE DATOS NoSQL

Hay algunas necesidades concretas donde las bases de datos SQL no cubren suficientemente las necesidades y por lo tanto se han desarrollado bases de datos no relacionales para poder cubrirlas. Muchas veces las bases de datos no relacionales relajan las restricciones impuestas por ACID o el esquema de datos necesario para poder cumplir el estándar SQL para poder de esta manera cumplir otros requisitos que las bases de datos no cumplen. Algunos ejemplos son:

1.2.4.1.1 Escalabilidad Horizontal:

Las bases de datos son extremadamente difíciles de escalar a más de una máquina. Aunque típicamente están diseñadas para aprovechar bien el incremento de recursos de una misma máquina, escalar la base de datos distribuyéndola en varias máquinas es un proceso complicado lleno de problemas, algunos de ellos prácticamente insoluble.

Si la base de datos entera cabe en una sola máquina y lo que se busca es escalar horizontalmente para aumentar el rendimiento de la misma, entonces la técnica utilizada es la replicación. Esto consiste en tener réplicas (llamadas esclavos) de la base de datos principal (llamada maestro) donde las escrituras se deben hacer todas en el equipo maestro y las lecturas se pueden distribuir entre los esclavos.

Esta solución para escalabilidad tiene ciertos problemas. No te permite escalar las escrituras, todas las escrituras se deben hacer en la misma máquina, y por tanto están limitadas a las escrituras que pueda hacer una máquina. Otro problema es que la replicación no es en tiempo real. Esto invalida la C de ACID, ya que la base de datos ya no está en un estado consistente, debido a que si se escribe un registro nuevo y se consulta en una de las réplicas, es posible que todavía no se haya replicado y por tanto ya se ha perdido el estado consistente que garantizan las bases de datos.

En caso de querer además escalar las escrituras, se tiene que utilizar una técnica llamada ¨sharding¨ o particionamiento horizontal **(6)**. Utilizando esta técnica, la base de datos se divide en varias partes donde cada porción se guarda en una máquina distinta.

De esta manera dependiendo de la tabla, o el fragmento de tabla al que se quiera escribir se escribe o se lee de una máquina u otra.

Esta solución incrementa exponencialmente la complejidad del sistema, y al mismo tiempo, hace que se pierda en la mayoría de las veces la A de ACID. Esto es debido a la problemática siguiente: las transacciones ya no son atómicas, debido a que es posible que una consulta tenga que escribir en varias particiones, haciendo imposible la garantía de que la transacción sea atómica y que todas la particiones estén en exactamente el mismo estado.

Debido a la complejidad de escalar horizontalmente, y la capacidad finita de la escalabilidad vertical, no es difícil ver por qué se han desarrollado proyectos que intentan crear una base de datos horizontalmente escalable. Ya que igualmente hay que renunciar a algunas de las restricciones ACID para escalar bases de datos relacionales, se entiende que crear un nuevo sistema de almacenamiento de datos pensado desde un principio para escalar horizontalmente pueda sacrificar desde el inicio algunas de las propiedades ACID.

Este es el caso de las soluciones NoSQL. Diseñadas para escalabilidad horizontal masiva, renuncian a algunas de las ventajas de bases de datos tradicionales. Sin embargo, estas ventajas igualmente se tendrían que renunciar si se intenta escalar la misma base de datos relacional a los niveles requeridos, por lo que no se considera gran pérdida.

Al ser diseñadas para este propósito específico, sabiendo desde un principio que sacrificarían las ventajas de una solución ACID no tienen la desventaja que acarrean los sistemas diseñados para cumplirlo, y que luego intentan escalarse más allá de lo que fueron diseñados para hacer.

➢ *1.2.4.1.2 Sin Esquema de datos:*

La mayoría de las aplicaciones evoluciona con el tiempo. Las distintas necesidades de negocio hacen que se guarden datos distintos, o que haga falta guardar datos adicionales. Los cambios en los esquemas de bases de datos relacionales son procesos extremadamente complicados. En una base de datos tradicional, añadir una columna requiere añadir ese campo a todas las filas existentes, que fácilmente pueden ser cientos de millones de filas. Este proceso puede dejar inoperativa a la base de datos durante un tiempo considerable, y en algunas bases de datos hasta necesita hacerse cuando la base de datos no brinde servicios.

Hay algunos negocios que pueden permitirse estar inoperativos durante una hora en ciertos horarios, pero hay otros que funcionan las 24 horas del día. Algunos ejemplos son los grandes portales web como Facebook, Twitter, o Google, que no se pueden permitir ningún tiempo de caída.

Las bases de datos NoSQL son Schema-less, lo que significa que no tienen un

esquema de datos pre-definido. Cada registro de la tabla puede tener campos distintos, la base de datos no limita ni controla cuantos campos ni de qué tipo se pueden guardar en cada registro.

Por ejemplo en una base de datos no relacional se guardan registros de tipo "persona". Estos registros tienen los campos "nombre" y "apellido". Después de varios millones de registros, se puede añadir el campo "teléfono" a cualquier registro, sin necesidad de variar el esquema de la base de datos, ya que este no tiene esquemas.

➢ *1.2.4.1.3 Latencia:*

Los sistemas de almacenamiento NoSQL Clave-Valor (o Key/Value) son sistemas sencillos que dados una clave, retoman el valor asociado a la clave. Son extremadamente rápidos y mantienen latencias muy bajas ya que están especializados en devolver los datos de la forma más rápida posible y todos los algoritmos internos están optimizados para la búsqueda de claves, típicamente con complejidad O(1). De esta forma tardan lo mismo para devolver una clave cuando hay 100 claves disponibles que cuando hay millones de claves disponibles.

1.2.4.2 TIPOS DE BASES DE DATOS NoSQL

Aunque sean muy difíciles de clasificar debido a las diferencias entre soluciones, existen algunas clasificaciones generales que se pueden hacer a las bases de datos no relacionales. Es posible que productos específicos tengan más de una clasificación ya que toman distintos conceptos e ideas de varias fuentes, pero en general, la siguiente tabla muestra los distintos tipos de bases de datos NoSQL que existen **(7)**:

Documentale	En Grafos	Clave/Valor	Orientada a Objeto	Multivalor	Tabular
CouchDB	Neo4j	Cassandra	ZoneObject Database	Extensible Storage Engine	Hbase
MongoDB	DEX	BigTable	db4o	Open QM	BigTable

RavenDz	AllegroGraph	Redis	GemStore S	Hipertable
BaseX	OrientDB	Dynamo	Objectivity /DB	
eXist	InfiniteGraph	Riak		
SimpleDB	SonesGraphDB	Proyect Voldemort		
IBM Lotus Dominio	HyperGraphDB			
Terrastore	InfoGrid			

Tabla 1 Tipos de bases de datos NoSQL

➢ *1.2.4.2.1 Documentales:*

Los almacenes de documentos guardan la información como un listado de documentos des-estructurados. Al acceder a un documento, se puede acceder a un número no especificado de campos con sus respectivos valores. Son muy rápidos para recuperar toda la información asociada al documento junto, y tienen un esquema de datos muy flexible. Sin embargo, suelen ser lentos para hacer consultas donde se buscan todos los documentos con un determinado campo, ya que estos no suelen tener índices.

➢ *1.2.4.2.2 En Grafos:*

Representan la información como nodos de un grafo y sus relaciones con las aristas del mismo, de manera que se pueda usar teoría de grafos para recorrer la base de datos ya que esta puede describir atributos de los nodos (entidades) y las aristas (relaciones).

➢ *1.2.4.2.3 Clave/Valor:*

Los almacenamientos clave-valor son uno de los tipos más simples de bases de datos no relacionales. Los caches de memoria típicamente utilizan el esquema clave-valor asocian una clave única (key) al valor que se quiere guardar (value). Este tipo de base de datos suele ser extremadamente rápido y optimizado para una

gran cantidad de accesos.

1.2.4.2.4 Multivalor:

En el caso particular de Extensible Storage Engine se utiliza para almacenar información en una secuencia lógica. La información puede ser recuperada de forma secuencial o accediendo a índices previamente definidos. Los cambios realizados en este tipo de base de datos se realizan en la transacción y con esto poder lograr operaciones seguras.

1.2.4.2.5 Orientada a Objeto:

La información se representa mediante objetos como los presentes en la programación orientada a objetos. Cuando se integra las características de una base de datos con las de un lenguaje de programación orientado a objetos, el resultado es un sistema gestor de base de datos (SGBD) orientada a objetos.

1.2.4.2.6 Tabular:

En el caso particular de BigTable, son tablas multidimensionales cuyas celdas están en su mayoría sin utilizar. Además, estas celdas disponen de versiones temporales de sus valores, con lo que se puede hacer un seguimiento de los valores que han tomado históricamente. Para poder manejar la información, las tablas se dividen por columnas, y son almacenadas como 'tabletas' de unos 100-200 Mbytes cada una. Cada máquina almacena 100 tabletas.

1.2.5 MODELO DIMENSIONAL

Básicamente el **Modelo Dimensional** es el nombre que se le da a una técnica utilizada especialmente en Almacenes de Datos. Este modelo difiere bastante del modelo Entidad-Relación que normalmente conocemos.

El **Modelo Dimensional** busca presentar la información de una manera estándar, sencilla y sobre todo intuitiva para los usuarios, además de que permite accesos a la información mucho más rápida por parte de los manejadores de bases de datos. **(8)**

1.2.5.1 COMPOSICIÓN DE UN MODELO DIMENSIONAL

Los Modelos Dimensionales se componen de:

1.2.5.1.1 Hechos: Los hechos representan tablas que describen el comportamiento del negocio. Son descritos usando dimensiones y medidas, por lo que representan la relación muchos a muchos que existe entre las dimensiones.

1.2.5.1.2 Medidas: Valores numéricos que describen el hecho que se está analizando.

1.2.5.1.3 Dimensiones: Son categorías que describen el contexto en el cual se analizan las medidas. Son las áreas temáticas o sujetos del negocio. Proveen un método general para organizar la información corporativa. Se definen como un grupo de uno o más atributos. Las dimensiones no comparten atributos. Las dimensiones definen los niveles de análisis o jerarquías.

1.2.5.1.4 Atributos: Los atributos son una agrupación de elementos dentro de una dimensión. Representan categorías o clases de elementos que tienen el mismo nivel lógico dentro de una dimensión.

La finalidad de los atributos es ver la información de cada dimensión a diferentes niveles de detalle y agrupar los datos para ser analizados.

1.2.5.1.5 Elementos: Son las instancias o valores de los atributos.

1.2.5.2 ARQUITECTURAS BÁSICAS

Los Modelos Dimensionales presentan dos arquitecturas de acuerdo con la normalización de sus dimensiones:

1.2.5.2.1 Estrella: El esquema estrella como bien indica su nombre se presenta en forma de estrella, con puntos radiales desde el centro. El centro de la estrella consiste de una o más tablas de hechos, y las puntas de la estrella son las tablas dimensionales. Este modelo entonces, resulta ser asimétrico, pues hay una tabla dominante en el centro con varias conexiones a las otras tablas. Las tablas dimensionales tienen sólo la conexión a la tabla de hecho y ninguna más.

1.2.5.2.2 Copo de nieve: Un esquema en copo de nieve es una estructura más

compleja que el esquema en estrella. Se da cuando las dimensiones se implementan con más de una tabla de datos. Aunque puede reducir espacio por la mínima redundancia de datos, tiene la contrapartida de peores rendimientos al tener que crear más tablas de dimensiones y más relaciones entre las tablas lo que tiene un impacto directo sobre el rendimiento.

1.3 ESTADO DEL ARTE

El movimiento NoSQL, surge a partir de las necesidades existente de proveer a los desarrolladores, herramientas de trabajo con finalidades muy específicas, debido a que las bases de datos relacionales se han convertido en la solución principal para cualquier necesidad de base de datos; un producto que intenta cubrir todas las necesidades de todos los clientes y que acaba sin poder cubrir las necesidades muy específicas de algunos. Las bases de datos relacionales ofrecen bajo rendimiento ante ciertas aplicaciones intensivas de datos:

- Indexación de un gran número de documentos
- Servir páginas en Sitios de mucho tráfico

Además están optimizadas para pequeñas pero frecuentes transacciones de lectura/escritura o largas transacciones con pocos accesos de escritura, sin embargo NoSQL puede dar servicio a grandes cargas de lectura/escritura. Debido a estas características el movimiento NoSQL está causando mucho impacto en los medios y la industria, y muchos responsables de sistemas están comenzando a interesarse y a comunicar su interés a los responsables de la toma de decisiones por este tipo de alternativas NoSQL, que se presentan como un concepto distinto al que se manejaba hasta ahora en las bases de datos relacionales.

En la actualidad existen muchas compañías de gran prestigio que han desarrollado aplicaciones basadas en este tipo de tecnología. Tanto los desarrolladores de la red social Twitter como los de Digg, sitio web principalmente sobre noticias de ciencia y tecnología, decidieron migrar de MySQL a una arquitectura de NoSQL basada en el proyecto de Cassandra, los desarrolladores de Digg alegan como motivo de este cambio lo siguiente:

“Nuestra principal motivación para alejarnos de MySQL es la creciente dificultad de

construir una aplicación de alto rendimiento con escrituras intensivas en un conjunto de datos que crece rápidamente, sin un final a la vista. A medida que nuestro sistema crece, es importante abarcar múltiples centros de datos para redundancia y rendimiento de la red, para agregar capacidad o reemplazar sin tiempos de caídas nodos que hayan fallado. Planeamos continuar usando hardware común y continuar asumiendo que fallará regularmente. Todo esto es crecientemente difícil con MySQL".

Cassandra fue desarrollado por Facebook una de las más difundidas redes sociales a nivel mundial, que en el año 2006 tenía 161 exaBytes (EB) de información almacenada en sus bases de datos, esto equivale a 322 millones de discos duros (HD) de 500 gigabytes (GB) y en el 2010 se concentraron 988 EB de información almacenada, que representa 1980 millones de HD de 500 GB, lo que indica que la información almacenada creció 6 veces en solo 4 años. Estos simples datos simbolizan el potencial de acceso a la información con que cuenta esta herramienta.

En correspondencia con el mencionado auge de estas tecnologías NoSQL, otro de los denominados "monstruos" de la web como Google, implementa BigTable y afirma que uno de sus clústeres más grande gestiona 6 petaBytes (PB) de datos sobre miles de servidores, esto equivale a 6.291.456 GB. HiperTable se emplea en un motor de búsqueda de la empresa de tecnología de búsqueda de acontecimientos Zvents para escribir 1.000.000.000 de celdas por días. Otros como la compañía estadounidense de comercio electrónico Amazon utiliza el sistema NoSQL Dynamo para gestionar sus datos y gestores documentales como CouchDB son usados en sistemas operativos para dispositivos móviles como Androide para el intercambio de música.

En Cuba, el concepto NoSQL no ha sido del todo estudiado por las principales empresas desarrolladoras de aplicaciones informáticas. No obstante, la Universidad de las Ciencias Informáticas (UCI), es una de las principales instituciones en el país que han logrado iniciar sus estudios con este tipo de tecnología, haciendo posible el despliegue de software como Sunshine, el cual es un sistema de gestión documental colaborativo que utiliza en su versión desplegada actualmente

MongoDB como sistema gestor de base de datos y que prevé para su nueva versión usar CouchDB.

1.4 ANÁLISIS DE LAS SOLUCIONES NoSQL

Las bases de datos NoSQL ofrecen una alternativa para las bases de datos relacionales, no un reemplazo. Cada una tiene su lugar, y simplemente entregan más alternativas de las cuales podemos elegir una. En los sistemas NoSQL solo podemos tener dos de las tres garantías del Teorema de CAP (la C, la A o la P), y por lo tanto es preciso elegir la más importante. Si lo que más importa es la coherencia, entonces se debería de optar por una base de datos relacional.

Se ha procedido a hacer un análisis de las distintas soluciones de bases de datos NoSQL de código abierto. Aunque los precursores del movimiento NoSQL son Google BigTable y Amazon Dynamo, ambas son de código cerrado y no están disponibles al público.

Estas soluciones propietarias no serán analizadas debido a su precio y que actualmente las soluciones de código abierto están muy avanzadas tecnológicamente. Solo se estudiarán en profundidad las herramientas NoSQL de código abierto con más influencia a nivel mundial, debido a la gran variedad y el gran número de ellas que existe en la actualidad.

1.4.1 APACHE CASSANDRA

Cassandra es una mezcla de almacenamiento clave-valor y bases de datos columnar creada por Facebook y hecha de código abierto en 2008. Está escrito en java por lo que funciona sobre cualquier sistema operativo, y utiliza Thrift (otro proyecto de Facebook) para serializar y comunicar los datos con programas externos. Esto permite usar Cassandra desde prácticamente cualquier lenguaje de programación. Está diseñada para gestionar cantidades muy grandes de datos distribuidos en una gran cantidad de máquinas comunes.**(9)**

1.4.1.1 MODELO DE DATOS

Cassandra ha sido descrita varias veces por el director del equipo que la desarrolló como una mezcla de BigTable de Google y Dynamo de Amazon, en parte porque

utiliza un modelo de datos columnar parecido al de BigTable, mientras que la infraestructura que permite añadir y restar nodos de forma automática y transparente es parecida a la de Dynamo. Uno de los creadores de Dynamo fue contratado por Facebook para desarrollar Cassandra.

Cassandra provee un servicio de clave-valor estructurado, ya que cada clave puede ir asociado a valores múltiples (llamados columnas), que a su vez van asociados en familias de columnas. Aunque las familias de columnas son fijas se pueden añadir y restar columnas a las claves en cualquier momento. Este modelo híbrido es como una mezcla del sistema columnar y el sistema basado por filas y es extremadamente poderoso, aunque a veces puede ser difícil de usar y entender.

Este sistema de claves es muy novedoso, y es una gran parte de la ventaja de trabajar con Cassandra, ya que da la rapidez de un sistema clave-valor, pero con la información profunda que puede proveer un sistema columnar.

1.4.1.2 CARACTERÍSTICAS

- ***1.4.1.2.1 Distribuido y Descentralizado:***

Cassandra es un sistema distribuido, lo cual significa que está en la capacidad de ejecutarse sobre múltiples máquinas mientras se presenta a los usuarios como un todo. De hecho, no es provechoso ejecutar Cassandra en una única máquina, aunque si puedes hacerlo. Por supuesto, aprovecharás todos sus beneficios si lo corres en múltiples servidores.

El hecho de que Cassandra sea descentralizado quiere decir que no tiene un punto de fallo, todos los nodos en un clúster (grupo de máquinas) funcionan del mismo modo. A lo cual se le denomina servidor simétrico, porque todas las máquinas hacen lo mismo, por definición no existe un administrador que coordine tareas entre los nodos.

La descentralización tiene dos ventajas clave: Es más fácil de usar que un Administrador/Esclavo, puede ser más fácil de operar y mantener un almacén de datos descentralizado que un Administrador/Esclavo ya que todos los nodos son iguales, y no se requieren conocimientos adicionales para escalar; configurar 50 máquinas no es diferente a configurar una sola. Un administrador puede ser un

simple punto de fallo. Como en Cassandra todos los nodos son idénticos, la inutilización de uno no interrumpe su funcionamiento. En pocas palabras, como Cassandra es distribuido y descentralizado, no hay un punto de fallo lo que genera una alta disponibilidad.

➢ *1.4.1.2.2 Escalabilidad Flexible:*

Uno de los puntos fuertes de Cassandra es su escalabilidad. Mientras muchos otros proyectos necesitan código adicional para poder crecer en capacidad, Cassandra ha sido diseñado para ser totalmente descentralizado, y crecer elásticamente añadiendo nodos adicionales.

Añadir un nodo a un clúster Cassandra es muy fácil, ya que se autoconfigura en la configuración de su nodo más próximo e inmediatamente se pone a copiar información de los demás nodos. Es igual de sencillo retirar un nodo de un clúster Cassandra. Simplemente apagar o desconectarlo, y los demás nodos se repartirán la carga de las consultas.

La facilidad de Cassandra de crecer y reducir el tamaño del clúster elásticamente, es uno de los puntos más fuertes de esta base de datos NoSQL. Los reportes iniciales de Twitter al integrar Cassandra para gestionar su base de datos de Tweets indican que saturaban las conexiones Ethernet entre los nodos de Cassandra y la propia aplicación de Twitter primero, antes de saturar la capacidad de Cassandra de responder a consultas, siendo esto un incremento de órdenes de magnitud de velocidad de transferencia a lo que previamente se conseguía con su anterior solución, que era un clúster de máquinas MySQL divididas por réplicas.

➢ *1.4.1.2.3 Alta Disponibilidad y Tolerancia a Fallos:*

En términos generales, la disponibilidad de un sistema está ligada a su habilidad de cumplir peticiones. Pero los computadores pueden experimentar muchas formas de fallo, desde componentes hardware descompuestos hasta problemas de conexión y comunicación. Existen computadores sofisticados y costosos que pueden tratar con este tipo de circunstancias. Para que un sistema sea altamente disponible, debe incluir múltiples computadores conectados, y el software que se ejecuta sobre ellos debe tener la capacidad de correr sobre un clúster e identificar posibles nodos

defectuosos.

Cassandra es altamente disponible, tú puedes reemplazar nodos defectuosos en un clúster sin tener que detener el sistema, y puedes también replicar la información a otros almacenes de datos para mejorar el desempeño local.

➢ *1.4.1.2.4 Consistencia Personalizable:*

Significa que tú decides el nivel de consistencia que necesitas en balance al nivel de disponibilidad. Existen tres modelos de consistencia: estricta, en la cual un lector recibe el valor más recientemente escrito, consistencia casual, la cual es una variación de la estricta, y la eventual, en la cual los datos deben pasar por las máquinas de un sistema distribuido, esto lleva tiempo, por eso decimos que "eventualmente" los datos son consistentes.

➢ *1.4.1.2.5 Orientado a Columnas:*

Cassandra no es relacional, representa las estructuras de datos como tablas de hash multidimensionales, en donde cada registro puede tener una o más columnas, aunque no todos los registros de un mismo tipo deben el mismo número de columnas, como si sucede en el modelo relacional. Cada registro contiene una llave única la cual permite el acceso a los datos.

Cassandra almacena los datos en tablas de hash multidimensionales, lo que significa que no es necesario tener por adelantado la representación de los datos que se va a usar y cuantos campos se necesitarán para los registros, esto es útil cuando la estructura de datos está sujeta a cambios frecuentes. Cassandra permite agregar campos para los registros aun cuando esté en servicio.

➢ *1.4.1.2.6 Libre Esquema:*

Cassandra requiere que definas un contenedor llamado "espacio clave" que contiene "familias de columnas". El espacio clave es esencialmente un nombre para mantener columnas familiares y propiedades de configuración. Las familias de columnas o columnas comunes son nombres para información asociada. Las tablas de datos son dinámicas, así que puedes agregar información usando las columnas que quieras, no necesitas definir las columnas que requieres por adelantado.

- ***1.4.1.2.7 Alto Desempeño:***

Cassandra fue diseñado específicamente para aprovechar al máximo las máquinas con procesadores multinúcleo, y para ejecutarse sobre muchas docenas de estas máquinas alojadas en múltiples almacenes de datos. Cassandra ha demostrado desempeñarse bien en la carga pesada de datos. Posee un alto rendimiento en escrituras por segundo. Cuantos más servidores sean agregados, se sacará mayor provecho de las propiedades de Cassandra sin sacrificar desempeño.

1.4.1.3 USUARIOS

Cassandra es utilizado por múltiples empresas importantes. Es un proyecto Apache de máximo nivel. Las empresas que se conocen que usan Cassandra actualmente son Facebook con más de 200 nodos, Digg, Twitter, Rackspace, IBM y Reddit.

1.4.2 HBASE

Hbase es una base de datos distribuida de código abierto no relacional. Está inspirada en el BigTable de Google, y es parte del proyecto Hadoop del Apache Software Foundation. Está escrito en Java, con lo que es portable a cualquier sistema operativo. Hbase se centra en el análisis de grandes cantidades de datos, y no en procesar datos rápidamente en tiempo real. **(10)**

Hbase fue desarrollada por la empresa Powerset que fue posteriormente adquirida por Microsoft.

Al hablar de Hbase hay que además hacer referencia a Hadoop y HDFS. HDFS es el sistema de archivos de Hbase, y se utiliza para guardar los archivos de forma distribuida y redundante y al mismo tiempo tener un acceso rápido a ellos.

Hadoop es un framework para escribir aplicaciones de tratamiento de datos distribuidas, y utiliza a Hbase como su base de datos. Hadoop permite hacer trabajos Map Reduce sobre un número ilimitado de nodos Hbase, y por tanto hacer un tratamiento masivo de datos. Los 3 (Hadoop, Hbase y HDFS) son proyectos de código abierto del Apache Foundation.

1.4.2.1 MODELO DE DATOS

Hbase guarda los datos organizados en columnas al estilo de BigTable de Google.

Soporta una variedad muy grande de protocolos para comunicarse con las distintas aplicaciones. Tiene una interfaz Thrift a sus datos, haciéndolos accesibles desde cualquier lenguaje de programación.

1.4.2.2 ESCALABILIDAD

Hbase es escalable a un número muy grande de nodos, hay clústeres en Yahoo! de Hbase con más de 250 máquinas. Sin embargo, Hbase requiere de software adicional para poder gestionar las máquinas que se van añadiendo al clúster, ya que estas no se auto configuran ni se auto gestionan.

1.4.2.3 CONSISTENCIA DE DATOS

Hbase no tiene un punto central de fallo, los datos están distribuidos en todos los nodos de la aplicación. Tiene parámetros configurables para regular el nivel de consistencia de los datos.

1.4.2.4 USUARIOS

Yahoo es el usuario más importante de Hbase, con casi 250 nodos en su centro de datos.

1.4.3 MONGODB

MongoDB es una base de datos de documentos, diseñada para reemplazar las bases de datos SQL tradicionales de uso general. Es de código abierto y escrita en C++. Actualmente está disponible para Windows, Linux y otros sistemas operativos.

Mongo fue diseñado por la empresa 10Gen de Nueva York, que actualmente ofrecen consultoría y soporte técnico para MongoDB. Aunque el desarrollo de MongoDB empezó en octubre del 2007, Mongo se hizo público en febrero de 2009.

La intención de Mongo es proporcionar más funcionalidad que la típicamente proporcionada por las bases de datos NoSQL de tipo clave-valor, permitiendo indexación de ciertos valores dentro del documento y manteniendo un rendimiento elevado y facilidad para distribuir la carga en varios servidores. **(11)**

1.4.3.1 MODELO DE DATOS

El modelo de datos sigue el de una base de datos de documentos. Cada

documento tiene su propio id, y un número indeterminado y flexible de campos con información. Sin embargo a diferencia de otros muchos sistemas de bases de datos NoSQL documentales, MongoDB permite indexar ciertos campos de datos dentro del documento. Esto redunda una velocidad mucho mayor para las búsquedas sobre esos campos, aunque afecta ligeramente la velocidad de inserción de documentos, ya que hay que escribir al índice además de escribir el documento.

MongoDB guarda cada documento en un formato que llaman BSON, que es un acrónimo de Binary JSON. Cada documento es un archivo JSON (un formato de representación des-estructurado parecido a XML) que a su vez es codificado en binario.

1.4.3.2 ESCALABILIDAD

Mongo escala usando sharding. Al romper la base de datos en varios equipos distintos esto permite crecer de forma prácticamente lineal. A diferencia de una base de datos relacional, los documentos no tienen relación entre sí, con lo que es extremadamente fácil de particionar los datos a través de una clave de particionamiento.

Sin embargo, el particionamiento en MongoDB no es tan sencillo de implementar como el crecimiento elástico de nodos de Cassandra, y requiere cierto trabajo al configurar los clientes de acceso al clúster Mongo.

1.4.3.3 CONSISTENCIA DE DATOS

MongoDB soporta replicaciones de datos. Al combinarlo con soporte de particionamiento de datos, esto permite una redundancia de datos. Sin embargo, le faltan sistemas de recuperación automática del maestro en caso de fallo. Una vez más, se puede conseguir redundancia total de datos en MongoDB, sin embargo, no es sencillo y requiere bastante trabajo de administración de sistemas para que todo el proceso sea relativamente automático.

MongoDB no tiene durabilidad de datos en un solo servidor. Esto quiere decir que MongoDB puede perder datos si no se configura con una réplica. Esto se debe a que MongoDB no cumple con las propiedades ACID, especialmente con la D de

durable.

En recientes publicaciones en su foro Dwight Merriman CEO de 10Gen explica que Mongo "*se usa normalmente on-line para lecturas y escrituras en tiempo real. No es para data warehousing o para el almacenamiento de datos cargados en lotes de forma offline*". **(12)**

1.4.3.4 USUARIOS

MongoDB es usado por SourceForge, Justin.tv, foursquare y Bit.ly

1.5 POSICIÓN ADOPTADA

Las bases de datos relacionales, como se ha expresado en epígrafes anteriores representan buenas herramientas a tomar en consideración con vistas a la elaboración de cualquier tipo de aplicación que requiera de una central de datos para la gestión de información, debido a las propiedades ACID que presentan, pero presenta limitaciones, especialmente a la hora de cargar grandes volúmenes de información, por lo que si se usa como alternativa para la realización de un almacén de datos, tendría el inconveniente de que las respuestas a peticiones clientes serían lentas. Luego de esta afirmación el presente trabajo investigativo enfatiza su estudio en la búsqueda de un sistema que sea capaz de cubrir estas limitaciones presentadas por las bases de datos relacionales.

Dado el estudio de epígrafes anteriores se considera como una alternativa de solución debido a su gran auge a nivel mundial y a las características y ventajas que presenta al SGBD Cassandra perteneciente al grupo de las bases de datos NoSQL.

1.6 HERRAMIENTAS DE DESARROLLO

1.6.1 HERRAMIENTAS COLABORATIVAS

1.6.1.1 Rapid SVN

RapidSVN es un cliente gráfico que permite manipular los repositorios de Subversión. Es una de las alternativas más conocidas para el sistema GNU/Linux para poder interactuar con el repositorio, es muy intuitivo y fácil de utilizar. Una de

las características más importantes de este sistema de control de versiones es que puede acceder al repositorio a través de diferentes redes lo que permite que varias personas puedan hacer uso del mismo conjuntamente, posibilitando que el trabajo se realice con mayor rapidez. A cierto nivel, la posibilidad de que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer porque la calidad del mismo vaya a verse afectada, si se ha hecho un cambio incorrecto a los datos, simplemente basta con deshacer ese cambio. Entre las ventajas de RapidSVN se encuentran:

- Sigue la historia de los archivos y directorios a través de copias y renombrados.
- Las modificaciones (incluyendo cambios a varios archivos) son atómicas.
- La creación de ramas y etiquetas es una operación más eficiente.
- Puede ser servido mediante Apache.
- Maneja eficientemente archivos binarios.
- Permite selectivamente el bloqueo de archivos. Se usa en archivos binarios que, al no poder fusionarse fácilmente, conviene que no sean editados por más de una persona a la vez.

Cuando se usa integrado a Apache permite utilizar todas las opciones que este servidor web provee en el momento de autentificar archivos **(13)**.

1.6.2 HERRANIENTAS DE DESARROLLO

1.6.2.1 POWER ARCHITECT

El Power Architect es una herramienta de modelado de datos creados por los diseñadores de almacenamiento de datos. Está basada en Java, pensada específicamente para el almacenamiento y repositorio de datos. Esta herramienta cuenta con una interfaz agradable al usuario y muy intuitiva.

1.6.2.1.1 Funcionalidades

- Permite a los usuarios realizar ingeniería inversa de bases de datos existentes.
- Realizar datos de perfiles en base de datos fuente y auto-generar metadatos ETL.
- Se conecta a múltiples fuentes de datos al mismo tiempo, incluso si son de los proveedores de bases de datos de la competencia.
- Compara datos de modelos y estructuras de bases de datos e identifica las discrepancias.
- Guarda el histórico de origen de cada columna.
- Guarda la estructura de fuente de datos instantáneamente en el diseño, para que pueda trabajar de forma remota.
- Todos los datos del proyecto se almacenan en un formato XML de fácil análisis.
- Genera Informes visuales.

1.6.2.2 CASSANDRA CLI

La Interfaz de Líneas de Comando de Cassandra (CLI) se utiliza para hacer un lenguaje básico de definición de datos (DDL) y la manipulación de datos (DML) dentro de un clúster de Cassandra. Para iniciar y conectarse a la CLI para una instancia en particular de Cassandra, se debe de iniciar una secuencia de comandos en un terminal donde se especifique el host y el puerto de conexión **(13)**. Se conectará a el nombre del clúster especificado en el archivo cassandra.yaml. El siguiente es un ejemplo de conexión a esta aplicación:

```
$ cassandra-cli -host localhost -port 9160
```

1.6.2.4 APACHE THRIFT

Thrift es un conjunto de herramientas y librerías software creadas por Facebook para acelerar el desarrollo e implementación de servicios **(15)**. El principal objetivo es permitir comunicaciones eficientes y fiables a través de lenguajes de programación mediante la abstracción de porciones de cada lenguaje en una

librería común, específicamente Apache Thrift permite a los desarrolladores definir los tipos de datos e interfaces de servicios en un archivo único en lenguaje neutral y generar todo el código necesario para construir clientes y servidores. Actualmente Thrift soporta los siguientes lenguajes:

C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk y OCaml.

1.6.2.5 DATANAMIC DATA GENERATOR

Datanamic ofrece un conjunto completo de herramientas para apoyar el diseño, desarrollo y administración de bases de datos PostgreSQL. Es una herramienta de PostgreSQL para generar datos, diseñar bases de datos y para sincronizar los datos y los esquemas.

Con la herramienta Datanamic de PostgreSQL usted puede modelar bases de datos, generar datos de prueba, instalar scripts, sincronizar los datos y sincronizar las estructuras de datos **(16)**.

1.6.2.6 CASSANDRA-UNIT

Cassandra-Unit es un sistema para desarrollar pruebas con aplicaciones que usen la base de datos Cassandra. Permite insertar datos en el sistema a partir de ficheros XML, JSON o YAML. Está construido sobre el framework Hector y posee Licencia LGPL **(17)**.

1.7 CONCLUSIONES PARCIALES

Durante este capítulo se realiza el esclarecimiento de algunos términos y conceptos relacionados a la problemática a resolver. Se adopta la posición acerca de la selección de la herramienta de trabajo más conveniente a utilizar y se efectúa un estudio de las principales herramientas y tecnologías que podrían ser utilizadas en busca de la solución a la problemática planteada en el presente trabajo investigativo.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1 INTRODUCCIÓN

En el presente capítulo se contribuirá significativamente al cumplimiento del objetivo general del trabajo de diploma. Se brindarán los argumentos pertinentes que avalan cada una de las propuestas ofrecidas. También se describen los procesos principales que tiene lugar para la optimización de la Base de Datos.

2.2 MODELO DIMENSIONAL

Dado el estudio realizado en el Capítulo 1 del presente documento, durante el cual se enfatizó de forma especial y detallada en la fundamentación teórica acerca de cuales son las causas, características y ventajas que contribuyen a elegir a la base de datos Cassandra, como herramienta de trabajo principal para dar respuesta a la problemática en cuestión, se considera que están definidas las bases teóricas fundamentales que avalen el comienzo del diseño de una propuesta de Modelo Dimensional que ayude a comprender cómo se podría comportar el sistema Cassandra ante un posible proyecto de Almacenes de Datos donde se generan grandes volúmenes de información.

El Modelo Dimensional propuesto, es el resultado de un caso de estudio confeccionado por el autor de la presente investigación, por lo que no presenta ninguna similitud con los modelos dimensionales que actualmente se están generando en los proyectos de Almacenes de Datos de la universidad. Dicho modelo está encaminado a la gestión de información de ventas de productos en todas las cadenas de tiendas de recaudo de divisas a nivel nacional.

Para dar solución al Modelo se hace necesario destacar aspectos importantes acerca del funcionamiento de la estructura relacional, cómo está estructurado su modelo de datos y que similitudes y diferencias podrían encontrarse en el sistema Cassandra.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.2.1 SOLUCIÓN EN BASES DE DATOS RELACIONALES

2.2.1.1 ANALISIS DE LA SOLUCIÓN

En primer lugar se debe decir que en las Bases de Datos Relacionales como Postgres o MySQL cada base de datos contiene tablas, cada una de estas tablas tiene su propio nombre y columnas, y cada columna su nombre. Para agregar datos a las tablas se asigna un valor para cada columna definida, a las que no se les asigna valores, se les agrega NULL. Las filas de las tablas corresponden a los registros y cada registro posee una llave primaria la cual permite el acceso a la información. Se pueden actualizar todos los registros o algunos de ellos si se desea. Este modelo se rige bajo el estándar SQL.

2.2.1.2 DISEÑO DE LA SOLUCIÓN

Dicho en síntesis el funcionamiento del sistema relacional, se procede a mostrar la representación del Modelo Dimensional propuesto atendiendo a su estructura.

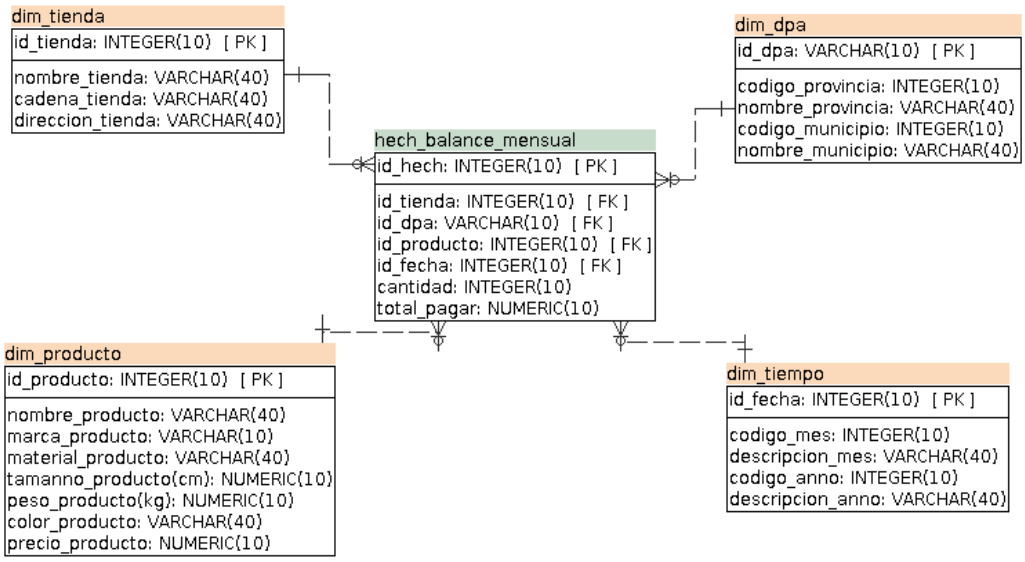


Figura 1 Solución del Modelo Dimensional en bases de datos relacionales

2.2.2 SOLUCIÓN EN BASES DE DATOS CASSANDRA

2.2.2.1 ANALISIS DE LA SOLUCIÓN

A diferencia de las bases de datos relacionales, la base de datos Cassandra tiene carencia de una estructura fija de tablas. Es decir, se componen de un conjunto de entidades básicas, pero estas carecen de una definición fija de atributos. Es como

si cada fila de cada tabla, en un modelo relacional, pudiera tener el número de columnas que quisiera. Vista esta afirmación desde el punto de que Cassandra no necesita almacenar un valor para cada columna al momento de crear un nuevo registro porque quizás no se conozcan los valores cada columna. Por ejemplo, algunas personas tienen un segundo número de teléfono y otras no. En vez de colocar NULL para los valores que no conocemos, lo cual gasta espacio, simplemente no se tiene en cuenta la columna para ese registro.

Como se puede apreciar en la Figura 2, la estructura del modelo de datos de Cassandra es la siguiente:

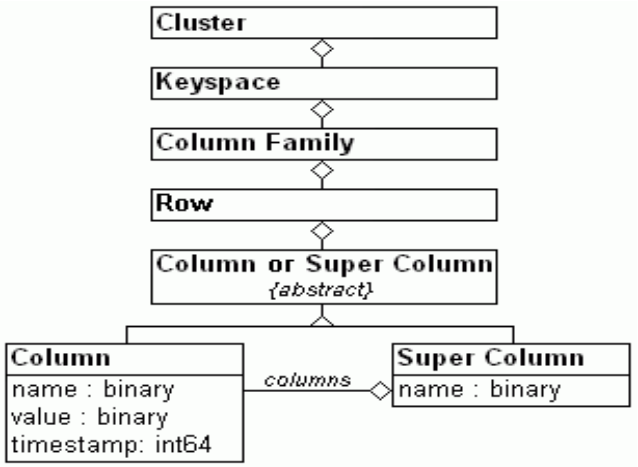


Figura 2 Estructura del Modelo de Datos de Cassandra

✓ ***Las Columnas (Columns***) son consideradas los elementos de menor granularidad al que se puede hacer referencia por un nombre. Lo interesante es que no es un valor escalar, sino una estructura compuesta por tres atributos: *name: binary*, *value: binary* y *timestamp: int64.*

No hay restricciones con respecto a lo que pueden contener los atributos. Todos los valores los deben suministrar los clientes de la base de datos, incluido el *timestamp*. El modelo apenas proporciona al gestor información acerca del dominio de la aplicación. No se define hasta el último detalle lo que puede insertarse o no en la base de datos. De hecho, en la práctica puede insertarse cualquier tupla de la forma (nombre, valor, fecha) que se quiera. Cassandra, a lo que se dedica en realidad, es a gestionar de forma muy eficiente colecciones distribuidas verdaderamente enormes de este tipo de tuplas, del orden de billones, de una

manera altamente escalable.

✓ ***Las Super Columnas (Super Columns)*** son una agregación de *columnas* que puede referenciarse por un nombre. Se implementa como una estructura que se compone de dos atributos: *name: binary* y *columns: lista*<*Column*>. En la práctica se le considera igual que una *columna*, sólo que en vez de almacenar un valor almacena una lista de columnas. Es un recurso bastante útil, ya que permite tener una colección de valores anidados asociados a otro valor.

✓ ***Las familias de Columas (Column Families)*** son una agregación de filas que se referencian con un nombre.

Se insiste en que no hay estructura predefinida. Cassandra mantiene colecciones ordenadas de objetos a los que se accede por un nombre (clave), que en la terminología de Cassandra se llama row (key). El modelo en realidad es un hash de varias dimensiones, donde las columnas se sitúan en el nivel más bajo, las súper columnas son un hash de columnas, las rows son un hash de columnas o súper columnas, y las column families son un hash de rows. Teniendo la cadena de nombres claves se puede acceder a todos los elementos que la componen.

El comportamiento de ordenación de Cassandra se establece en función de cómo los nombres de las columnas de este tipo de base de datos serán comparados para su ordenación cuando los resultados son retornados al cliente. Las columnas son ordenadas mediante la sentencia "with comparator" definida en la Column Family que las contiene, y se pueden escoger los siguientes tipos de ordenación:

- AsciiType: Compara directamente los bytes, validando que la entrada pueda ser analizada como US-ASCII. US-ASCII es un mecanismo de codificación de caracteres basado en el orden léxico del alfabeto Inglés. En él se definen los 128 caracteres, 94 de los cuales se pueden imprimir.

- BytesType: Este es el valor predeterminado, y ordena comparando directamente los bytes, omitiendo el paso validación. BytesType es el valor por defecto por una razón: proporciona la correcta clasificación de la mayoría de los tipos de datos (UTF-8 e incluye ASCII).

- LongType: Ordena por tipos long numéricos de 8 bytes(64 bits).

- IntegerType: Ordena por valores enteros.

- UTF8Type: Una cadena de caracteres usa UTF8 como codificador de caracteres. Aunque se podría creer este tipo de ordenación como uno bueno para establecer por defecto, porque es confortable para programadores que usan lenguaje XML u otro mecanismo de intercambio de datos común que requiere codificación. En realidad, en Cassandra se debe utilizar UTF8Type sólo si usted quiere que sus datos estén validados.

✓ ***La fila (Row)*** es una agregación de Columns o Super Columns que se referencia con un nombre. No hay más, sólo un nombre contenido dentro de un String. Ese nombre es la "clave" (key) que identifica de forma unívoca a un registro.

En este punto es importante no confundir una row con una super column. La row es sólo una palabra "clave", no es una estructura, carece de atributos.

La mayoría de las personas que usan esta base de datos se sienten cómodas pensando en las rows como el equivalente a la clave primaria de cada registro en las tablas del modelo relacional.

✓ ***Los Espacios de Claves (Keyspace)*** son una agregación de column families que puede referenciarse por un nombre. No son una estructura con atributos, es tan sólo otro contenedor al que se accede por un String con su nombre.

La mayoría de la gente parece sentirse cómoda pensando en los keyspaces como el equivalente a los esquemas (conjunto de tablas) en el modelo relacional.

✓ ***El Clúster*** es el elemento de más alto nivel que puede referenciarse por un nombre. Es de naturaleza más física que los anteriores, más relacionado con el hardware, ya que agrupa los nodos (máquinas) sobre los que se ejecuta Cassandra. Puede contener uno o más keyspaces.

Esta es a muy grandes rasgos toda la estructura del modelo. Se espera que haya quedado claro que Cassandra almacena pares de datos compuestos por una clave y un valor (más un timestamp), que no permite hacer joins, y es más, que no permite ejecutar ningún tipo de sentencia SQL.

No obstante, utilizar este esquema no significa renunciar a las buenas prácticas

aprendidas a lo largo de años de uso de los tradicionales modelos relacionales. Por ejemplo, para las claves de filas nada nos impide seguir utilizando por comodidad algún tipo generación secuencial o identificador universal (UUID) generado automáticamente, como se hace en la actualidad para las claves primarias con las columnas AUTOINCREMENT.

2.2.2.2 DISEÑO DE LA PROPUESTA DE SOLUCIÓN

Realizado el análisis se considera la siguiente estructura, como la propuesta de solución que más se adecua al Modelo Dimensional clásico de las bases de datos relacionales dadas las características que presenta el sistema Cassandra:

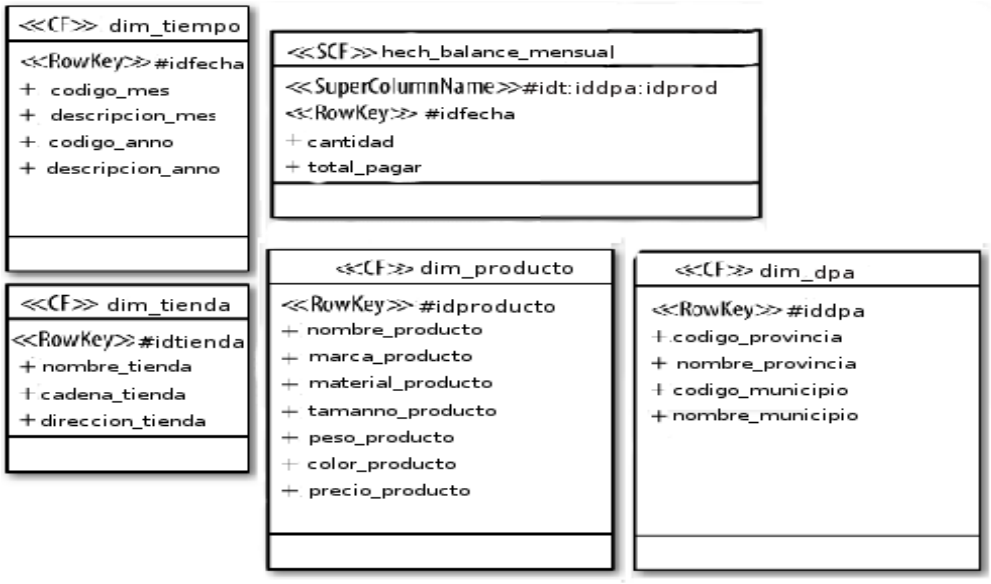


Figura 3 Solución del Modelo Dimensional en bases de datos Cassandra

Como se puede apreciar, el modelo tal y como se representa en una base de datos relacional, parte de una composición de 5 estructuras de datos. En este caso las dimensiones se crean mediante el elemento column family, que establece una serie de columnas que como se ha explicado, su diferencia con el modelo relacional radica en que su estructura no es fija, por lo que al momento de insertar un registro, se puede tener un registro con todos los atributos de esa column family y al siguiente poder insertar solo uno de ellos, o cuantos se desee. Las columnas establecidas en Cassandra pueden almacenar cualquier tipo de valor debido a que Cassandra no ofrece restricciones al momento de la inserción al no tener en cuenta si es una cadena de caracteres, un entero, un valor numérico u otro tipo de datos.

Cada familia de columnas, establecida como dimensión del Modelo Dimensional, cuenta con un identificador de fila (RowKey), que es usado de forma similar a las llaves primarias en los sistemas relacionales. Aunque vale volver a resaltar que los identificadores de filas son sólo palabras "claves", no son una estructura, carecen de atributos y le sucede lo mismo que con las columnas, no hay restricción de tipos de datos.

Si se desea garantizar la integridad de la base de datos restringiendo el tipo de datos de cada uno de estos elementos, se debe de establecer del lado del cliente con validaciones JavaScript u otro tipo de lenguaje.

Además de las dimensiones, se define una estructura de tipo Super Columna para los hechos, en el caso del modelo propuesto, sería hech_balance_mensual. El identificador de fila de este hecho, garantiza la indexación de cada venta por la fecha en que se realizó, ya que intenta simular una llave foránea con la dimensión de tiempo (dim_tiempo), buscando la existencia de su RowKey en algún identificador de fila de esta dimensión. Se dice que intenta simularla, porque en ningún momento se garantiza su integridad ya que no es algo establecido para el modelo de datos de Cassandra. Esta tarea debe de realizarse mediante un lenguaje de programación, externo a la Interfaz de Líneas de Comandos (CLI) de Cassandra, que establecería vínculos con el mismo mediante el conjunto de librerías Thrift desarrollado para aplicaciones con este tipo de base de datos.

Por otra parte debido a su estructura, cada Super Columna contiene una lista de columnas, a las cuales se puede acceder gracias al nombre de la súper columna (SuperColumnName) específico para cada uno de los registros. En el modelo en cuestión, el SuperColumnName de la súper columna hech_balance_mensual, está descrito del modo que sea una integración de cada una de las llaves de filas de las dimensiones: dim_tienda, dim_dpa, dim_producto, sucesivamente, garantizando de esta manera la relación con las mismas de la misma forma en que se puede realizar el vínculo con la dimensión dim_fecha, solo que el valor del SuperColumnName debería ser divido en las porciones de cadenas de caracteres que identifiquen a cada una de los identificadores de filas de las dimensiones. Los valores de ese listado de columnas dan a conocer la cantidad de productos

vendidos y el precio total de la venta.

2.2.2.3 IMPLEMENTACIÓN DE LA SOLUCIÓN

Durante el presente epígrafe se muestra la forma en que se implementa el Modelo Dimensional utilizando el cliente de Cassandra.

Creación de la dimensión: **dim_tiempo**

```
create column family dim_tiempo with comparator= UTF8Type and
column_metadata= [
{column_name: codigo_mes, validation_class: LongType},
{column_name: descripcion_mes, validation_class: UTF8Type},
{column_name: codigo_anno, validation_class: LongType},
{column_name: descripcion_anno, validation_class: UTF8Type}
];
```

Creación de la dimensión: **dim_dpa**

```
create column family dim_dpa with comparator= UTF8Type and
column_metadata= [
{column_name: codigo_provincia, validation_class: LongType},
{column_name: nombre_provincia, validation_class: UTF8Type,
index_type:KEYS},
{column_name: codigo_municipio, validation_class: LongType},
{column_name: nombre_municipio, validation_class: UTF8Type,
index_type: KEYS}
];
```

Creación de la dimensión: **dim_tienda**

```
create column family dim_tienda with comparator= UTF8Type and
column_metadata= [
{column_name: nombre_tienda, validation_class: UTF8Type,
index_type: KEYS},
{column_name: cadena_tienda, validation_class: UTF8Type},
{column_name: direccion_tienda, validation_class: UTF8Type}
];
```

Creación de la dimensión: **dim_producto**

```
create column family dim_producto with comparator= UTF8Type
and column_metadata=
[
{column_name: nombre_producto, validation_class: UTF8Type,
index_type: KEYS},
{column_name: marca_producto, validation_class: UTF8Type,
index_type: KEYS},
{column_name: material_producto, validation_class: UTF8Type},
{column_name: tamanno_producto, validation_class: UTF8Type},
{column_name: peso_producto, validation_class: UTF8Type},
{column_name: color_producto, validation_class: UTF8Type},
{column_name: precio_producto, validation_class: UTF8Type}
];
```

Creación del hecho: **hech_balance_mensual**

```
create column family hech_balance_mensual with column_type=
'Super' and comparator= UTF8Type and subcomparator= UTF8Type
and column_metadata=
[
{column_name: cantidad, validation_class: LongType},
{column_name: precio_total, validation_class: UTF8Type}
];
```

2.3 PROCESOS DE OPTIMIZACIÓN

2.3.1 VISTAS MATERIALIZADAS

Los sistemas de bases de datos NoSQL están diseñados para la escalabilidad. El lado negativo de esto es que como el nombre sugiere, no hay soporte para SQL. Esto puede creerse una seria limitación de estos sistemas debido a que se piensa ¿Cómo es posible seleccionar, unir y agrupar en estos sistemas?. En este epígrafe, se explica cómo todas estas operaciones pueden ser implementadas de forma natural y eficientemente en uno de los más famosos sistemas NoSQL que existen en el mundo, Cassandra.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Considerando el Modelo Dimensional propuesto donde existen como dimensiones 4 familias de columnas, se tendrán como ejemplo 2 de ellas para el desarrollo de estas vistas. Estas son las dimensiones dim_dpa y dim_tienda, como se muestra en la figura.

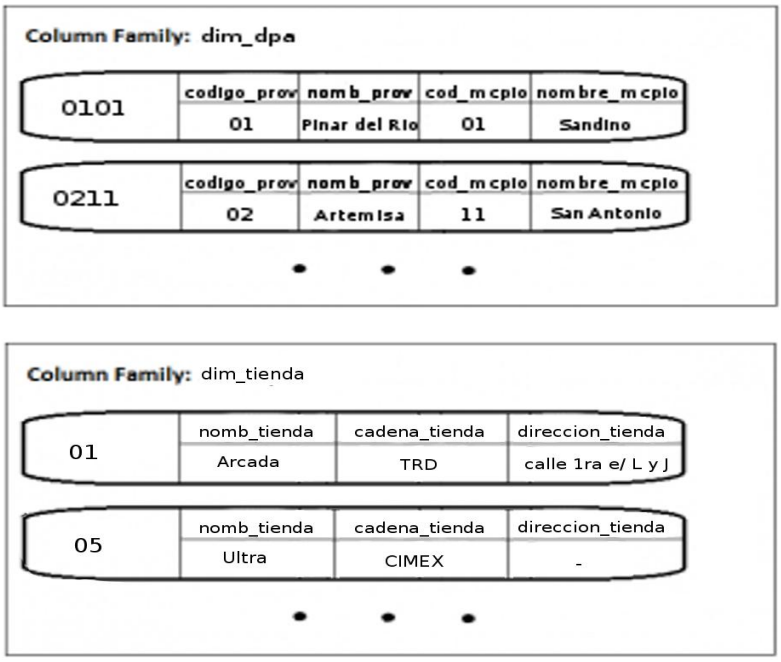


Figura 4 Ilustración de datos en las dimensiones dim_dpa y dim_tienda

Como se puede apreciar, en dim_dpa, se almacena cada provincia con su código, nombre, el código de sus municipios y el nombre de los mismos, tomándose como identificador único de sus filas el id_dpa que no es más que la combinación de los códigos de la provincia con el de los municipios. De igual forma, en dim_tienda, el identificador de fila es el id de la tienda, que da acceso a datos como el nombre de la tienda, la cadena comercial a la que pertenece y su dirección.

2.3.1.1 Seleccionar (SELECT)

Para que Cassandra pueda soportar algo parecido a esta sentencia SQL:

```
select * from dim_tienda where nombre_tienda = 'Arcada';
```

Se necesita agregar una column family más nombrada nombre_dimtienda, en el cual la llave resulta ser el nombre de la tienda y los nombres de las columnas son los IDs de las tiendas que tienen ese nombre (Ver Figura 5). Los valores de las columnas no son usados aquí, y pueden ser un arreglo de bytes vacios (denotados de esta manera " - "). Todo el tiempo cuando una tienda es insertada o eliminada

de dim_tienda, se necesita actualizar la column family nombre_dimtienda. Para ejecutar la consulta solo se necesita obtener todas las columnas para la llave 'Arcada' de la column family nombre_dimtienda.

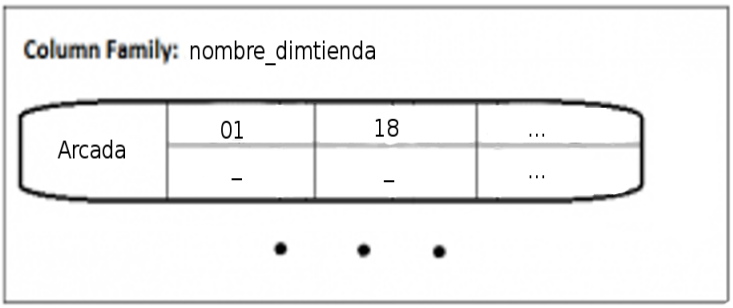


Figura 5 Column Family agregada a la solución

Se debe decir que nombre_dimtienda es esencialmente un índice que permite ejecutar la consulta eficientemente y es escalable ya que debe ser distribuido a todos los nodos de Cassandra. Se puede, para acelerar aún más la consulta de forma redundante, almacenar información sobre las tiendas en nombre_dimtienda.

En ese caso los IDs de la tienda son los nombres de la Super Column que contiene las columnas de dim_tienda (Ver Figura 6).

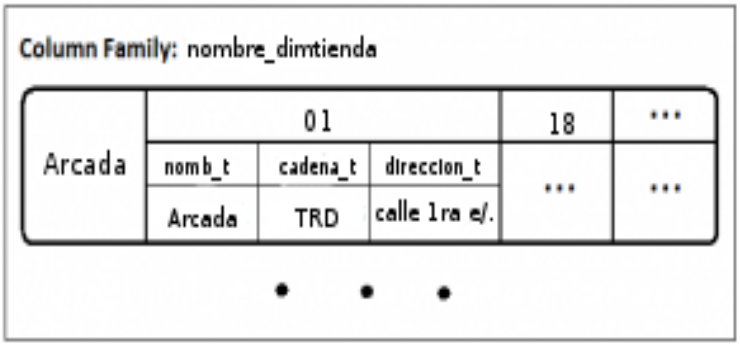


Figura 6 Solución segundaria utilizando estructuras de Súper Columnas

2.3.1.2 Unir (JOIN)

Si se desea que Cassandra soporte una consulta como la siguiente:

```
select * from dim_tienda t, dim_dpa d where t.id_dpa = d.id_dpa;
```

Como se debe conocer, los JOINs construyen registros que representan las

relaciones entre las entidades. Estas relaciones pueden ser fácilmente representadas a través de anidación. Para hacer eso en Cassandra, se debe agregar una column family que se denominará en este ejemplo dpa_tienda en el cual su llave es el ID de dpa y los nombres de las columnas son los IDs correspondientes a las tiendas.

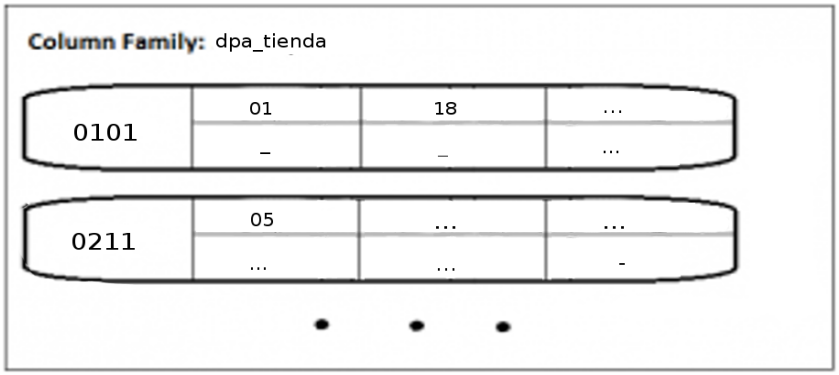


Figura 7 Solución que ofrece Cassandra en sustitución de los JOINS

2.3.1.3 Agrupar (GROUP BY)

```
select count(*) from dim_tienda group by cadena_tienda;
```

Desde el punto de vista del GROUP BY su implementación es similar a la de selección descrita anteriormente. Solo se necesita agregar una column family que se denominará a conveniencia en este ejemplo cadena_dimtienda, con las cadena como llave y los nombres de las columnas sería los IDs de las tiendas que pertenecerían a esa cadena. En este caso se contarán los nombres de las columnas para cada llave.

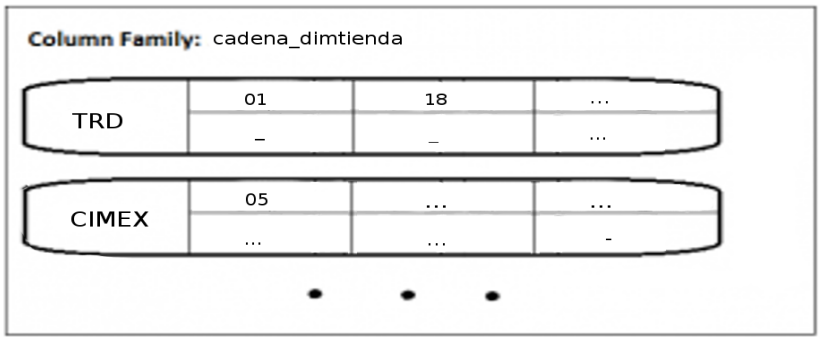


Figura 8 Solución que ofrece Cassandra para agrupamiento

2.3.2 MULTINODOS EN CASSANDRA

Como se ha descrito en el capítulo anterior, Cassandra es un sistema distribuido, lo que significa que puede ejecutarse sobre múltiples máquinas mientras se presenta a los usuarios como un todo. Aunque se puede realizar, no es provechoso ejecutar Cassandra en un solo nodo porque se estaría prescindiendo de una de las principales ventajas de estos sistemas con respecto a otros como los relacionales: su carácter descentralizado, que como bien se ha explicado permite que no exista un punto de fallo en el sistema.

El presente epígrafe establece un estudio detallado del comportamiento interno del sistema Cassandra, atendiendo a la comunicación entre sus nodos y al particionamiento de la información que en él se registra. Además ofrece una guía con las principales instrucciones a seguir para añadir nodos al clúster de Cassandra, con vistas a aumentar su rendimiento.

2.3.2.1 COMUNICACIÓN ENTRE NODOS

Cassandra usa un protocolo llamado Gossip para descubrir la localización y estado acerca de los otros nodos que integran el clúster del sistema. Gossip es un protocolo de comunicación peer-to-peer en el cual los nodos periódicamente intercambian información acerca de si mismos y acerca de otros nodos que conocen.

En Cassandra, el proceso Gossip corre cada segundo y permite intercambiar los mensajes de estado a un máximo de tres nodos del clúster. Cada nodo intercambia información acerca de su estado y también acerca de otros nodos que conocen, permitiendo el rápido estudio del comportamiento de los datos en los nodos que componen el clúster. Un mensaje Gossip tiene una versión asociada, de modo que durante un intercambio de datos, la información más antigua se sobrescribe con el estado más actual de un nodo en particular.

2.3.2.2 MIEMBROS DEL CLÚSTER Y NODOS SEMILLAS (Seeds)

Cuando un nodo inicia por primera vez, se debe de editar su fichero de configuración para determinar el nombre del clúster de Cassandra y a qué nodo llamado semilla es al que pertenece, para ponerse en contacto para obtener

información acerca de los otros nodos en el clúster. El fichero de configuración que tiene estos puntos de contacto es el llamado cassandra.yaml.

Un nodo semilla de Cassandra es un nodo que está en contacto con otros nodos cuando estos se ponen en marcha por primera vez y se unen al clúster. Un clúster o anillo puede tener varios nodos semillas. Cassandra utiliza el antes mencionado protocolo gossip para descubrir la ubicación y la información de estado acerca de los otros nodos que participan en un clúster de Cassandra. Cuando un nodo se inicia por primera vez, entra en contacto con un nodo semilla para arrancar el proceso de comunicación gossip.

Para prevenir las particiones en las comunicaciones del protocolo gossip, todos los nodos del clúster deben tener la misma lista de nodos semillas en su archivo de configuración.

La designación del nodo semilla no tiene otro propósito que el inicio del proceso de comunicación gossip para los nuevos nodos que se unen al clúster. Los nodos semillas no son un punto único de fallo, ni tampoco tienen ningún otro propósito especial en las operaciones del clúster más allá de la secuencia de arranque de los nodos.

Para saber hasta que rango de datos es responsable, un nodo debe conocer sus propios tokens (símbolos) y aquellos que pertenecen a los otros nodos del clúster. Cuando se inicializa un nuevo grupo de máquinas, se deben generar tokens para todo el clúster y asignar un token inicial para cada nodo antes de la puesta en marcha. Cada nodo hará público su token a los demás.

2.3.2.3 PARTICIONAMIENTO DE DATOS EN CASSANDRA

Cuando se inicia un clúster de Cassandra, se debe seleccionar cómo los datos serán divididos a través de los nodos en el clúster. Esto se hace mediante la elección de un partidor para el clúster.

Cassandra utiliza un algoritmo de Tabla de Hash Distribuido (DHT) para determinar cuando los datos se almacenan en un nodo con un token determinado. Cada servidor de Cassandra se configura asignando un valor simbólico en la sentencia initial_token del fichero de configuración. Esta asignación debe estar regida por la

siguiente fórmula matemática:

i*(2127)/ número_de_nodos con i comenzando desde 0 hasta (número_de_nodos - 1)**

Esta fórmula permite que los datos almacenados en el sistema se distribuyan de forma equitativa sobre todos los nodos que comprende el clúster.

El valor 2 a la potencia de 127 representa el espacio de nombres DHT de 128 bits utilizado por Cassandra. La siguiente tabla muestra la asignación correcta de tokens para un clúster de 4 máquinas.

Servidor	Token Inicial	Rango Responsable
0	0	2**127/4 * 3 - 0
1	2**127/4	0 - 2**127/4
2	2*(2**127/4)	2**127/4 - 2*(2**127/4)
3	3*(2**127/4)	2*(2**127/4) - 3*(2**127/4)

Tabla 2 Distribución de rango de responsabilidad de datos por nodos en el clúster

El total de datos administrados por el clúster es representado con un espacio circular o anillo. El anillo está dividido en intervalos iguales al número de nodos, con cada nodo siendo responsable de uno o más rangos del total de datos. Antes de que un nodo pueda unirse al anillo, debe tener asignado el token, el cual determina las posiciones del nodo en el anillo y el rango de datos del cual es responsable.

Los datos de las familias de columnas son particionados a través de nodos basándose en su identificador de fila. Para determinar el nodo donde estará la primera réplica, el anillo camina en el sentido de las agujas del reloj hasta que se localiza el nodo con un valor de token mayor que el del identificador de fila. Cada nodo es responsable de su región del anillo y la de su precursor.

Con los nodos ordenados en orden simbólico, el último nodo es considerado como el predecesor del primer nodo.

Por ejemplo, considere un clúster simple de 4 nodos donde todos los identificadores de filas administrados por el clúster son números en el rango de 0 a 100. A cada nodo se le asigna un token que representa un punto en ese rango. En

este ejemplo los valores de los tokens son 0, 25, 50 y 70. El nodo con el menor símbolo acepta claves de fila inferiores a su símbolo y superiores al nodo con mayor valor de token, tal como se puede observar en la siguiente ilustración:

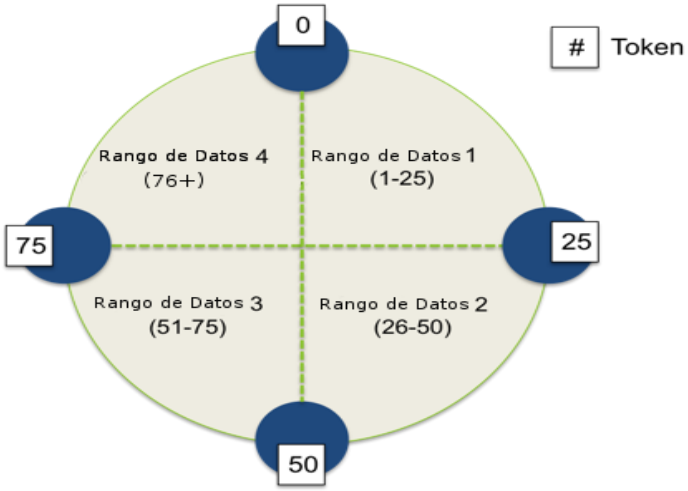


Figura 9 Ejemplo de comportamiento de datos en un clúster Cassandra

2.3.2.4 DESCRIPCIÓN DE LOS TIPOS DE PARTICIONADO

A diferencia de casi todas las opciones de configuración en Casandra, el particionador no puede ser cambiado sin recargar todos sus datos. Es importante elegir y configurar la herramienta de particionado correcta antes de inicializar el clúster.

Cassandra ofrece una serie de particionadores, pero el particionador aleatorio es la mejor opción para la mayoría de las implementaciones de Cassandra.

➢ *2.3.2.4.1 Particionamiento Aleatorio*

El particionamiento aleatorio (RandomPartitioner) es la estrategia de partición por defecto para un clúster de Cassandra, y en casi todos los casos es la elección correcta. Esta estrategia utiliza hashing consistente para determinar que nodos almacenará cada fila en particular. Los hashing consistentes aseguran que cuando los nodos son añadidos al clúster se afecte el conjunto mínimo de datos.

Para distribuir los datos uniformemente a través de un número de nodos, un algoritmo de hash crea un valor hash MD5 del identificador de la fila. A cada nodo en el clúster se le asigna un token que representa un valor hash dentro de ese rango. Un nodo es propietario de las filas con menor valor hash que su número de

token.

➢ *2.3.2.4.2 Particionamiento Ordenado*

El uso de un particionador ordenado asegura que las claves de fila se almacenen de forma ordenada. A menos que sea absolutamente necesario en su aplicación, se recomienda la elección de la herramienta de particionado aleatorio sobre un particionador ordenado.

Usar particionamiento ordenado no es recomendable por las siguientes razones:

1. **Escrituras secuenciales pueden causar los puntos calientes:** Si su aplicación tiende a escribir o actualizar un bloque secuencial de filas a la vez, entonces la escritura no se distribuirá en todo el clúster, todos ellos irán a un solo nodo. Esto es con frecuencia un problema para las aplicaciones que tratan con datos de tiempo.
2. **Una mayor carga administrativa para equilibrar la carga del clúster:** Un particionador ordenado requiere administradores para calcular manualmente los rangos de fichas en base a sus estimaciones de las distribuciones de claves filas. En la práctica, esto requiere el movimiento activo de los tokens de los nodos para acomodar la distribución real de datos una vez que se han cargado.
3. **Balanceo de carga desigual para las familias de columnas múltiples:** Si su aplicación tiene múltiples familias de columnas, lo más probable es que las familias de las columnas tienen diferentes claves de fila y diferentes distribuciones de datos. Un particionador ordenado está equilibrado para una familia de columnas por lo que puede causar puntos calientes y la distribución desigual de las otras familias de columnas en el mismo clúster.

2.3.2.5 REPLICACIÓN DE DATOS EN CASSANDRA

La replicación es el proceso de almacenamiento de copias sobre múltiples nodos para asegurar en todo momento la disponibilidad y tolerancia a fallos de los datos. Cuando se crea un Keyspace en Cassandra se debe decidir la estrategia de colocación de réplicas, el cual determina el número de réplicas y cómo estas son distribuidas a través de los nodos en el clúster. La estrategia de replicación

depende de un clúster debidamente configurado para ayudar a determinar la ubicación física de los nodos y su proximidad entre sí.

El número total de réplicas de todo el clúster hace referencia al factor de replicación. Un factor de replicación de 1 significa que sólo hay una copia de cada fila. Un factor de replicación de 2 significa dos copias de cada fila. Todas las réplicas son igualmente importantes, no hay réplica primaria o principal en términos de cómo leer y escribir el manejo de las peticiones.

Como una regla general, el factor de replicación no debería de exceder el número de nodos en el clúster. Sin embargo, es posible incrementar el factor de replicación y luego agregar el número deseado de nodos después.

➢ *2.3.2.5.1 Estrategias de colocación de réplicas en Cassandra*

1. **SimpleStrategy:** Coloca las réplicas de los datos a todos los nodos que están juntos en el mismo anillo.
2. **NetworkTopologyStrategy:** Configura el número de réplicas por Centro de Datos. Es exclusivamente una estrategia de colocación de réplicas para clústers con múltiples Centros de Datos. A continuación se muestra un ejemplo de cómo puede ser configurada esta estrategia:

   ```
   update keyspace tesis with strategy_options=[{DC1:3, DC2:3}];
   ```

   En este ejemplo se muestra cómo se configura el clúster para que dados 2 Centros de Datos, se coloquen 3 réplicas de los datos almacenados en el Espacio de Claves `tesis` por cada uno.

El presente trabajo investigativo realizó mayor énfasis en la generación de un clúster con estrategia de colocación de réplica simple debido a que es la que más se adecua a las actuales condiciones tecnológicas que existen en la universidad.

A continuación se muestra la sentencia a utilizar para el uso de la estrategia de colocación simple, definiéndose el factor de replicación a utilizar.

```
create keyspace tesis with placement_strategy =
'org.apache.cassandra.locator.SimpleStrategy'   and strategy_options =
[{replication_factor: #}];
```

2.3.2.6 CREANDO UN CLÚSTER MULTINODOS EN CASSANDRA

Los valores predeterminados del fichero cassandra.yaml perteneciente al directorio /conf de la carpeta de instalación de Cassandra son muy buenos para levantar el sistema por primera vez y hacerlo correr en un solo nodo. Sin embargo, es inadecuado para su uso en un grupo de múltiples nodos. La configuración y los procesos que se indican en este epígrafe son la forma más sencilla para crear un clúster de varios nodos.

➢ *2.3.2.6.1 Trabajando con el primer nodo:*

Los valores por defecto del fichero **cassandra.yaml** tienen configuradas las direcciones de escucha (para la conexión entre nodos) y Thrift (acceso de los clientes) de forma local, como se muestra a continuación:

listen_address: locahost

rpc_address: localhost

A medida que la dirección de escucha sea utilizada para la comunicación dentro del clúster, debe ser cambiada a una dirección enrutable para que los otros nodos puedan llegar a ella. Por ejemplo, suponiendo que su ordenador tiene una interfaz Ethernet con la dirección 10.208.2.40, tendría que cambiar la dirección de escucha de esta manera:

listen_address: 10.208.2.40

La interfaz Thrift puede ser configurada usando una dirección especificada, al igual que la dirección de escucha, o el uso del comodín 0.0.0.0. Lo que hace Cassandra para escuchar a los clientes en todas las interfaces disponibles. Esta dirección se actualiza de la siguiente manera:

rpc_address: 10.208.2.40

Tal vez la máquina tenga una segunda tarjeta de red con IP 10.208.2.122 para así dividir el tráfico de la red interna del clúster del de Thrift para mejor rendimiento. Configurándose de la siguiente manera:

rpc_address: 10.208.2.122

Si la entrada DNS de su servidor es correcta, es seguro usar un nombre de host en

lugar de una dirección IP. Del mismo modo, la información de la instrucción seed debe cambiar su dirección.

Por defecto estaba configurada de la siguiente manera:

seeds:

```
- 127.0.0.1
```

La nueva configuración sería:

seeds:

```
- 10.208.2.40
```

Una vez que estos cambios fueron realizados, simplemente se debe reiniciar Cassandra en ese nodo. Se debe usar la sentencia **netstat** (`netstat -ant | grep 7000`) para verificar que Cassandra está escuchando en la dirección correcta, apareciendo una línea como esta:

`tcp 0 0 10.208.2.40.7000 *.* ESCUCHAR`

Si **netstat** muestra que Cassandra todavía escucha en 127.0.0.1.7000, entonces o bien el proceso de Cassandra anterior no fue terminado correctamente o no se está editando el archivo cassandra.yaml.

2.3.2.6.2 Trabajando con el resto de los nodos:

Los otros nodos del clúster usarán el fichero cassandra.yaml casi idéntico al del primer nodo configurado. Por lo que se debe usar la configuración del primer nodo como la base de estos cambios en lugar de los valores por defecto de este fichero. El primer cambio es convertir en automático el bootstrapping, esto hará posible la unión del nodo al anillo e intentará tomar el control de una amplia gama de espacios.

auto_bootstrap: true

El segundo cambio es a la dirección de escucha, ya que no debe ser también el bucle de retorno y no puede ser el mismo que cualquier otro nodo. Suponiendo que el segundo nodo dispone de una interfaz Ethernet con la dirección 10.208.2.30, se establece su dirección de escuchar con:

listen_address: 10.208.2.30

Por último, debe de actualizar la dirección del Thrift para aceptar conexiones de clientes, como con el primer nodo, ya sea con una dirección específica o comodín:

rpc_address: 10.208.2.30

Se debe tener en cuenta el hecho de no realizar cambios en la configuración de la sección Seeds, que es la que los nuevos nodos utilizan para usar el primer nodo para el arranque. Una vez que se realicen estos cambios, inicie Casandra en el nuevo nodo y automáticamente se unirá el anillo.

2.3.2.7 REEMPLAZO DE NODOS CAÍDOS

Para reemplazar un nodo del clúster que haya muerto –por fallas de hardware, por ejemplo– se debe añadir un nuevo nodo en su lugar mediante el parámetro `-Dcassandra.replace_token=<token>`, esta instrucción permite que el nuevo nodo añadido al anillo asuma la posición del token del nodo que ha muerto. Para poder reemplazar nodos de esta manera:

- El token que se le asigna al nuevo nodo debe de corresponder con el token del nodo caído – si se trata de reemplazar un nodo asignándole el token de un nodo activo se lanzaría una excepción en el sistema–
- El token que se le asigna al nuevo nodo debe ser parte del anillo.
- El nuevo nodo que se une al clúster no puede contener datos –el directorio de datos debe estar vacío si se desea realizar un reemplazo–

➢ *2.3.2.7.1 Pasos para reemplazar un nodo caído:*

1. Confirmar la existencia del nodo caído usando el comando *nodetool ring* sobre cualquiera de los otros nodos activos del clúster. La Figura 10 se muestra un ejemplo, en el cual se observa que el nodo con dirección IP 10.46.123.12 está en estado caído.

```
$ nodetool ring -h localhost

Address         DC          Rack        Status State   Load            Owns    Token
10.46.123.11    datacenter1 rack1       Up     Normal  179.58 KB       16.67%  0
10.46.123.12    datacenter1 rack1       Down   Normal  315.21 KB       16.67%  28356863910078205288614550619314017621
10.46.123.13    datacenter1 rack1       Up     Normal  267.71 KB       16.67%  56713727820156410577229101238628035242
10.46.123.14    datacenter1 rack1       Up     Normal  315.21 KB       16.67%  85070591730234615865843651857942052863
10.46.123.15    datacenter1 rack1       Up     Normal  292.36 KB       16.67%  113427455640312821154458202477256070485
10.46.123.16    datacenter1 rack1       Up     Normal  300.02 KB       16.67%  141784319550391026443072753096570088106
```

Figura 10 Ejemplo de nodo caído en un clúster

2. Preparar al nodo de reemplazo instalando Cassandra y configurando correctamente el fichero de configuración cassandra.yaml.

3. Iniciar Cassandra en el nuevo nodo haciendo uso de la propiedad `-Dcassandra.replace_token=<token>` asignándole el mismo token que hasta ese momento usaba el nodo caído. En el caso de la figura de ejemplo sería de esta forma:

```
$ cassandra -Dcassandra.replace_token= 28356863910078205
288614550619314017621
```

4. El nuevo nodo iniciará en estado hibernado y comenzará a obtener los datos pertenecientes a la partición que ocupa dentro del clúster. Durante ese tiempo este nodo no aceptará escrituras y será visto como caído por los otros nodo en el clúster. Cuando la carga se haya completado, el nodo será visto como activo por los otros miembros del anillo.

5. Una vez que el nuevo nodo está activo, es recomendable usar el comando **nodetool repair** sobre cada keyspace para asegurarse de que el nodo es completamente consistente. Por ejemplo:

```
$ nodetool repair -h 10.46.123.12 keyspace_name -pr
```

2.4 CONCLUSIONES PARCIALES

En el presente capítulo, se describió el modelo de datos del sistema Cassandra con vista a obtener una visión de la propuesta a desarrollar y basado en el mismo comenzar con el flujo de trabajo de implementación. Se abordó sobre la propuesta de diseño de un Modelo Dimensional con el uso de este sistema atendiendo a las principales restricciones y ventajas con que cuenta el modelo de datos del mismo, realizándose un análisis metódico de aspectos esenciales que avalan su solución. Además se realizó una profunda investigación sobre los principales procesos que permiten el aumento del rendimiento con el uso de esta herramienta de trabajo.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN

3.1 INTRODUCCIÓN

En el presente capítulo se realizará la validación funcional de la propuesta de solución ofrecida durante el desarrollo del trabajo. Se utilizarán diferentes herramientas para verificar los tiempos de respuestas de las consultas y el rendimiento de la aplicación demostrativa.

3.2 VALIDACIÓN MEDIANTE CRITERIOS DE ESPECIALISTA.

Con el objetivo de validar la propuesta de solución que ofrece el sistema Cassandra ante las limitaciones que tienen los sistemas relacionales, se hizo necesaria la contribución de 6 especialistas en el área de investigación con probados conocimientos en el tema de bases de datos, a los cuales se les realizó una serie de preguntas formuladas en un cuestionario (Ver Anexo 1).

Las respuestas dadas por dichos especialistas, los cuales presentan como promedio más 5 años de experiencia en el uso de herramientas de gestión de información arrojan (Ver Tabla 3) que el 66,6% de los encuestados otorga el máximo valor en la escala, atendiendo a la capacidad que presenta el modelo de datos del sistema Cassandra para simular un modelo multidimensional, de la misma manera el 33,3% le otorga un valor alto de 4 puntos. La totalidad de los encuestados opinan que las bases de datos NoSQL de tipo clave-valor agilizan los procesos de acceso a grandes volúmenes de información y que el poder de escalabilidad de Cassandra es superior al de los sistemas relacionales.

El 66,6% de los especialistas piensan que es imprescidible el estudio del carácter simétrico de los clústers que se generan en Cassandra, mientas que el 33,3% opina que es necesario. Todos los encuestados consideran que los sistemas relacionales aún perdiendo alguna de las garantías ACID para poder replicar los datos en otro ordenador no tendrán un mejor comportamiento que el sistema Cassandra, al cual le otorgan un alto nivel a la velocidad de acceso a grandes volúmenes de datos.

Los encuestados no consideran que el aprendizaje del modelo de datos del sistema Cassandra sea más rápido que el relacional y le otorgan a este sistema no relacional un nivel alto de 4 puntos en cuanto a su capacidad de tolerancia a fallos.

	Preg 1				Preg 2		Preg 3			Preg 4		
	Valor				Si	No	Sup	Igual	Inf	Imp	Nec	Innec
E1	5				x		x			x		
E2	5				x		x			x		
E3	5				x		x			x		
E4	4				x		x				x	
E5	4				x		x			x		
E6	5				x		x				x	
%	5	4	3	2								
	66,6	33,3			100	0	100	0	0	66,6	33,3	

	Preg 5		Preg 6				Preg 7		Preg 8		
	Si	No	Valor				Si	No	Alta	Media	Baja
E1		x	4					x	x		
E2		x	4					x	x		
E3		x	4					x	x		
E4		x	4					x	x		
E5		x	4					x	x		
E6		x	4					x	x		
%			5	4	3	2					
	0	100	0	100	0	0	0	100	100	0	0

Tabla 3 Resultados de la encuesta a especialistas

Leyenda: E- Especialista
Preg- Pregunta

Valorando los resultados de la encuesta a los especialistas, se considera que a pesar de la complejidad que tiene el estudio del modelo de datos de Cassandra, resulta muy importante la presente investigación debido a las actuales limitaciones que presentan los sistemas de bases de datos relacionales.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN

3.3 VALIDACIÓN DEL CLÚSTER DE CASSANDRA.

Cassandra, como bien se explicó durante el capítulo anterior probablemente no sea la mejor opción si necesitas únicamente ejecutar un simple nodo. Cassandra está diseñado para distribuirse en múltiples máquinas operando juntas y presentándose como una única entidad al usuario final. Su principal ventaja con respecto a otros sistemas es su alto nivel de escalabilidad a medida que el volumen de datos y las demandas de los usuarios crece. Así, la estructura más externa de Cassandra es el clúster, el cual como se había descrito anteriormente representa el conjunto de máquinas donde se distribuye la información. El objetivo de este epígrafe es demostrar la realización de un clúster con múltiples nodos en este sistema.

El clúster implementado está compuesto por 3 nodos, cada uno de ellos tiene almacenada una porción del volumen total de registros de la base de datos donde como se puede observar en la Figura 11, el mayor peso de la carga lo tiene el host con la dirección ip 10.208.2.115 con un 66,12% de total de registros del sistema.

```
root@Produccion:/opt/cassandra/bin# /opt/cassandra/bin/nodetool -host 10.208.2.40 -p 7199 ring
Address         DC          Rack        Status State   Load            Owns    Token
                                                                               152280516958630826558320640298900719891
10.208.2.40     datacenter1 rack1       Up     Normal  1,17 MB         20,76%  174574042883143471896907989945626391 35
10.208.2.12     datacenter1 rack1       Up     Normal  1,13 MB         13,13%  397899334735270909490804231053805907 85
10.208.2.115    datacenter1 rack1       Up     Normal  3,56 MB         66,12%  152280516958630826558320640298900719891
```

Figura 11 Ilustración del Clúster de 3 nodos implementado durante la investigación

A continuación se muestra el listado de las conexiones activas, tanto entrantes como salientes de uno de los nodos en ejecución cuando el sistema Cassandra está funcionando en su totalidad para el clúster implementado.

```
root@Produccion:/opt/cassandra/bin# netstat -ant | grep 7000
tcp        0      0 10.208.2.40:7000        0.0.0.0:*               ESCUCHAR
tcp        0      0 10.208.2.40:7000        10.208.2.115:54879      ESTABLECIDO
tcp        0      0 10.208.2.40:33252       10.208.2.12:7000        ESTABLECIDO
tcp        0      0 10.208.2.40:42503       10.208.2.12:7000        ESTABLECIDO
tcp        0      0 10.208.2.40:39086       10.208.2.115:7000       ESTABLECIDO
tcp        0      0 10.208.2.40:7000        10.208.2.12:41212       ESTABLECIDO
tcp        0      0 10.208.2.40:42979       10.208.2.115:7000       ESTABLECIDO
```

Figura 12 Ilustración de las conexiones activas del Clúster de 3 nodos implementado

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN

3.4 PRUEBAS DE RENDIMIENTO

Para comprobar el rendimiento de la base de datos Cassandra y su correcto funcionamiento es necesario realizar determinadas pruebas, asegurando así que la misma cumpla con los requisitos definidos. Las pruebas persiguen el objetivo de simular cargas en los sistemas PostgreSQL y Cassandra y observar cómo se comportan estas bases de datos ante estas cargas.

3.4.1 Pruebas de Volumen

Las pruebas de volumen se centran en analizar el comportamiento de ambas bases de datos verificando si alcanzan su límite de almacenamiento y pueda causar fallas. Para ello fue necesario el uso del generador de datos automáticos Datanamic para PostgreSQL y la aplicación denominada Cassandra-Unit para importar ficheros yaml al sistema Cassandra. Estos ficheros fueron generados mediante una aplicación script desarrollada por el autor de la presente investigación.

Los datos generados para cada uno de los sistemas están estructurados de la misma forma y cargados hacia tablas -familias de columnas en el caso del sistema Cassandra- con la misma cantidad de columnas en ambas bases de datos con el objetivo de que no se afecte la calidad de la prueba y que se establezca un resultado lo más fiable y acorde posible a la realidad.

Las tablas fueron llenadas con un rango de datos equivalentes a los podrán alcanzar estas bases de datos en un período no menor a un año. Las tablas tienen almacenados 600000 registros.

Al introducir los datos en las tablas no se presentaron problemas de límites de capacidad, desbordamiento de columnas, atributos o tipos de datos. La utilización de estas herramientas permitió verificar la integridad de los datos.

3.4.2 Pruebas de Carga

Las pruebas de carga que se establecen en este capítulo están orientadas a dar respuesta a la idea a defender del presente trabajo de diploma. Estas pruebas se centran fundamentalmente en la realización de una comparación entre el sistema de bases de datos NoSQL Cassandra, objeto de estudio durante el desarrollo de la presente investigación y el sistema relacional PostgreSQL, teniendo como principal

parámetro de medición la velocidad de respuesta de estos sistemas a medida que aumenta su volumen de datos. Las consultas realizadas a ambas bases de datos no tuvieron variación, en ambos sistemas se realizaron las peticiones con idéntico nivel de complejidad.

Se realizaron 4 fases de prueba, aumentándose paulatinamente la carga de datos en ambas bases de datos, cada fase consta de 3 iteraciones con el objetivo de establecer el tiempo promedio que demoran en acceder a los datos requeridos en cada una de las consultas de ambas bases de datos.

3.3.2.1 Rendimiento con 3000 registros almacenados

La primera fase de la prueba se realizó con volúmenes de datos relativamente bajos a los cuales cualquier aplicación respetable a nivel mundial debe superar, se cargaron 3000 registros hacia cada uno de los sistemas y estos fueron los resultados

- ***Comportamiento en PostgreSQL***

Tal como se puede evidenciar en la Tabla 4, el sistema PostgreSQL presenta un comportamiento bastante estable sobre un segundo y fracción en el mapeo de los datos. El tiempo promedio de respuesta luego de la realización de las 3 iteraciones en este sistema con 3000 datos registrados es de 1, 235 segundos.

	ITERACIONES		
	1ra	**2da**	**3ra**
Tiempo (s)	1,560	1,084	1,063

Tabla 4 Rendimiento del sistema PostgreSQL ante una carga de 3000 registros

Las ilustraciones que se muestran en los Anexos 2 y 3 representan las evidencias de las pruebas realizadas al sistema PostgreSQL para esta cantidad de datos.

- ***Comportamiento en Cassandra***

En sentido general, el comportamiento del sistema Cassandra para cada fase se torna algo inestable, debido a que la presente prueba se realizó sobre un clúster de 3 ordenadores anidados. Los tiempos están en dependía del tráfico de red y la

cantidad de procesos que se encuentren en ejecución en ese momento en cada una de las máquinas miembros del anillo. A pesar de eso, como se puede evidenciar en la Tabla 5, Cassandra puede lograr tiempos de respuesta similares e incluso inferiores a los presentados por el sistema PostgreSQL en la misma instancia. El tiempo de respuesta promedio para las 3 mejores iteraciones de Cassandra es de 1,051.

	ITERACIONES		
	1ra	2da	3ra
Tiempo (s)	1,035	1,034	1,084

Tabla 5 Rendimiento del sistema Cassandra ante una carga de 3000 registros

Las evidencias de las pruebas realizadas al sistema Cassandra para esta cantidad de datos se pueden observar en los Anexos 4 y 5 del presente documento.

3.3.2.2 Rendimiento con 10000 registros almacenados

Para la segunda fase, tal y como se establece en el enunciado del epígrafe se cargaron 10000 registros en ambas bases de datos y estos fueron los resultados que se obtuvieron.

- ***Comportamiento en PostgreSQL***

El sistema PostgreSQL continúa teniendo resultados bastante estables sobre los 3 segundos y medio aproximadamente (Tabla 6). El tiempo de respuesta promedio para este sistema en esta fase es de 3,562 segundos aproximadamente.

	ITERACIONES		
	1ra	2da	3ra
Tiempo (s)	3,553	3,555	3,577

Tabla 6 Rendimiento del sistema PostgreSQL ante una carga de 10000 registros

- ***Comportamiento en Cassandra***

Cassandra mantiene tiempos de respuesta bastante similares a los de PostgreSQL en esta instancia, sus tiempos oscilan entre por los 3 segundos y medio también, aunque con una tendencia a disminuir en ocasiones (Ver Tabla 7). El tiempo de respuesta promedio para las 3 mejores iteraciones del sistema Cassandra en esta fase es de 3,498 aproximadamente.

	ITERACIONES		
	1ra	**2da**	**3ra**
Tiempo (s)	3,543	3,501	3,450

Tabla 7 Rendimiento del sistema Cassandra ante una carga de 10000 registros

3.3.2.3 Rendimiento con 30000 registros almacenados

El comportamiento de ambas bases de datos luego de que se triplicara la cifra almacenada en la fase anterior es como se muestra a continuación:

- ***Comportamiento en PostgreSQL***

El promedio de tiempo de respuesta de PostgreSQL luego de realizadas las 3 iteraciones (Ver Tabla 8) es de 11,076.

	ITERACIONES		
	1ra	**2da**	**3ra**
Tiempo (s)	10,524	12,055	10,649

Tabla 8 Rendimiento del sistema PostgreSQL ante una carga de 30000 registros

Las ilustraciones que se muestran en los Anexos 6 y 7 representan las evidencias de las pruebas realizadas al sistema PostgreSQL para esta cantidad de datos.

- ***Comportamiento en Cassandra***

El sistema Cassandra tal y como se puede observar en la Tabla 9 evidencia una ligera mejoría en comparación con los tiempos de respuesta para la misma cantidad de datos del sistema PostgreSQL, aunque aún no es base suficiente para

demostrar el verdadero potencial de respuesta de esta herramienta ante el mapeo de grandes volúmenes de datos. El promedio de tiempo de respuesta de Cassandra ante 30000 registros insertados es de 10,365 aproximadamente.

ITERACIONES

	1ra	2da	3ra
Tiempo (s)	10,377	10,354	10,363

Tabla 9 Rendimiento del sistema Cassandra ante una carga de 30000 registros

Los resultados de estas pruebas se pueden evidenciar en los Anexos 8 y 9 del presente documento.

3.3.2.4 Rendimiento con 60000 registros almacenados

La última fase de prueba de rendimiento para cada uno de los sistemas se realizó empleando una carga hacia ambas bases de datos de 60000 registros, la cual permitió luego del estudio y comparación de las fases anteriores y los resultados que en la presente se muestran dar las conclusiones finales de la comparación.

- ***Comportamiento en PostgreSQL***

El sistema PostgreSQL logra mapear 60 000 registros a un promedio de 21,122 segundos como resultado de lo evidenciado por las 3 iteraciones de prueba para esta instancia (Ver Tabla 10).

ITERACIONES

	1ra	2da	3ra
Tiempo (s)	21,073	21,097	21,196

Tabla 10 Rendimiento del sistema PostgreSQL ante una carga de 60000 registros

Los resultados de estas pruebas se pueden evidenciar en los Anexos 10 y 11 del presente documento.

- ***Comportamiento en Cassandra***

Los tiempos de respuesta de Cassandra para esta fase son inferiores a los

mostrados por la base de datos PostgreSQL. El promedio es de 17, 846 segundos aproximadamente (Ver Tabla 11).

	ITERACIONES		
	1ra	2da	3ra
Tiempo (s)	18,143	16,787	18,609

Tabla 11 Rendimiento del sistema Cassandra ante una carga de 60000 registros

Las evidencias de las pruebas realizadas al sistema Cassandra para esta cantidad de datos se pueden observar en los Anexos 12 y 13 del presente documento.

Luego de realizadas las 4 fases de prueba para ambos sistemas, se procede a realizar un análisis detallado de los resultados obtenidos. Tal como lo muestra la Figura 12, a medida que aumenta el volumen de datos en cada base de datos, la capacidad de respuesta de Cassandra se hace cada vez mejor en comparación con los tiempos que presenta PostgreSQL.

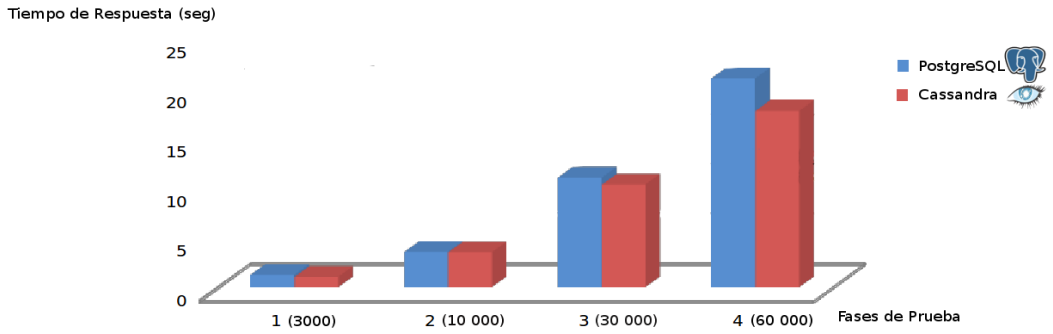


Figura 13 Gráfico del rendimiento de los sistemas PostgreSQL y Cassandra en cada fase de prueba

CONCLUSIONES PARCIALES

En este capítulo se analizó un conjunto de criterios a tener en cuenta para validar la propuesta de solución brindada que servirá como base material de estudio para los siguientes proyectos de gestión de datos en la universidad. Las pruebas realizadas demostraron la disminución de los tiempos de respuestas y el mejor rendimiento del sistema Cassandra en comparación con PostgreSQL principalmente a medida que aumenta considerablemente el volumen de datos. Además se demostró la

capacidad que presenta Cassandra para escalar horizontalmente, lo cual posibilita una mayor tolerancia a fallos de las aplicaciones que se generen con su uso, dándose solución al problema de investigación que arrojó la presente investigación: la búsqueda de un sistema altamente escalable que permita el acceso rápido a la información con grandes volúmenes de carga de datos y tolerante a fallos.

CONCLUSIONES GENERALES

Concluido el presente trabajo de investigación se puede afirmar que se ha cumplido con el objetivo del mismo, ofrecer una base teórica y práctica sólida que se tenga en consideración para los siguientes proyectos que se realicen en la universidad, esencialmente aquellos destinados a la gestión de información.

La propuesta de optimización elaborada fue validada mediante criterios de especialistas y la realización de pruebas de carga demostrando que los resultados obtenidos tributan a un rendimiento superior de dicha propuesta en comparación con los sistemas relacionales.

Como conclusión general, puede afirmarse que se le dio cumplimiento a todos los objetivos planteados al inicio del trabajo y se validó la idea a defender materializada con las pruebas realizadas a la solución propuesta.

RECOMENDACIONES

Se recomienda principalmente luego del desarrollo del presente trabajo investigativo, seguir profundizando en los estudios destinados a las herramientas que utiliza el sistema Cassandra para establecer las conexiones cliente-servidor, cuya investigación quedó limitada debido a que el presente trabajo se centró principalmente en la demostración de las potencialidades de rendimiento que ofrece este sistema en comparación con las soluciones relacionales y por cuestiones de tiempo no fue posible abundar en el estudio de las mismas .

REFERENCIAS BIBLIOGRÁFICAS

1. CITMATEL, UEB Servicios Web. *Discurso del comandanate Fidel Castro .* [En línea] 2008. [Citado el: 12 de 12 de 2011.] http://www.cuba.cu/gobierno/discursos/1960/esp/f150160e.html.

2. ARQHYS. [En línea] 2010. [Citado el: 23 de 1 de 2012.] http://www.arqhys.com/construcciones/escalabilidad.html.

3. Pérez Valdés, Damián. ¿Qué son las bases de datos?, 2008. Disponible en: http://www.maestrosdelweb.com/principiantes/%C2%BFque-son-las-bases-de-datos

4. Victoria. 2008. Definición ABC. [En línea] 2008. [Citado el: 13 de 1 de 2012.] http://www.definicionabc.com/tecnologia/base-de-datos.php.

5. Mi Tecnológico [En línea] 2009. [Citado el: 25 de 1 de 2012.] http://www.mitecnologico.com/Main/PropiedadesDeAtomicidadConsistenciaAislamientoYDurabilidadAcid.

6. Pensamientos Ágiles. [En línea] 7 de 8 de 2009. [Citado el: 26 de 1 de 2012.] http://brigomp.blogspot.com/2009/08/sharding-si-o-no.html.

7. NoSQL [En línea] 2010. [Citado el: 26 de 1 de 2012.] http://es.wikipedia.org/wiki/NoSQL.

8. Inteligencia del Negocio para Todos [En línea] 6 de 9 de 2010. [Citado el: 28 de 1 de 2012.] http://inparatodos.blogspot.com/2010/09/que-es-un-modelo-dimensional.html.

9. Apache Cassandra [En línea] 2009. [Citado el: 31 de 1 de 2012.] http://cassandra.apache.org/.

10. Apache Hbase [En línea] 2012. [Citado el: 2 de 2 de 2012.] http://hbase.apache.org/.

11. MongoDB [En línea] 2009. [Citado el: 2 de 2 de 2012.] http://www.mongodb.org/.

12. My ComputerPro [En línea] 30 de 3 de 2010. [Citado el: 28 de 1 de 2012.] http://www.muycomputerpro.com/2010/03/30/actualidadnoticiasnosql-no-cambiara-el-mundo_we9erk2xxdb6rqkk96nhqlaxhzqozy8gqpjs5j44nnxnr458jaiujitw3a02ml8g/.

13. RapidSVN [En línea] 30 de 3 de 2010. [Citado el: 3 de 2 de 2012.] http://rapidsvn.com/index.php/Documentation.

14. Foundation, Apache Software. DataStax. Getting Started Using the Cassandra CLI. [En línea] 2011. [Citado el: 4 de 2 de 2012.] http://www.datastax.com/docs/0.8/dml/using_cli..

15. Apache Thrift [En línea] 2010. [Citado el: 4 de 2 de 2012.] http://thrift.apache.org/.

16. Datanamic [En línea] 2008. [Citado el: 5 de 3 de 2012.] http://www.datanamic.com/products/postgresql-tools.html.

REFERENCIAS BIBLIOGRÁFICAS

17. Hector-Users. [En línea] 2012. [Citado el: 17 de 3 de 2012.] http://groups.google.com/group/hector-users/browse_thread/thread/6b274a6498584da7.

BIBLIOGRAFÍA

1. **CITMATEL, UEB Servicios Web**. *Discurso del comandanate Fidel Castro .* [En línea] 2008. [Citado el: 12 de 12 de 2011.] http://www.cuba.cu/gobierno/discursos/1960/esp/f150160e.html.

2. **ARQHYS**. [En línea] 2010. [Citado el: 23 de 1 de 2012.] http://www.arqhys.com/construcciones/escalabilidad.html.

3. **Pérez Valdés, Damián.** ¿Qué son las bases de datos?, 2008. Disponible en: http://www.maestrosdelweb.com/principiantes/%C2%BFque-son-las-bases-de-datos

4. **Victoria. 2008.** Definición ABC. [En línea] 2008. [Citado el: 13 de 1 de 2012.] http://www.definicionabc.com/tecnologia/base-de-datos.php.

5. **Mi Tecnológico** [En línea] 2009. [Citado el: 25 de 1 de 2012.] http://www.mitecnologico.com/Main/PropiedadesDeAtomicidadConsistenciaAislamientoYDurabilidadAcid.

6. **Pensamientos Ágiles.** [En línea] 7 de 8 de 2009. [Citado el: 26 de 1 de 2012.] http://brigomp.blogspot.com/2009/08/sharding-si-o-no.html.

7. **NoSQL** [En línea] 2010. [Citado el: 26 de 1 de 2012.] http://es.wikipedia.org/wiki/NoSQL.

8. **Inteligencia del Negocio para Todos** [En línea] 6 de 9 de 2010. [Citado el: 28 de 1 de 2012.] http://inparatodos.blogspot.com/2010/09/que-es-un-modelo-dimensional.html.

9. **Apache Cassandra** [En línea] 2009. [Citado el: 31 de 1 de 2012.] http://cassandra.apache.org/.

10. **Apache Hbase** [En línea] 2012. [Citado el: 2 de 2 de 2012.] http://hbase.apache.org/.

11. **MongoDB** [En línea] 2009. [Citado el: 2 de 2 de 2012.] http://www.mongodb.org/.

12. **My ComputerPro** [En línea] 30 de 3 de 2010. [Citado el: 28 de 1 de 2012.] http://www.muycomputerpro.com/2010/03/30/actualidadnoticiasnosql-no-cambiara-el-mundo_we9erk2xxdb6rqkk96nhqlaxhzqozy8gqpjs5j44nnxnr458jaiujitw3a02ml8g/.

13. **RapidSVN** [En línea] 30 de 3 de 2010. [Citado el: 3 de 2 de 2012.] http://rapidsvn.com/index.php/Documentation.

14. **Foundation, Apache Software. DataStax.** Getting Started Using the Cassandra CLI. [En línea] 2011. [Citado el: 4 de 2 de 2012.] http://www.datastax.com/docs/0.8/dml/using_cli..

15. **Apache Thrift** [En línea] 2010. [Citado el: 4 de 2 de 2012.] http://thrift.apache.org/.

16. **Datanamic** [En línea] 2008. [Citado el: 5 de 3 de 2012.] http://www.datanamic.com/products/postgresql-tools.html.

BIBLIOGRAFÍA

17. Hector-Users. [En línea] 2012. [Citado el: 17 de 3 de 2012.] http://groups.google.com/group/hector-users/browse_thread/thread/6b274a6498584da7.

18. Cassandra Wiki [En línea] 2009. [Citado el: 15 de 2 de 2012.] http://wiki.apache.org/cassandra/CassandraCli.

19. Datastax [En línea] 2010. [Citado el: 18 de 2 de 2012.] http://www.datastax.com/docs/0.8/dml/using_cli.

20. Getting Started with Cassandra [En línea] 2008. [Citado el: 26 de 2 de 2012.] http://jonathanhui.com/cassandra-cli-client.

21. Hewitt, Eben. Noviembre 2010. *Cassandra The Definitive Guide.* United States of America : O'Reilly, Noviembre 2010.

22. Jonathan [En línea] 2010. [Citado el: 26 de 2 de 2012.] http://jonathanhui.com/cassandra-cli-client.

23. Scribd [En línea] 2010. [Citado el: 3 de 2 de 2012.] http://es.scribd.com/doc/76613613/Apache-Cassandra.

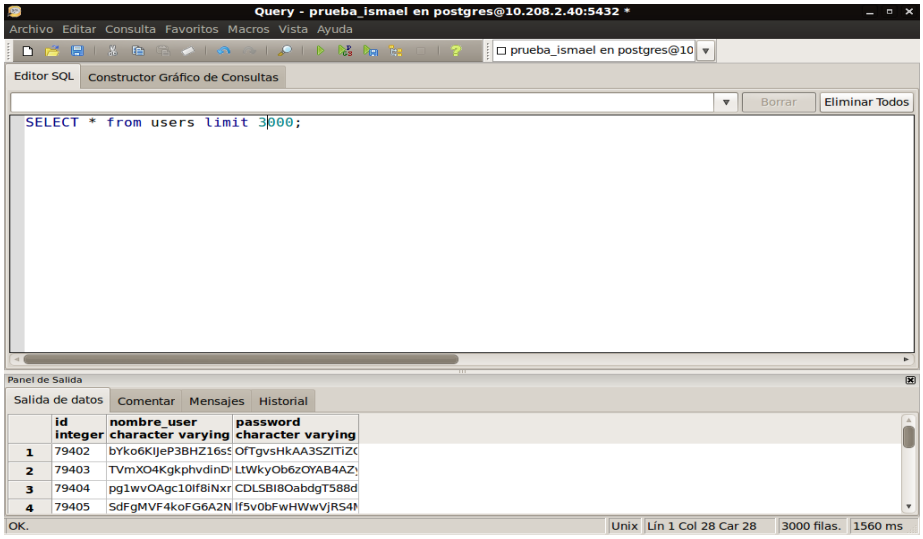
ANEXOS

ANEXO 1: Encuesta realizada a especialistas del centro con vistas a la validación de la propuesta de solución

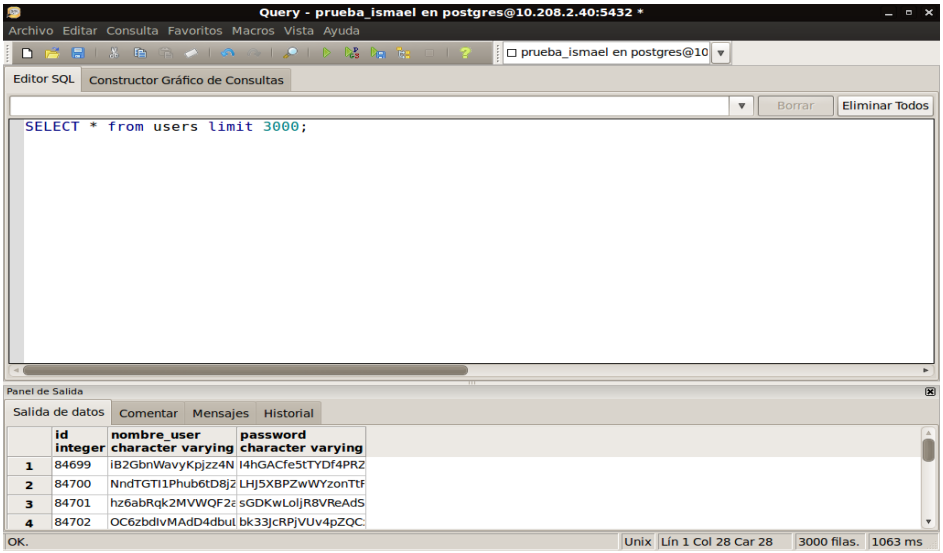
Encuesta para la validación de la propuesta
1. Considera usted que el modelo de datos de Cassandra tiene potencialidades para simular un modelo multidimensional. Evalúelo en un rango de 5 a 2. __5 __4 __3 __2 En caso de seleccionar 2, argumente su selección
2. ¿Cree usted que el modelo de datos clave-valor agiliza el proceso de acceso a grandes volúmenes de información? Si – No. Justifique en cualquiera de los casos. __Si __No
3. ¿Cómo clasificarías la escalabilidad de Cassandra respecto a los sistemas de bases de datos relacionales? __Superior __ Igual __ Inferior
4. ¿Cómo considera usted el estudio del carácter simétrico de los clústers que se generan en Cassandra atendiendo a las dificultades en la replicación de datos de los sistemas relacionales? __Imprescindible. __Necesario __Innecesario
5. ¿Considera usted que la curva de aprendizaje del modelo de datos de Cassandra es superior a la de los modelos relacionales tradicionales para un usuario conocimientos del tema? __Si __No
6. ¿Qué nivel de calidad en cuanto a tolerancia a fallos y disponibilidad de la información le otorgaría usted al sistema Cassandra. Evalúelo en un rango de 5 a 2. __5 __4 __3 __2
7. Considera usted que perdiendo alguna de las garantías ACID (Aislamiento, Consistencia, Durabilidad y Atomicidad) para poder replicar los datos en otro ordenador los sistemas relacionales tendrán un mejor comportamiento que el

sistema Cassandra. __Si __No
8. ¿Cómo evalúa usted la velocidad de acceso a grandes volúmenes de datos en el sistema Cassandra? __Alta __Media __Baja

ANEXO 2: Iteración 1 de la fase 1 de pruebas en el sistema PostgreSQL

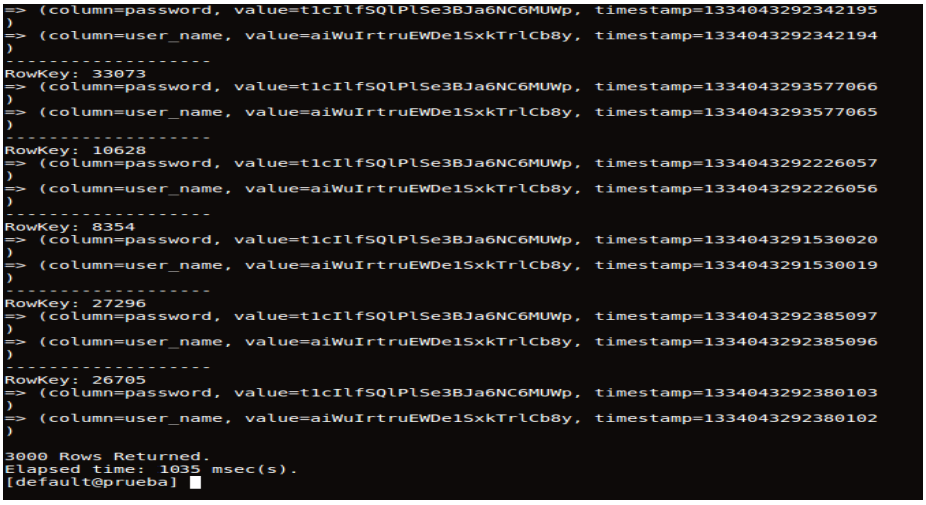


ANEXO 3: Iteración 3 de la fase 1 de pruebas en el sistema PostgreSQL

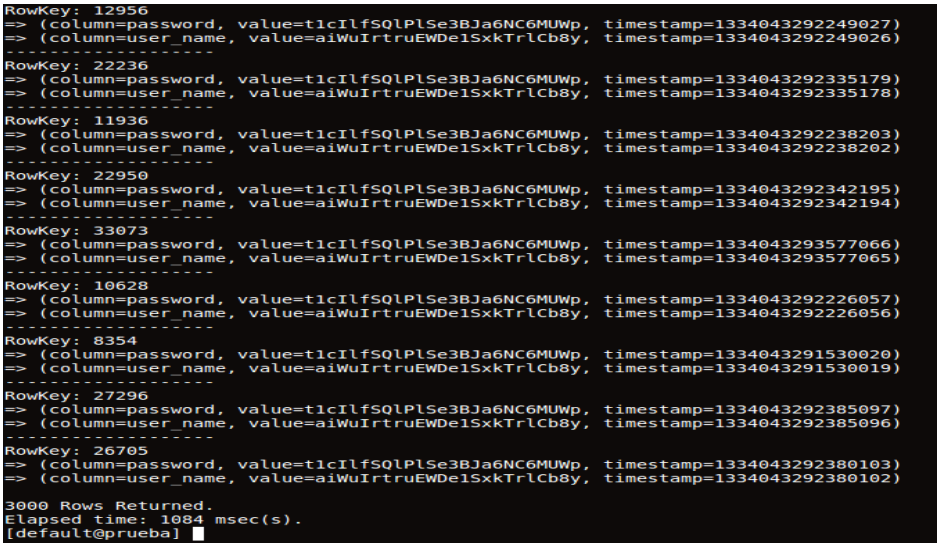


ANEXOS

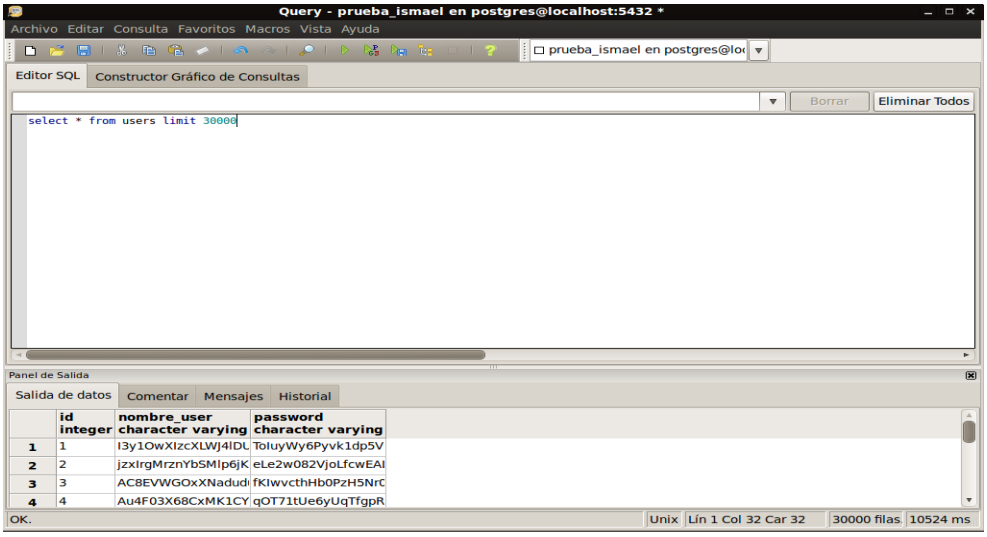
ANEXO 4: Iteración 1 de la fase 1 de pruebas en el sistema Cassandra.



ANEXO 5: Iteración 3 de la fase 1 de pruebas en el sistema Cassandra.

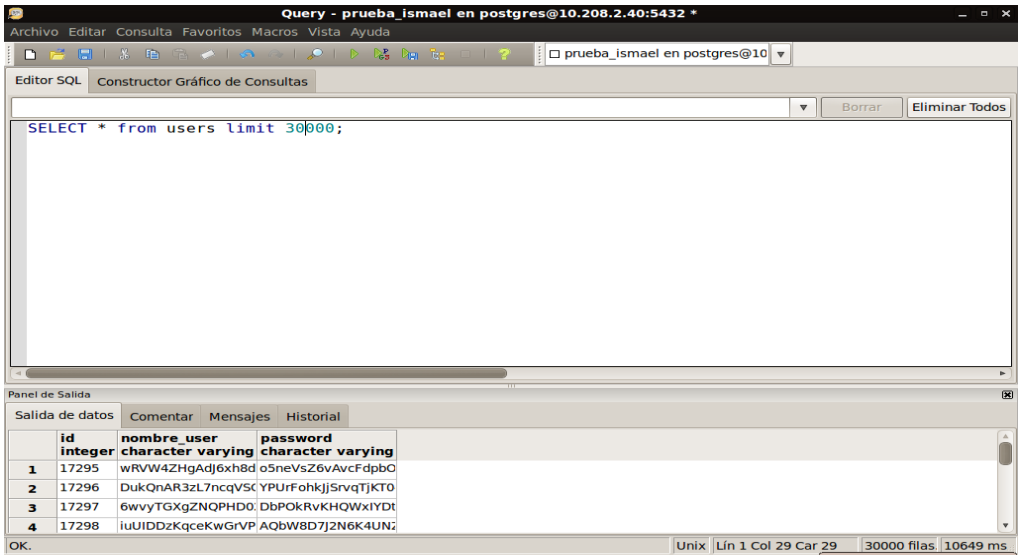


ANEXO 6: Iteración 1 de la fase 3 de pruebas en el sistema PostgreSQL

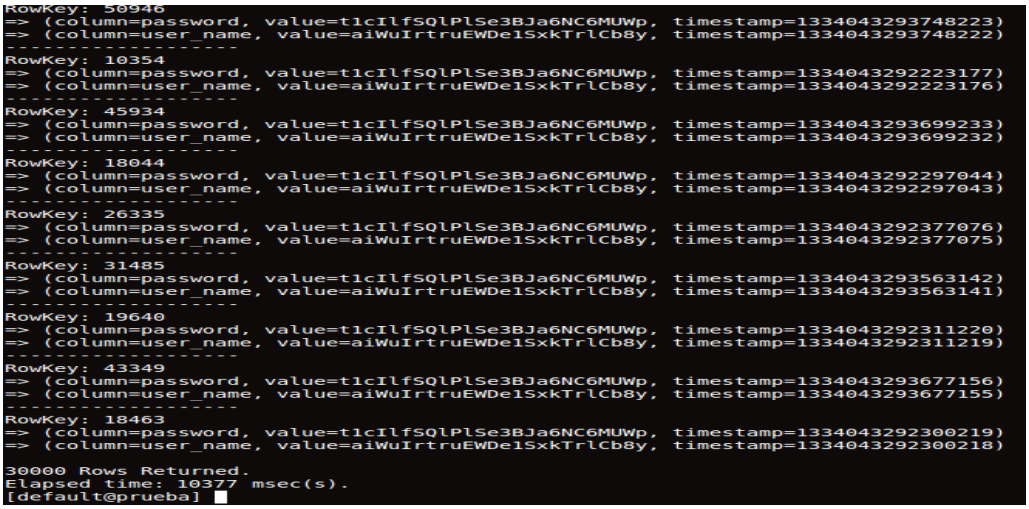


ANEXOS

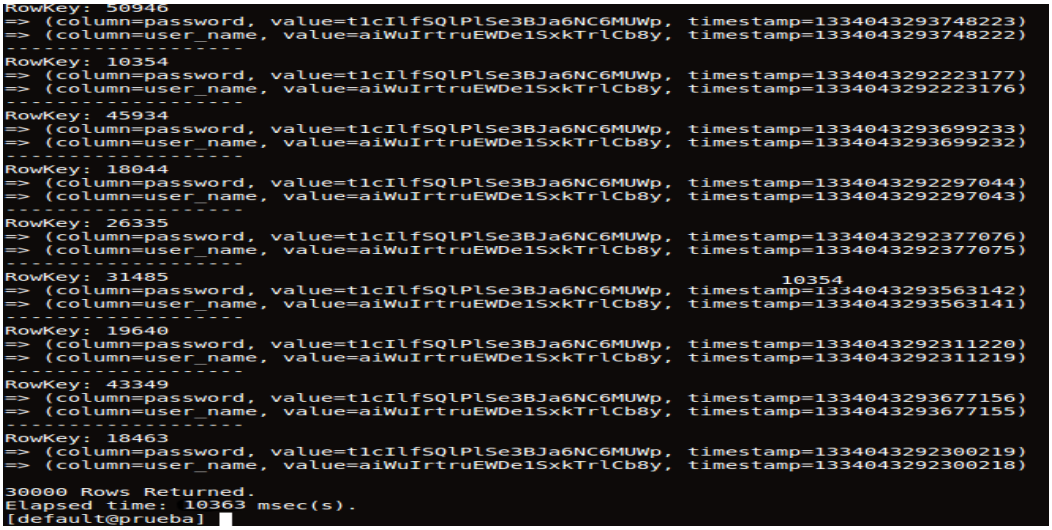
ANEXO 7: Iteración 3 de la fase 3 de pruebas en el sistema PostgreSQL



ANEXO 8: Iteración 1 de la fase 3 de pruebas en el sistema Cassandra

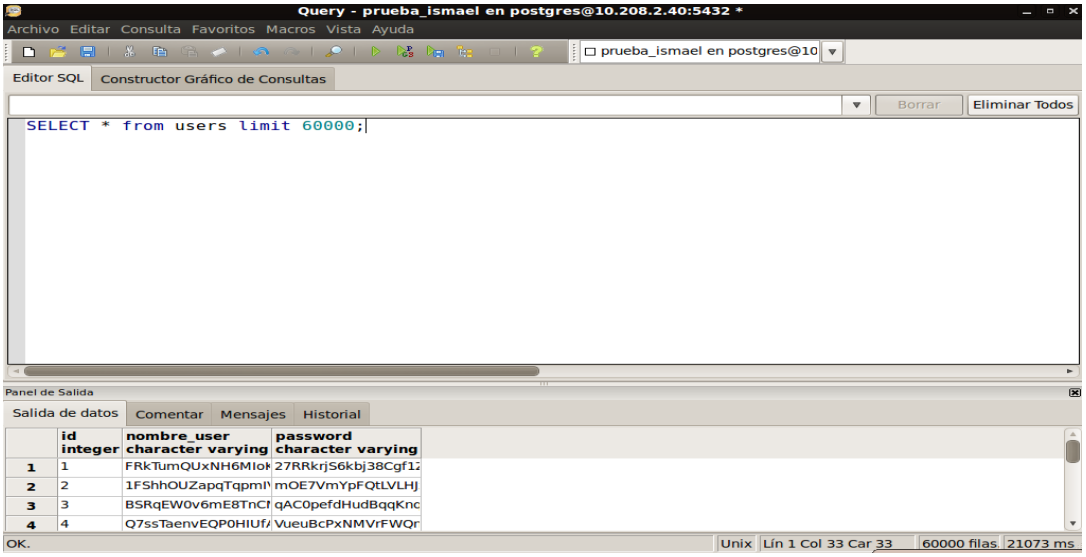


ANEXO 9: Iteración 3 de la fase 3 de pruebas en el sistema Cassandra

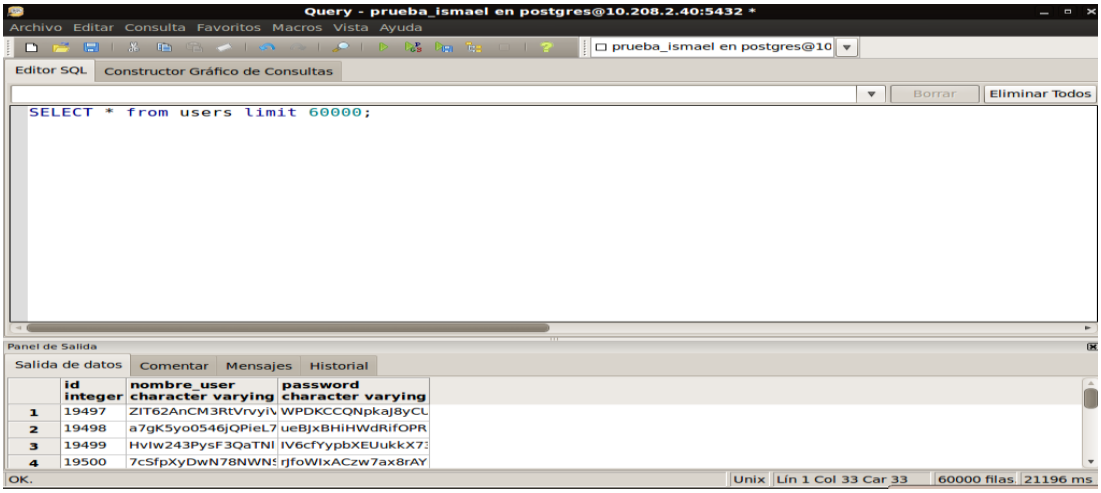


ANEXOS

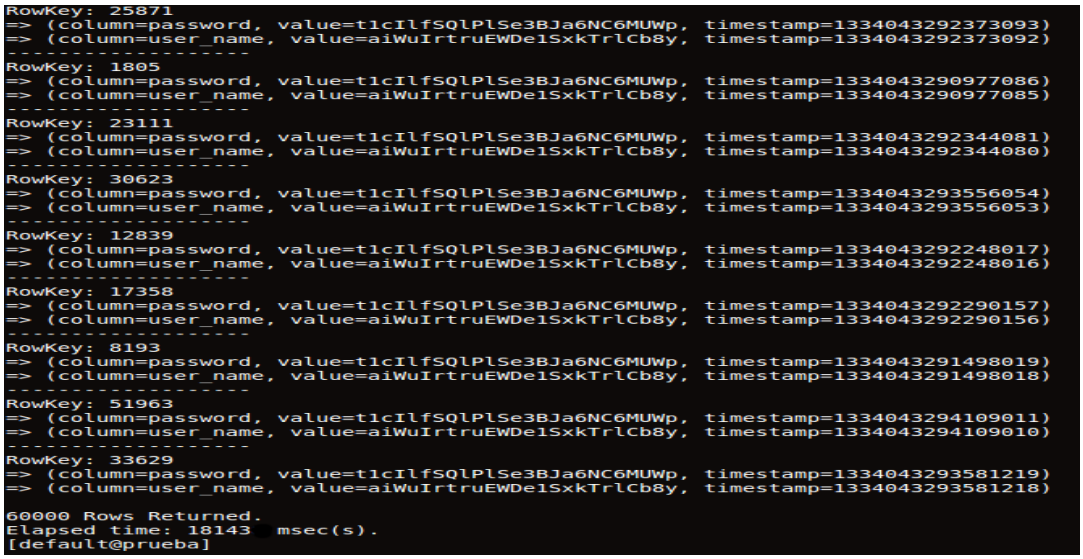
ANEXO 10: Iteración 1 de la fase 4 de pruebas en el sistema PostgreSQL



ANEXO 11: Iteración 3 de la fase 4 de pruebas en el sistema PostgreSQL



ANEXO 12: Iteración 1 de la fase 4 de pruebas en el sistema Cassandra



ANEXO 13: Iteración 3 de la fase 4 de pruebas en el sistema Cassandra