UNIVERSIDAD DEL BÍO-BIO
FACULTAD DE CIENCIAS EMPRESARIALES
ESCUELA DE INGENIERÍA CIVIL INFORMÁTICA

UNIVERSIDAD DEL BÍO-BÍO

"Estudio y diseño de un Data Mart Clínico NonSQL que permita comparar los tiempos de respuesta versus un ambiente basado en SQL"

Memoria para optar por el título de Ingeniero Civil en Informática.

Autor:
Celeste Monserrat Torres Varela

Profesor Guía:
Sr. Manuel Crisosto Muñoz

Concepción, Abril 2013

Agradecimientos:

De primera instancia a mis Padres, Mireya Varela y Norberto Torres, por su apoyo incondicional.

Mi hermana Angela por no hacer mi vida monótona y mis hermanos por estar presente en todo momento.

A Leonel por ser una parte importante en mi vida, por su comprensión y contención durante todo este tiempo.

A mi amigo Alfredo Arévalo, por su preocupación, ayuda y apoyo constante durante todos los años académicos.

A Francisco Ortiz, por su buena disposición, apoyo, paciencia y por hacer posible este proyecto.

Resumen

Este estudio se realiza con la finalidad de comparar dos modelos de base de datos: el tradicional modelo de datos relacional y el denominado NoSQL, que una de sus características principales es su modelado flexible, poca rigidez al almacenar datos y capacidad de gestión de grandes cantidades de datos.

La comparativa se realiza basado un modelo multidimensional denominado Data Marts, el cual almacena información histórica y relevante de las entidades.

Se implementan ambos modelos y, finalmente, se realizan pruebas sobre ellos, para obtener información cuantitativa sobre sus rendimientos.

Abstract

This study was performed in order to compare two database models: the traditional relational data model and the so-called NoSQL, which one of its main features is its flexible modeling, low stiffness to store data and ability to manage large amounts of data.

The comparison is performed based on a multidimensional model called data marts, which stores historical information and relevant entities.

Both models are implemented and, finally, tests are realized on them, to obtain quantitative information about his performances.

INDICE

CONTENIDO **PAGINA**

INDICE ILUSTRACIONES

CONTENIDO **PAGINA**

INDICE TABLAS

CONTENIDO **PAGINA**

INTRODUCCION

Hoy, en que los computadores y las tecnologías de información generan una gran cantidad de datos, se vuelve esencial la gestión óptima de ellos.

Las nuevas aplicaciones pertenecientes a la web 2.0 como Twitter, Facebook, Amazon, entre otros, requieren de motores de almacenamiento que logren gestionar cientos y miles de Terabytes y Zetabytes de datos.

Cisco Visual Networking Index(VNI), a mediados del 2012 pronosticaba que para el 2016 el tráfico IP se multiplicaría por 4 veces y alcanzaría 1,3 Zetabytes[1], sin embargo, la primera semana de febrero del 2013 publicó un informe elaborado por la empresa de investigación IDC, patrocinado por EMC, en el cual se superan todas las estimaciones en más del 14%. [0]

Según el estudio, para el año 2020 la cantidad de datos digitales a nivel mundial alcanzará los 40 Zetabytes. Para entender esta cifra, varios medios de comunicación han realizado la comparación entre los datos y la cantidad de granos de arena en todas las playas del mundo: los datos resultarían mayores. [0]

Por lo cual, para estas aplicaciones ha surgido un movimiento denominado NoSQL, que por medio de una comunidad de Software Libre ha desarrollado soluciones escalables que permiten manejar esta problemática.

Este trabajo presenta un modelo de base de datos no relacional, mencionando su historia, características y sus diversas funcionalidades.
Luego, se adapta un modelo relacional a un entorno no relacional, implementado en el motor de NOSql de Apache-Cassandra. La adaptación de los modelos se realiza con la finalidad de comparar los tiempos de respuestas que se obtienen en ambos motores de base de datos, pero bajo una filosofía distinta de almacenamiento, como son las bases de datos NoSQL.

Finalmente, se realizan pruebas de rendimiento de ambos diseños, los que son comparados cuantitativamente para así poder determinar el modelo que responde de mejor manera a las consultas realizadas.

[1] Zetabyte(ZB)= 1.099.511.627.776 GB

1 MARCO TEORICO

El origen de las bases de datos está ligado a la necesidad de organizar datos. En los inicios, la información estaba delegada a los archivos, como depósitos de información y a los que se accedía a bajo nivel para las operaciones más elementales de lectura y escritura. Con el tiempo, la información requerida se hacía mayor, y la complejidad de gestionarla también creció. Se comenzó a aplicar técnicas como índices para accesos más rápidos y para el ordenamiento de la misma. Pero, las aplicaciones debían tratar los archivos de forma artesanal y personalizada. [1]

El modelo relacional, postulado por Frank Codd en 1970, surge tras el crecimiento de la información y la difícil forma de gestionarlas.
En este modelo señala que la normalización[2] de los datos permite organizar la información de manera eficiente dentro de una base de datos, separándolas en tablas de información de la forma más pequeña posible e independiente, para evitar la repetición de los datos. Para acceder a toda la información, las tablas se relacionan unas y otras mediante campos clave comunes, que suelen ser números únicos.

Este modelo de almacenamiento es organizado y óptimo, define unas reglas que evitan la redundancia de la información y facilitan la consistencia de los datos. Este sistema de ingeniería de la información es el que ha regido las bases de datos durante los últimos 40 años. [1]

En este capítulo se tratan los temas de las bases de datos en sí, con una historia desde sus origen hasta la actualidad, descripción de las bases de datos y sus distintos modelo y finalmente el análisis de lo que corresponde a las Base de Datos Relacionales, Multidimensionales, las NoSQL y Apache-Cassandra que es el modelo NoSQL con el que se realiza el estudio.

[2] Normalización es el proceso de organizar de manera eficiente los datos dentro de una base de datos. (Sergio Sánchez http://www.slideshare.net/sesa78/normalizacion-de-base-de-datos-14102278)

1.1 HISTORIA BASE DE DATOS

Los orígenes de las bases de datos se remontan a la antigüedad, en esa época ya existían bibliotecas y toda clase de de documentos que servían para almacenar registros. Además, también se utilizaban para recoger información sobre las cosechas y censos.

Sin embargo, su búsqueda era lenta, poco eficaz y no se contaba con la ayuda de máquinas que pudiesen reemplazar el trabajo manual.

El uso de sistemas de bases de datos automatizadas, se desarrolló a partir de la necesidad de almacenar grandes cantidades de datos, para su posterior consulta, por lo cual, se idearon formas para almacenar gran volúmenes de datos y se crearon las Bases de Datos.

Para entender lo que son las bases de datos, a continuación se profundiza en su historia.

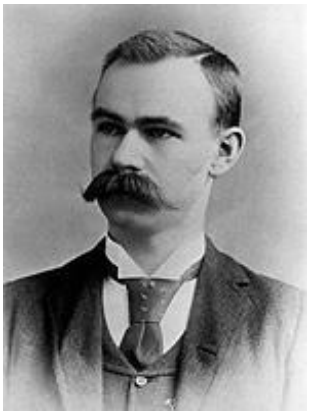


Ilustración 1: Herman Hollerith

En 1884, se realizó el censo poblacional en Estados Unidos. En esa época, los censos se realizaban de forma manual, por lo que la realización de este proceso llevaba mucho tiempo. Ante esta situación Herman Hollerith[3] creó una máquina tabuladora o censadora, basada en tarjetas perforadas, reduciendo el tiempo que demoraba el desarrollo del censo. Gracias a este invento, fue nombrado el primer ingeniero estadístico de la historia. [2]

- **Década de 1950.**

  Se desarrollaron las cintas magnéticas para almacenar los datos. El procesamiento de datos consistía en leer datos de una o más cintas y escribir datos en una nueva cinta. Los datos también se podían introducir desde paquetes de tarjetas perforadas e impresos en impresoras. Por ejemplo, los aumentos de sueldo se procesaban introduciendo los aumentos en las tarjetas perforadas y leyendo el paquete de tarjetas perforadas en sincronización con una cinta que contenía los detalles maestros de los salarios. Los

[3] Ingeniero estadístico estadounidense. Es considerado el primer informático de la historia, por el logro de un tratamiento automático de la información.

registros debían estar igualmente ordenados. Los aumentos de sueldo tenían que añadirse a los sueldos leídos de la cinta maestra, y escribirse en una nueva cinta; esta última convirtiendose en la nueva cinta maestra. Las cintas (y los paquetes de tarjetas perforadas) sólo se podían leer secuencialmente, y los tamaños de datos eran mucho mayores que la memoria principal; así, los programas de procesamiento de datos tenían que procesar los datos según un determinado orden, leyendo y mezclando datos de cintas y paquetes de tarjetas perforadas. [3]

- **Década de 1960.**

  El amplio uso de los discos fijos a finales de la década de 1960 cambió en gran medida el escenario del procesamiento de datos, ya que los discos fijos permitieron el acceso directo a los datos.

  La ubicación de los datos en disco no era importante, porque se podía acceder a cualquier posición del disco en sólo decenas de milisegundo.

  Con los discos se pudieron desarrollar las bases de datos de red y jerárquicas, que permitieron que las estructuras de datos tales como listas y árboles pudieran almacenarse en disco. Los programadores pudieron construir y manipular estas estructuras de datos.

  Sin embargo, dado que almacenamiento de en disco era costoso, se comenzó a almacenar en cintas magnéticas. Como los datos eran utilizados por diferentes aplicaciones, se debía reorganizar la infomarción, reordenando los datos según su identificador y fusionando después la información. Normalmente se procesaban los datos en bloques, es decir, todos los registros se procesan en un mismo tiempo. [4]

  Este mecanismo era eficaz en situaciones en que era necesario obtener información para producir informes una o dos veces al mes, pero no para el uso diario.

  Los problemas de redundancia en la introducción de datos, y por tanto, de mayor probabilidad de error, así como la reorganización de la información dado que el acceso era secuencial, fueron resueltos parcialmente con la introducción de los archivos de acceso directo y, particularmente, de los archivos secuenciales indexados (Indexed Sequential Access Method -ISAM) que se utilizaron ampliamente en los años 70.

A diferencia de la accesibilidad de datos de manera secuencial, los archivos de acceso directo permiten la recuperación de los registros aleatoriamente, por lo tanto, pueden recuperarse inmediatamente. Los archivos ISAM son los archivos más utilizados en procesos de tipo comercial. Estos archivos permiten que uno o más campos de datos - llamados conjuntamente clave - se utilicen precisamente para indicar qué registro se recuperará.

Esas bases de datos eran demasiado complejas e inflexibles y sólo eran utlizadas por personal muy calificado. Aunque, para escribir los programas de aplicación se utilizaban lenguajes de alto nivel se disponía también de instrucciones y de subrutinas especializadas para tratar las bases de datos que requerían que el programador conociera muchos detalles del diseño físico, y que hacían que la programación fuera muy compleja. Puesto que los programas estaban relacionados con el nivel físico, se debían modificar continuamente cuando se hacían cambios en el diseño y la organización de la base de datos. La preocupación básica era maximizar el rendimiento.

En esta época se llevó a cabo la alianza de IBM y American Airlines para desarrollar SABRE, un sistema operativo que manejaba las reservas de vuelos, transacciones e infomaciones sobre los pasajeros de la compañía American Airlines. [1]

- **Década de 1970.**

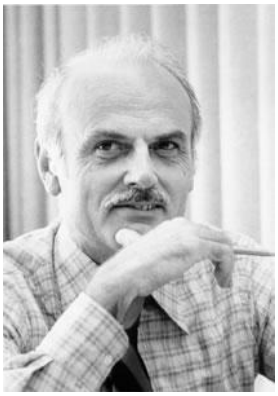


Ilustración 2: Edgar Frank Codd

Edgar Frank Codd, en el artículo "Un modelo relacional de datos para grandes bancos de datos compartidos" ("A Relational Model of Data for Large Shared Data Banks") en 1970, define el modelo relacional y publica una serie de reglas para la evaluación de administradores de sistemas de datos relacionales y, así nacen las bases de datos relacionales. La simplicidad del modelo relacional y *la posibilidad de ocultar completamente los detalles de* implementación al programador fueron realmente atractivas. Codd obtiene posteriormente el prestigioso premio Turing de la ACM (Association of Computing Machinery) por su trabajo. A partir de los aportes de Codd, Larry Ellison desarrolla el Relational Software System, lo que actualmente se conoce como Oracle Corporation. Creando así un sistema de gestión de bases de datos relacional con el mismo nombre que dicha compañía, el cual es un sistema

de administración de base de datos, que destaca por sus transacciones, estabilidad, escalabilidad y multiplataforma. [3]

Cabe destacar, que ORACLE se considera como uno de los sistemas de bases de datos más completos que existen en el mundo, y aunque su dominio en el mercado de servidores empresariales es casi total hasta hace relativamente poco, actualmente sufre la competencia del SQL Server de la compañía Microsoft y de la oferta de otros Sistemas Administradores de Bases de Datos Relacionales con licencia libre como es el caso de PostgreSQL, MySQL o Firebird que aparecen posteriormente en la década de 1990.[2]

Actualmente, según datos entregados en http://db-engines.com a fecha de Abril del 2013, la base de datos más utilizada corresponde a ORACLE, seguida de MySQL y Microsoft SQLServer. [5]

- **Década de 1980.**

El modelo relacional no se usaba inicialmente en la práctica, debido a sus inconvenientes por el rendimiento, las bases de datos relacionales no pudieron competir con el rendimiento de las bases de datos de red y jerárquicas existentes. Esta situación cambió con el System R, un proyecto innovador en IBM Research que desarrolló técnicas para la construcción de un sistema de bases de datos relacionales eficiente. El prototipo de System R completamente funcional condujo al primer producto de bases de datos relacionales de IBM: SQL/DS. Los primeros sistemas de bases de datos relacionales, como DB2 de IBM, Oracle, Ingres y Rdb de DEC, jugaron un importante papel en el desarrollo de técnicas para el procesamiento eficiente de consultas declarativas.

En los principios de la década de 1980 las bases de datos relacionales llegaron a competir con los sistemas de bases de datos jerárquicas y de red incluso en el área de rendimiento. Las bases de datos relacionales fueron tan sencillas de usar, que finalmente reemplazaron a las bases de datos jerárquicas y de red, los programadores que usaban estas bases de datos estaban forzados a tratar muchos detalles de implementación de bajo nivel y tenían que codificar sus consultas de forma procedimental. Aún más importante, debían tener presente el rendimiento durante el diseño de sus programas, lo que implicaba un gran esfuerzo. En cambio, en una base de datos relacional, casi todas estas tareas de bajo nivel se realizan automáticamente por la base de datos, liberando al programador en el nivel lógico.

Desde su escalada en el dominio en la década de 1980, el modelo relacional ha conseguido el reinado supremo entre todos los modelos de datos.
La década de 1980 también fue testigo de una gran investigación en las bases de datos paralelas y distribuidas, así como del trabajo inicial en las bases de datos orientadas a objetos. [3]

- **Década de 1990.**

  Al acabar la década de los 80's, los sistemas de base de datos relacionales ya se utilizaban prácticamente en todas las empresas.

  Para la toma de decisiones se crea el lenguaje SQL (Structured Query Language, estandarizándose posteriormente), que es un lenguaje programado para consultas. El programa de alto nivel SQL es un lenguaje de consulta estructurado que analiza grandes cantidades de información el cual permite especificar diversos tipos de operaciones frente a la misma información, a diferencia de las bases de datos de los 80's que eran diseñadas para las aplicaciones de procesamiento de transacciones, que eran intensivas en actualizaciones.

  SQL comenzó a ser el estándar de la industria, ya que las bases de datos relacionales con su sistema de tablas (compuesta por filas y columnas) pudieron competir con las bases jerárquicas y de red, como consecuencia de que su nivel de programación era sencillo y su nivel de programación era relativamente bajo.

  Los grandes distribuidores de bases de datos incursionaron con la venta de bases de datos orientadas a objetos.

  En esta década surge el crecimiento explosivo de World Wide Web(www). Las bases de datos se implantaron mucho más extensivamente que nunca antes. Los sistemas de bases de datos tienen ahora soporte para tasas de transacciones muy altas, así como muy alta fiabilidad y disponibilidad 24×7 (disponibilidad 24 horas al día y 7 días a la semana, que significa que no hay tiempos de inactividad debidos a actividades de mantenimiento planificadas). Los sistemas de bases de datos también tuvieron interfaces Web a los datos. [1]

- **Año 2000.**

El siglo XXI trajo una nueva tendencia en las bases de datos: el NoSQL. Esta tendencia introduce una línea no relacional significativamente diferente de las clásicas. Por lo general no requieren esquemas fijos, evitan las operaciones *join*[4] almacenando datos desnormalizados y están diseñadas para escalar horizontalmente.

Recientemente ha habido una gran demanda de bases de datos distribuidas[5] con tolerancia a fallos, que de acuerdo con el Teorema CAP(Acrónimo de Consistencia, Disponibilidad y Tolerancia a fallos) establece la imposibilidad de conseguir un sistema distribuido que simultáneamente proporcione consistencia, disponibilidad y tolerancia a fallos.[6]

Un sistema distribuido puede satisfacer sólo dos de las tres restricciones a la vez. Por dicha razón, muchas de las bases de datos NoSQL usan la llamada consistencia eventual para proporcionar disponibilidad y tolerancia al particionado, con un nivel máximo de consistencia de datos. [7]

En los últimos años, los avances en la tecnología han conducido a diversas aplicaciones y sistemas de bases de datos nuevos. La tecnología de los medios de comunicación nuevos hace posible almacenar digitalmente imágenes, clips de audio y flujos de vídeo. Estos tipos de archivos se están convirtiendo en un componente importante de las bases de datos multimedia. Los sistemas de información geográfica (GIS,*Geographic information systems)* pueden almacenar y analizar mapas, datos meteorológicos e imágenes de satélite. Los almacenes de datos y los sistemas de procesamiento analítico en línea (OLAP, *online analytical processing)* se utilizan en muchas compañías para extraer y analizar información útil de bases de datos mucho más grandes para permitir la toma de decisiones. Las tecnologías de tiempo real y bases de datos activas se utilizan para controlar procesos industriales y de fabricación. Y las técnicas de búsqueda en las bases de datos se están aplicando a la WWW para mejorar la búsqueda de la información que los usuarios necesitan para navegar por Internet.

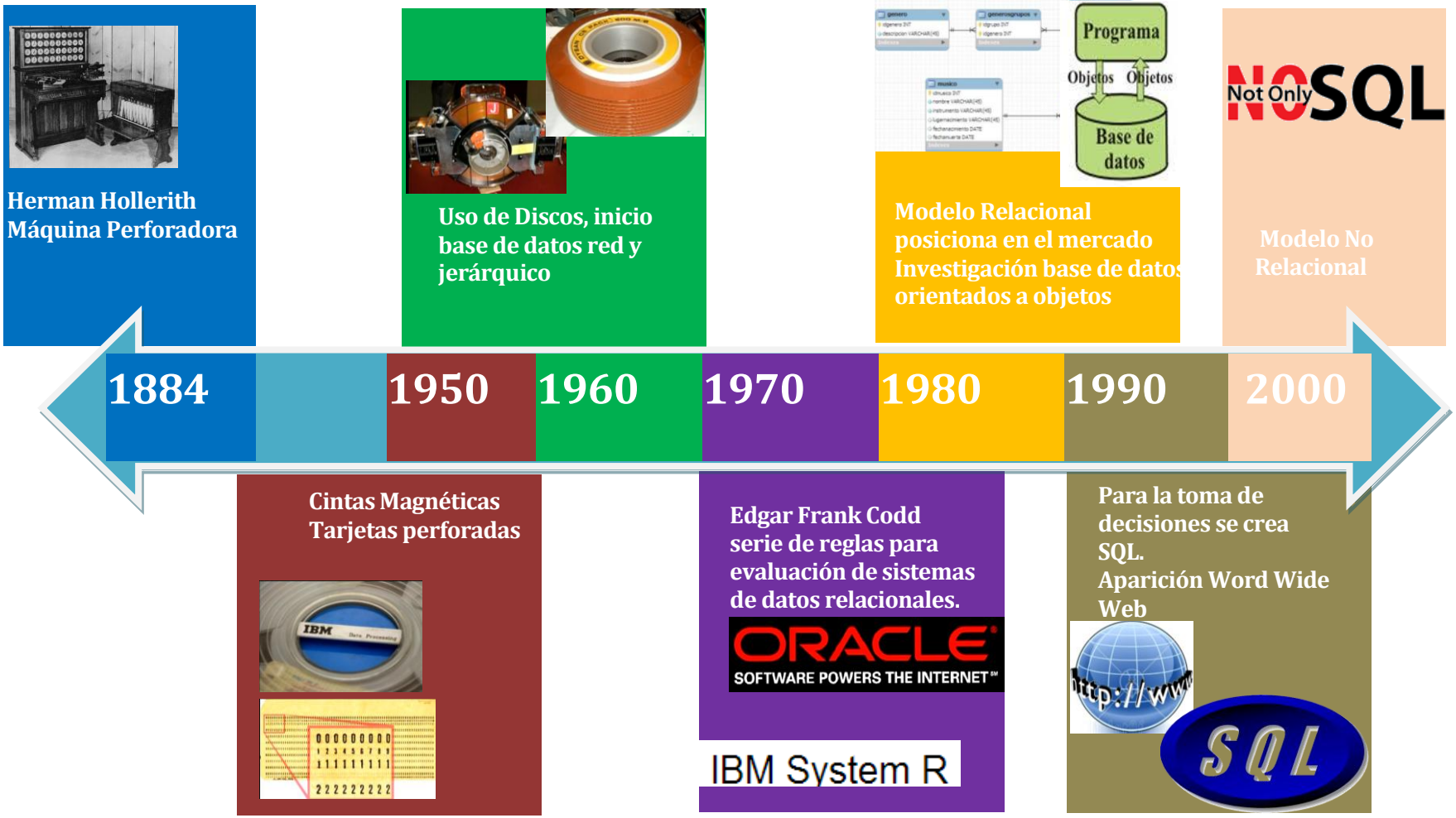
En la Ilustración 3: Línea de Tiempo, se grafica el avance de las bases de datos, de acuerdo a lo comentado anteriormente.

[4] Cláusula del SQL que combina registros de dos o más tablas en una base de datos.(Wikipedia)

[5] Una **base de datos distribuida** (BDD) es un conjunto de múltiples bases de datos lógicamente relacionadas las cuales se encuentran distribuidas en diferentes espacios lógicos (Ej. un servidor corriendo 2 máquinas virtuales) e interconectados por una red de comunicaciones.(Wikipedia)

1.1.1 Línea de Tiempo de Base Datos

Ilustración 3: Línea de Tiempo



Fuente: Elaboración propia a partir de bibliografía utilizada.

1.2 BASE DE DATOS

El concepto de base de datos surge a fines de la década de los 60' como propuesta de solución a un conjunto de problemas técnicos y administrativos presentes en el manejo de archivos. A medida que los sistemas de información se vuelven más complejos y se extienden a nuevas áreas de la operación de una empresa o institución, las dificultades en mantener su funcionamiento y costo bajo control se acentuaban. [3]

Antes de la creación de las bases de datos, una manera de mantener la información en un computador era almacenarlas en archivos administrados por el sistema operativo. Para permitir a los usuarios manipular la información, el sistema mantenía programas de aplicación que manipulaban los archivos.
Estos programas de aplicación se escribían por programadores de sistemas en respuesta a las necesidades de la organización. Si las necesidades se incrementaban, se añadían nuevos programas de aplicación al sistema.
Los registros permanentes eran almacenados en varios archivos y se escribían diferentes programas de aplicación para extraer registros y para añadir registros a los archivos adecuados.
Antes de la llegada de los sistemas de gestión de bases de datos (SGBDs), las organizaciones normalmente almacenaban la información usando tales sistemas.
Mantener información de la organización en un sistema de procesamiento de archivos tiene una serie de inconvenientes importantes.[3]

- **Redundancia:** Este problema surge cuando las aplicaciones se tratan como proyectos separados, ya que se almacenan datos en sectores distintos, pero no hay coordinación que permita descubrir los requerimientos de información comunes en varias aplicaciones, duplicándose los datos en algunos casos. Esto conlleva dos consecuencias, que es el costo de almacenamientos sea elevado por el hecho de almacenar más de una vez los datos y el segundo que es bastante importante y corresponde a la inconsistencia, porque al tratarse como aplicaciones distintas puede ocurrir el caso en que se actualicen los datos de uno y no del otro, por ende, al momento de consultar la base estará inconsistentes. [3][8]

- **Dificultad de acceso a los datos:** La información almacenada en los sistemas de archivos no siempre está asequible fácilmente, ya que depende del diseño de la aplicación donde se gestionan los datos, y puede darse el caso que la información solicitada no se encuentre accesible. Por ejemplo, un resumen de cuenta, utiliza los nombres, dirección y

teléfono del cliente, existe la posibilidad que se desee saber que clientes viven en una determinada área, lo que será difícil de obtener, ya que como esta petición no fue prevista cuando el sistema original fue diseñado y no existe un programa de aplicación que satisfaga lo solicitado, implicará intervención del personal para diseñar la nueva solución. [3][8]

- **Problemas de validez:** Los valores de los datos almacenados en los archivos deben satisfacer restricciones de consistencia, por ejemplo, la edad no puede ser menor a 0. Para esto los desarrolladores realizan validaciones añadiendo el código en los respectivos programas.[3]

- **Problemas de Atomicidad**: Corresponde al caso en que se quisiera insertar valores a una base de datos, y ocurriera un inconveniente que imposibilita la inserción de todos los valores, en este caso, no se debería realizar la acción**.**

- **Problemas de seguridad:** No todos deben tener acceso a la base de datos. Como los programas de aplicación se añaden al sistema de una forma ad hoc, es difícil garantizar tales restricciones de seguridad.[3]

Para solucionar lo mencionado, el enfoque que proponen las base de datos es tomar el conjunto de toda la información que es relevante para una organización, apropiadamente organizada, codificada y colocarla en un gran repositorio llamado *base de datos,* con esto, ningún programa de aplicación tiene acceso directo a los archivos que componen la base de datos, sólo el sistema que gestiona la base de datos. [8]

Por lo tanto, **¿Qué es una base de datos?**

Elmasri, y otros, 2007 y Silberschatz, y otros, 2002 coinciden en que una base de datos es una colección de datos relacionados y tienen las siguientes propiedades implícitas:

- Una base de datos representa algún aspecto del mundo real, lo que en ocasiones se denomina mini-mundo o universo de discurso UOD, (Universe of discourse), los cambios realizados en el UOD, se deben reflejar en la base de datos.
- Es una colección de datos lógicamente coherente con algún tipo de significado inherente
- Se diseña, construye y rellena con datos par aún propósito específico.[3][9]

Unos de los propósitos fundamentales de un sistema de base de datos es proporcionar a los usuarios una visión abstracta, ya que los detalles de cómo se almacenan y mantienen los datos no se encuentran a la vista de las personas que la utilizan.

Con el transcurso del tiempo han surgido varios modelos de Bases de Datos.
Un modelo de datos: una colección de herramientas conceptuales para describir los datos, las relaciones, la semántica y las restricciones de consistencia.[3]
Algunas son:

- Base de Datos jerárquicos
- Base de Datos de red
- Base de Datos transaccionales
- Base de Datos relacionales
- Base de Datos multidimensionales
- Base de Datos orientadas a Objetos

A continuación, se especifica el modelo de datos relacional y multidimensional, ya que ambos son utilizados para la confección y realización de este estudio.

1.2.1 Base de Datos Relacionales

Una base de datos relacional es un conjunto de tablas relacionadas entre sí, que se le asigna un nombre exclusivo. Cada tabla contiene filas, denominadas tuplas y cada fila representa a una relación entre un conjunto de valores. [3].

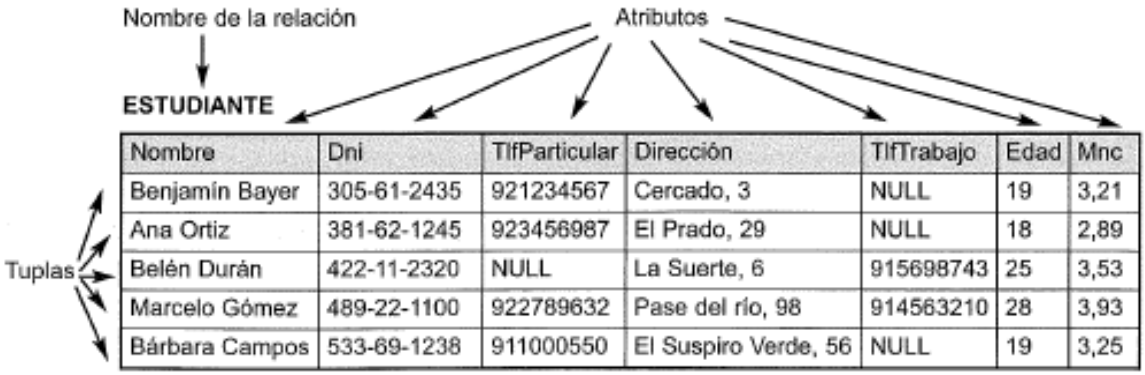
El tipo de dato que describe los valores de una columna se representa por un dominio.

Dominio: Es un conjunto de valores atómicos en el modelo relacional, el tipo de dato y formato. Por ejemplo Nombres Conjunto de caracteres que representa el nombre de una persona con un máximo de 100 caracteres [6].

PACIENTE(Correlativo: entero, Nombre: cadena, Apellido: cadena , Dirección: cadena)

En la Ilustración 4: Ejemplo Atributos y Tuplas, se ejemplifica los conceptos de tuplas y atributos.

Ilustración 4: Ejemplo Atributos y Tuplas



Nombre	Dni	TlfParticular	Dirección	TlfTrabajo	Edad	Mnc
Benjamín Bayer	305-61-2435	921234567	Cercado, 3	NULL	19	3,21
Ana Ortiz	381-62-1245	923456987	El Prado, 29	NULL	18	2,89
Belén Durán	422-11-2320	NULL	La Suerte, 6	915698743	25	3,53
Marcelo Gómez	489-22-1100	922789632	Pase del río, 98	914563210	28	3,93
Bárbara Campos	533-69-1238	911000550	El Suspiro Verde, 56	NULL	19	3,25

Fuente:Ilustración obtenida de Elmasri & Navathe, Base de Datos Fundamentos de Sistemas, Capítulo 5

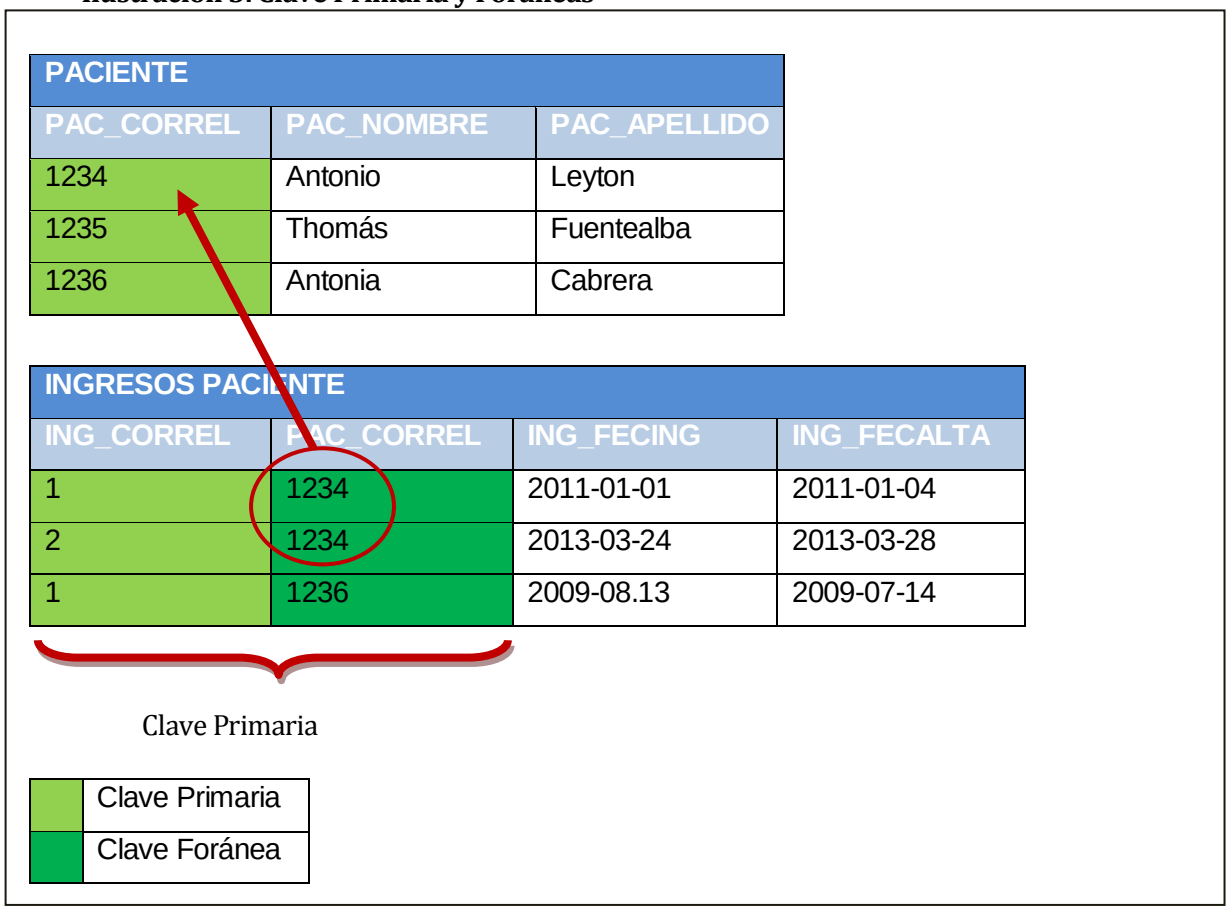
Claves

Clave primaria: Mendelzon, en su libro Introducción a las Bases de Datos relacionales la define como: "Un conjunto de atributos de la entidad tal que no existen dos instancias de la entidad que tengan el mismo conjunto de valores en ese conjunto de atributo" [8]. Una tabla puede estar compuesta por más de una clave primaria, sin embargo, la clave no puede ser repetida en la tabla. Por ejemplo la clave de un paciente puede ser un correlativo que se asigna a un único paciente.

Clave Foránea: Una clave foránea es una referencia a una clave en otra tabla, determina la relación existente en dos tablas. Las claves foráneas no necesitan ser claves primarias en la tabla donde están y sí a donde están referenciadas.

En la Ilustración 5 se ejemplifica el uso de claves primarias y foráneas, en la tabla Pacientes se visualiza el atributo PAC_CORREL como clave primaria, y no existen dos iguales y en la Tabla Ingresos Pacientes, la clave primaria se componen de dos atributos, ING_CORREL y PAC_CORREL, este último corresponde a la clave primaria de Pacientes, sin embargo, en la tabla Ingresos Paciente es también una clave foránea y una componente de clave primaria, se puede ver que la clave 1234 se repite en la segundo tabla, pero en la combinación de ING_CORREL y PAC_CORREL no ocurre.

Ilustración 5: Clave Primaria y Foráneas



Fuente: Elaboración propia

Lenguaje de Consulta

Es un lenguaje utilizado para consultar los valores almacenados en la base de datos.

En este caso se define el lenguaje SQL, bajo su nombre original SEQUEL (Structured English Query Languaje), diseñado en la década de los 70, como parte del proyecto de investigación de Sistema R de IBM, fue uno de los primeros intentos en construir un sistema relacional a gran escala, pero ya en la década de los 90 se consolida el lenguaje SQL.

SQL es un lenguaje de bases de datos global: cuenta con sentencias para definir datos, consultas y actualizaciones.

Por tanto, se comporta como DDL[6] y como DML[7]. Además, dispone de características para definir vistas en la base de datos, especificar temas de seguridad y autorización, definir restricciones de

[6] Lenguaje de descripción de datos(Wikipedia)

[7] Lenguaje de manipulación de datos , lenguajes de programación de bases de datos para recuperar, insertar, borrar y actualizar datos en una base de datos(Wikipedia)

integridad, y especificar controles de transacciones. También tiene reglas para incrustar sentencias de SQL en un lenguaje de programación de propósito general, como Java, COBOL o C/C++.

1.2.2 Modelo Multidimensional

Son bases de datos ideadas para desarrollar aplicaciones muy concretas, como creación de **Cubos OLAP**[8]. Básicamente no se diferencian demasiado de las bases de datos relacionales (una tabla en una base de datos relacional podría serlo también en una base de datos multidimensional), la diferencia está más bien a nivel conceptual, en las bases de datos multidimensionales los campos o atributos de una tabla pueden ser de dos tipos, representan dimensiones de la tabla o métricas que se desean estudiar.

La administración y el manejo de la información es un factor primordial dentro de una empresa. Cada vez es más necesario disponer de herramientas para poder gestionar los datos y transformarlos de tal manera de que sean útiles y sirvan para la toma de decisiones de la organización.
Para cubrir lo anterior se debe disponer de repositorios de calidad, con datos de confianza en un formato entendible y reutilizable.

A continuación se especifican las características y puntos relevantes de ambos temas:

1.2.3 Data Warehouse (DW):

Es un gran almacén de datos que está estructurado para analizar la información a diferente nivel de detalle de todos los procesos de negocios que tiene la organización.

Es la Base de Datos llamada estratégica o multidimensional. Una vez diseñadas es poblada mediante el ETL (Extract Transform and Load) a partir de las Bases de Datos operacionales. Su diseño va orientado a recopilar toda la información de la empresa en un único modelo de negocio que de soporte a las necesidades de información en la organización [13].

Este almacén**,** es un gran repositorio de datos que apoya la toma de decisiones dentro de una organización, manteniendo datos resumidos, consolidados e históricos.

[8] **OLAP** es el acrónimo en inglés de **procesamiento analítico en línea** (*On-Line Analytical Processing*). Es una solución utilizada en el campo de la llamada Inteligencia empresarial (*Business Intelligence*) cuyo objetivo es agilizar la consulta de grandes cantidades de datos(Wikipedia)

Sus fuentes de datos son múltiples y variadas, las cuales terminan complementándose en una base de datos amplia, homogénea y fácilmente manipulable.

Su utilidad principal es convertirse en un acceso universal de datos para consultas, análisis (Data Mining[9]) y generación de informes digitales con datos consolidados.

1.2.3.1 Características

Se pueden diferenciar cuatro características importantes:

- Orientado a tema
- Integrado
- De tiempo variante
- No Volátil

1.2.3.1.1 Orientado a temas

Esta característica corresponde a que la información se clasifica en base a los aspectos que son de interés de la empresa.
En el DW se excluye la información que no será utilizada por el sistema de soporte de decisiones. También es importante mencionar que en este modelo los datos se miden en un espectro de tiempo y las relaciones que se pueden encontrar son bastantes. [11]

1.2.3.1.2 Integrado

Esta es una de las características más importante, la información de un DW debe estar integrada, lo que significa que los datos que procesa el DW deben mantenerse todos con el mismo formato, independiente que su origen sean fuentes diferentes. Por lo tanto, deben resolver problemas como conflictos de nombres e inconsistencias entre las diferentes unidades de medidas. [14]

La Ilustración 6: Integrar datos antes de ingresar DW. [15] muestra lo descrito anteriormente.

Por ejemplo, en el caso del Sexo de un paciente, en algunas fuentes de datos puede estar presente como F o M, en otras Femenino o Masculino, o bien Hombre o Mujer, pero lo que interesa es que al almacén las tres lleguen con el mismo formato.

[9] La **minería de datos** es un campo de las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos.(Wikipedia)

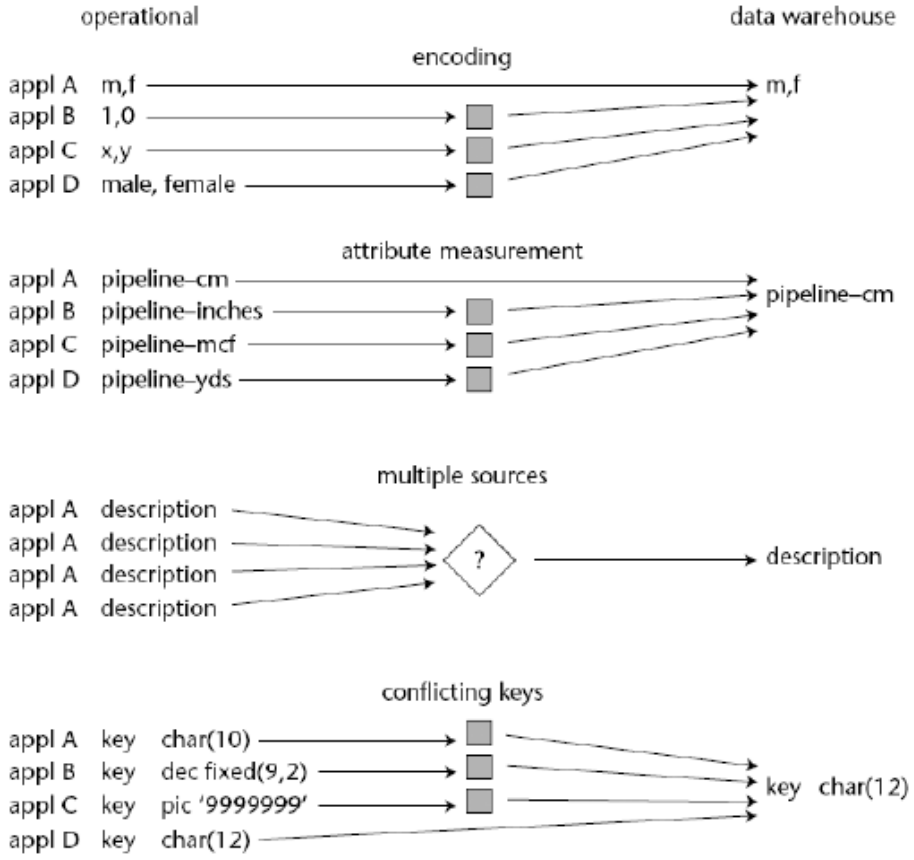


Ilustración 6: Integrar datos antes de ingresar DW. [15]

1.2.3.1.3 De tiempo variante

La estructura de un DW contiene un elemento de tiempo (dimensión).

Los datos históricos son de poco uso en el procesamiento operacional. Sin embargo, en un DW son relevantes, debido a que se incluyen para reflejar variaciones y evaluar tendencias en relación al tiempo.

El tiempo variante se presenta de varias maneras :(Ver Ilustración 7: Variante de Tiempo)

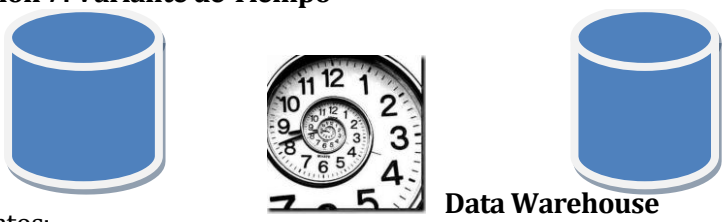
- **Datos con horizonte de tiempo:** En el ambiente operacional es bastante más corto, con un rango desde los 60 a 90 días. A diferencia de DW, que su rango fluctúa entre 5-10 años.

- **Estructura de claves:** En DW cada elemento clave debe contener por obligación un elemento de tiempo (día, mes, entre otros), ya que el análisis de los datos se realizan respecto a periodo de tiempo. En cambio las BD operacionales la clave puede o no tener asociado un elemento tiempo. Este punto se hace referencia a que los datos ocurren en un periodo de tiempo en particular, no al hecho que se puedan almacenar el tiempo en que se manipuló el registro.

  Por ejemplo, si un paciente ingresa a la clínica el 2013-02-02, pero estos datos fueron ingresados al DW el 2013-02-03, si bien este último es importante para saber el tiempo de actualización, no necesariamente se ingresará como un atributo de la tabla, ya que el periodo de tiempo que importa corresponde al ingreso realizado por el paciente a la clínica.

- **Actualización datos:** Una vez registrada la información en DW, no puede actualizarse. En una DW la información es una serie de snapshots(vistas simultaneas). A diferencia de la información de las BD operacionales, puede ser actualizada cuando se desee.[11]

Ilustración 7: Variante de Tiempo



Operacional
Valor Actual datos:
Horizonte tiempo: 60-90 días
Clave puede o no tener un elemento de tiempo.
Los datos pueden ser actualizados

Data Warehouse
Datos instantáneos:
Horizonte tiempo: 5-10 años
Clave contiene un elemento de tiempo.
Una vez que el snapshot se realice, no se puede actualizar el registro

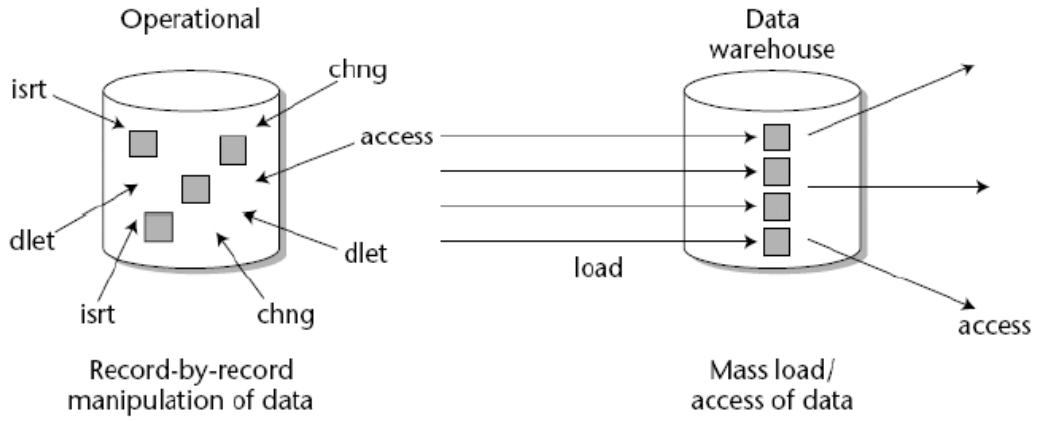
Fuente: Elaboración propia según bibliografía consultada

1.2.3.1.4 No volátil

Esta característica del DW significa que una vez que los datos ingresan al almacén, no se deben cambiar, ya que el propósito es permitir el análisis de los datos extraídos en un periodo de tiempo determinado. [14]

En la Ilustración 8: Actualizaciones BD operacional y DW [15] se muestra que en el ambiente operacional las actualizaciones se realizan regularmente. Mientras que el DW existen dos tipos de operaciones, carga inicial y el acceso a ellos mismos.

Ilustración 8: Actualizaciones BD operacional y DW [15]



En la Tabla 1 se realiza un resumen comparativo entre las bases de datos operacionales y la de un Data Warehouses

Tabla 1: Comparación entre BD Operacional y DW

Base de Datos Operacional	Data Warehouse
Orientado información operativa	Orientado a la información estratégica
Orientado a Aplicación	Orientado a la toma de decisiones
Actual	Actual + Histórico
Detallada	Resumida y consolidados
Cambia continuamente	Estable
Transacciones pequeñas	Transacciones complejas con grandes cantidades de datos.

Fuente: Elaboración propia a partir de bibliografías consultadas.

1.2.3.2 Objetivos

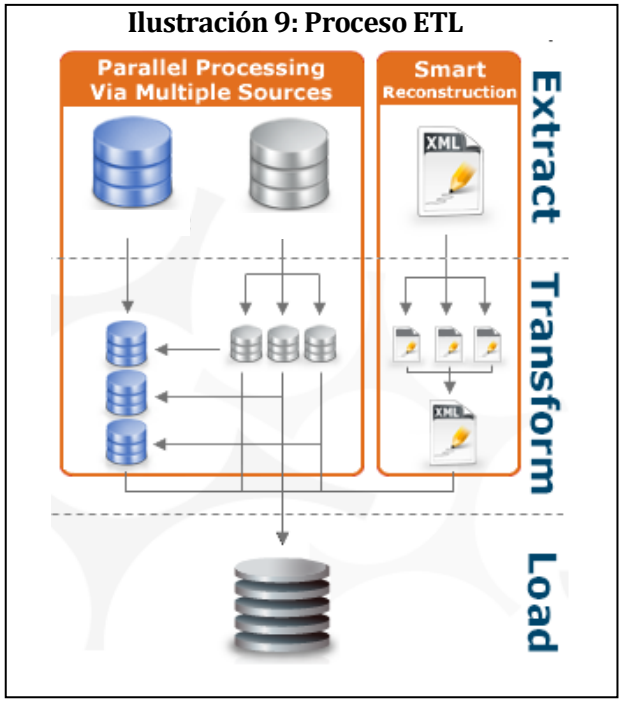
Los DW están enfocados para ser utilizados en el área estratégica dentro de las empresas, por lo tanto, se puede señalar los siguientes objetivos

- Lograr que la información sea fácilmente accesible.
- Presentar la información de la organización de manera consistente.
- Adaptarse a los cambios en el negocio, necesidades de los usuarios, etc.
- Proveer seguridad de los datos.
- Ser la base para mejorar la toma de decisiones. [15]

Como el DW es un repositorio que almacena datos de varias fuentes, es necesario realizar un proceso de Extracción, Transformación y Carga (ETL), para lograr obtener datos limpios y estandarizados. (Ver Ilustración 9: Proceso ETL)

- **Extraer** la de información de los distintos repositorios iniciales.
- **Transformar** la información de acuerdo a los estándares de la organización.
- **Cargar** la información de las bases de datos operaciones hacia las base de datos multidimensionales. [16]

Ilustración 9: Proceso ETL



Fuente: Ilustración modificada y obtenida desde http://www.probandocodigo.com/2008/11/quees-un-etl.html

1.2.4 Data Marts

Un Data Marta es un subconjunto de datos derivado del Data Warehouse. Está diseñado para soportar requerimientos analíticos específicos de una determinada unidad de negocios. Es un repositorio menos ambicioso que un DW, pero que cumple con las mismas características y necesita de los mismos procesos para su construcción. [16]

En algunas circunstancias se comienza a confeccionar primero los Data Marts y luego el Data Warehouse, o viceversa, todo depende de las necesidades de la organización.

El Data Marts tiene las mismas características de un DW, pero al enfocarse en una sola área de la empresa, se puede destacar algunas ventajas:

- Más simples de implementar que un Data Warehouse.
- Pequeños conjuntos de datos, por ende, menos recursos.
- Encuentra con mayor rapidez las necesidades de la Unidad del Negocio.
- Consultas más rápidas por menor volumen de datos. [17]

1.3 BASE DE DATOS NO RELACIONALES (NONSQL)

NoSQL es, literalmente, una combinación de dos palabras: No y SQL.

NoSQL es una tecnología o producto que contrarresta SQL. Los creadores y primeros usuarios del término NoSQL probablemente querían decir No RDBMS o No relacional, sin embargo, el término NoSQL fue el adoptado.

Tiempo después, algunos han propuesto NonRel como una alternativa a NoSQL. Unos pocos han tratado de rescatar el término original, proponiendo que NoSQL es en realidad un acrónimo que se expande a "Not Only SQL"(No solo SQL). [18]

Sea cual sea el significado literal, NoSQL se utiliza hoy en día como un término general para las bases de datos y almacenes de datos que no siguen los populares y bien establecidos principios de RDBMS[10] y, con frecuencia se relacionan con grandes conjuntos de datos accedidos y manipulado en una escala Web.

Esto significa que NoSQL, no es un producto único o una sola tecnología, representa una clase de productos y una colección de diversos conceptos sobre el almacenamiento y manipulación de datos. [18]

NoSQL es uno modelo de datos que difiere al relacional, y consta de un modelo libre de esquemas, donde el principal enfoque es almacenar y recuperar grandes cantidades de datos, y no en las relaciones en sí.

Carlo Strozzi usó el término NoSQL en 1998 para referirse a su base de datos. Era una base de datos open-source, ligera, que no ofrecía un interface SQL, pero sí seguía el modelo relacional. [19]

Strozzi sugiere que, ya que el actual movimiento NoSQL Se sale completamente del modelo relacional, debería, por tanto, haberse llamado 'NoREL', o algo así.[20]

[10] RDBMS Acrónimo de Relational Database Manager System o Sistema de Gestión de Base de Datos Relacionales[21]

El NOSQL fue introducido a principios del 2009 por un empleado de Rackspace, Eric Evans, cuando Johan Oskarsson, miembro de Last.fm[11] propuso organizar un evento para discutir sobre bases de datos open-source distribuidas. [23]

1.3.1 Características

Las NOSQL abarcan una amplia variedad de tecnologías de diferente base de datos, sin embargo, tienen características en común, las cuales se describen a continuación

1.3.1.1 Libres de Esquema o Dinámico

Existen bases de datos NoSQL que son libres de esquemas, es decir, no necesitan la creación de un modelo de datos rígido en el cual almacenar los registros, otorgando flexibilidad en la manipulación y almacenamiento de datos de diferente estructura. Esto se contrasta con las bases de datos relacionales en las cuales es obligación el uso de esquemas y no es posible insertar datos en ellas sin que se cuente con los nombres, tipos y tamaños de las columnas en sus respectivas tablas.

Por otro lado, existen bases de datos que dejan ésta elección al usuario, permitiendo el uso o no de esquema.

1.3.1.2 Escalabilidad

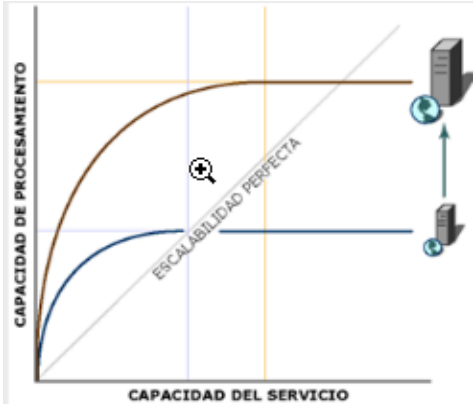
Michael Nygard, en su libro Release It! Design and Deploy Production-Ready Software, 2007, define la escalabilidad como la posibilidad de añadir recursos físicos a un sistema de computación, los denominados nodos, con el fin de obtener un mejor rendimiento.

La Escalabilidad puede de ser de dos tipos vertical y horizontal.

- Escalabilidad vertical corresponde a la adición de los recursos de una sola computadora. Puede ser que sea más memoria, un procesador más rápido o los discos más rápidos y más grandes. Este tipo de escalabilidad es la utilizada en los modelos relaciones. Ver Ilustración 10: Escalabilidad Vertical

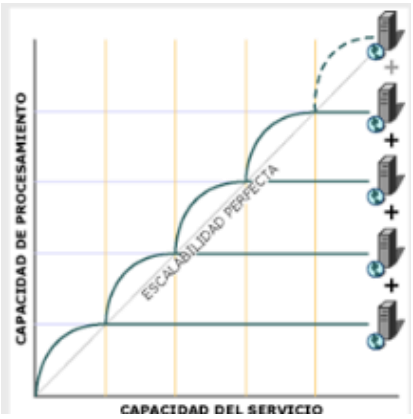
[11] Last.fm es un servicio de recomendaciones musicales [22]

Ilustración 10: Escalabilidad Vertical



- Escalabilidad Horizontal: A diferencia de la escalabilidad vertical, la horizontal consiste en añadir recursos físicos a una red de computadores través de la adición de más ordenadores. En el caso ideal, los nuevos ordenadores proporcionan un aumento lineal en el desempeño. Escalabilidad utilizada en NOSql. [24]

Ilustración 11: Escalabilidad Horizontal



1.3.1.3 Disponibilidad

Las bases de datos relacionales tienen la propiedad de estar siempre en un estado coherente. Esto significa no realizar nuevas operaciones de escritura hasta que la operación de escritura actual esté acabada.

Gilbert, y otros, (Jun. 2002) definen la disponibilidad como: cada petición recibida por un nodo sin fallas da lugar a una respuesta. Un nodo es un computador que forma parte de un sistema construido de varios computadores. Vale la pena señalar que esta definición de disponibilidad sólo se aplica a los nodos sin fallas y no hace ninguna limitación en cuánto tiempo puede transcurrir entre la solicitud y la respuesta. Generalmente los requerimientos de disponibilidad de los sistemas de software se establecen mediante la negociación de un acuerdo de nivel de servicio. [25]

1.3.1.4 Tolerancia a Fallos

Consiste en que a la ocurrencia de algún tipo de falla no afecte el sistema entero ni a los demás nodos, sino que sólo al nodo en que ocurre, y de esta manera seguir con las operaciones normales aún bajo circunstancias anormales.

Las bases de datos relacionales tratan los fallos del hardware como las excepciones y se requiere hardware especial para lograr tolerancia a fallos. La tolerancia a fallos a través de la replicación de datos no es una funcionalidad nativa de las bases de datos relacionales, teniendo que ser provista por software externo que añada esta característica.

Si bien la definición sólo toma en cuenta los fallos de hardware, un sistema fiable debe estar preparado también para fallos de software. Si ocurre un fallo en el hardware, el nodo, por lo general no responde, y ese es el caso fácil. Sin embargo, los fallos de software hacen que un nodo se comporte erróneamente pero sigue retornando valores, por lo cual, es más difícil de detectar su falla[26].

1.3.2 Clasificación

Existen distintas bases de datos en el marco del NoSQL, pero no todas trabajan de la misma manera, por lo cual, se clasifican con respecto a la implementación de ellas. A continuación se describen las utilizadas con mayor frecuencia y las que se encuentran descritas en la literatura.

1.3.2.1 Documentales

Los almacenes de documentos guardan la información como un listado de documentos desestructurados. 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.

Utiliza formato JSON o XML. Útil para aplicaciones web complejas y para todo tipo de aplicación que utiliza documentos y necesita flexibilidad. [28]

1.3.2.2 Almacenes de Claves o Clave –Valor

Los almacenamientos clave-valor, o *key-value*, son uno de los tipos más simples de base de datos no relacional. Los caches de memoria típicamente utilizan el esquema *key-value* para guardar la información y permitir el rápido acceso. Los almacenamientos *key-value* asocian una clave única (*key*) al valor que se quiere guardar (*value*).
Varias implementaciones de los almacenamientos *key-value* tienen funcionalidad adicional, pero a un nivel básico, el *key-value* solo requiere una clave y un valor. Este tipo de base de datos suele ser extremadamente rápido y optimizado para una gran cantidad de accesos. [18]

1.3.2.3 Orientados a Columnas

Las bases de datos columnares suelen estar optimizados para guardar grandes cantidades de datos, para consultas de agregación y reporte. Estas bases de datos suelen ser muy rápidas en consultas de agregación de datos o data mining, sin embargo, no suelen ser usadas en entornos on-line donde la latencia y tiempo de respuesta de las consultas suele ser crítico. [29]

1.3.2.4 Grafos

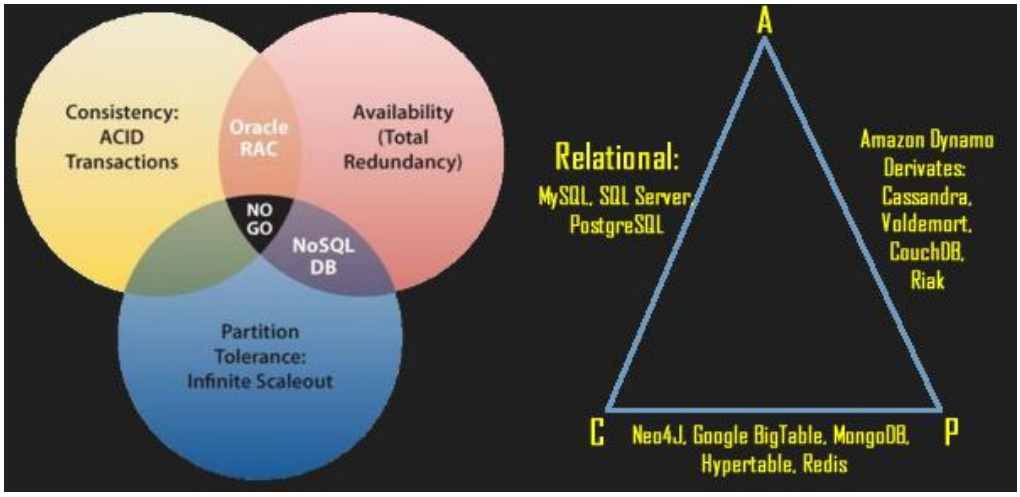
Las bases de datos de grafo organizan la información en grafos dirigidos. Están optimizadas para hacer operaciones de consulta de relaciones entre miembros, y para esta tarea en específico son extremadamente rápidas. [30]

1.3.3 Limitación de Atomicidad y transacciones de integridad.

En esta sección se habla del teorema CAP (Consistencia, Disponibilidad y Tolerancia a fallos), también denominado Teorema de Brewer que establece que para un sistema de cómputo distribuido, puede garantizar solo 2 operaciones simultáneamente.

- **Consistency (Consistencia):** Cada cliente puede ver la misma cantidad o vista de los datos
- **Availability (Disponibilidad):** Cada cliente puede leer y escribir
- **Partition tolerance (Tolerancia a Fallos):** Sistema siga funcionando, a pesar de haber fallos en algunas partes de él. [18]

Ilustración 12: Teorema CAP



Fuente: Ilustración obtenida de http://www.researchbeta.com/

En la Ilustración 12: Teorema CAP, se visualiza el teorema CAP y las bases de datos que se enmarcan en la definición.

Con las características mencionadas anteriormente, en la Tabla 2 se expone una comparativa entre las Base de datos tradicionales con las NoSQL.

Tabla 2: Comparativa NOSQL/BD relacional

Base de datos	Relacional	NoSQL
Modelo de almacenamiento datos	Almacenamiento como filas y tablas.	Varía dependiendo del tipo de Base de datos NoSQL. Clave-Valor: Similar a la relacional, pero tiene solo dos columnas ("clave" y "valor"). Documentos: Almacena todos los datos juntos en un documento que pueden ser anidados jerárquicamente, en JSON,XML u otro formato,
Esquemas	Tipo estructura de datos, se fijan por adelantado. Si se desea agregar otro atributo, se realiza mientras la base de datos está fuera de línea.	Normalmente dinámico. Atributos se añaden sin necesidad de bajar el servicio.
Escalado	Solo verticalmente, o sea un servidor más poderoso. Es posible la transmisión sobre muchos servidores, pero generalmente significa ingeniería adicional	Horizontalmente, para aumentar la capacidad se deben añadir más nodos al servidor, y la base de datos se propaga automáticamente. Verticalmente.
Consistencia	Configurable para consistencia fuerte	Depende del producto. Algunos proporcionan consistencia fuerte y otros eventual
Tolerancia a fallos	Bajo. Fallo en el nodo y generalmente hará fallar la consulta	Alta. Configurados para que la pérdida de algunos nodos no interrumpa funcionamiento global
Ejemplos	MySQL[12], PostgreSQL[13], Oracle Database[14], SQLServer[15]	MongoDB[16], HBase[17], Cassandra[18], Neoj[19]

Fuente: Elaboración propia a partir de bibliografía consultada

[12] http://www.mysql.com/
[13] http://www.postgresql.org/
[14] http://www.oracle.com
[15] http://www.microsoft.com
[16] http://docs.mongodb.org
[17] http://hbase.apache.org/
[18] http://cassandra.apache.org/
[19] http://www.neo4j.org

En conclusión, las Bases de Datos no relacionales son una tecnología de gran relevancia para nuestros tiempos, las redes sociales y las necesidades empresariales requieren de sistemas que sean capaces de administrar grandes cantidad de datos. Sin embargo, no se puede decir que estas bases de datos NOSql son mejores que las BD relacionales, ya que cada una está destinada a distintos segmentos y usos. Por la tanto, es de gran relevancia al momento de implementar algunas de las alternativas evaluar con gran cuidado la aplicación a realizar, ya que la elección incorrecta provocaría que la base de datos no respondiera correctamente a lo solicitado.

1.4 APACHE CASSANDRA

Esta sección cubre los aspectos importantes de la base de datos NoSQL Cassandra y el porqué se opta por esta base de datos no relacional para el estudio.

Se definen sus principales características, modelo de datos, componentes, arquitectura y almacenamiento de los datos.

Se presenta una analogía de las base de datos NonSQL con las bases de datos relacionales con respecto a sus componentes, se trata el tema de cómo Cassandra, siendo un modelo flexible en esquema, que no cumple con las propiedades del ACID[20], aborda el tema de la consistencia e integridad de los datos.

Finalmente se presenta la interfaz proporcionada al cliente y su sintaxis con similitudes al SQL estándar, el denominado CQL.

1.5 ¿Qué es Cassandra?

Ilustración 13: Apache-Cassandra

"Apache Cassandra es un sistema gestor de base de datos distribuidas, de código abierto y está diseñado para manipular grandes cantidades de datos a través de varios servidores, proporcionando un servicio de alta disponibilidad sin puntos únicos de fallo. Este sistema consta de almacenamiento de datos distribuido que se diferencia de los sistemas de base de datos relaciones. [32]

Cassandra se inició primero como un proyecto de incubación de Apache en Enero del 2009. Tiempo después los desarrolladores liderados por Jonathan Ellis, director del proyecto Apache Cassandra, lanzaron la versión 0.3 de Cassandra y han seguido haciendo lanzamientos desde esa época." [31]

La última versión estable lanzada es Apache Cassandra 1.2.4 lanzada el 11 Abril 2013.

Según Adrian Garcete:

> "Apache Cassandra es una base de datos no relacional distribuida y basada en un modelo de almacenamiento de clave valor escrita en Java" [32]

[20] **ACID** es un acrónimo de **A**tomicity, **C**onsistency, **I**solation and **D**urability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.(Wikipedia)

Sin embargo, Eben Hewitt, en su libro Cassandra The Definitive Guide menciona que: Cassandra se denomina con frecuencia como una base de datos orientados a columnas. [31]

En ambos casos existe una clasificación diferente al tipo de base de datos. Sin embargo, no se puede discutir que corresponde una base de datos no relacional, ya que no mantiene una estructura ni un modelo relacional fijo.
Cada fila contiene columnas, pero no necesariamente todas mantienen la misma cantidad de columnas.
Cada fila es identificada por una clave única, que hace que sus datos sean accesibles y únicos.

Eben Hewitt, en su libro Cassandra The Definitive Guide menciona que Cassandra almacena los datos en lo que se puede considerar como una tabla hash multidimensional, lo que significa que no es necesario decidir exactamente la estructura de la base de datos, sino que puede irse modificando con el transcurso del tiempo.[31]

1.6 ¿Por qué Cassandra?

Se opta por comparar Cassandra con Sql Server, porque para la realización de este estudio se tiene a disposición sólo un computador, y esta base de datos NoSQL se puede utilizar en un solo nodo, como en varios, a diferencia de otras base de datos no relacionales que necesitan un computador para ser ejecutado como un nodo esclavo donde se ejecutan las consultas, y otro como un nodo maestro que corresponde a un servidor que almacena las configuraciones.
Otro criterio para la elección de esta base de datos fue el hecho de que se debe realizar la comparativa de los tiempos de respuestas de un modelo de DataMarts, el cual está destinado para el análisis de datos. Por lo tanto, es importante que la base de datos tenga soporte para el análisis de la información, y en este caso existen framework que trabajan sobre Cassandra, como lo es Pentaho[21] y DataStax[22]. Por lo tanto, se tiene soporte por si llegaran a ser positivos los datos arrojados.

A continuación se define las características y el modelo de Apache Cassandra.

[21] http://www.pentaho.com/big-data/
[22] http://www.datastax.com/

1.6.1 Modelo de Datos

El modelo de datos de Cassandra es un esquema dinámico, orientado a columnas. Esto significa que a diferencia del modelo relacional no es necesario modelar todas las columnas requeridas en la aplicación, ya que cada fila no requiere tener la misma cantidad de columnas.

El modelo de datos Cassandra está diseñado para datos distribuidos en una escala muy grande. Aunque es natural querer comparar el modelo de datos de Cassandra con una base de datos relacional, es un error, ya que en realidad son muy diferentes.
En una base de datos relacional, los datos se almacenan en tablas, las que se relacionan con otras tablas.
Los datos se normalizan para reducir la redundancia de datos, y las tablas se unen en claves comunes para satisfacer una consulta determinada. [33]

En Cassandra, un *espacio de claves* o keyspace es el contenedor de datos de la aplicación, de forma similar a una base de datos o un esquema de una base de datos relacional. En el interior del espacio de claves existen uno o más grupos de *familia de columna*, que son análogas a las tablas. Las Familias de Columnas contienen *columnas*, y un conjunto de columnas relacionadas se identifica por una fila suministrada por la aplicación *clave*.

Cassandra no contiene claves foráneas, y no mantiene relaciones con demás familias de columnas. Cada familia tiene un conjunto de columnas autónomas que están destinadas a ser visitada en conjunto para satisfacer consultas específicas. De hecho, una buena forma de modelar la base de datos es diseñar una familia de columnas por consultas a responder, ya que las familias de columna pueden optimizar el rendimiento de lectura.

1.6.1.1 Keyspaces

Es un contenedor de datos, de forma similar a un esquema en una base de datos relacional. Keyspace se utilizan para agrupar familias de columnas.
Al definir un Keyspace se pueden configurar los siguientes parámetros:

- Partitioner: Determina como se distribuyen los datos a través de los nodos del clúster.
- Replication factor: Establece el número de nodos que actuarán como copias de un conjunto de datos, o sea la cantidad de veces que estarán repetidos los datos en el clúster.
- Placement strategy: Parámetro que establece el modo de replicación de los datos en el cluster.

1.6.1.2 Columnas

1.6.1.2.1 Standars Columns:

La columna es el nivel más pequeño de los datos en Cassandra. Consta de 3 elementos: column name, value y timestamp(marca de tiempo).

Column name: es el indentificador con el que se puede acceder para obtener o modificar el valor que contiene. Es único y no pueden existir dos iguales en el mismo conjunto de columnas.

Una columna debe tener un nombre, el cual puede ser una etiqueta estática (por ejemplo, "nombre" o "e-mail",) o puede ser configurada de forma dinámica (creada y asignada en el momento de ejecución cuándo se requiera).

Value: Corresponde al dato de la columna. Es el único elemento modificable por el usuario. Se puede definir la validación del tipo de dato que contiene.

Timestamp: Cassandra usa timestamp de columna para determinar la actualización más reciente de una columna. La timestamp es proporcionada por la aplicación cliente. La última marca de tiempo siempre es la primera retornada al solicitar los datos, por lo que si varias sesiones de clientes actualizan las mismas columnas en una fila al mismo tiempo, la actualización más reciente es el que finalmente persisten.

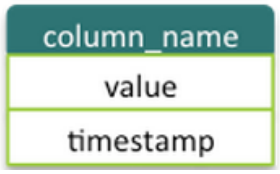


Ilustración 14: Standard Columns [33]

1.6.1.2.2 Composite Columns

O también denominadas como columnas compuestas, es utilizada en Cassandra para almacenar filas agrupadas. Todas las filas lógicas con la misma clave de partición se almacenan en forma de una sola fila, ancho físico. Con este diseño, Cassandra soporta hasta 2 mil millones filas por columnas.

Las columnas compuestas comprenden plenamente desnormalizar las tablas utilizando claves primarias compuestas.

1.6.1.2.3 Counter Columns

Un contador es un tipo especial de columna que se utiliza para almacenar un número incremental que cuenta las ocurrencias de un evento o proceso en particular. Por ejemplo, es posible utilizar una columna de contador para contar el número de veces que se visita una página.

Un contador de familias de columna debe utilizar CounterColumnType como el validador (el tipo de valor de la columna).

Un contador columnas es diferente de las columnas estándar, ya que un contador está definido en la aplicación utilizada, y el cliente actualiza el valor de la columna mediante el incremento o decremento de ella. Una actualización de cliente a una columna de contador pasa el nombre del contador y el valor del incremento o decremento, sin requerir marca de tiempo.

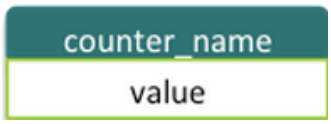


Ilustración 15: Counter Columns[33]

1.6.1.3 Row key

Corresponde a un identificador único de cada fila en una familia de columnas. Es similar a la clave primaria de una tabla relacional

A diferencia del modelo relacional, dos rows de un mismo conjunto pueden tener diferentes números de columnas y no es necesario tener columnas vacías.

1.6.1.4 Column Family

Una familia de columnas es similar a una tabla en el modelo relacional.

Las familias de columnas pueden y debe definir los metadatos de las columnas, pero las columnas reales que forman una fila se determinan por la aplicación del cliente.

Cada fila se identifica por una fila única row key(similar a la primary key en la base de datos relacional) que es implícitamente indexada.

Cada fila puede tener un conjunto diferente de columnas. Existen dos tipos:

1.6.1.4.1 Column Family static:

Una familia de columna estática utiliza un conjunto estática de columnas y es similar a una tabla de base de datos relacional. Por ejemplo, una columna que guarda los datos de familias de usuario podría tener columnas para el nombre de usuario, dirección, correo electrónico, número de teléfono y así sucesivamente. Aunque las filas tienen generalmente el mismo conjunto de columnas, no se requiere que tengan todas las columnas definidas. (Ver Ilustración 16: Column Family Static [33])

Ilustración 16: Column Family Static [33]

1.6.1.4.2 Column Family Dinamic:

Una familia columna dinámica aprovecha las características de Cassandra y la capacidad de generar columnas mientras se vayan requiriendo, evitando así crear columnas que no se utilizarán. Una familia columna dinámica le permite pre-calcular conjuntos de resultados y guardarlos en una sola fila para la recuperación de datos eficiente. Cada fila de datos está destinada a satisfacer una determinada consulta, algo así como una vista materializada. Por ejemplo, una familia de columnas que rastrea a los usuarios que se suscriben al blog de un usuario en particular es dinámica.

En lugar de definir los metadatos de cada columna, una columna dinámica familiar define la información de tipo de nombres de columna y valores (comparadores y validadores), pero los nombres de columna y los valores reales son fijados por la aplicación cuando una columna se inserta. (Ver Ilustración 17**: Column Family Dinamic [33]**)

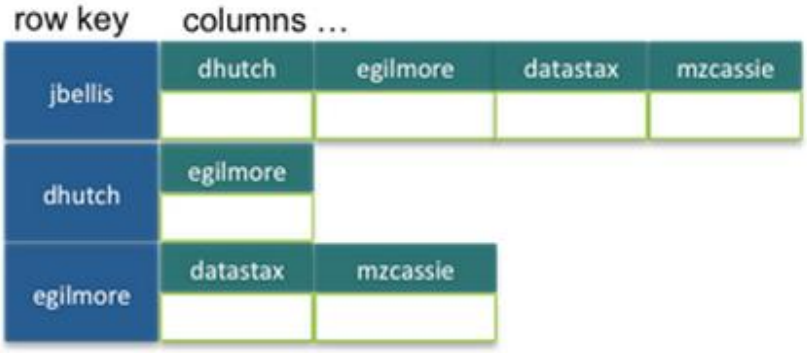


Ilustración 17: Column Family Dinamic [33]

1.6.1.5 Índices

Un índice es una estructura de datos que permite la búsqueda rápida y eficiente de una consulta con una determinada condición. Existen dos tipos

1.6.1.5.1 Índices primarios:

En el diseño de bases de datos relacionales, una clave primaria es la clave única para identificar cada fila de una tabla. Un índice de clave principal, al igual que cualquier índice, acelera el acceso aleatorio a los datos de la tabla. La clave primaria también garantiza la exclusividad de registro, y también puede controlar el orden en que los registros se agrupan físicamente, o almacenados por la base de datos.

En Cassandra, el índice principal para una familia de columna es el índice de cada row key. Cada nodo mantiene este índice para los datos que maneja.

1.6.1.5.2 Índices secundarios:

Estos índices hacen referencia a los valores de columna. Se implementó estos índices como una tabla oculta, separada de las tablas que contienen valores de indexación.

La creación de estos índices será necesaria en aquellas filas donde sus valores no varían en gran cantidad, no así para los campos que almacenan datos únicos. Por ejemplo, en una tabla paciente listar todos los del sexo masculino, con una consulta de este estilo arrojaría

gran volumen de datos, a diferencia que si se quisiere filtrar por el correo electrónico, esta última devolvería una fila, ya que este típicamente es único para cada usuario.

Una ventaja de los índices secundarios es la facilidad operativa de crear y mantener el índice. Los índices secundarios se construyen de forma automática, sin necesidad de detener el servicio.

Para entender de mejor manera y asimilar la migración de lo relacional a no relacional, en la Tabla 3 se señalan las "similitudes" en ambas Base de datos.

Tabla 3: Analogía Cassandra

Cassandra	Base Datos Relacional
Keyspace	Base de Datos
Column Family	Tablas de Datos
Row Key	Primary key
Column	Atributo de una tabla

Fuente: Elaboración propia, a partir de bibliografía consultada.

1.6.2 Consistencia

Cassandra es eventualmente consistente en las escrituras y lecturas de los registros, lo que significa que se puede configurar el número de copias que se desea tener de los datos. Por lo tanto, según la configuración la base de datos puede contener datos replicados que no sean iguales, logrando que una consulta pueda arrojar distintos resultados.
Existen dos niveles de consistencia, las de escritura y lectura.

1.6.2.1 Consistencia Escritura

Al realizar una escritura en Cassandra, antes de devolver una confirmación de recibo, se debe comprobar que se haya logrado el nivel de consistencia especificado. Los niveles de consistencia son:

- **ANY:** Asegura que la escritura se realizará en al menos un nodo antes de responder al cliente de la operación.

- **ONE:** Asegura que la escritura se realizará en al menos el log de commit y la tabla de memoria de una réplica antes de responder al cliente de la operación.
- **TWO:** Asegura que la escritura se realizará en al menos el log de commit y la tabla de memoria de dos réplicas antes de responder al cliente con el resultado de la operación.
- **THREE:** Asegura que la escritura se realizará en al menos el log de commit y la tabla de memoria de tres réplicas antes de responder al cliente con el resultado de la operación.
- **QUORUM:** Asegura que la escritura se realizará en el log de commit y la tabla de memoria de (N/2)+1 réplica antes de responder al cliente con el resultado de la operación, siendo N el número total de nodos que forman el cluster.
- **LOCAL QUORUM**: Asegura que la escritura se realizará en el log de commit y la tabla de memoria de (R/2)+1 nodos en el clúster del datacenter en el que se están modificando los datos antes de responder al cliente con el resultado de la operación. Este nivel de consistencia tiene sentido si la base de datos está distribuida en varios datacenters. Es necesario configurar el parámetro Placement strategy con el valor NetworkTopologyStrategy, siendo R el factor de replicación.
- **EACH QUORUM:** Asegura que la escritura se realizará en el log de commit y la tabla de memoria de (R/2)+1 nodos en el clúster de cada datacenter antes de responder al cliente con el resultado de la operación, siendo R el factor de replicación. Este nivel de consistencia tiene sentido si la base de datos está distribuida en varios datacenters. Es necesario configurar el parámetro Placement strategy con el valor de NetworkTopologyStrategy.
- **ALL:** Asegura que la escritura se realizará en el log de commit y la tabla de memoria de las N réplicas antes de responder al cliente con el resultado de la operación. Hay que tener en cuenta que si alguna de estas réplicas falla, la operación también fallaría.

1.6.2.2 Consistencia de Lectura

Al realizar una petición de Lectura, también se debe especificar la consistencia deseada y comprobar que se haya logrado el nivel de consistencia especificado, al cumplirse se devuelven los resultados al cliente.

- **ONE:** Devuelve el resultado obtenido en la primera réplica que responda. Para este nivel se debe saber que en segundo plano se estará ejecutando una comprobación de consistencia que su función es buscar la réplica más reciente de los datos, por si hubiese algún dato obsoleto y así actualizarlo con el más reciente. Con esta comprobación, en operaciones de lecturas posteriores se obtendrá el resultado correcto incluso si la lectura

inicial obtuvo un valor antiguo y, por lo tanto, no válido. Esta es la operación llamada ReadRepair.

- **TWO:** Consulta en dos réplicas y devuelve los datos más recientes de dos de los más cercanos réplicas.
- **THREE:** Consulta en tres réplicas y devuelve el dato más reciente. Y, la comprobación se realiza en segundo plano.
- **QUORUM:** Devuelve el registro con el timestamp más reciente después de un quórum de réplicas ha respondido.
- **LOCAL QUORUM:** Devuelve el registro con el timestamp más reciente después de un quórum de réplicas en el centro de datos actual como nodo coordinador ha reportado. Evita la latencia de inter-comunicación de datos central.
- **EACH QUORUM:** Devuelve el registro con la timestamp más reciente después de un quórum de réplicas en cada centro de datos de la agrupación ha respondido.
- **ALL:** Devuelve el registro con la timestamp más reciente después de que todas las réplicas han respondido. La operación de lectura falla si una réplica no responde.

La elección de un nivel de consistencia para lecturas y escrituras va a depender de los requerimientos de la aplicación, ya que para obtener resultados consistentes (siempre la lectura de los datos escritos más recientemente) es necesario escribir y leer en todos los nodos posibles, pero la latencia puede ser mayor (el tiempo que toma para que los datos solicitados para ser devueltos o para la escritura de tener éxito).

Si la latencia es una prioridad, considere un nivel de consistencia de ONE (sólo un nodo réplica con éxito debe responder a la petición de lectura o escritura). Existe una mayor probabilidad de que se estén leyendo datos antiguos con este nivel de consistencia (como las réplicas de contacto para lecturas no siempre tienen las de escrituras más recientes). Para algunas aplicaciones, esto puede ser un compromiso aceptable. Si es absolutamente necesario que nunca falle una escritura, también se puede considerar un nivel de coherencia de escritura ANY. Este nivel de coherencia tiene la mayor probabilidad de no leer la devolución de los valores más recientes escritos.

1.6.3 Transacciones y Control de concurrencia

Cassandra no ofrece totalmente compatibilidades con transacciones ACID.

Ale Mendelzon, en su capítulo de control de paralelismo y recuperación menciona:
ACID (acrónimo para las cuatro propiedades que un sistema gestor de base de datos garantiza que se ejecute bajo su control):
Atomicidad: Visión de todo o nada, garantizando que si la transacción falla por cualquier motivo, deshará los efectos de las actualizaciones para la transacción haya realizado.
Consistencia: Es un estado que satisface todas las restricciones de integridad de una base de datos.
Aislamiento (Isolation**)**: Cada transacción se ejecuta en aislamiento, sin interferir en otra.
Durabilidad: Garantiza que las actualizaciones de la base de datos realizadas por transacciones serán durables y públicas. [8]

Como Cassandra es una base de datos no relacional, y no soporta JOINS ni claves foráneas, por lo tanto, no ofrece consistencia en el sentido ACID.

1.6.3.1 Atomicidad en Cassandra

En primer lugar, Cassandra no considera la posibilidad de agrupar varias actualizaciones de una fila en una sola operación, lo que en otros modelos es conocido como una transacción. Tampoco considera retroceder cuando una operación de escritura no es realizada correctamente en una réplica, esto basado en que las escrituras se realizan de forma independiente en cada réplica y resultado de cada operación (éxito o error) no es conocido por las restantes réplicas.
Un ejemplo de esto, es que si se utiliza un nivel de consistencia de escritura de quórum con un factor de replicación de 3, Cassandra enviará la escritura de 2 réplicas. Si la escritura falla en una de las réplicas pero, tiene éxito por la otra, Cassandra informará de un error de escritura para el cliente. Sin embargo, la escritura no se revierte automáticamente a la otra réplica.
Pero esta "descoordinación" entre réplicas, se subsana el hecho de que Cassandra usa la timestamp para determinar la actualización más reciente de una columna. La timestamp es proporcionada por la aplicación cliente. La última timestamp siempre es la devuelta al solicitar los datos, por lo que si varias sesiones de cliente actualizar las mismas columnas en una fila al mismo tiempo, la actualización más reciente es el que finalmente persisten.

1.6.3.2 Consistencia Sintonizable en Cassandra

Como se dijo anteriormente, Cassandra no utiliza transacciones, por lo tanto, no existe bloqueo o dependencias transaccionales al momento de actualizar varias filas o familias de columnas al mismo tiempo. Cassandra soporta sintonización entre la disponibilidad y la coherencia, y siempre da tolerancia partición. De igual manera, Cassandra se puede ajustar para darle consistencia fuerte en el sentido del teorema de CAP, y así los datos se hacen consistentes a través de todos los nodos en un clúster de base de datos distribuida.

1.6.3.3 Aislamiento en Cassandra

En versiones anteriores a Cassandra 1.1, cuando un usuario actualizaba una fila, otro usuario era capaz de ver las actualizaciones parciales de dicha fila. Por ejemplo, si un usuario estaba escribiendo una fila con dos mil columnas, otro usuario podría leer esa misma fila y ver algunas de las columnas, pero no todas las columnas si es que la escritura estaba todavía en curso.

Actualmente, es posible obtener un aislamiento completo a nivel de fila, ya que los que escribir un cliente no será visible para cualquier usuario hasta que se hayan completado. De un punto de vista ACID, esta mejora ahora da soporte Cassandra AID transaccional.

1.6.3.4 Durabilidad en Cassandra

Antes de que un proceso de escritura sobre un nodo de réplica sea reconocido como un éxito, esta escritura se graba tanto en la memoria como en un registro de confirmación. En el caso que por accidente, falle el servidor, se produce que las tablas de memoria se vacíen al disco y el registro de confirmación se repite para recuperar cualquier pérdida por escritura.

1.6.4 Arquitectura Cassandra

Cassandra está diseñado para el manejo de grandes volúmenes de datos a través de varios nodos.

El tema de las fallas en Cassandra es controlada con un sistema de peer-to-peer[23] distribuido, donde los datos se distribuyen entre todos los nodos del clúster. Los datos se escriben en una estructura en memoria llamada *memtable,* una vez que la estructura de memoria está llena se escriben en un archivo llamado *SStable* en el disco. Todas las escrituras son automáticamente particionadas y replicadas en todo el clúster.

Cassandra es una base de datos orientadas a filas, su arquitectura permite a cualquier usuario autorizado conectarse a algún nodo en cualquier centro de datos y acceder a datos utilizando el lenguaje CQL, que es una sintaxis similar a SQL. Los desarrolladores pueden acceder a través al CQL con el comando ./cqlsh.

1.6.4.1 **Comunicación entre nodos (goosip):** Es un protocolo de comunicación peer-to-peer para descubrir y compartir información de la ubicación y estado de los nodos en un clúster Cassandra.

1.6.4.2 Particionado

Un particionado determina la forma de distribuir los datos entre los nodos del clúster (incluyendo las réplicas). Básicamente una herramienta de particionado es una función hash para calcular el token (en el hash) de una row key (clave de fila). Cada fila de datos se identifica de forma única por una row key y es almacenada en el clúster dependiendo del valor del token.

Cassandra consta de los siguientes particionados:

- **Murmur3Partitioner:**

  El Murmur3Partitioner (org.apache.cassandra.dht.Murmur3Partitioner) es el particionado por defecto de Cassandra. El Murmur3Partitioner utiliza la función MurmurHash. Esta función crea un valor hash de 64 bits de la clave de fila. El rango posible de valores hash es de -2^{63} a 2^{63}.

[23] Una red peer-to-peer, red de pares, red entre iguales, red entre pares o red punto a punto (P2P, por sus siglas en inglés) es una red de computadoras en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí.(Wikipedia)

- **RandomPartitioner:**

  El RandomPartitioner (org.apache.cassandra.dht.RandomPartitioner) distribuye uniformemente los datos a través de los nodos utilizando un valor hash MD5[24] de la clave de fila. Los posibles valores del hash es de 0 a 2 ^{127} -1.

- **ByteOrderedPartitioner:**

  El ByteOrderPartitioner (org.apache.cassandra.dht.ByteOrderedPartitioner) está destinado para un particionado ordenado. Por ejemplo, si se realizan particiones alfabéticamente, se puede asignar un token utilizando su representación hexadecimal de 41. Este tipo de particionado permite realizar búsqueda por parcialmente o completa de las claves, lo cual no se puede realizar con las demás particiones.

1.6.4.3 **Réplica de estrategia de colocación :** Almacena copias de Casandra (réplicas) de datos en varios nodos para garantizar la fiabilidad y tolerancia a fallos. Una estrategia de replicación determina que nodos colocar para réplicas. La primera réplica de los datos no es más que la primera copia, no es único en todo sentido.

1.6.5 Hashing

Esta sección proporciona detalles de cómo funciona el mecanismo de hashing de Cassandra y cómo distribuye los datos en un clúster.

Para el ejemplo se utiliza el particionado Murmur3Partitioner y el hash se basa en la clave principal. En la Tabla 4 se encuentran los datos a utilizar.

Jim	Edad: 36	Coche: camaro	Sexo: M
Villancico	Edad: 37	Coche: bmw	Sexo: F
Johnny	Edad: 12		Sexo: M
Suzy	Edad: 10		Sexo: F

Tabla 4: Datos

[24] **MD5** (abreviatura de *Message-Digest Algorithm 5*, Algoritmo de Resumen del Mensaje 5) es un algoritmo de reducción criptográfico de 128 bits(Wikipedia)

Cassandra asigna un valor hash para cada clave primaria:

La clave primaria	Murmur3 valor hash
Jim	-2245462676723223822
Villancico	7723358927203680754
Johnny	-6723372854036780875
Suzy	1168604627387940318

Tabla 5: Asigna Valor Hash

Cada nodo en el clúster es responsable de una serie de datos basado en el valor de hash.

En la Tabla 6 se aprecia que los nodos (A, B, C y D) mantienen rangos de inicio y final de cada nodo, donde serán almacenados los datos según el valor del hashing dado y que se muestra en la Tabla 5.

Nodo	Murmur3 inicio del rango	Murmur3 final del rango
A	-9223372036854775808	-4611686018427387903
B	-4611686018427387904	-1
C	0	4611686018427387903
D	4611686018427387904	9223372036854775807

Tabla 6: Rango Nodos

Según los datos del valor hash asignado en la Tabla 5 y los valores de los rangos de almacenamiento de datos de los nodos de la Tabla 6, la distribución se especifica en la Tabla 7.

Nodo	Comience rango	Fin rango	La clave primaria	Valor hash
A	-9223372036854775808	-4611686018427387903	Johnny	-6723372854036780875
B	-4611686018427387904	-1	Jim	-2245462676723223822
C	0	4611686018427387903	Suzy	1168604627387940318
D	4611686018427387904	9223372036854775807	Villancico	7723358927203680754

Tabla 7: Distribución de datos

1.6.5.1 Cassandra-cli

Cassandra introdujo una versión estable de su interfaz de cliente de línea de comandos, cassandra-cli, que se puede utilizar para la definición común de datos (DDL), la manipulación de datos (DML), y la exploración de datos. Aunque no destinados al desarrollo de la aplicación, es una buena manera de empezar la definición de su modelo de datos y familiarizarse con Cassandra. Ver USO CASSANDRA-CLI

1.6.5.2 CQL

CQL sintaxis se basa en SQL. Aunque CQL tiene muchas similitudes con SQL, eso no cambia el modelo de datos subyacente Cassandra.

1.7 Hector

Hector es una API de java para Apache Cassandra,

Se puede encontrar información en

- http://en.wikipedia.org/wiki/Hector
- http://en.wikipedia.org/wiki/Cassandra

1.8 Pig

Pig es una plataforma para el análisis de grandes conjuntos de datos, que utiliza un lenguaje de alto nivel (llamando Pig Latin).

Con Pig se pueden realizar consultas de agregación de datos en Cassandra.

Este proyecto fue incorporado en la última versión estable de Apache Cassandra.

2 DEFINICIÓN DE LA EMPRESA

2.1 Descripción de la empresa

Infovida S.A nace en 1993, como una empresa informática orientada a prestar servicios a la industria de la salud en general. Aunque su interés primordial y focalización fue el área de las Isapres en donde desarrolló innumerables aplicaciones para la gestión operativa de las mismas.

A contar del año 2000, Infovida comienza a incursionar en el desarrollo de software administrativo creando aplicaciones de última generación para la administración financiera contable, remuneraciones y control de asistencia.

2.2 Descripción del problema

Los Data Warehouse y Data Mart almacenan grandes volúmenes de datos y su estructura la compone una base multidimensional sobre un motor propio, que se comunica con la base de datos a través de SQL. Es por esto que, con consultas muy complejas los tiempos de respuesta van aumentando a medida que se complejiza el Data Mart y, con el pasar del tiempo pueden ser insoslayables, incluso con la aplicación de planes preventivos de mantención y optimización, es por esto, que se requiere contar con una solución que permita almacenar grandes volúmenes de información mejorando los tiempos de respuesta.

2.2.1 Situación Actual

Actualmente, Infovida utiliza el motor de Base de Datos SQLServer 2008 con grandes cargas diarias de datos transaccionales por parte de las Clínicas pertenecientes al Holding Masvida, lo que con el tiempo ocasionará que la latencia en procesos de consulta comience a aumentar.

3 DEFINICIÓN DEL PROYECTO

El proyecto se realizara con el fin de obtener conocimientos acerca de los nuevos sistemas de base de datos NonSQL, a su vez poder ser una referencia para Infovida S.A. respecto a los resultados del comportamiento obtenidos al aplicarlos.

3.1 Objetivos

Objetivo General:

- Realizar un estudio comparativo de los tiempos de respuesta entre las bases de datos no relacionales (NonSQL) y las multidimensionales, mediante el diseño y la implementación de un Data Mart Clínico.

Objetivos Específicos:

- Investigar y analizar diferentes bases de datos NonSQL existentes.
- Implementar de un Data Mart Clínico con el modelo de datos multidimensional.
- Implementar de un Data Mart Clínico con el modelo de datos no relacional (NonSQL).
- Analizar el rendimiento y tiempos de respuestas de ambos modelos, para la elaboración de un estudio comparativo

3.2 Aportes

El aporte de esta investigación es la siguiente:

- Ayudar a la toma de decisiones por parte del Holding Masvida, con datos actualizados que permitan generar información en el momento adecuado. Lo anterior, considerando una mejora en los tiempos de respuesta que ocurren cuando se procesan grandes volúmenes de datos.
- Entregar un conocimiento de las bases de datos del tipo NonSQL, que es reciente dentro de las bases de datos tradicionales y sobre los cuales la información es aún escasa.

3.3 Límites

En cuanto a los límites del proyecto es necesario mencionar que no se desarrollará una aplicación particular que permita visualizar el Data Mart NonSQL, sino que se utilizará la herramienta que proporciona el propio motor para realizar las pruebas.
Las pruebas se realizarán en un solo computador, ya que no se disponen de más recursos para realizar el estudio.

3.4 Metodología

La metodología a utilizar será la destinada a encontrar soluciones a los problemas de ingeniería, por el hecho de que se realizará un análisis y comparaciones de los tiempos de respuestas de las consultas, y no se desarrollará un software en si.

Para llevar a cabo lo anteriormente mencionado, se leerán artículos y documentaciones, se seleccionará el modelo más adecuado según lo investigado para luego evaluar y reportar los resultados obtenidos.

Esta metodología se puede desglosar de la siguiente manera:

- Identificación de un problema
- Recopilación de información
- Búsqueda de soluciones de acuerdo al tema que se quiere desarrollar
- Evaluación y selección de soluciones

4 ACRÓNIMOS, ABREVIACIONES Y SIGNIFICADOS

ACID :Acrónimo de Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

ACM : Association of Computing Machinery.

CACHÉ : El caché de CPU, es una área especial de memoria que poseen los ordenadores.

CAP : Acrónimo de consistencia, disponibilidad y tolerancia a fallos.

CLUSTER : Se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de hardwares comunes y que se comportan como si fuesen una única computadora.

CODASYL : Acrónimo para "Conference on Data Systems Languages", un consorcio de industrias informáticas formado en 1959 con el objeto de regular el desarrollo de un lenguaje de programación estándar que pudiera ser utilizado en una multitud de ordenadores.

CQL : Acrónimo de Cassandra Query Language. Sintaxis parecida al SQL estándar, pero destinada a ser utilizada en Cassandra.

DATA MINING : Minería de datos es un campo de las ciencias de la computación referido al proceso que intenta descubrir patrones en grandes volúmenes de conjuntos de datos.

DDL : Lenguaje de descripción de datos.

DML : Lenguaje de base de datos para recuperar, insertar, borrar y actualizar datos en una base de datos.

DW : Data Warehouse. Almacén de datos orientado para el análisis de los datos.

ETL : Extract, Transform and Load. Proceso de extracción, transformación y carga de datos.

HTTP : Hypertext Trasfer Protocol. Protocolo usado en cada transacción de la World Wide Web.

IBM : International Business Machines. Empresa multinacional estadounidense que fabrica y comercializa hardware y software para computadoras.

INGRES : Soporte comercial de código abierto SQL para la gestión de base de datos relacional destinada a soportar grandes aplicaciones comerciales y gubernamentales.

ISAM : Indexed Sequential Access Method. Método para almacenar información a la

que se pueda acceder rápidamente.

JNI : Java Native Interface. Framework de programación que permite que un programa escrito en Java ejecutado en la máquina virtual de java pueda interactuar con programas escritos en otros lenguajes.

JOIN : Cláusula del SQL que combina registros de dos o más tablas en una base de datos.

JRE : Java Runtime Environment. Conjunto de utilidades que permite la ejecución de un programa en Java.

METADATO : Datos que describen otros datos.

MYSQL : Sistema de gestión de base de datos relacional de código abierto.

NoSql : Modelo de datos no relacional.

OLAP : Procesamiento Analítico Online.

QUERIES : Lenguaje informático utilizado para realizar consultas en bases de datos y sistemas de información.

RDBM : Relational Database Management System o RDBMS - Sistema de Gestión de Base de Datos Relacional o SGBDR. Tipo de SGBD (o DBMS en inglés) para bases de datos relacionales (que emplea el modelo de datos); o sea, soporte de tablas relacionadas.

SNAPSHOTS : Vistas simultáneas.

SQL : Structure Query Language, Lenguaje de consulta estructurado de acceso

THRIFT : Llamada a procedimiento remoto (RPC) desarrollada en el marco de Facebook para "escalar en lenguajes de desarrollo de servicios".

UOD :Universe of discourse.

WWW :Word Wide Web.

5 ANÁLISIS

En este capítulo se aborda el tema de análisis de modelo multidimensional utilizado para la obtención de datos.

Se analiza el diseño actual, para comprender el funcionamiento y así poder replicarlo en el modelo de datos de Cassandra.

Se señala el cómo se obtiene la información para poblar la Base de Datos, su modelo físico y el tipo de consulta que se realizan en ellas.

5.1 Modelo Actual

Se describe el modelo que se encuentra actualmente en uso.

5.1.1 Funcionamiento

Data Marts clínico tiene como finalidad proveer a la clínica de información relevante relacionada con los ingresos de los pacientes.

El modelo se utiliza desde Diciembre del 2012, y mantiene el registro de todos los pacientes ingresados a consulta desde el año 2009 a la fecha. Dicha información se utiliza para la generación de Informes mediante una herramienta online llamada ReportPortal.

Estos informes se confeccionan según plantillas utilizadas por cada clínica y generan reportes anuales y mensuales. Por ejemplo, el total de camas utilizadas en un periodo, la cantidad de pacientes ingresados (ambulatorios u hospitalarios), cantidad de operaciones realizadas, entre otros.

Para apoyar la comprensión de los datos, cada informe incluye un gráfico con datos estadísticos.

Tipos de Informes:

Ilustración 18: Informes de Gestión

5.1.2 Componentes:

El DataMarts implementado en la actualidad sigue el modelo estrella, que consiste en una tabla de hechos anexada con todas las dimensiones a utilizar, en este caso, está modelado de la siguiente manera:

- Tablas de Hechos:

  FACT_INGRESO(Ingreso de pacientes con los valores de cada cargo asignado por ficha)

  FACT_INGRESO_DURACION (Duración del paciente en el pabellón en minutos),

  FACT_UTILI_HAB(Cantidad de días en la habitación).

- Dimensiones: Compuesta por 15 dimensiones, que contempla DIM_PACIENTE, DIM_PRESTADOR, DIM_EQUIPO_MEDICO, DIM_ISAPRE, DIM_CONVENIO, entre otros. Para mayor información ver Anexo: Diccionario de Datos

  El la Ilustración 19: Lista de Dimensiones, se lista las dimensiones creadas, y en la Ilustración 21 las medidas creadas, las medidas corresponde a los valores numéricos a desplegar en los reportes.

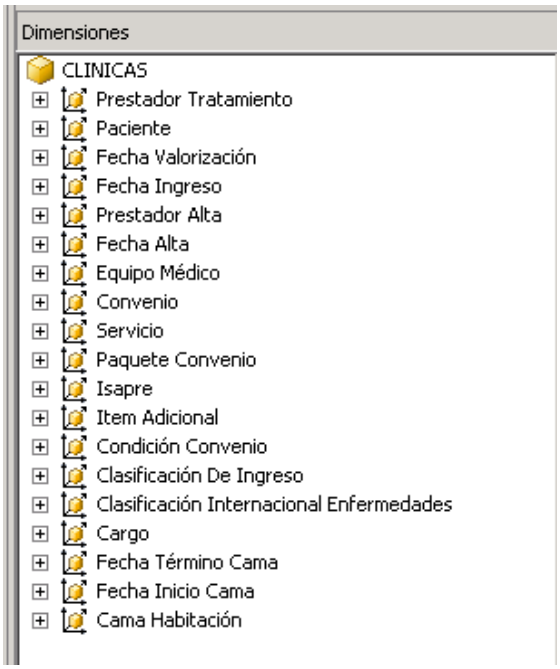


Ilustración 19: Lista de Dimensiones

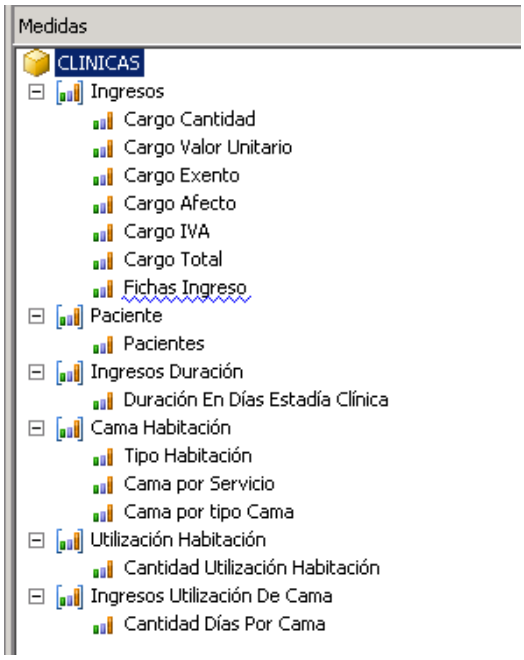


Ilustración 20: Lista de Medidas

- **Datos:**

  En la Tabla 8, desglosa la cantidad de registros por cada tabla de ingreso del modelo relacional.

  Ver definición de Tuplas y Atributos en Ilustración 4: Ejemplo Atributos y Tuplas**.**

Tabla de Hechos	Cantidad Tuplas	Cantidad Atributos	Total
Fact_ingreso	**2.892.921.-**	**30.-**	**86.787.630.-**
Fact_ingreso_duracion	**17.276.-**	**11.-**	**190.036.-**
Fact_ingreso_utili_hab	**36.677.-**	**8.-**	**293.416.-**
Total registros			**87.271.082.-**

Tabla 8: Cantidad registros RDMS[25]

[25] Los Datos contabilizados corresponden sólo a las tablas de hechos, ya que los relacionados con las dimensiones la cantidad de registros no se verán afectados.

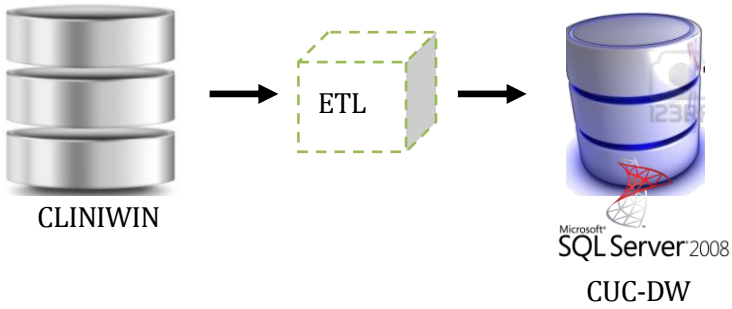
5.1.3 Flujo de Datos

En la Ilustración 21: Flujo Datos Data Marts se grafica el flujo de los Datos desde la base de datos operacional al DataMart.

5.1.3.1 Proceso ETL.

Datos se extraen desde la base de datos operacional de la clínica, denominada CLINIWIN. Son transformados y cargados a la base de Datos CUC-DW, que corresponde a la base utilizada para el Data Marts, alojada en el motor de Base de Datos de SQL Server.

Ilustración 21: Flujo Datos Data Marts



Fuente: Elaboración propia.

La Transformación de Datos desde la Base de Datos operacional CLINIWIN hacia CUC-DW se realiza para mantener datos unificados dentro del almacén y que sean entendibles para el usuario.

Modelo Operacional : Tabla	Valor	Data Marts
Paciente, Prestador	Sexo: F y M	Femenino y Masculino respectivamente
Convenio	Tipo: O, P, I, F, N Exento: S, N, null	Otros, Particular, Isapre, Fonasa Exento, No exento, Sin especificar
Servicio	Clasificación 1: CEM, END, SKI, SIM, IAP, ICA, IFP, IMA, IOF, IOT, SVA, QUI. Clasificación 2: MAT, PED,	Centro Médico Servicio Médico

	URG, UGO, SUG, UTA, UTN, PAB, SMT, POS, SPB, MED, NEO, CAM, EST, SNN, SMQ	
	Clasificación 3: BIO, CAU, LAB, SLB, SUE	Laboratorio
	Null	Otros Servicios
	Estado: S, N, null	Vigente, No vigente y Desconocido
Isapre	Tipo: F, P, I, O, Null	Fonasa, Particular, Isapre, Otros y Desconocido
Prestador	Acreditado: S, N	Acreditado, No Acreditado
Item Adicional	Tipo: P ,E , O, null	Pabellón, Equipo Médico, Otros y Sin Tipo respectivamente.
Equipo Médico	Acreditado: S, N	Acreditado, No Acreditado

La extracción, transformación y cargas se realiza mediante código confeccionado para cada caso, no se utiliza ninguna herramienta para dicha operación.

Para realizar este estudio se debió transformar los datos de Pacientes y Médicos, ya que corresponde a información confidencial. Ver TRANSFORMACION DATOS

5.2 Modelo físico

Para más información del modelo de datos Ver ANEXO D: Diccionario de Datos

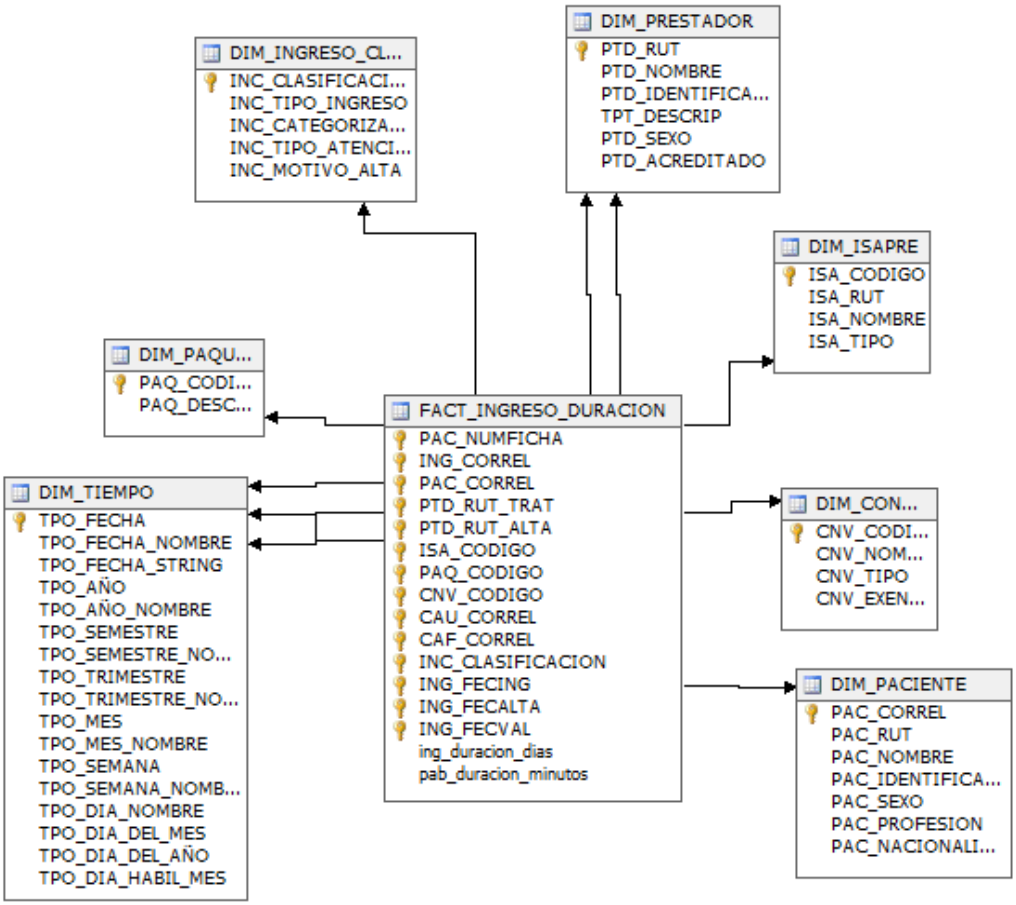


Ilustración 22: Modelo Fact_ingreso_Duración

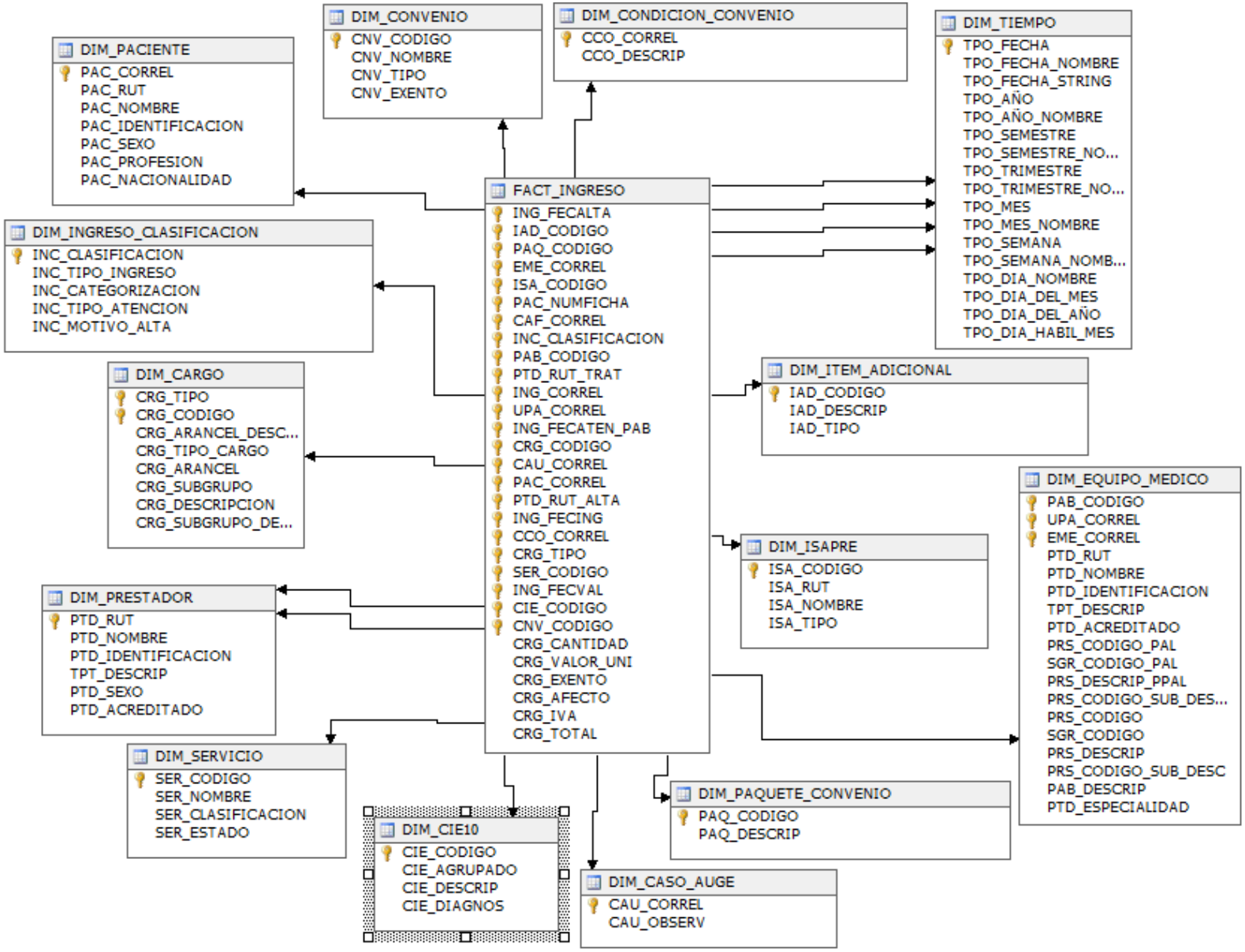


Ilustración 23: Modelo FACT_INGRESO

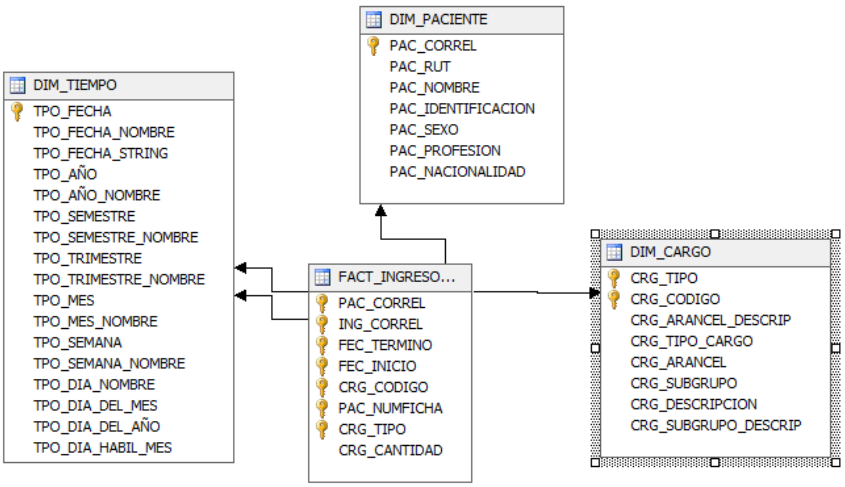


Ilustración 24: FACT_INGRESO_HABITACION

5.2.1 Debilidades:

La tabla FACT_INGRESO, Ver Ilustración 23: Modelo FACT_INGRESO, posee 30 atributos, de las cuales, 24 son llaves foráneas que provienen de varias tablas y las otras 6 restantes contienen datos propios de la tabla. Sin embargo, en el motor de Base de Datos de SQLServer, es posible declarar solo 16 claves primarias.

La tabla no posee una llave primaria que logre identificar unívocamente a una tupla, haciendo que en cada actualización se deban modificar todas las tuplas que tienen la información redundante.

6 DISEÑO

Tras el estudio del modelo relacional realizado en el Capítulo de Análisis, se está en condiciones de adaptar el modelo de datos relacional a un modelo no relacional en la base de datos de Cassandra.

6.1 Modelo Propuesto

Aprovechando las características proporcionadas por el modelo de Cassandra, las tres tablas de hechos del modelo relacional actual se unen, consolidándose todos los datos en una sola familia de columnas.

En el modelo actual existe una gran cantidad de datos que no contienen valores. Por lo tanto, se propone realizar una limpieza de ellos antes de efectuar la inserción, lo que significa que si el dato no se encuentra, simplemente la columna no es creada en la base de datos.

Cassandra cuenta un método de organización de los datos a nivel de disco, el cual permite almacenar de acuerdo a un orden dado, este método se denomina ByteOrdererPartitioner (Ver Particionado) y será el utilizado para almacenar el formato de la clave primaria, el cual otorgará un mejor rendimiento de la extracción de datos.

La clave primaria mantendrá el formato (Año-Mes-Día)+ Correlativo del ingreso del paciente. De esta manera, todos los pacientes del mismo Año-Mes-Día, serán almacenados en bloques del disco contiguos, logrando realizar búsquedas más rápidas.

Debido que el CQL de Cassandra todavía es muy limitado, se agregan nuevas columnas correspondientes al año, mes y días del ingreso de cada paciente, con la finalidad de poder filtrar por años y mes, ya que estos son los datos más consultados. También se crea una nueva columna que une el correlativo, ingreso y ficha de cada paciente, ya que con la creación de un índice en este campo se mejoran los tiempos de respuestas de las consultas de un paciente en particular.

De igual manera, se crean índices secundarios para la realización de búsquedas de datos por las diferentes columnas, ya que en Cassandra se puede acceder a los valores solo por su clave, por lo tanto, los índices secundarios cumple la función de optimizar la búsqueda.

- **Datos:**

Tabla de Hechos	Cantidad Tuplas	Cantidad Columnas	Total columnas
Fact_ingreso	**2.892.921.-**	**(variable)**	**53.472.620.-**
Fact_ingreso_duracion	**17.276.-**	**1.-**	**17.276.-**
Fact_ingreso_utili_hab	**36.677.-**	**2.-**	**73.354.-**
Total columnas			**53.563.250.-**

Tabla 9: Cantidad registros Cassandra[26]

6.1.1 Debilidades:

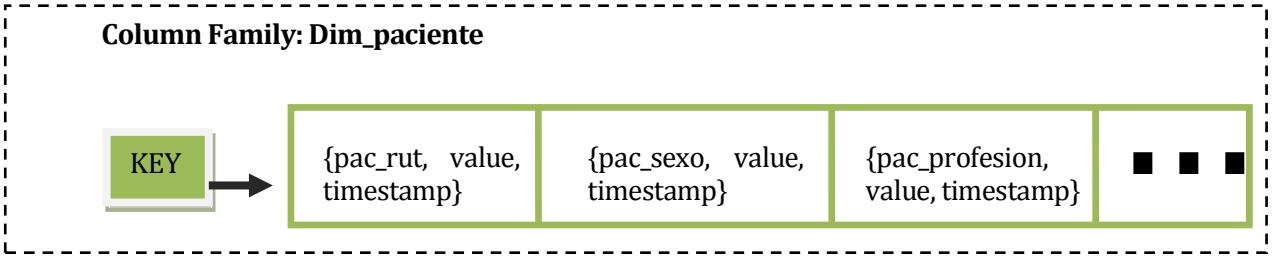
Debido a la naturaleza de orientación de columnas de la base de datos Cassandra y su CQL es limitado, no es posible formular consultas complejas de agregación como se realiza en SqlServer, ya que Cassandra está destinado a la consulta de datos sin procesar, no al análisis de ellos mismo.

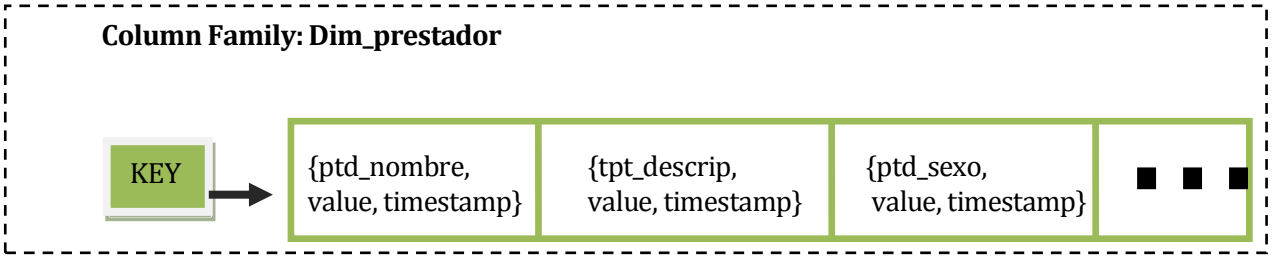
6.2 Diagrama de datos NonSQl

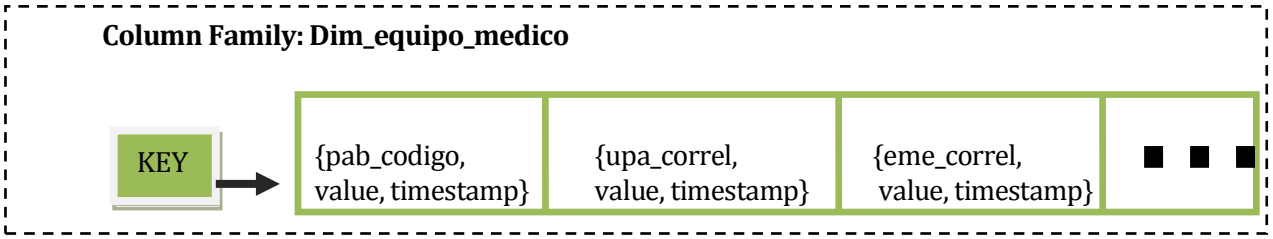
Como se menciona anteriormente el modelo de Cassandra es flexible en cuanto a los esquemas, por lo tanto, no todas las familias de columnas contienen la misma cantidad de columnas.

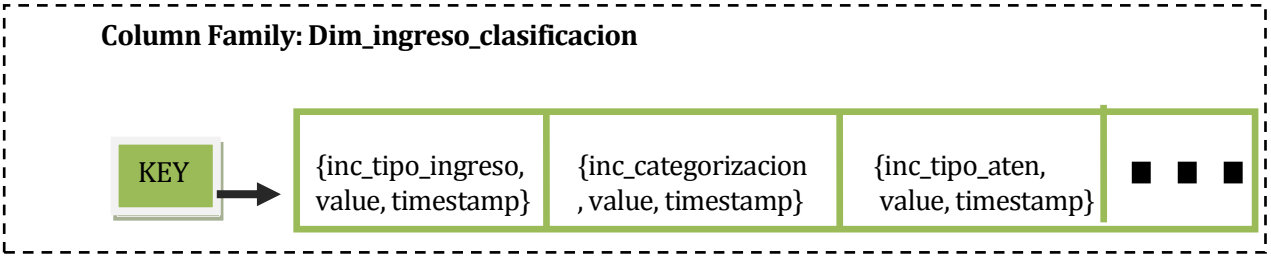
[26] Los Datos contabilizados corresponden sólo a las tablas de hechos, ya que los relacionados con las dimensiones la cantidad de registros no se verán afectados

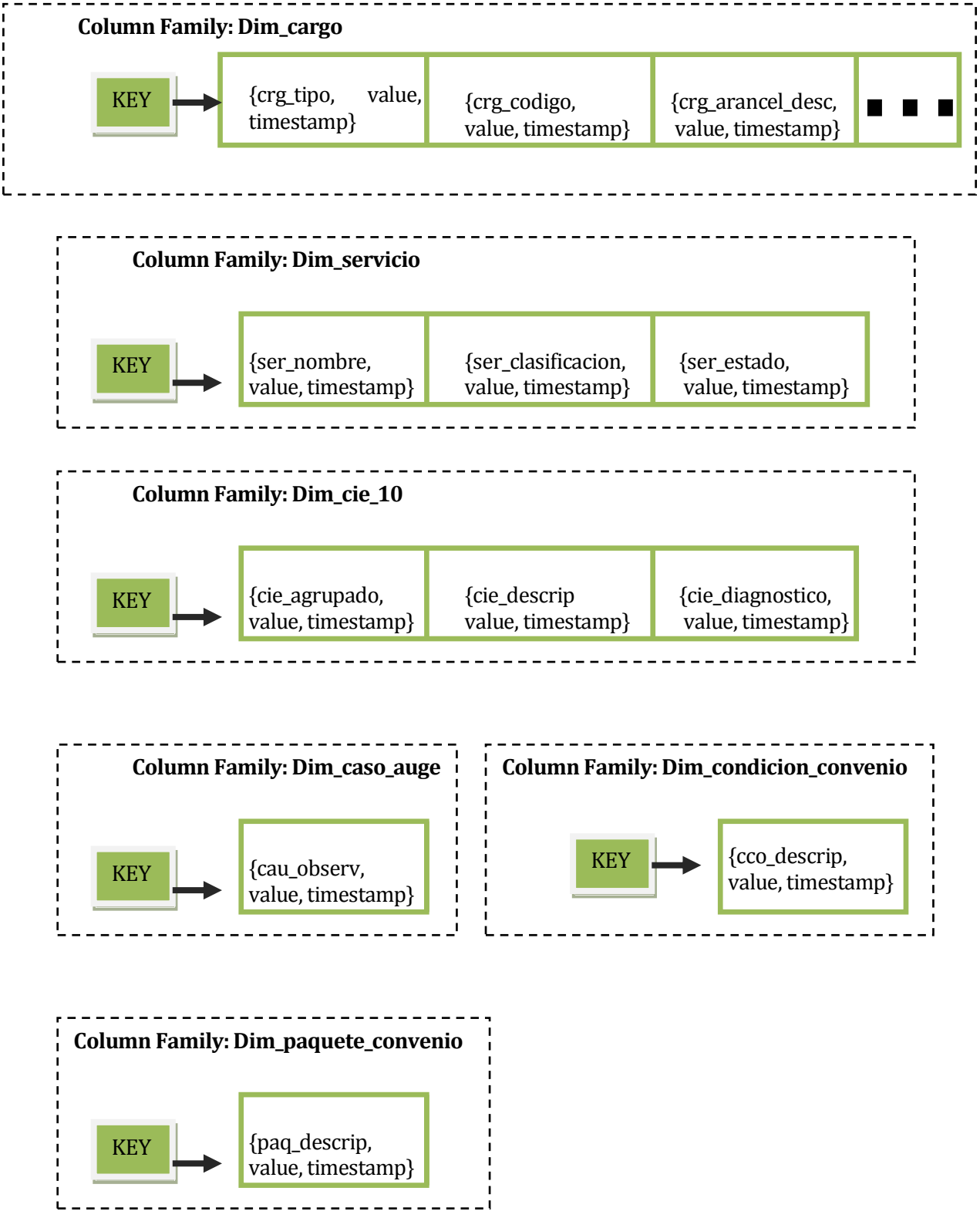
A continuación, se listan las familias de columnas del modelo propuesto. La familia de columna Fact_ingreso, puede variar en cuanto a la cantidad de columnas que puede contener.











Fuente: Gráfica del modelo de Datos de Cassandra fue extraída desde http://www.ibm.com/developerworks/br/library/os-apache-cassandra/

6.3 Conclusión

Según el modelo de Cassandra diseñado, la cantidad de registros es menor al modelo actual.

En el modelo propuesto disminuye en gran cantidad los registros almacenados en la base de datos, SqlServer: 87.271.082.-, Cassandra : 53.563.250.-, que corresponde a una disminución de un 38,62%. Si bien es cierto, el modelo de Cassandra es más flexible respecto al almacenamiento de datos, ya que las row key pueden o no tener la misma cantidad de columnas asociados, no se puede aseverar que tendrá tiempos de respuestas más rápidos que los de SQLServer.

En el capítulo de pruebas se verifica si modelo no relacional implementado en Cassandra es más eficiente en la búsqueda y extracción de datos que el modelo relacional en SQLServer.

7 IMPLEMENTACIÓN

Para llevar a cabo el estudio, la base de Datos Cassandra se instala en un entorno de Linux, en este caso, Ubuntu 12.04. Sin embargo, su instalación no es cubierta por este estudio, ya que eso conlleva a la inmersión de muchos conceptos que solo lograrían desviar el objetivo de este trabajo.
Para instalación y configuración ver Anexo A: Instalación y configuraciones.
Los códigos fueron escrito en el programa Eclipse, para la creación de un proyecto y las librerías a exportar para la conexión con Cassandra Ver Anexo B: Proyecto Eclipse.

Modelo NoSQL.

Debido a que Cassandra es una base de datos clave valor, se debe asignar en estricto rigor una clave primaria, la que corresponde fecha/tiempo+correlativo del ingreso del paciente.

Cassandra no dispone de claves foráneas. Sin embargo, existen los denominados índices secundarios, los cuales permiten realizar búsquedas mediante los valores de las columnas asignadas como índices, por lo tanto, si bien no se realiza automáticamente la comprobación de la clave, si se puede hacer mediante código escrito en java, el cual inserta los datos solamente en los casos en que estos estén presentes en las demás familias de columnas.

La consistencia en esta base de datos no se encuentra predefinida, pero si es configurable, como se mencionó anteriormente existen 2 tipos, de lectura y escritura.

Como las pruebas se realizaron en un solo nodo la configuración de la consistencia de lectura corresponde a ONE (ver 1.6.2.2 Consistencia de Lectura), lo que significa que al encontrar el dato, lo retorna de inmediato. En el caso que se dispusiera más de un nodo en el clúster, se debe configurar la consistencia del tipo ALL, que consiste en que devuelve el registro más reciente, después que todas las réplicas hayan respondidos. Si no se realiza de esta manera puede ocurrir que al momento de actualizar un valor, no se realice en todos los nodos, lo que influye al momento de la lectura, ya que puede darse el caso en que el valor encontrado sea el nodo donde no se realizó la actualización, devolviendo valores distintos en un mismo caso. Aunque lo

mencionado últimamente depende de los requerimientos del modelo, ya que puede darse el caso que la consistencia no sea un factor que se pueda negociar, y se prefiera la rapidez, lo que implica que la consistencia de lectura puede ser ONE.

Cassandra, aunque dispone de un lenguaje similar al SQL, el llamado CQL, no se pueden realizar todas las operaciones que el SQL estándar, por ejemplo las consultas de agregación.

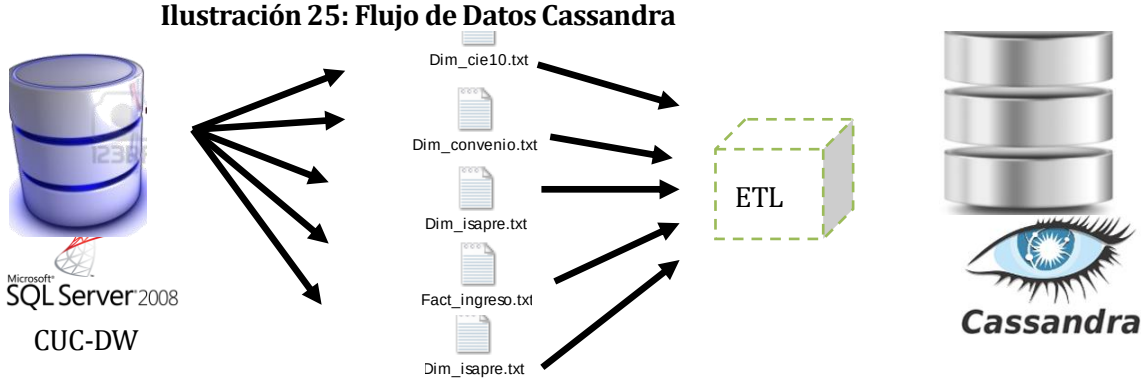
7.1.1 Extración de Datos

Para realizar las pruebas en el nuevo modelo de Cassandra se exportan los datos de cada tabla desde SQLServer hacia archivos en formato .txt.(Ver EXPORTAR DATOS)

La extracción, transformación y cargas se realiza mediante código confeccionado para cada caso, no se utiliza ninguna herramienta para dicha operación.

Finalmente, se extraen los datos, se adaptan al nuevo modelo y se cargan para empezar a realizar las pruebas.

Ilustración 25: Flujo de Datos Cassandra



Fuente: Elaboración propia.

En este caso, la transformación de datos hace referencia a la omisión de aquellos que no están presentes en el modelo relacional, por ejemplo, los valores ingresados en el caso que corresponden a valores nulos dentro de la base de datos operación.

7.1.2 Creación de índices Secundarios.

En Cassandra para realizar algún tipo de búsqueda se debe hacer accediendo por la clave única de las columnas, pero existen los denominados índices Secundarios, que sirven para poder realizar búsqueda mediantes ellos.

Éstos se pueden crear al momento de crear la familia de columnas o después de la inserción de los datos.

Por ejemplo: En la Ilustración 26: Creación metadato familia de columnas con índices secundarios, se visualiza la creación de la familia de columna, con el comparador de los atributos, validación que tendrá la clave, y la validación por defecto, que corresponde si es que se agrega una columna que no se encuentre en el metadato.

Para la creación del índice sólo se le agrega a la columna *index_type:KEYS,* y se crea automáticamente y se va almacenando al momento de la inserción.

```
create column family fact_ingreso with
 comparator = UTF8Type and
 key_validation_class=UTF8Type and
 default_validation_class= UTF8Type and
 column_metadata =
 [
{column_name: clave_paciente, validation_class: UTF8Type, index_type: KEYS},
{column_name: ing_year, validation_class: LongType, index_type: KEYS}
{column_name: ing_month, validation_class: UTF8Type},
{column_name: ing_day, validation_class: UTF8Type},
{column_name: pac_numficha, validation_class: LongType}
]
```

Ilustración 26: Creación metadato familia de columnas con índices secundarios

O también, se actualiza la familia de columnas después de la inserción de datos, lo cual se realiza en tiempo de ejecución. Ver Ilustración 27: Creación índice secundarios actualizando familia de columnas.

```
UPDATE COLUMN FAMILY fact_ingreso WITH
comparator = UTF8Type AND
column_metadata =
[
{column_name: clave_paciente, validation_class: UTF8Type, index_type: KEYS},
{column_name: ing_year, validation_class: LongType, index_type: KEYS}
]
```

Ilustración 27: Creación índice secundarios actualizando familia de columnas

Lo anterior se realiza en la Shell de cassandra-cli, también se puede realizar desde código java.

8 PRUEBAS

Después de realizar la implementación se ejecutan las pruebas.

8.1 Entorno de Hardware y Software

Las pruebas realizadas en Apache-Cassandra y en SQLServer, se efectuaron en un solo Notebook, con procesador Intel Core i3, 3200MB de RAM, 800MB de Swap y en un entorno de Linux,Ubuntu 12.04.

8.2 Especificación de Pruebas

Si bien este estudio contempla una comparación entre los tiempos de respuesta entre Cassandra y SqlServer, para la realización de lo anterior se elaboran pruebas comparativa en cuanto al rendimiento propio de Cassandra, midiéndose en consultas que incluyan índices secundarios, y claves, ya que el modelo confeccionado su clave primaria consta de las fechas de ingreso de los pacientes, siendo esto de gran importancia, porque un DataMarts, se enmarca en los hechos en cierto periodo de tiempo.

También se realizan pruebas sobre dos versiones distintas de Cassandra, las que corresponden a la 1.2.1 y la 1.2.4, ambas con un particionado de ByteOrderPartitioner. Estas pruebas se efectúan para comprobar las mejoras de la localización de las RowKey en relación a ambas versiones.

Según los resultados arrojados, la versión que obtenga mejores tiempos de respuestas se comparan con los resultados arrojados por SQLServer 2008.

8.2.1 Dominio

El dominio de datos se encuentra especificado en el capítulo 5.1 Modelo Actual para el modelo relacional y 6.1 Modelo Propuesto para el modelo no relacional.

8.2.2 Consultas de Selección

Se crearon 4 tipos de consultas de selección, las que se efectúan en el modelo relacional.

Estas consultas se realizan con el fin de determinar el tiempo que demora en extraer dichos resultados.

Es importante mencionar que para el modelo de Cassandra, las consultas se adecuaron para poder evaluar el rendimiento de la extracción de datos con respecto a las claves y/ o índices secundarios creados, sin embargo, existen consultas en que no se pueden adaptar a ambas.

Para obtener los datos desde Cassandra se utilizó la API de Hector.

Cada consulta se ejecuta tres veces (C1, C2 y C3) con la propósito de evaluar el comportamiento que mantiene ambas base de datos.

Consultas a realizar:

- **Q1:** Seleccionar el cargo exento, cargo afecto, cargo IVA y el cargo total de pacientes que realizaron ingreso en el mes de Mayo del 2011.
- **Q2:** Seleccionar el paciente que tiene como número de ficha 154, ingreso de 1 y con un correlativo 160.
- **Q3:** Seleccionar los pacientes ingresados el año 2010 y que tienen un convenio MVCCERO2.
- **Q4:** Seleccionar pacientes ingresados entre Enero y Junio 2010.

8.2.2.1 Consultas en el marco de SqlServer

```
DECLARE
    @antes   DATETIME,
    @despues DATETIME
SET @antes = GETDATE()
SELECT CRG_AFECTO, CRG_EXENTO, CRG_IVA, CRG_TOTAL
        FROM FACT_INGRESO
        WHERE YEAR(ING_FECING)=2011 AND MONTH(ING_FECING)=5
SET @despues = GETDATE()
SELECT DATEDIFF(ms,@antes,@despues)
```

Ilustración 28: Q1 para SqlServer

```
DECLARE
    @antes   DATETIME,
    @despues DATETIME
SET @antes = GETDATE()
SELECT *
        FROM FACT_INGRESO
        WHERE PAC_NUMFICHA=154 AND ING_CORREL=1 AND PAC_CORREL=160
SET @despues = GETDATE()
SELECT DATEDIFF(ms,@antes,@despues)
```

Ilustración 29: Q2 para SqlServer

```
DECLARE
    @antes   DATETIME,
    @despues DATETIME
SET @antes = GETDATE()
SELECT *
        FROM FACT_INGRESO
        WHERE YEAR(ING_FECING)='2010' AND CNV_CODIGO='MVCCERO2'
SET @despues = GETDATE()
SELECT DATEDIFF(ms,@antes,@despues)
```

Ilustración 30: Q3 para SqlServer

```
DECLARE
    @antes   DATETIME,
    @despues DATETIME
SET @antes = GETDATE()
SELECT *
        FROM fact_ingreso
        WHERE ING_FECING>'20091231' AND ING_FECING<'20100701'
SET @despues = GETDATE()
SELECT DATEDIFF(ms,@antes,@despues)
```

Ilustración 31: Q4 para SqlServer

8.2.2.2 Consultas en marco de Cassandra

```
CqlQuery<String, String,Long> cqlQuery1 = new CqlQuery<String, String, Long>(keyspace,stringSerializer,
                                                                              stringSerializer,
longSerializer);
cqlQuery1.setQuery("select crg_afecto, crg_exento, crg_iva, crg_total
                    from fact_ingreso
                    where key>='2011-05' and key<'2011-06' limit 82000");
System.out.println("Tiempo: "+cqlQuery1.execute().getExecutionTimeMicro());
```

Ilustración 32: Q1 para Cassandra, búsqueda por clave

```
1. CqlQuery<String, String,Long> cqlQuery1 = new CqlQuery<String, String, Long>(keyspace,stringSerializer,
                                                                                 stringSerializer,
   longSerializer);
2. cqlQuery1.setQuery("select crg_afecto, crg_exento, crg_iva, crg_total
3.                       from fact_ingreso
4.                       where ing_year='2011' and ing_month='5' limit 82000");
5. System.out.println("Tiempo:"+cqlQuery1.execute().getExecutionTimeMicro());
```

Ilustración 33: Q1 para Cassandra, búsqueda por índices secundarios

```
1. CqlQuery<String, String,Long> cqlQuery2 = new CqlQuery<String, String, Long>(keyspace,stringSerializer,
                                                                                 stringSerializer,
   longSerializer);
2. cqlQuery2.setQuery("select *
3.                       from fact_ingreso
4.                       where where clave_paciente='154-1-160'");
5. System.out.println("Tiempo: "+cqlQuery2.execute().getExecutionTimeMicro());
```

Ilustración 34: Q2 para Cassandra búsqueda por índice secundario

```
1. CqlQuery<String, String,Long> cqlQuery3 = new CqlQuery<String, String, Long>(keyspace,stringSerializer,
                                                                                 stringSerializer,
   longSerializer);
2. cqlQuery3.setQuery("select *
3.                       from fact_ingreso
4.                       where ing_year=2010 and cnv_codigo='MVCCERO2'LIMIT 8000");
5. System.out.println("Tiempo: "+cqlQuery3.execute().getExecutionTimeMicro());
```

Ilustración 35: Q3 para Cassandra, búsqueda por índices secundarios

```
1. CqlQuery<String, String,Long> cqlQuery3 = new CqlQuery<String, String, Long>(keyspace,stringSerializer,
                                                                                 stringSerializer, longSerializer);
2. cqlQuery3.setQuery("select *
3.                       from fact_ingreso
4.                       where where key>='2010-01' and key<='2010-06' limit 71000");
5. System.out.println("Tiempo: "+cqlQuery3.execute().getExecutionTimeMicro());
```

Ilustración 36: Q4 para Cassandra, búsqueda por clave

En esta sección se aprecia:

La consulta **Q2** no contiene consulta por clave, esto es porque se desea obtener un paciente en particular, y, como la clave primaria corresponde a la fecha del ingreso del paciente, la búsqueda se debe hacer solamente por un índice secundario creado en esa columna.

La consulta **Q3** solo contiene la búsqueda por índices secundarios creados, es porque en Cassandra no se puede combinar la búsqueda por un rango de la clave y por índice, por ese hecho sólo se realiza filtrando según los índices secundarios.

9 RESULTADOS

En este capítulo se presentan los resultados de las consultas confeccionadas.

La presentación de ellos se divide en 3 grupos:
Primer grupo consta de los tiempos obtenidos en las consultas por SqlServer.
Segundo grupo corresponde a la comparativa realizada sólo en Cassandra.
Tercer grupo, la comparación de ambas base de datos.

En cada uno de los grupos se elaboran gráficos de acuerdos a los datos obtenidos en cada consulta y su interpretación correspondiente.

9.1 SqlServer

Los datos de la Tabla 10: Tabla Tiempos respuestas SqlServer están expresados en Milisegundos

	SqlServer 2008		
	C1	C2	C3
Q1	13946	1590	1230
Q2	12746	460	453
Q3	14823	933	860
Q4	29553	3366	2590

Tabla 10: Tabla Tiempos respuestas SqlServer

9.1.1.1 Análisis de desempeño ejecución de consultas

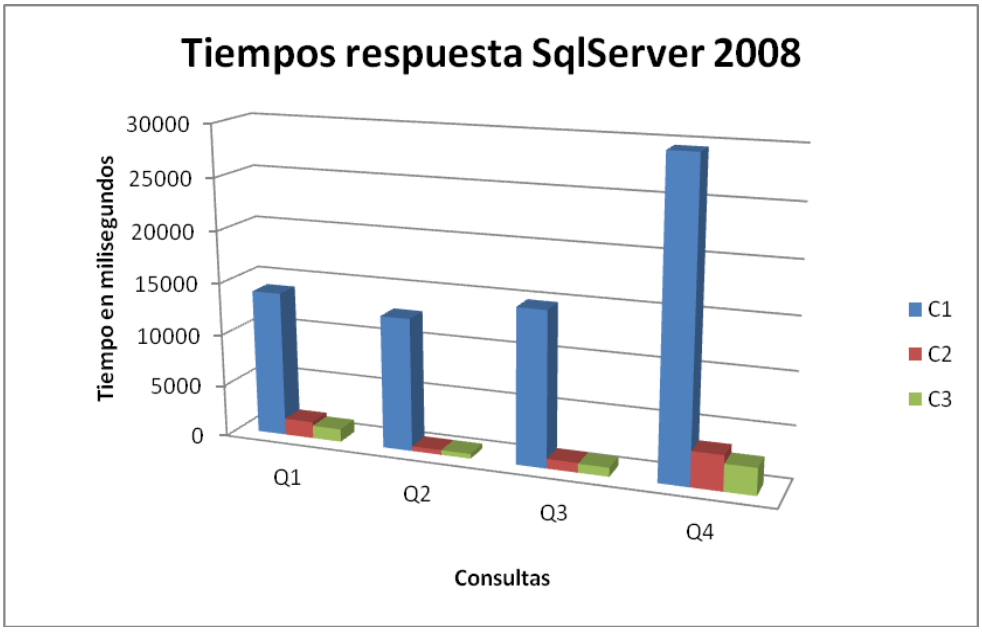


Ilustración 37: Gráfico Barras resumen tiempos respuestas SqlServer 2008

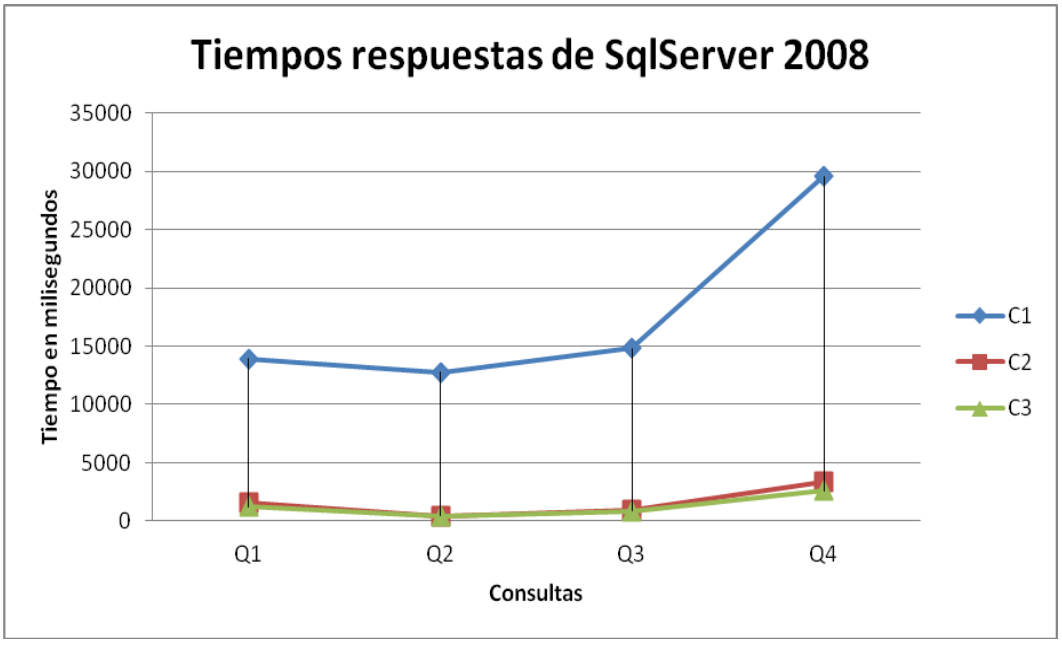


Ilustración 38: Gráfico de puntos tiempos respuestas SqlServer 2008

Interpretación.

En la Ilustración 37: Gráfico Barras resumen tiempos respuestas SqlServer 2008 e Ilustración 38: Gráfico de puntos tiempos respuestas SqlServer 2008 se grafican los resultados obtenidos en relación a los tiempos de respuestas de las 4 consultas confeccionadas, repitiendo cada una tres veces.

Como se aprecia en ambos gráficos Q4 es la consulta que demoró mayor tiempo en devolver los resultados, sin embargo, no es la que devuelve la mayor cantidad de filas entre las 4, ya que Q1 devuelve una muestra de 83.317 filas a diferencia de Q4 con 70.195, lo cual ocurre porque Q4 corresponde a un rango de 6 meses en un determinado año, a diferencia de Q1 que consulta por un mes y un año en particular.

También se aprecia que la primera vez que se realiza las diferentes consultas, sus tiempos de respuesta son bastante mayores en comparación a la segunda y tercera repetición. Las dos últimas varían levemente en los tiempos.

La disminución de los tiempos de respuesta ocurre ya que el motor de base de datos de SqlServer, realiza estadísticas de los datos consultados, donde almacena los valores arrojados, por lo tanto, al volver a ejecutar la consulta su respuesta un notoriamente menor.

9.2 Apache-Cassandra

	BYTEORDERPARTITIONER											
	Cassandra 1.2.1						Cassandra 1.2.4					
	key			Index			key			Index		
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
Q1	3296	2584	2431	4786	4215	4495	2962	3110	2540	5743	5871	6731
Q2	-	-	-	7	6,8	5,7	-	-	-	5,1	4,6	4,8
Q3	-	-	-	2455	2209	2216	-	-	-	2580	2172	2349
Q4	6283	9582	4050	-	-	-	5164	4173	4081	-	-	-

Tabla 11: Tiempos de Respuestas partición ByteOrderPartitioner en Cassandra

Los datos expresados en la Tabla 11: Tiempos de Respuestas partición ByteOrderPartitioner en Cassandra se miden en Milisegundos.

9.2.1 Análisis de desempeño de Cassandra según versión

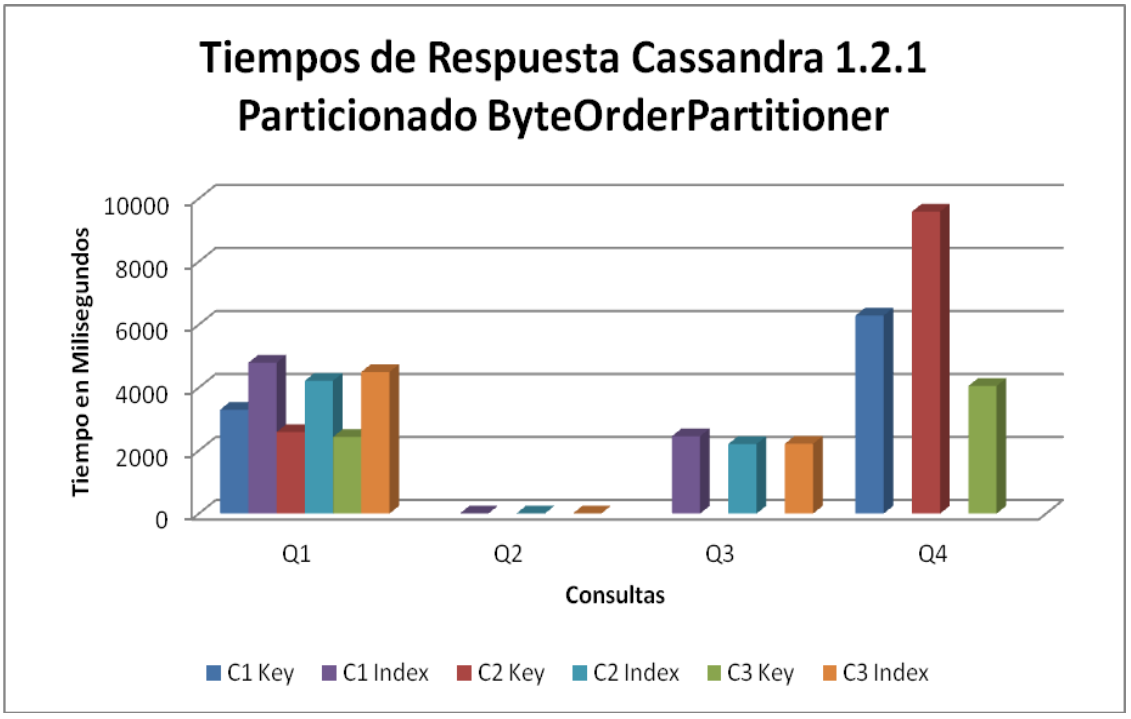


Ilustración 39: Tiempos de respuestas Cassandra 1.2.1 particionado ByteOrderPartitioner

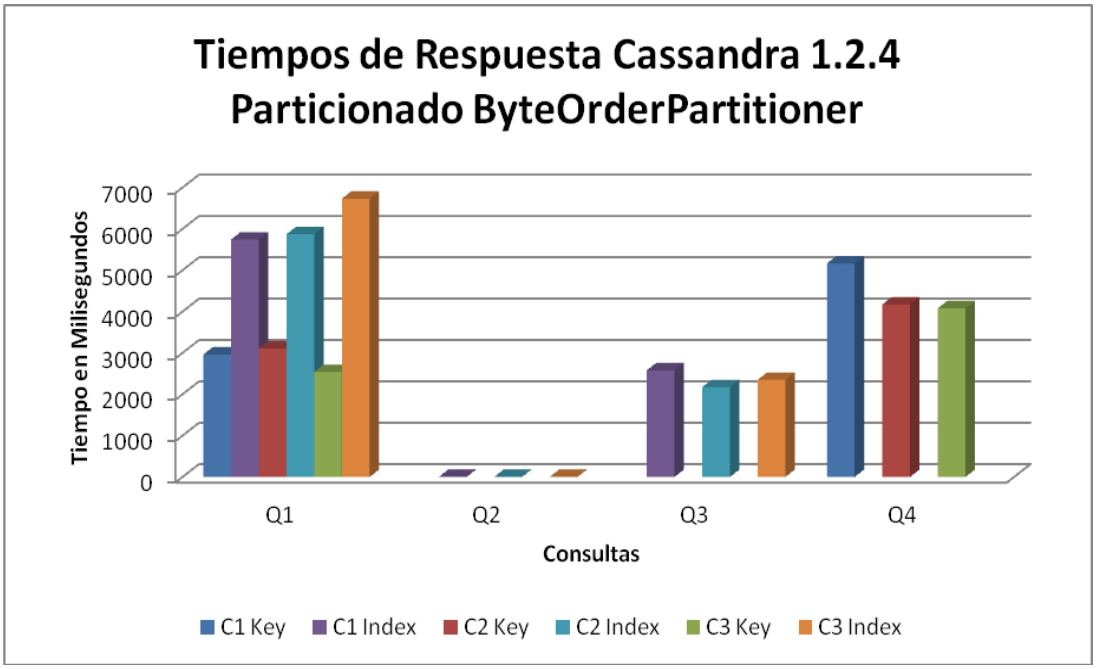


Ilustración 40: Gráfico de Tiempos de Respuestas Cassandra 1.2.4 con particionado ByteOrderPartitioner

Interpretación:

En la Ilustración 39: Tiempos de respuestas Cassandra 1.2.1 particionado ByteOrderPartitioner e Ilustración 40: Gráfico de Tiempos de Respuestas Cassandra 1.2.4 con particionado ByteOrderPartitioner se muestra la gráfica de los tiempos de respuestas de las 4 consultas efectuadas en Apache-Cassandra en la versión 1.2.1 y 1.2.4 respectivamente.

Se dividen en Key e Index, lo que quiere decir, que se confeccionó una consulta destinada a evaluar los resultados obtenidos en relación a las claves y otras a los índices secundarios creados. Sin embargo, según el modelamiento de la base de datos y la confección de clave primaria, se da el caso que para la consulta Q2 y Q3 no fue posible elaborarla en relación a la clave, solamente a los índices.

Un punto importante, y que queda en evidencia con las pruebas realizadas, es el hecho que ambas versiones sus tiempos de respuestas son menores cuando se ejecuta una búsqueda por las claves que el arrojado por el uso de índices secundarios, independiente que la consulta se haya realizado por una parte de la clave.

9.2.2 Análisis de desempeño según Claves e Índices secundarios

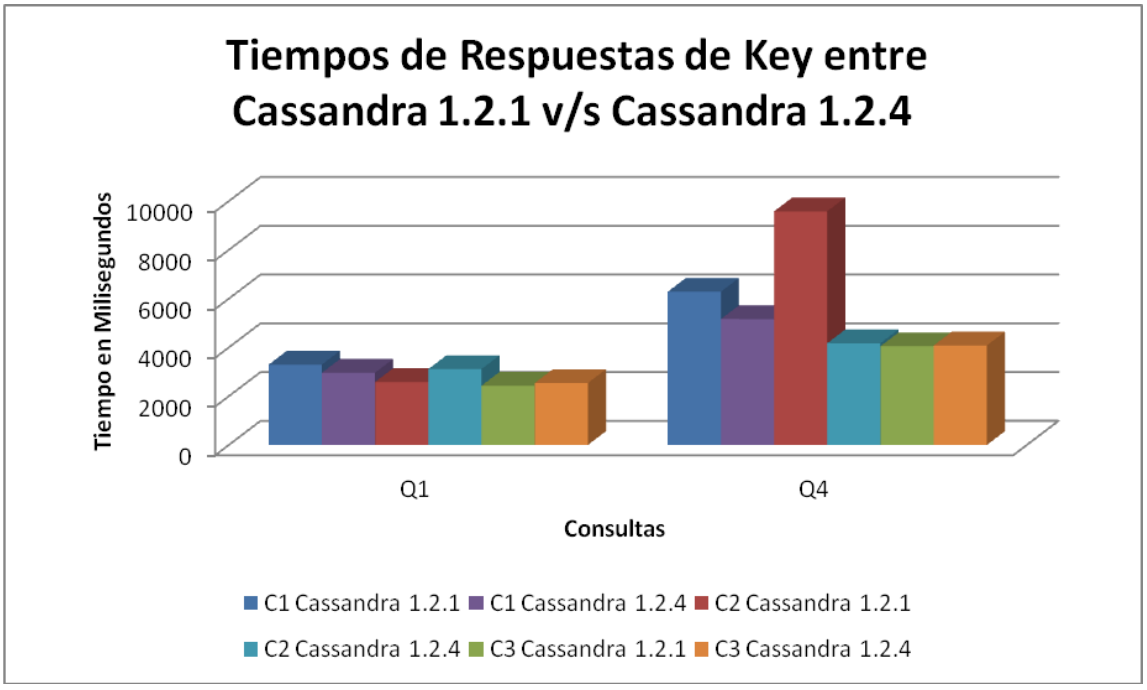


Ilustración 41: Gráfico de Tiempos de Respuestas en relación a consultas con claves entre Cassandra 1.2.1 v/s Cassandra 1.2.4

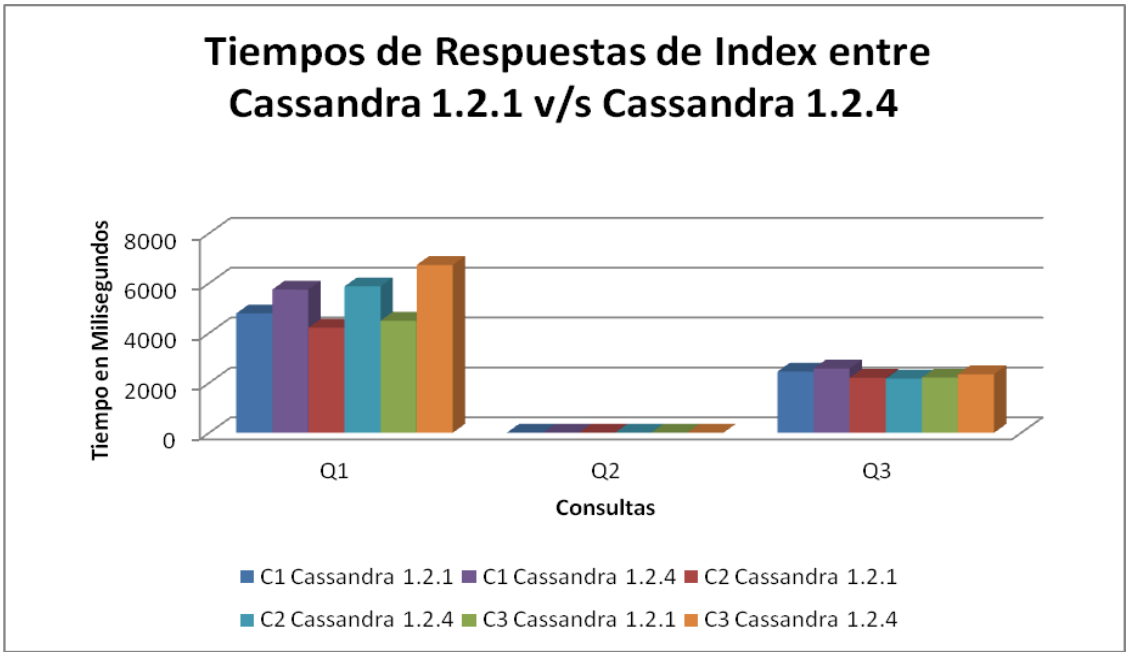


Ilustración 42: Gráfico de Tiempos de Respuestas en relación a consultas con índices entre Cassandra 1.2.1 v/s Cassandra 1.2.4

Interpretación.

En la Ilustración 41: Gráfico de Tiempos de Respuestas en relación a consultas con claves entre Cassandra 1.2.1 v/s Cassandra 1.2.4 se grafican los tiempos de respuestas de las búsquedas por las claves de las columnas en las dos versiones comparadas.

Como se explicó anteriormente, sólo se confeccionaron consultas donde se utilizaran la clave en la búsqueda los cuales corresponden a las consultas Q1 y Q4.

Como se visualiza, en Q1 la variación es bastante leve en las tres ejecuciones. Sin embargo, en Q4, en la segunda vez que se ejecuta la consulta, en la versión 1.2.1 el tiempo aumenta considerablemente, ya que en primera instancia demora 6283 milisegundos, a diferencia de la segunda vez en consultar, que se elevó a 9582.

En la Ilustración 42: Gráfico de Tiempos de Respuestas en relación a consultas con índices entre Cassandra 1.2.1 v/s Cassandra 1.2.4 se comparan los resultados obtenidos en relación a los índices secundarios. En estas pruebas la versión 1.2.1 arrojó levemente un mejor resultado que la versión 1.2.4.

9.2.3 Apache-Cassandra v/s SqlServer2008

Los tiempos de la Tabla 12 se midieron en milisegundos.

	Cassandra			SqlServer		
	1.2.4			2008		
	C1	C2	C3	C1	C2	C3
Q1	3111	2962	2540	13946	1590	1230
Q2	5,1	4,6	4,8	12746	460	453
Q3	2580	2172	2349	14823	933	860
Q4	5164	4173	4081	29553	3366	2590

Tabla 12: Tiempos de respuesta Cassandra 1.2.4 v/s SqlServer

9.2.3.1 Análisis de desempeño primera ejecución

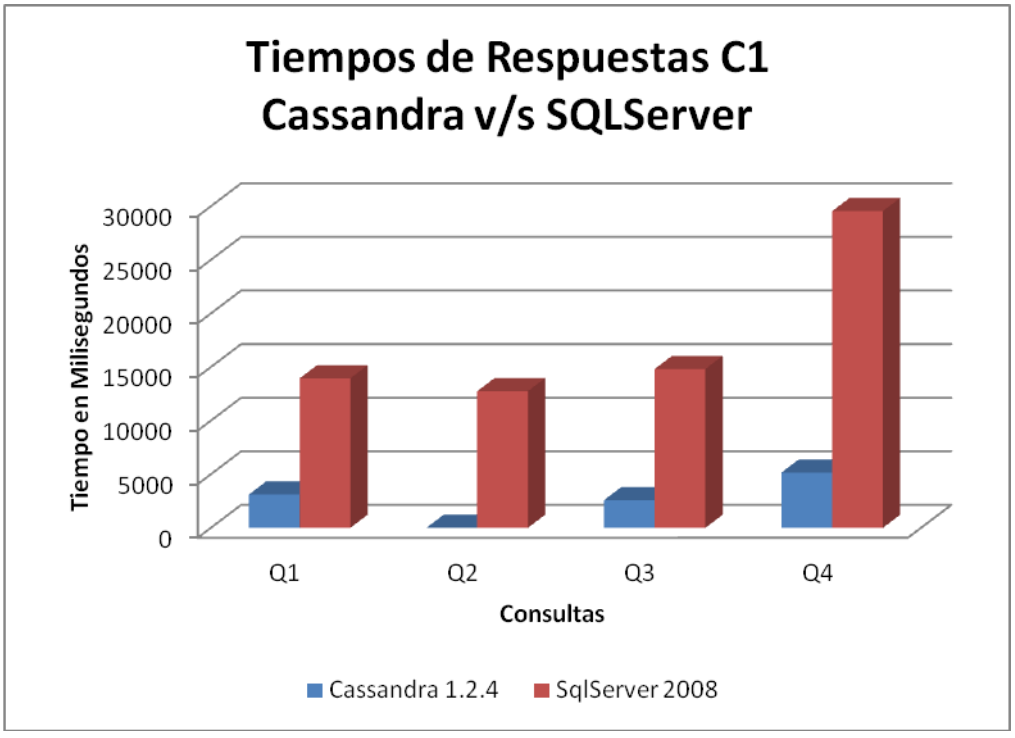


Ilustración 43: Gráfico Barras C1 entre Cassandra y SQLServer

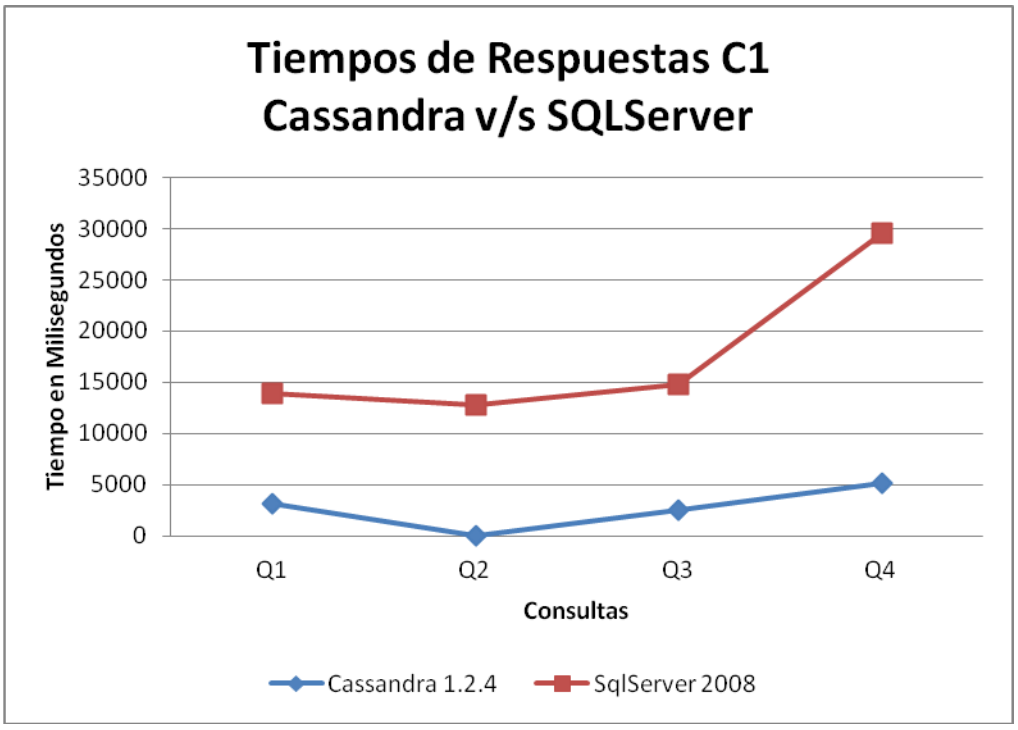


Ilustración 44: Gráfico Líneas C1 entre Cassandra y SQLServer

Interpretación

En la Ilustración 43: Gráfico Barras C1 entre Cassandra y SQLServer e Ilustración 44: Gráfico Líneas C1 entre Cassandra y SQLServer, se grafica los tiempos de respuestas obtenidos al realizar por primera vez la consultas. El comportamiento de Cassandra es considerablemente superior en relación a los valores arrojados por SqlServer, manteniendo su mejora en las 4 consultas.

En Q1 su tiempos de respuestas equivale a un 76% mas lento que SQLServer.

En Q2 la superioridad de Cassandra es muy notaria, ya que demora 5,1 milisegundos en retornar el valor consultado, a diferencia de SQLServer que lo realiza en 12.746 milisegundos, significando un 99,95% mas lento que Cassandra. En esta consulta quede en evidencia la rapidez y el buen funcionamiento de los índices secundarios al momento de realizar un búsqueda.

Finalmente en las Q3 y Q4, Cassandra es 82,52% más rápido al retornar lo consultado.

9.2.3.2 Análisis de desempeño segunda ejecución

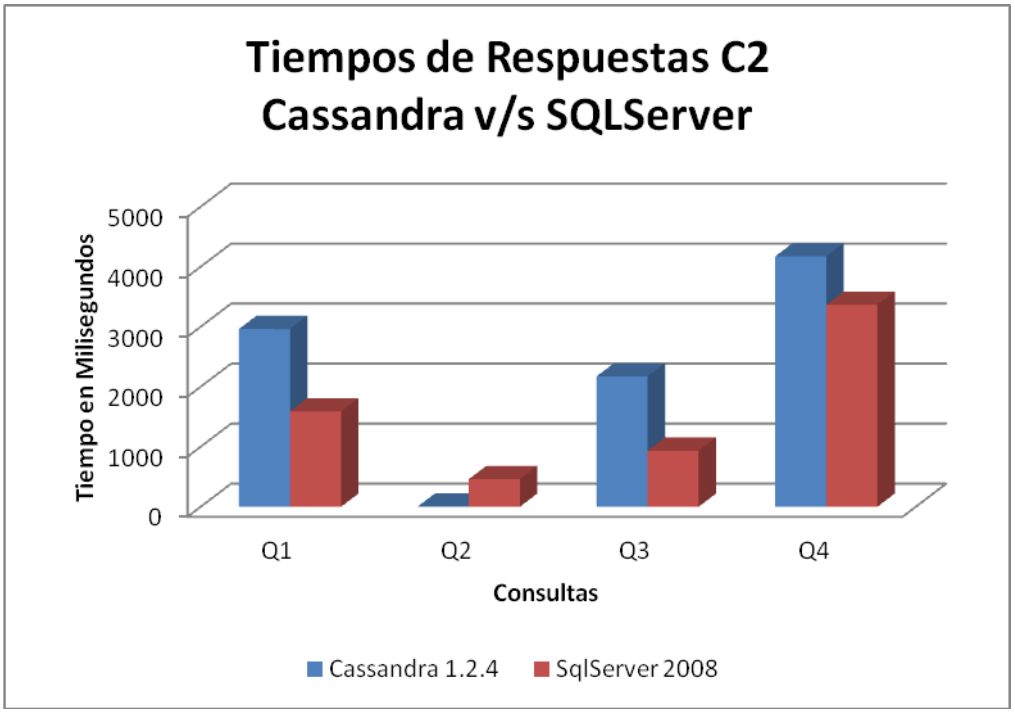


Ilustración 45: Gráfico Barras C2 entre Cassandra y SQLServer

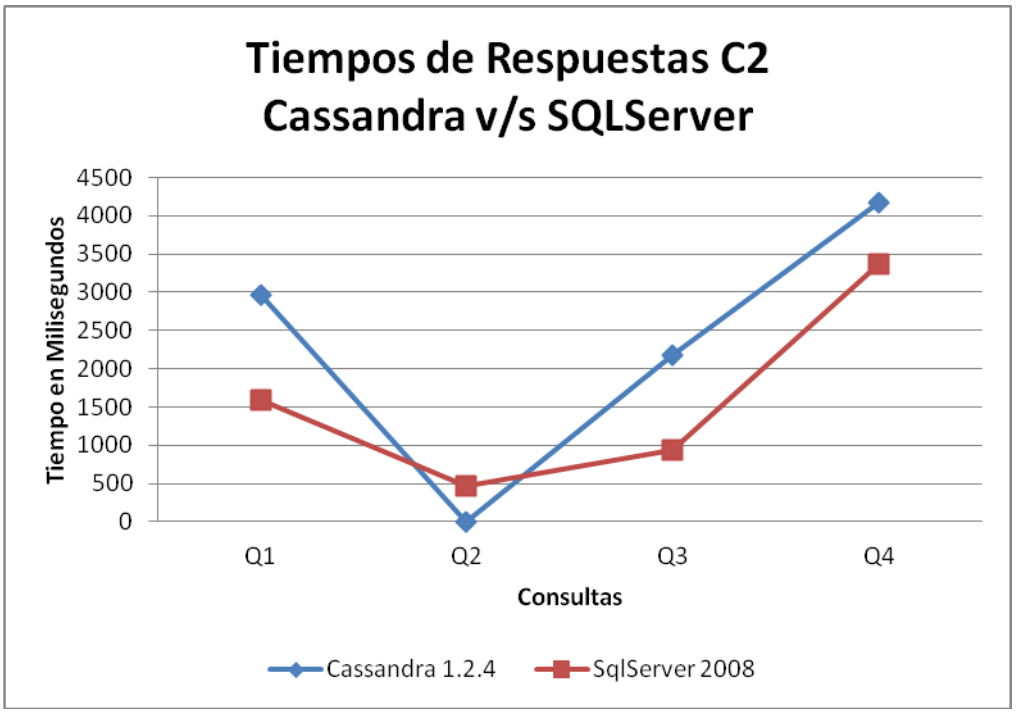


Ilustración 46: Gráfico Líneas C2 entre Cassandra y SQLServer

Interpretación

En las Ilustración 45: Gráfico Barras C2 entre Cassandra y SQLServer e Ilustración 46: Gráfico Líneas C2 entre Cassandra y SQLServer se grafican el comportamiento de las bases de datos.

El escenario de la primera consulta cambia rotundamente al ser ejecutada por segunda vez.

En ambos gráficos se puede apreciar las notorias mejoras alcanzadas por SQLServer en su segunda ejecución. Logrando a ser superior a la Cassandra en sus tiempos de respuestas.

Q1 es un 51%, Q3 un 57% y en Q4 un 19%, más rápida que la efectuada en Cassandra. Pero, Q2 sigue siendo más eficiente que SQLServer, con la diferencia que corresponde a un 99%.

Lo anterior es debido a que las consultas son almacenadas en el caché y en las estadísticas realizadas por SQLServer, y no son accedidas desde el disco directamente.

9.2.3.3 Análisis de desempeño tercera ejecución

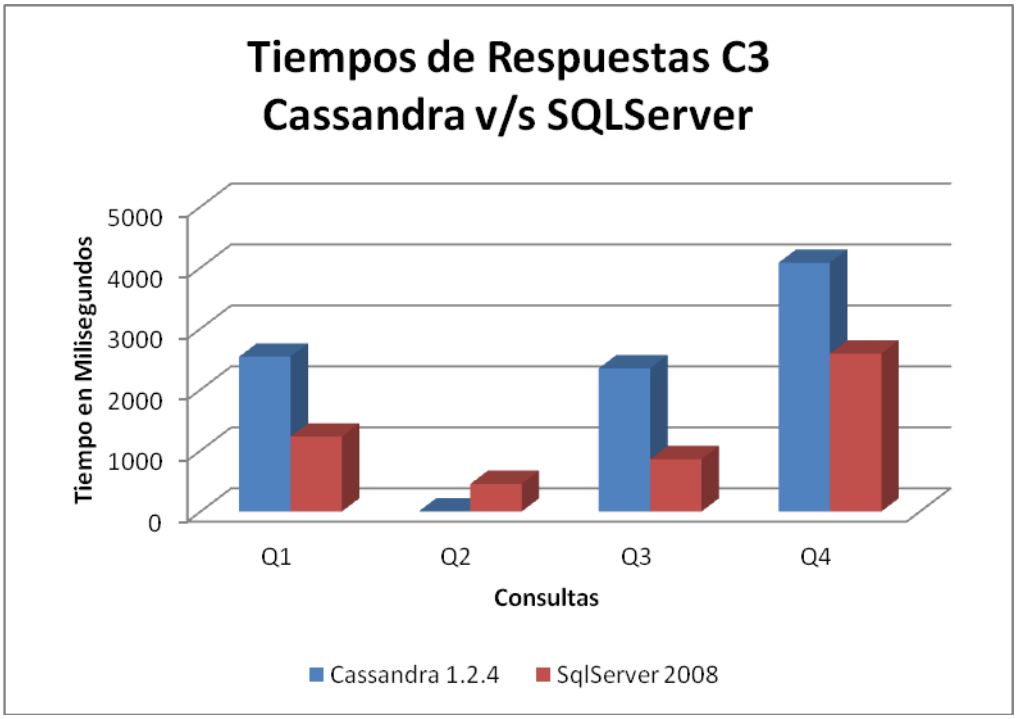


Ilustración 47: Gráfico Barras C3 entre Cassandra y SQLServer

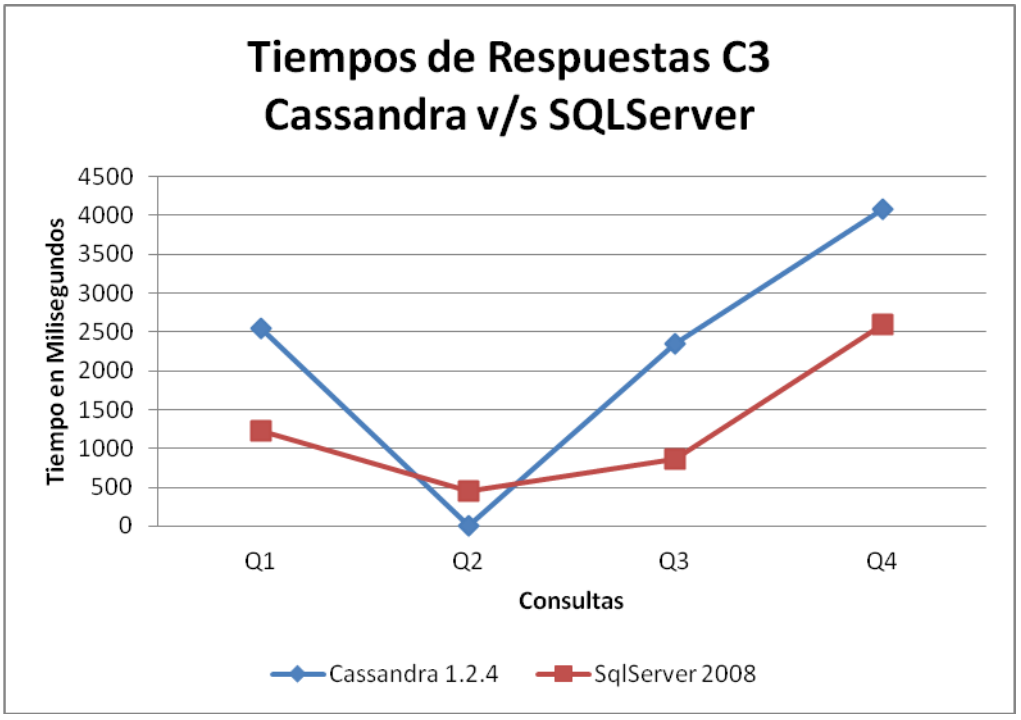


Ilustración 48: Gráfico Líneas C3 entre Cassandra y SQLServer

Interpretación

En la Ilustración 47: Gráfico Barras C3 entre Cassandra y SQLServer e Ilustración 48: Gráfico Líneas C3 entre Cassandra y SQLServer se grafican los resultados obtenidos al ejecutar por tercera vez las consultas.

En ellos se puede apreciar que se mantiene la tendecia ocurrida en la segunda ejecución. SQLServer predomina en 3 de las 4 consultas realizadas. Por lo tanto, el comportamiento de este motor es muy eficaz al momento de la generación de las estadísticas y almacenamiento en caché tras haber realizado anteriormente consultas sobre ellas.

Pero también, en Q2 sigue siendo predominante Cassandra, logrando nuevamente ser superior en un 98%.

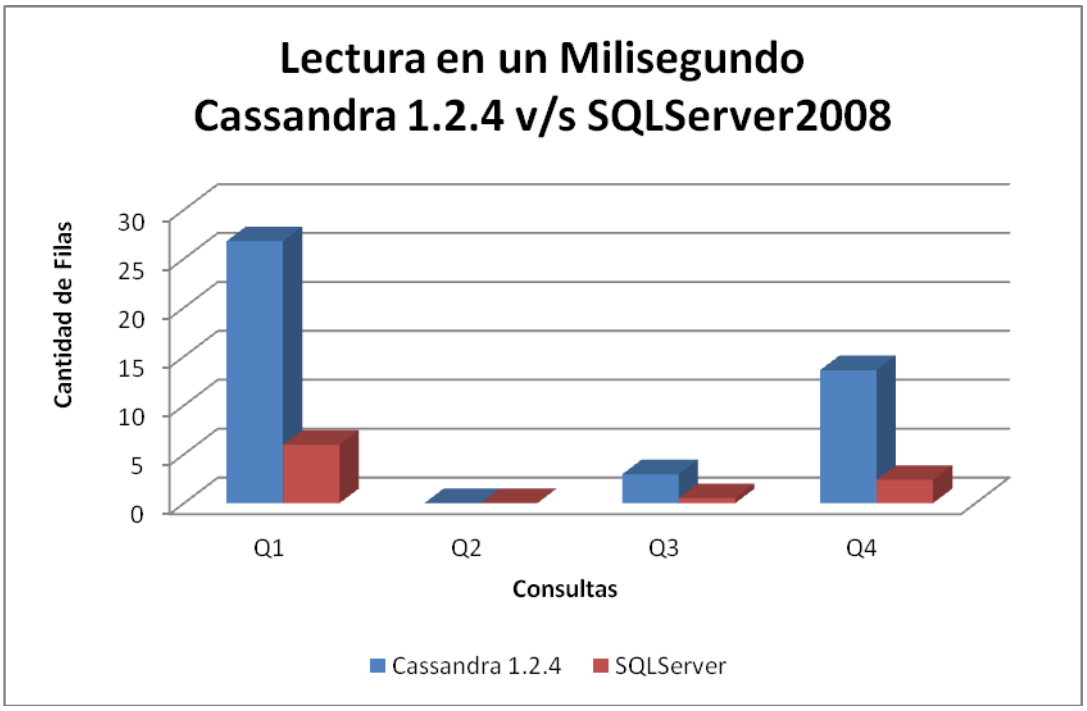


Ilustración 49: Lectura en un Milisegundo Cassandra 1.2.4 v/s SQLServer 2008

Interpretación.

La Ilustración 49: Lectura en un Milisegundo Cassandra 1.2.4 v/s SQLServer 2008 se la grafica la cantidad de filas o tuplas leídas en 1 milisegundo entre Cassandra 1.2.4 y SQLServer2008.

El gráfico se confeccionó considerando solamente la primera ejecución de ambas base de datos, y los datos se obtuvieron dividiendo la cantidad de filas por el tiempo que demoró en la ejecución.

Como se aprecia Cassandra lee mayor cantidad de filas que SQLServer en un milisegundo.

En Q1 lee un total de 26,8 filas por milisegundos, por lo tanto, en un segundo 26.781 filas, a diferencia de SQLServer que realiza 5,97 por milisegundos, y 5974 por segundos.

En Q2 como los valores son demasiado pequeños al devolver un registro, no se señala el cálculo.

En Q3 lee un total de 2,95 filas por milisegundos, por lo tanto, en un segundo 2948 filas, a diferencia de SQLServer que realiza 0,51 por milisegundos, y 513 por segundos.

En Q4 lee un total de 13,6 filas por milisegundos, por lo tanto, en un segundo 13593 filas, a diferencia de SQLServer que realiza 2,3 por milisegundos, y 2375 por segundos.

10 CONCLUSIÓN

El estudio arrojó resultados favorables para ambos motores de Base de Datos pero en escenarios distintos.

En la primera ejecución de las consultas, demostró mayor eficiencia de base de datos NoSQL, obteniendo los tiempos de respuestas de todas las consultas considerablemente más rápida que el motor de las base de datos relacionales. Exceptuando la consulta 2, que en las 3 ejecuciones se comportó de mejor manera en ambos modelos.

De igual manera, Cassandra demostró solidez en la obtención de registros únicos, alcanzando altas velocidades, como ocurrió en la consulta 2, que retornaba un paciente en particular. También es importante mencionar que la búsqueda por un parte de la clave es superior a la búsqueda que se puede realizar por un índice secundario.

La generación de la clave que identificaría la fila fue la correcta, ya que al mezclar la fecha de ingreso del paciente y un correlativo, la búsqueda parcial por clave fue bastante mejor que la realizada por un índice.

Si bien el modelo de Cassandra contemplaba un 40% menos de registros, las consultas se confeccionaron para medir de igual manera ambos modelos. Sin embargo, para no quedar con la interrogante de que la disminución de registros fue la conllevó a unos tiempos de respuestas favorables, se realizó una prueba insertando todos los datos del DataMart del modelo relacional para corroborar si fue o no el factor de éxito, lo cual siguió dando mejores resultados que los arrojados por el motor de base de datos relacional. Ver ANEXO E: PRUEBAS SEGUNDO MODELO DE CASSANDRA

En el capítulo de Análisis se mencionó que la reducción de registros no necesariamente significaría un decremento en los tiempos de respuesta, lo cual quedó en evidencia al momento de ejecutar por segunda y tercera vez las consultas, ya que en este escenario, el motor del modelo relacional arrojó resultados favorables, en cuanto a consultas complejas y/o prolongadas en un rango de tiempo, pero la brecha de ambos no alcanzó a superar los resultados arrojados en la primera ejecución. En este punto es importante mencionar que SQLServer se comportó favorablemente en la segunda y tercera vez que se realizaron las

consultas ya que los datos extraídos no fueron leídos sobre el disco, sino de la memoria caché que almacena el motor, por lo tanto, no es de gran relevancia para este estudio.

Si bien SQLServer obtuvo mejores resultados en la segunda y tercera ejecución, es importante señalar que existen consultas que se realizan solamente una vez, por lo que implica, que no siempre podrá obtener mejores tiempos, ya que no estará en función a las estadísticas y caché almacenados por el motor. Con ese punto de vista, la base de datos de Cassandra sería la mejor opción. Lo cual no sería así en una transaccional, donde constantemente se consultan los datos.

Algunos factores que implicaron un menor rendimiento de Cassandra en relación a SqlServer en la segunda y tercera ejecución de las consultas son:

- Cassandra está diseñada para funcionar en clústers de a lo menos 3 nodos. Si bien es posible ejecutarlo en uno, no es lo ideal.
- El universo de datos utilizados no es el ideal para Cassandra, ya que las bases de datos NOSql fueron diseñadas para trabajar sobre la franja de los Terabytes, que es la línea en donde la escalabilidad se vuelve esencial y las bases de datos relacionales no logran desempeñarse bien (diseñada sólo para Gigabytes).

Por otra parte, los aspectos favorables de Cassandra que este estudio comprobó son:
-Facilidad de Configuración: Activar la funcionalidad de estructura distribuida de Cassandra es simple y sólo requiere de dos pasos. Esto permite que la tolerancia a fallos por replicación de datos sea una tarea simple, ya que no sólo es sencilla de configurar sino que también es realizada automáticamente por Cassandra.

También es importante mencionar la disponibilidad, que debido a la replicación, es asegurada a lo largo de los nodos, haciendo que las escrituras no tengan que esperar por otras operaciones de escrituras, alcanzando un notable desempeño en esas tareas.

Cassandra es una base de datos clave-valor, por lo que su conjunto de funcionalidad para la consulta de datos es limitado e impidiendo la ejecución de complejas consultas CQL, es más ni siquiera permite consultas de agregación. Es por ello que es necesario disponer de herramientas externas o frameworks que permitan equiparar, en ese aspecto, a Sql Server. Para ello, el proyecto Pig provee de una capa superior que permite la ejecución de complejas

consultas de agregación, entre otras funcionalidades. Sin embargo, para este estudio no corresponde a un factor negativo, ya que el análisis de datos en un DataMarts también es realizado con herramientas externas.

Un aspecto que dificultó el estudio, corresponde a la escasa documentación sobre la API de java y de la base de datos de Cassandra.

11 BIBLIOGRAFÍA

[0]. **Iocit.com** http://www.iocit.com. *IOCIT.* [En línea] [Citado el: 07 de Mayo de 2013.] http://www.iocit.com/40-zetabytes-para-el-2020/

[1]. **Hernampérez, Rafael.** http://rafinguer.blogspot.com. *Base de Datos.* [En línea] [Citado el: 06 de Febrero de 2013.] http://rafinguer.blogspot.com/2010_03_01_archive.html.

[2]. Histinf. Historia de la Informática[En línea] 4 de Enero de 2011. [Citado el: 19 de Febrero de 2013.] http://histinf.blogs.upv.es/2011/01/04/historia-de-las- bases-de-datos/.

[3] **Silberschatz, Abraham, Korth, Henry F y Sudarshan, S.** *Fundamentos de Base de Datos.* Madrid : MacGraw-Hill, 2002. ISBN: 0-07228637.

[4] Introducción a las Bases de Datos [En línea] [Citado el: 19 de Febrero de 2013.] http://informatica.uv.es/docencia/biblioguia/BD/ficheros/tema1.pdf. PDF.

[5] Db-engines.[En línea][Citado el: 28 de Abril de 2013] http://dbengines.com/en/ranking

[6] Dbpedias. [En línea][Citado el: 28 de Abril de 2013] http://dbpedias.com/wiki/NoSQL:Consistency_Models_in_Non-Relational_Databases

[7]. Wikipedia. [En línea] 28 de Enero de 2013. [Citado el: 19 de Febrero de 2013.] http://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_bases_de_datos.

[8]. **Mendelzon, Ale.** *Introducción a las base de datos relacionales.* Buenos Aires : PRENTICEE HALL, 2000. ISBN: 987-97892-2-9.

[9]. **Ramez, Elmasri y Shamkant, Navathe.** *Fundamentos de Sistemas de Base de Datos.* Madrid : PEARSON EDUCACION, 2007. ISBN: 978-84-7829-085-7.

[10]. **O'Brein, James y Marakas, George.** *Sistemas de Información Gerencial.* s.l. : Mc Graw Hill, 2006.

[11]. **Cesares, Claudio.** personal.lobocom. [En línea] [Citado el: 16 de Febrero de 2013.] http://personal.lobocom.es/claudio/gen006.htm..

[12]. **Imhoff, Claudia, Galemmo, Nicholas y Geiger, Jonathan.** *Mastering Data Warehouse Design- Relational and Dimensional Techniques.* Indianapolis : Wiley Publishing, 2003. ISBN: 0-471-32421-3.

[13]. **Ralph, Kimball y Ross, Margy.** *The Data Warehouse toolkit: the complete guie to dimensional modeling.* New York : John Wiley & Sons, Inc, 2002. ISBN: 0-471-20024-7.

[14]. **Oracle.** Oracle. *Data Warehousing Concepts.* [En línea] 2002. [Citado el: 5 de Marzo de 2013.] http://docs.oracle.com/cd/B10501_01/server.920/a96520/concept.htm#49871.

[15]. **Caniupán, Mónica.** *Base de Datos Operacionales. Tópicos Avanzados de Base de Datos.* [PDF] Concepción : Universidad Bio-Bio, 2012.

[16]. **Moss, Larissa T y Atre, Shaku.** *Business Intelligence Roadmap: The complete Project Lifecycle for Decision-Support Aplications.* Boston : Addison Wesley, 2003. ISBN: 0-201-78420- 3.

[17]. **Mendez, A, y otros.** Fundamentos de Data Warehouse. *REPORTES TÉCNICOS EN INGENIERÍA DEL SOFTWARE.* CAPIS-EPG-ITBA (http:// http://www.itba.edu.ar/capis/rtis), 2003, ISSN: 1667-5002.

[18]. **Tiwari, Shashank.** *Professional NoSQL.* Canadá : John Wiley & Sons, Inc, 2011. ISBN: 978-0-470-94224-6.

[19]. **Lith, Adam y Mattson, Jakob.** *Investigating storage solutions for large data: A comparasion of well perfoming and scalable data storage solutions for real time extraction and batch insertion of data.* [PDF] Goterborg : Chalmers University of Technology Department of Computer Science and Engineering, 2010. Pags. 70.

[20]. **Strozzit.it.** NOSQL: Relational Database Management System: Home Page. [En línea] 2008. [Citado el: 21 de Febrero de 2013.] http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page.

[21] Alegsa. [En línea] 2012 Argentina [Citado el: 29 de Abril de 2013] (http://www.alegsa.com.ar/Dic/RDBMS.php)

[22]. Lastfm. [En línea] 15 de Enero de 2013. [Citado el: 4 de Marzo de 2013.] http://www.lastfm.es/about.

[23]. **Amorós Martínez, Luis Miguel y Arbós Lino, Noemí.** *Tesis VirtualEPSC, el mundo virtual 2.0 del Campus del Baix Llobregat.* s.l. : Universitat Politécnica de Catalunya, 2010.

[24]. **Sutinen, Olli.** *NoSQL – Factors Supporting the Adoption of Non-Relational Databases.* s.l. : University of Tampere, 2010.

[25]. **Gilbert, S y Lynch, N.** Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. s.l. : SIGACT News 33, (Jun. 2002), págs. 51-59.

[26]. **Lamport, L, Shostak, R y Pease, M.** *The Byzantine Generals Problem.* s.l. : ACM Trans. Program. Lang, Jul. 1982. 382-401.

[27]. **Nube, Computación en la.** Computación en la Nube. [En línea] 2013. [Citado el: 19 de Febrero de 2013] http://www.computacionennube.org/.

[28]. **10gen.** 10gen. [En línea] Copyright © 2013 10gen, 2013. [Citado el: 19 de Febrero de 2013.] http://www.10gen.com/nosql-vs-sql.

[29] The Apache Software Foundation. [En línea] 2013. [Citado el: 19 de Febrero de 2013] http://hbase.apache.org

[30] Neo Technology, Inc. Neo4j[En línea]2013[Citado el: 19 de Febrero de 2013] http://www.neo4j.org/learn/graphdatabase

[31]. **Hewitt, Eben.** Cassandra The Definitive Guide. [aut. libro] Eben Hewit. *Cassandra The Definitive Guide.* United States of America : O'REILLY, 2010, pág. xvii.

[32]. **Garcete, Adrian.** *Base de Datos Orientados a columnas.* Paraguay : Universidad Católica "Nuestra Señora de la Asunción", Septiembre 2012.

[33]. **Datastax.** Datastax. *Documentación Oficial Cassandra 1.2.2.* [En línea] [Citado el: 5 de Marzo de 2013.] http://www.datastax.com/docs/1.2/index.

12 ANEXO A: INSTALACIÓN Y CONFIGURACIONES

12.1 Apache-Cassandra

12.1.1 Requisitos:

Antes de instalar se debe verificar lo siguiente

- Oracle Java SE Runtime Environment (JRE)6. Java 7 no es recomendado
- Instalación de Java NAtive Access (JNA)

12.1.2 INSTALACIÓN JRE

Cassandra está programado en Java, por lo tanto, requiere de Java Runtime Environment(JRE). Como mínimo usar la versión 1.6.0_32 de JRE, Java 7 no es recomendado.

1.- Comprobar la versión existente del JRE de Oracle. Se puede revisar la versión escribiendo en consola el comando java -version

```
root@Laptop:~# java -version
java version "1.6.0_24"
OpenJDK Runtime Environment (IcedTea6 1.11.5) (6b24-1.11.5-0ubuntu1~12.04.1)
OpenJDK Server VM (build 20.0-b12, mixed mode)
```

2.- Actualizar el JRE desde de la consola, con las siguientes sentencias:
Ver Ilustración 50: Agregar Repositorio

2.1.-
```
$ sudo add-apt-repository ppa:webupd8team/java
```

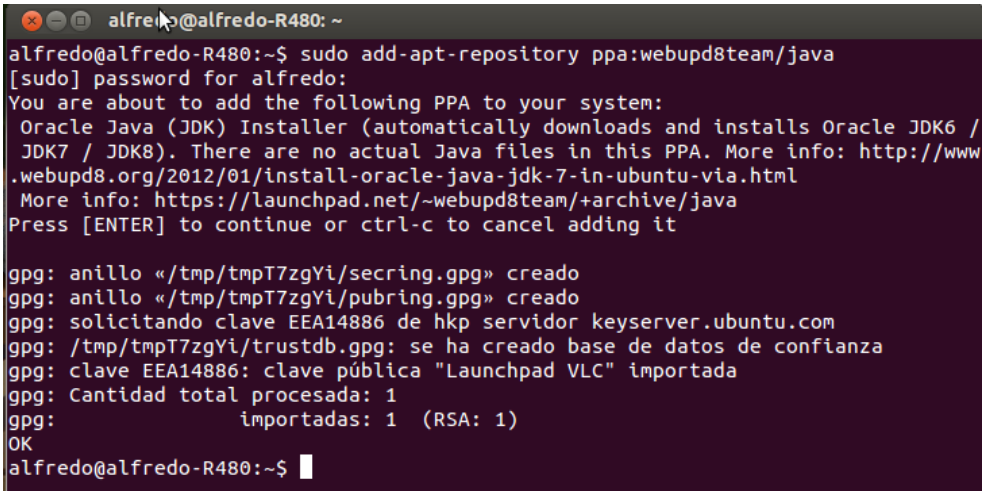


Ilustración 50: Agregar Repositorio

2.2.- $ sudo apt-get update

2.3.- $ sudo apt-get install oracle-java6-installer Ver. Ilustración 51: Instalar Java 6.

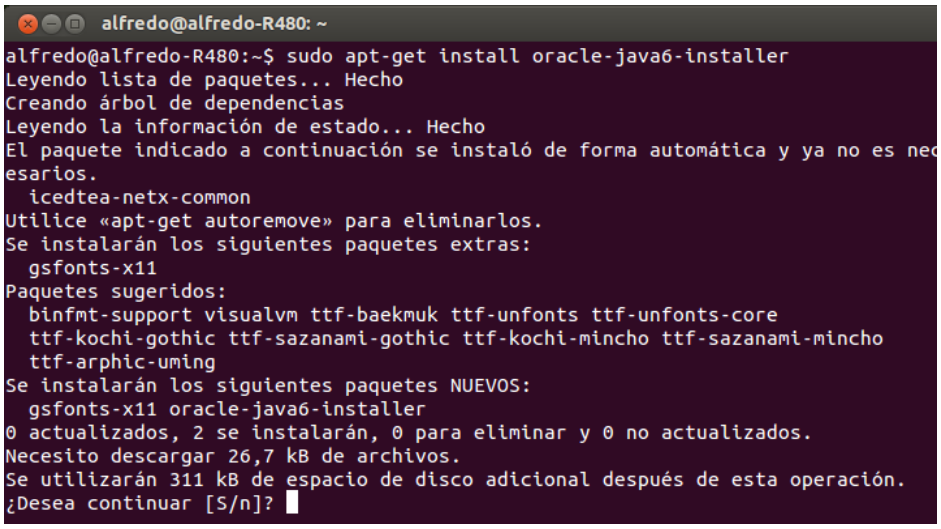


Ilustración 51: Instalar Java 6

2.4 Seguir los pasos y aceptar los términos.

Ilustración 52: Configuración Java 6

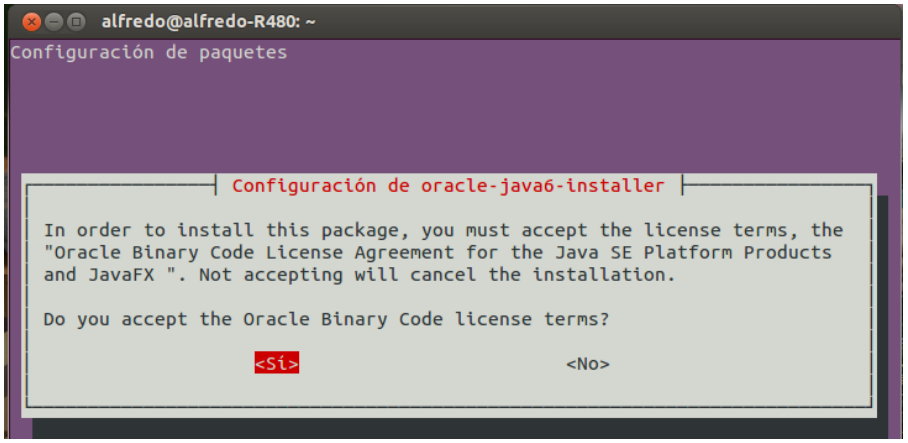


Ilustración 53: Aceptar Licencia Oracle

2.5 Volver a realizar el paso 1.

2.6 Para poder iniciar eclipse con el nuevo JRE actualizado se deberá modificar el archivo eclipse.ini que se encuentra /etc.

```
$ sudo gedit /etc/eclipse.ini
```

Agregarles las siguientes líneas al documento y guardar.

```
-vmargs

-D java.library.path= /usr/lib/jni
```

Conceder permiso para lectura e iniciar normalmente Eclipse.

```
$ sudo chmod 755 /etc/eclipse.ini
```

12.1.3 INSTALACION DE JNA

El Java Native Access puede mejorar el uso de la memoria de Apache-Cassandra. Una vez instalado y configurado, Linux no intercambia la JVM, por lo que evita problemas relacionados con el rendimiento.

```
$ sudo apt-get install libjna-java
```

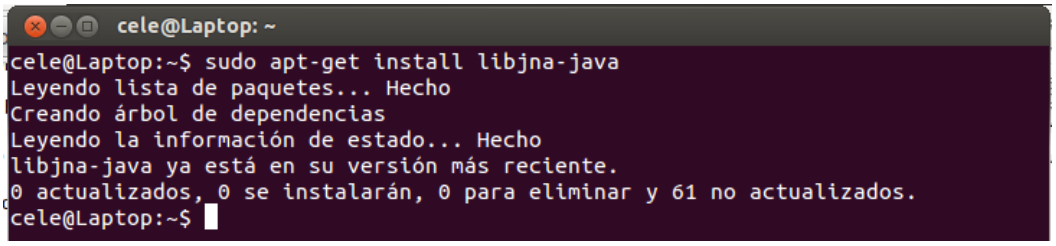


Ilustración 54: Estado JNA

12.2 INSTALACIÓN CASSANDRA

La instalación de esta herramienta se realiza de varias formas, que se pueden encontrar en la página oficial de Apache-Cassandra http://wiki.apache.org/cassandra/ como también en http://www.datastax.com/docs/1.2/index.

Proceso de instalación:

- Descarga versión de Cassandra a utilizar desde la página oficial http://cassandra.apache.org/download/, en este caso es la apache-cassandra-1.2.1-bin.tar.gz que consiste en Cassandra binary tarball distribution

Ilustración 55: Sitio Oficial Cassandra

- Se navega en la shell hasta la carpeta de descarga de Apache-Cassandra y se descomprime el .tar.gz como se muestra en la Ilustración 56**: Descomprimir Apache-Cassandra**

```
cele@Laptop:~$ sudo tar -xvzf apache-cassandra-1.2.1-bin.tar.gz
```

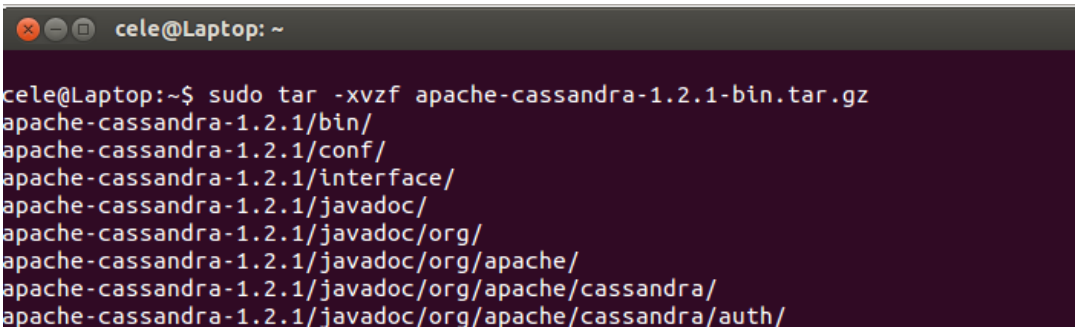


Ilustración 56: Descomprimir Apache-Cassandra

12.2.1 Configuración memoria(opcional).

Antes de iniciar el servicio se puede configurar el uso de la memoria asignada.

Este es un paso que se puede obviar, ya que Apache-Cassandra asigna una memoria basada en la memoria física del sistema, utilizando en algunos caso ¼ o ½ de la memoria disponible, por lo tanto, si se desea se puede modificar el archivo cassandra-env.sh, que se encuentra en el directorio /conf.

Ver **Ilustración 57: Directorio cassandra-env.sh**

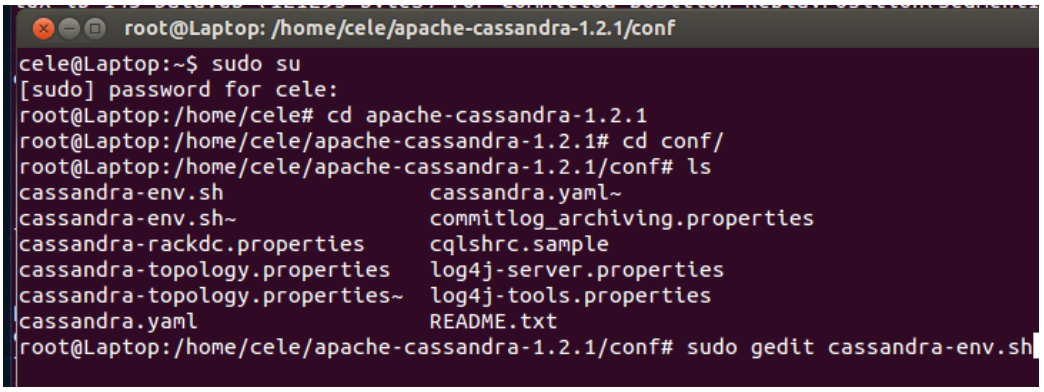


Ilustración 57: Directorio cassandra-env.sh

Descomentar y modificar lo señalado en la Ilustración 58**: Cassandra-env.sh**, asignándole la memoria deseada para que sea utilizada.

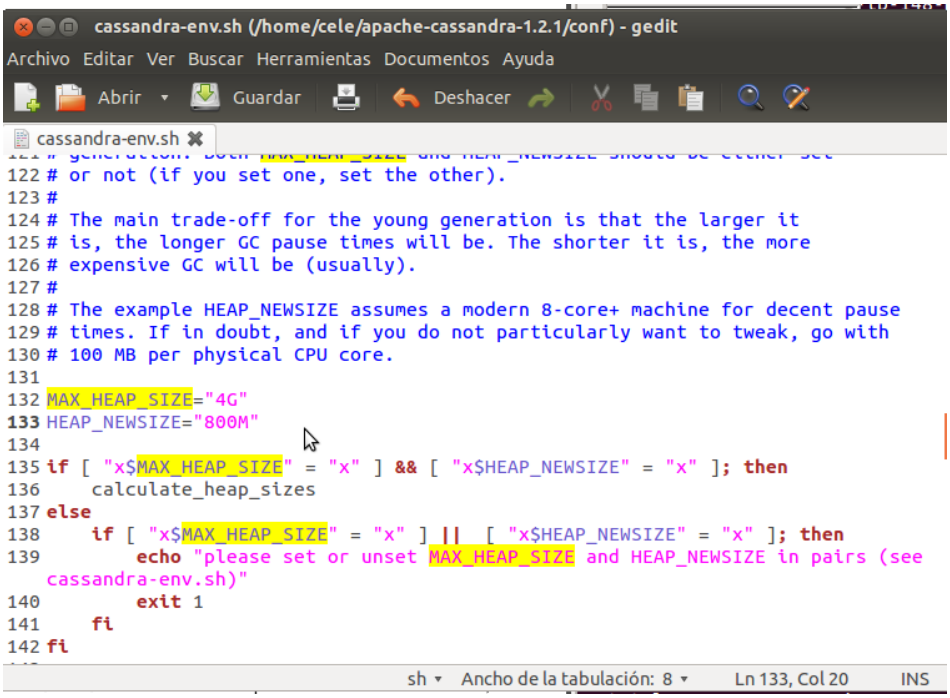


Ilustración 58: Cassandra-env.sh

Como regla general, se debe establecer HEAP_NEWSIZE a ¼ de MAX_HEAP_SIZE.

12.3 INICIAR APACHE-CASSANDRA

Dirigirse al directorio descomprimido de Apache-Cassandra e iniciar desde /bin, se puede realizar de dos maneras

```
root@Laptop:/home/cele/apache-cassandra-1.2.1# bin/cassandra
```

En el caso que no generar algún tipo de error, el servicio se iniciará en primer plano. Para detenerlo se debe presionar “CTRL + C”

```
root@Laptop:/home/cele/apache-cassandra-1.2.1# bin/cassandra -f
```

Con la sentencia anterior el servicio se inicia en segundo en segundo plano. Para detenerlo se puede realizar mediante el siguiente comando 'pkill -f Cassandra Daemon' o también consultado con

'ps aux | grep cassandra' , identificar el pid del servicio y luego 'kill <pid_cassandra>'

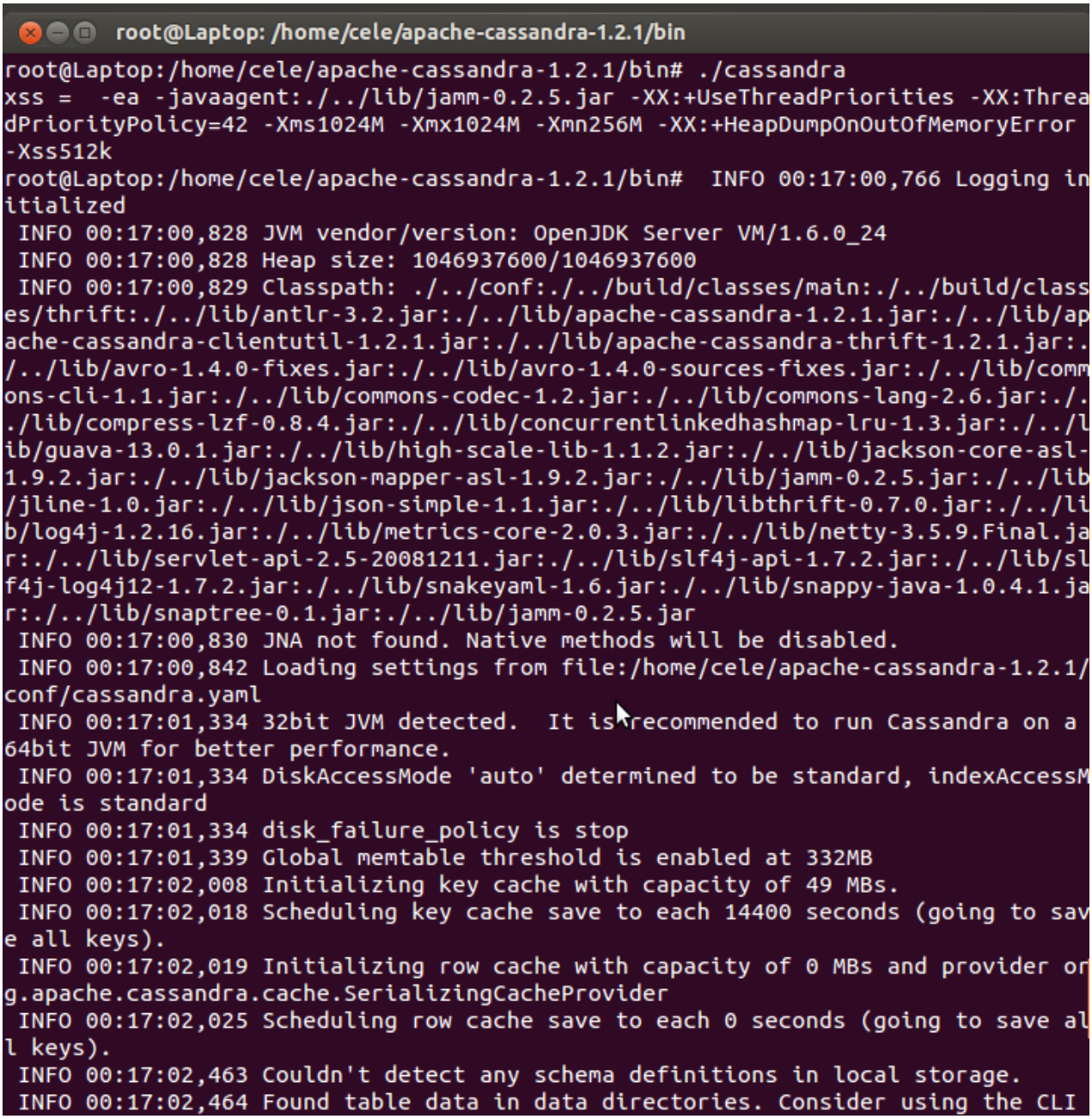


Ilustración 59: Inicio cassandra

12.4 USO CASSANDRA-CLI

Cassandra-cli es una interfaz de líneas de comandos interactiva de Cassandra.
Se inicia desde el directorio bin/:

```
cele@Laptop:~/apache-cassandra-1.2.1$ cd bin/
cele@Laptop:~/apache-cassandra-1.2.1/bin$ ./cassandra-cli
```

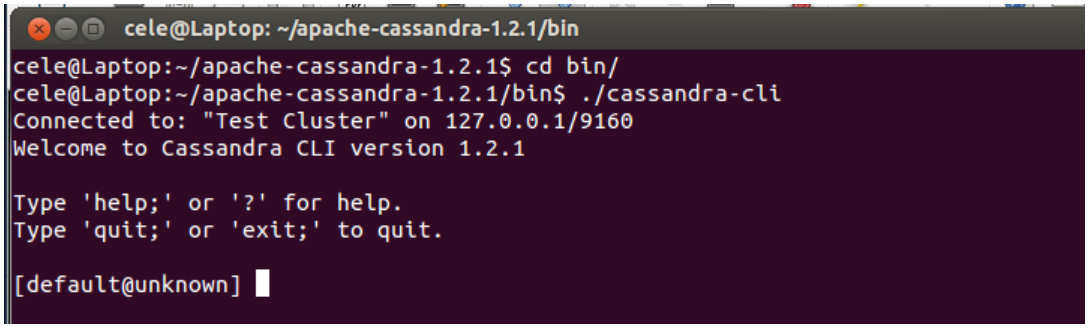


Ilustración 60: Conección cassandra-cli

12.5 CONFIGURAR CLUSTER CASSANDRA

La inicialización de un Clúster en Cassandra implica configurar cada nodo de manera que está dispuesto a unirse al clúster. Después de configurar los nodos, iniciar cada uno secuencialmente comenzando con el nodo de seed (s).

Configuración por defecto de Cassandra es el inicio de servicio en un solo nodo.

Puede utilizar inicializar un clúster de Cassandra con uno o más centros de datos. Los datos se replica a través de los centros de datos de forma automática y transparente; ningún trabajo de ETL es necesario mover datos entre diferentes sistemas o servidores. Puede configurar el número de copias de los datos en cada centro de datos y Cassandra se encarga del resto, replicando los datos para usted.

12.5.1 Requisitos

Antes de iniciar el clúster se debe configurar correctamente cada nodo a utilizar y haber realizado los siguientes puntos:

- Instalar Cassandra en un solo nodo.
- Elegir el nombre para el clúster.
- Obtener la dirección IP de cada nodo.
- Determinar los nodos seed.
- En el caso de tener un firewall abrir los puertos necesarios.

12.5.2 Configuración para un único data center

Instalación de un clúster con solo dos nodos en un centro de datos. Sin embargo, se recomienda tener más de un nodo seed por el centro de datos.

Establecer las propiedades en las máquinas de cada nodo en el archivo cassandra.yaml, ubicado en el directorio de apache-casandra /conf.

12.5.2.1 En este caso las ip son:

- 192.168.1.2
- 192.168.1.3(seed).

12.5.2.2 Detener la ejecución de Cassandra y borrar los datos

```
$ sudo aux | grep cassandra  (encontrar el <pid> proceso cassandra)

$ sudo kill <pid>(detener proceso)

$ sudo rm –rf /var/lib/cassandra/* (borra datos predefinidos de los directorios)
```

12.5.2.3 Modificar propiedades para cada nodo en el archivo cassandra.yaml

Node0

```
nombre_clúster: 'TestCluster'
num_tokens: 256
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parámetros:
        - Semillas: "192.168.1.3"
listen_address: 192.168.1.3
rpc_address: 0.0.0.0
endpoint_snitch: RackInferringSnitch
```

Node1

```
nombre_clúster: 'TestCluster'
num_tokens: 256
seed_provider:
  - class_name: org.apache.cassandra.locator.SimpleSeedProvider
    parámetros:
        - Semillas: "192.168.1.3"
listen_address: 192.168.1.2
rpc_address: 0.0.0.0
endpoint_snitch: RackInferringSnitch
```

12.5.2.4 Iniciar los nodos seed y luego el resto de los nodos

```
root@Laptop:/home/cele/apache-cassandra-1.2.1# bin/cassandra
```

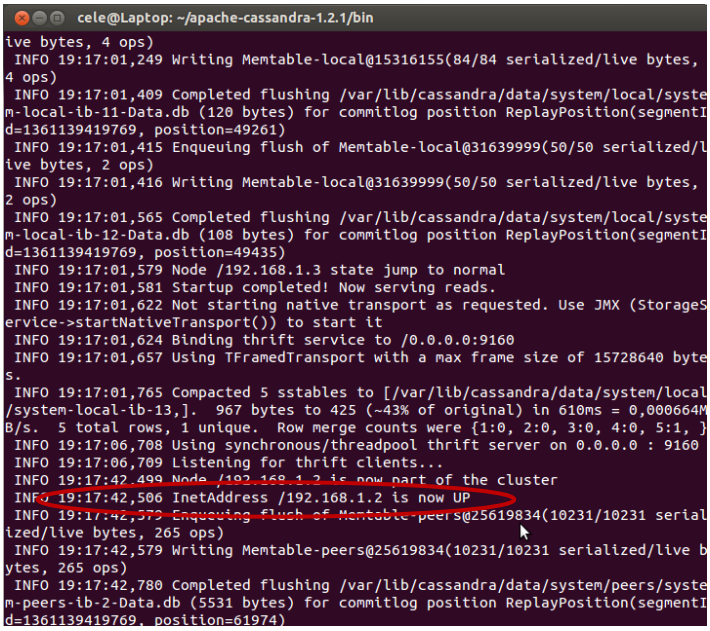


Ilustración 61: Monitoreo nodo1

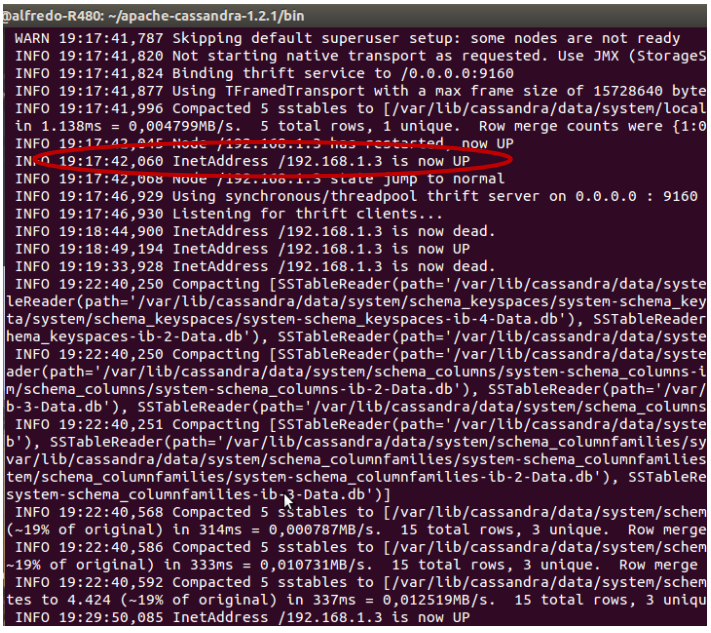


Ilustración 62: Monitoreo Nodo 1

12.5.2.5 Para comprobar que está en marcha ejecutar ./nodetool status

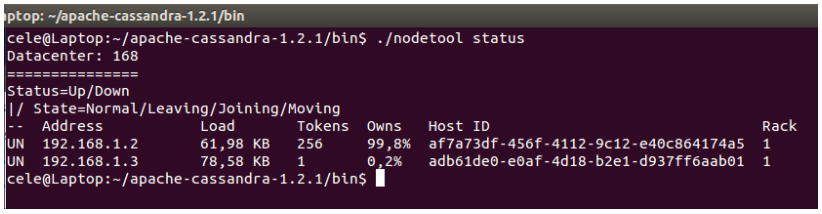


Ilustración 63: Estado nodo

13 ANEXO B: PROYECTO ECLIPSE

Eclipse es un IDE (*Integrated Development Environment*, entorno integrado de desarrollo) para Java muy potente. Es libre y fue creado originalmente por IBM. Se está convirtiendo en el estándar de facto de los entornos de desarrollo para Java. Otros IDE comerciales como JBuilder han anunciado que su próxima versión se basará en Eclipse. Y es que Eclipse no es tan sólo un IDE, se trata de un marco de trabajo modular ampliable mediante complementos (*plugins*). De hecho, existen complementos que nos permiten usar Eclipse para programar en PHP, Perl, Python, C/C++, etc. (Guia Ubuntu)

Para insertar los datos a la base de datos de Cassandra desde un txt, se realiza un proyecto desde eclipse, en que se deben importar las librerías necesarias para poder conectar e interactuar con Java y Apache-Cassandra

13.1 Crear Proyecto

Iniciar Eclipse → File → New → Proyect Java

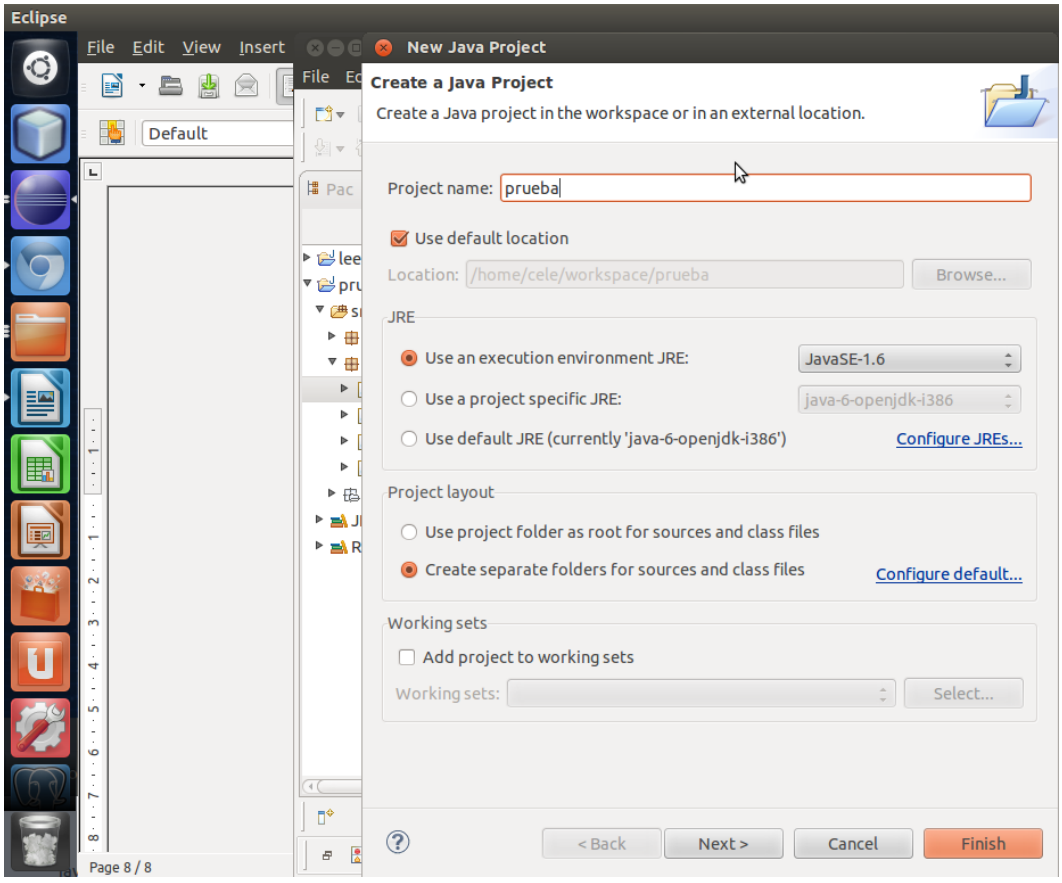


Ilustración 64: Creación Proyecto

Después de la creación de un proyecto se exportan las librerías que se utilizan JRE System Library → Build Path → Configure Build Path.

Seleccionar Export Jar Extern y añadir los .jar correspondientes.

En el directorio descomprimido de Apache-cassandra, carpeta /bin se almacenan todas las librerías necesarias para realizar un proyecto en Apache-cassandra. Ver Ilustración 65: Librerías Apache-Cassandra.

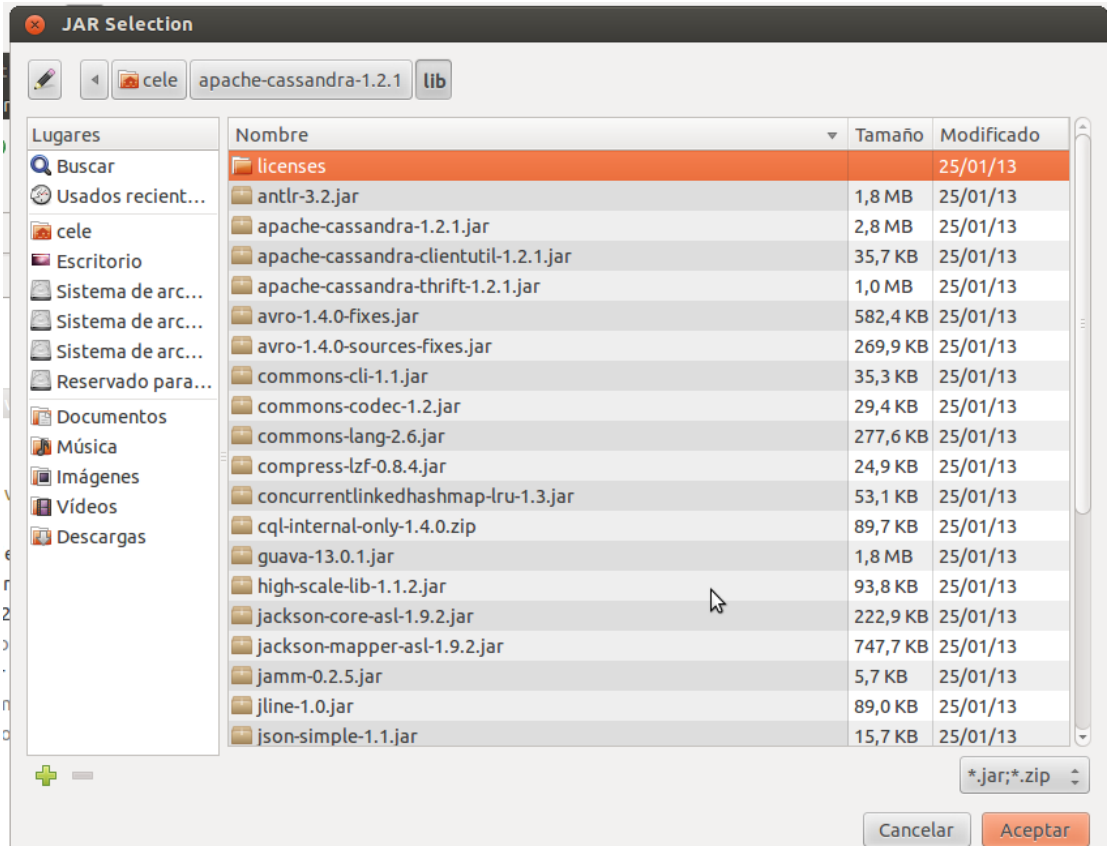


Ilustración 65: Librerías Apache-Cassandra

Importar las siguientes librerías

- apache-cassandra-clientutil-1.2.1.jar
- apache-cassandra-thrift-1.2.1.jar
- apache-cassandra-1.2.1.jar
- libthrift-0.7.0.jar
- slf4j-log4j12-1.7.2.jar
- log4j-1.2.16.jar
- slf4j-api-1.7.2.jar

14 ANEXO C: SQLSERVER 2008 R2

14.1 TRANSFORMACION DATOS

La Base de Datos contiene información reales, por lo tanto, se debe transformar los datos que identifican a los pacientes y médicos.

```
//Asigna un numero aleatorio como Rut//
UPDATE    DIM_PACIENTE
SET       PAC_RUT= ABS(CAST(NEWID() as binary(9)) % 1000000) + 1
WHERE     PAC_RUT<>'       1K'

UPDATE    DIM_PRESTADOR
SET       PTD_RUT= ABS(CAST(NEWID() as binary(10)) % 1000000) + 1
WHERE     PTD_RUT<>'       1K'

//Modifica los rut del equipo médico, insertando los valores nuevos de Dim_prestador//
UPDATE    DIM_EQUIPO_MEDICO
SET       PTD_RUT= prestador.ptd_rut
FROM      DIM_PRESTADOR prestador, DIM_EQUIPO_MEDICO e
WHERE     e.PTD_IDENTIFICACION=prestador.PTD_IDENTIFICACION

//Elimina los espacios en blanco de la derecha e izquierda//
UPDATE    FACT_INGRESO
SET       PTD_RUT_ALTA=ltrim(RTRIM(PTD_RUT_ALTA))

UPDATE    FACT_INGRESO
SET       PTD_RUT_TRAT=ltrim(RTRIM(PTD_RUT_TRAT))

UPDATE    FACT_INGRESO_DURACION
SET       PTD_RUT_ALTA=ltrim(RTRIM(PTD_RUT_ALTA))

UPDATE    FACT_INGRESO_DURACION
SET       PTD_RUT_TRAT=ltrim(RTRIM(PTD_RUT_TRAT))

//Modifica los datos de ingreso con los valores de Dim_prestador//
UPDATE    FACT_INGRESO_DURACION
SET       PTD_RUT_TRAT=p.ptd_rut
FROM      DIM_PRESTADOR p, FACT_INGRESO_DURACION f
WHERE     p.PTD_IDENTIFICACION=f.PTD_RUT_TRAT

UPDATE    FACT_INGRESO_DURACION
SET       PTD_RUT_ALTA=p.ptd_rut
FROM      DIM_PRESTADOR p, FACT_INGRESO_DURACION f
WHERE     p.PTD_IDENTIFICACION=f.PTD_RUT_ALTA
```

```
UPDATE   DIM_PRESTADOR
SET      PTD_IDENTIFICACION= left(ltrim(rtrim(PTD_IDENTIFICACION)),
         len(ltrim(rtrim(PTD_IDENTIFICACION))) - 2)+RIGHT(PTD_IDENTIFICACION,1)
WHERE    PTD_IDENTIFICACION<>'       1K'

UPDATE   FACT_INGRESO
SET      PTD_RUT_TRAT=p.ptd_rut
FROM     DIM_PRESTADOR p, FACT_INGRESO f
WHERE    p.PTD_IDENTIFICACION=f.PTD_RUT_TRAT

UPDATE   FACT_INGRESO
SET      PTD_RUT_ALTA=p.ptd_rut
FROM      DIM_PRESTADOR p, FACT_INGRESO f
WHERE    p.PTD_IDENTIFICACION=f.PTD_RUT_ALTA

UPDATE   DIM_PRESTADOR
SET      PTD_IDENTIFICACION= RIGHT(ptd_identificacion,1)

UPDATE   DIM_PRESTADOR
SET      PTD_IDENTIFICACION=SUBSTRING(PTD_IDENTIFICACION,1,10)
WHERE    SUBSTRING(PTD_IDENTIFICACION,1,3)<>'000' and
         SUBSTRING(PTD_IDENTIFICACION,1,2)<>'00'and
         SUBSTRING(PTD_IDENTIFICACION,1,1)<>'0'

UPDATE   DIM_PRESTADOR
SET      PTD_IDENTIFICACION=SUBSTRING(PTD_IDENTIFICACION,4,7)
WHERE    SUBSTRING(PTD_IDENTIFICACION,1,3)='000'

UPDATE   DIM_PRESTADOR
SET      PTD_IDENTIFICACION= SUBSTRING(PTD_IDENTIFICACION,3,8)
WHERE    (SUBSTRING(PTD_IDENTIFICACION,1,2)='00')  and
         SUBSTRING(PTD_IDENTIFICACION,1,3)<>'000'

UPDATE   DIM_PRESTADOR
SET      PTD_IDENTIFICACION=SUBSTRING(PTD_IDENTIFICACION,2,9)
WHERE    SUBSTRING(PTD_IDENTIFICACION,1,1)='0' and
         SUBSTRING(PTD_IDENTIFICACION,1,3)<>'000 ' and
         SUBSTRING(PTD_IDENTIFICACION,1,3)<>'00'
```

14.2 EXPORTAR DATOS

SQLSERVER 2008 R2 tiene la funcionalidad de poder exportar los datos que contienen las tablas de una base de datos. Dispone de un amplio formato de destino, entre los cuales están los archivos de textos.

Para efectuar lo anterior se siguen los siguientes pasos:

PASO 1: Conectar el servidor de SQLServer correspondiente.

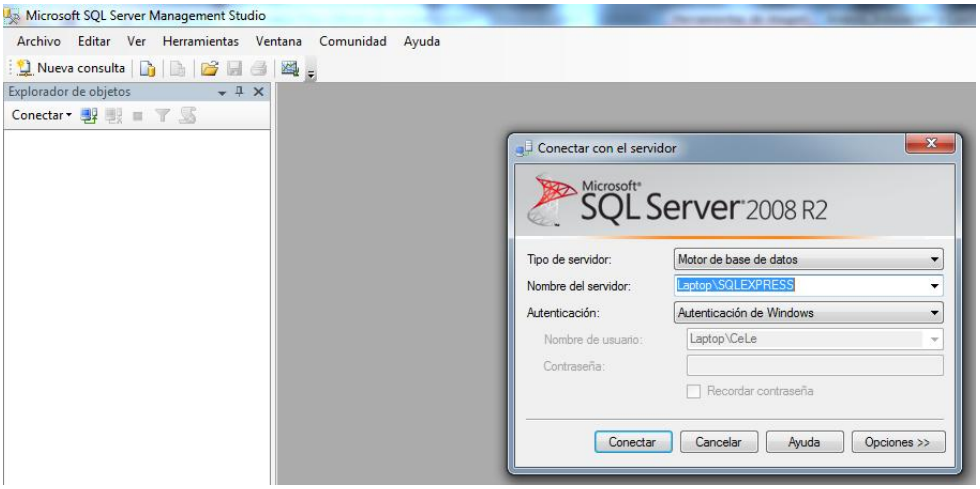


Ilustración 66: Conectar SQLServer 2008 R2

PASO 2: Seleccionar la Base de Datos donde se encuentra las tablas que se exportan y selección Exportar Datos.

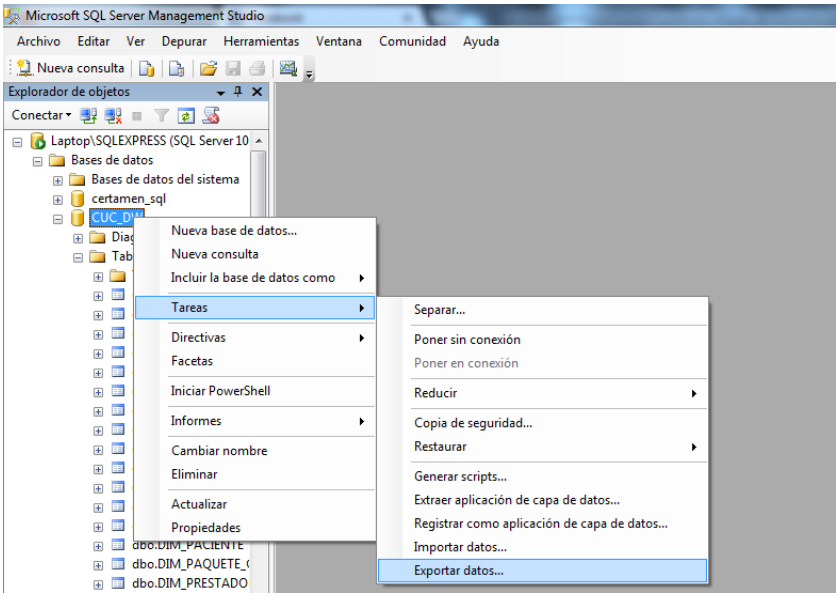


Ilustración 67: Ruta Exportar Datos

PASO 3: Seleccionar el origen de los Datos, en el caso que no se seleccionó la Base de Datos correcta que contienen los datos, se puede cambiar la Base a utilizar.

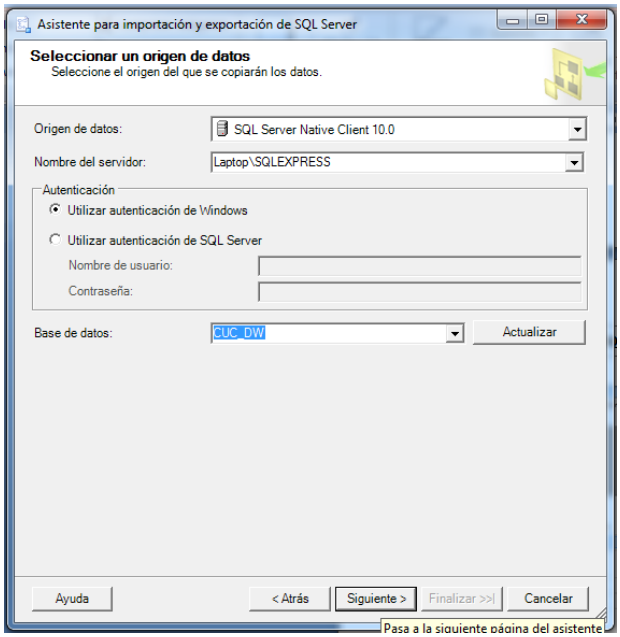


Ilustración 68: Seleccionar Datos Origen

PASO 4: Seleccionar tipo en que se exportan los datos, en este caso es un archivo plano, luego se debe seleccionar el archivo donde se guardarán los datos y por último se puede marcar si se desea dejar en la primera fila el nombre de las columnas de los datos.

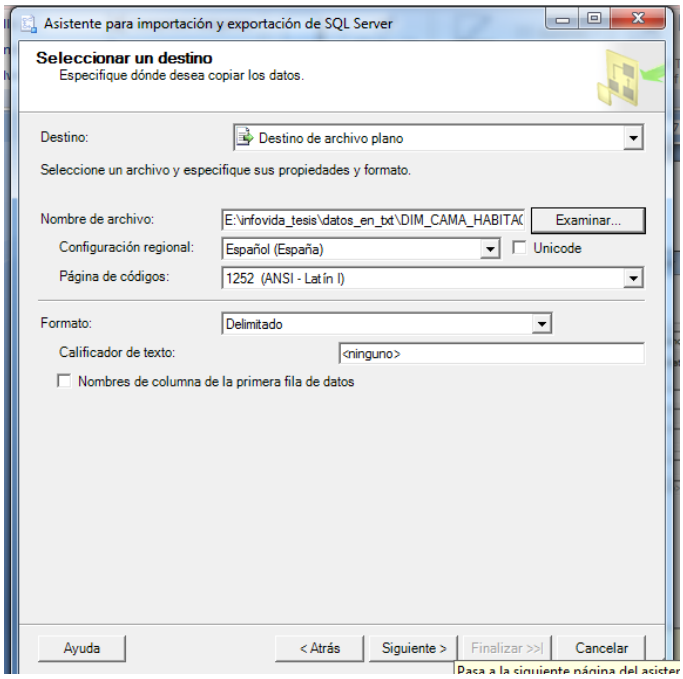


Ilustración 69: Seleccionar Destino

PASO 5: Seleccionar la copia de los datos de una tabla o escribiendo una consulta en particular

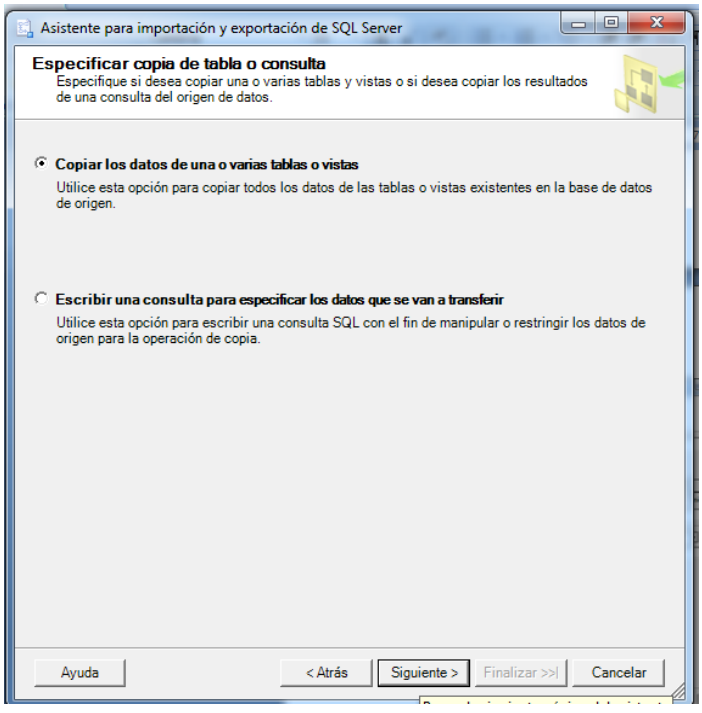


Ilustración 70: Copia de tabla o consulta

PASO 5: Configurar los delimitadores de las filas

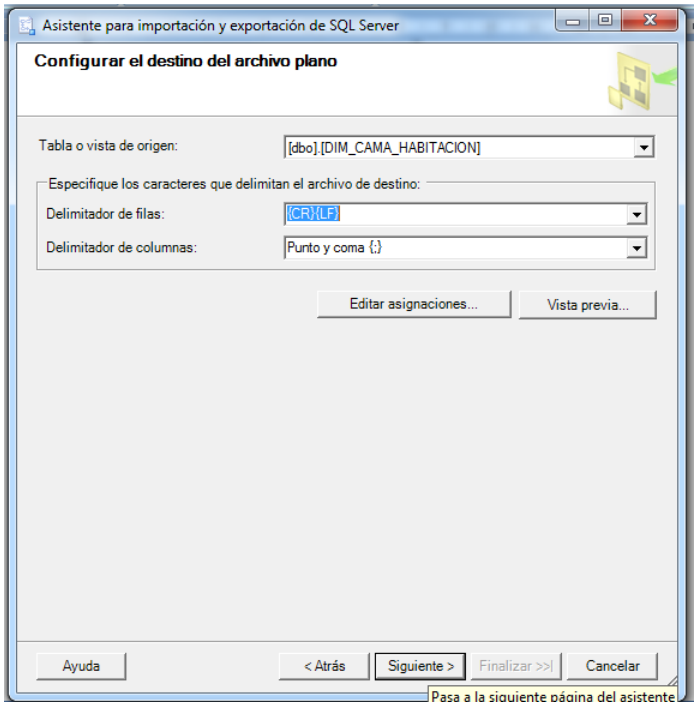


Ilustración 71: Delimitadores filas y columnas

PASO 6: Seleccionar si ejecutar el paquete inmediatamente y seleccionar Siguiente o Finalizar

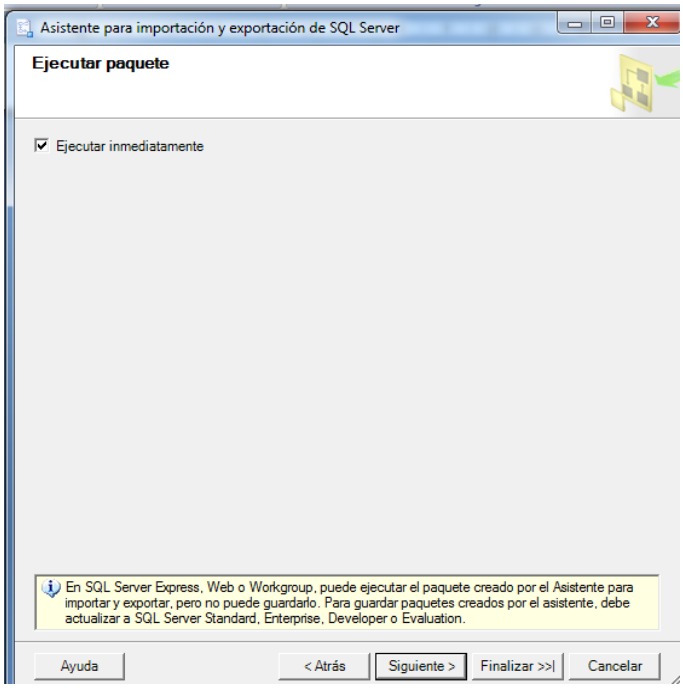


Ilustración 72: Ejecutar Paquete

PASO 7: Seleccionar Finalizar

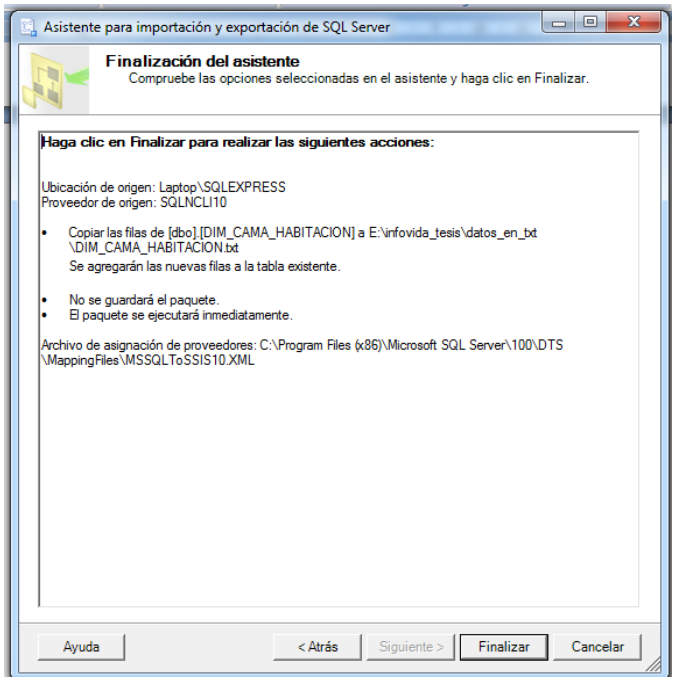


Ilustración 73: Finalización Asistente

PASO 8: Estado de la finalización.

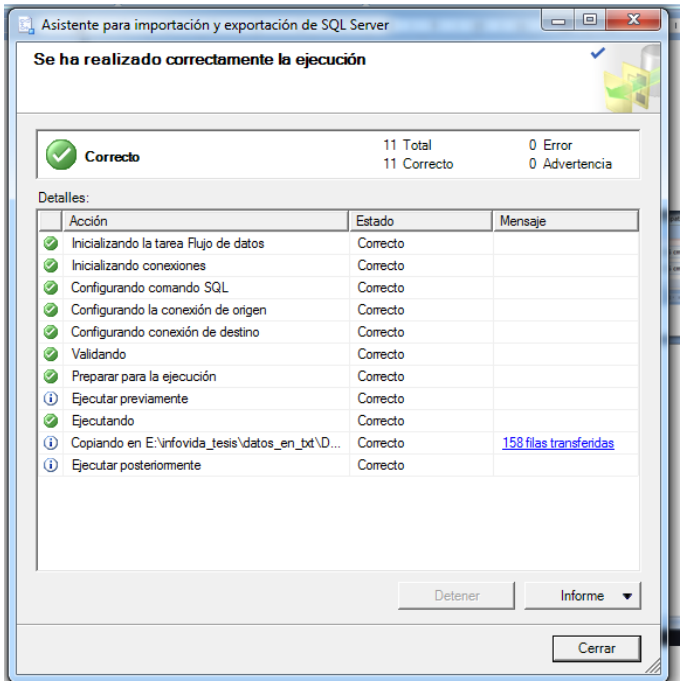


Ilustración 74: Estado Exportación

15 ANEXO D: DICCIONARIO DE DATOS

15.1 Tabla DIM_PACIENTE

- **Descripción de la tabla DIM_PACIENTE:**

  Almacena los datos del paciente, con una clave única que corresponde a su correlativo.

- **Lista de columnas de la tabla DIM_PACIENTE**

Nombre	Tipo de dato
PAC_CORREL	Numeric(10,0)
PAC_RUT	Variable characters (11)
PAC_NOMBRE	Variable characters (153)
PAC_FECHA_NAC	DateTime
PAC_IDENTIFICACION	Variable characters (166)
PAC_SEXO	Variable characters (11)
PAC_PROFESION	Variable characters (101)
PAC_NACIONALIDAD	Variable characters (250)

15.2 Tabla DIM_CARGO

- **Descripción de la tabla DIM_CARGO:**

  Almacena los datos pertenecientes a los cargos que se atribuyen al paciente.

- **Lista de columnas de la tabla DIM_CARGO**

Nombre	Tipo de dato
CRG_TIPO	Variable characters (3)
CRG_CODIGO	Numeric(10,0)
CRG_ARANCEL_DESCRIP	Variable characters(90)
CRG_TIPO_CARGO	Variable characters(11)

CRG_ARANCEL	Variable characters(50)
CRG_SUBGRUPO	Variable characters(20)
CRG_DESCRIP	Variable characters(250)
CRG_SUBGRUPO_DESCRIP	Variable characters(90)

15.3 Tabla DIM_EQUIPO_MEDICO

- **Descripción de la tabla DIM_EQUIPO_MEDICO:**

  Almacena los datos de los distintos equipos médicos de la clínica. En esta tabla se especifica el pabellón y el uso del pabellón por cada equipo médico

- **Lista de columnas de la tabla DIM_EQUIPO_MEDICO**

Nombre	Tipo de dato
PAB_CODIGO	Variable characters (5)
UPA_CORREL	Numeric(5,0)
EME_CORREL	Numeric(4,0)
PTD_RUT	Variable characters (10)
PTD_NOMBRE	Variable characters (50)
PTD_IDENTIFICACION	Variable characters (62)
TPT_DESCRIP	Variable characters (100)
PTD_ACREDITADO	Variable characters (24)
PRS_CODIGO_PAL	Characters(7)
SGR_CODIGO_PAL	Character(4)
PRS_DESCRIP_PPAL	Variable characters (250)
PRS_CODIGO_SUB_DESC_PPAL	Variable characters (50)
PRS_CODIGO	Character(7)
SGR_CODIGO	Character(4)
PRS_DESCRIP	Variable characters (250)
PRS_CODIGO_SUB_DESC	Variable characters (40)
PAB_DESCRIP	Variable characters (25)
PTD_ESPECIALIDAD	Variable characters (30)

15.4 Tabla DIM_CONVENIO

- **Descripción de la tabla DIM_CONVENIO:**

  Almacena los datos de los distintos convenios que tiene la clínica con sus pacientes.

- **Lista de columnas de la tabla DIM_CONVENIO**

Nombre	Tipo de dato
CNV_CODIGO	Variable characters (15)
CNV_NOMBRE	Variable characters (250)
CNV_TIPO	Character(12)
CNV_EXENTO	Variable characters (16)

15.5 Tabla DIM_SERVICIO

- **Descripción de la tabla DIM_SERVICIO:**

  Almacena los datos de los distintos prestados por la clínica, con su código, tipo de servicio y su estado (Vigente o No vigente).

- **Lista de columnas de la tabla DIM_SERVICIO**

Nombre	Tipo de dato
SER_CODIGO	Variable characters (3)
SER_NOMBRE	Variable characters (150)
SER_CLASIFICACION	Variable characters (20)
SER_ESTADO	Variable characters (12)

15.6 Tabla DIM_ISAPRE

- **Descripción de la tabla DIM_ISAPRE:**

  Almacena los datos de isapres y el tipo que pertenece (Isapre, Fonasa y Otros).

- **Lista de columnas de la tabla DIM_ISAPRE**

Nombre	Tipo de dato
ISA_CODIGO	Numeric(5,0)
ISA_RUT	Character(10)
ISA_NOMBRE	Variable characters (100)
ISA_TIPO	Variable characters (11)

15.7 Tabla DIM_PRESTADOR

- **Descripción de la tabla DIM_PRESTADOR:**

  Almacena los datos de prestadores, tipo de prestación que realiza y su estado de acreditación.

- **Lista de columnas de la tabla DIM_PRESTADOR**

Nombre	Tipo de dato
PTD_RUT	Variable characters (10)
PTD_NOMBRE	Variable characters (50)
PTD_IDENTIFICACION	Variable characters (63)
TPT_DESCRIP	Variable characters (100)
PTD_SEXO	Variable characters (11)
PTD_ACREDITADO	Variable characters (24)

15.8 Tabla DIM_TIEMPO

- **Descripción de la tabla DIM_TIEMPO:**

  Almacena los datos de las fechas.

- **Lista de columnas de la tabla DIM_TIEMPO**

Nombre	Tipo de dato
TPO_FECHA	DateTime
TPO_FECHA_NOMBRE	Variable characters (25)
TPO_FECHA_STRING	Variable characters (10)
TPO_AÑO	Int
TPO_AÑO_NOMBRE	Variable characters (20)
TPO_SEMESTRE	Int
TPO_SEMESTRE_NOMBRE	Variable characters (20)
TPO_TRIMESTRE	Int
TPO_TRIMESTRE_NOMRE	Variable characters (20)
TPO_MES	Variable characters (10)
TPO_MES_NOMBRE	Variable characters (25)
TPO_SEMANA	Int
TPO_SEMANA_NOMBRE	Variable characters (20)
TPO_DIA_NOMBRE	Variable characters (20)
TPO_DIA_DEL_MES	Int
TPO_DIA_DEL_ANO	Int
TPO_DIA_HABIL_MES	Character(7)

15.9 Tabla DIM_CONDICION_CONVENIO

- **Descripción de la tabla DIM_CONDICION_CONVENIO**

  Almacena los datos condición de los convenios

- **Lista de columnas de la tabla DIM_CONDICION_CONVENIO**

Nombre	Tipo de dato
CCO_CORREL	Numeric(5,0)
CCO_DESCRIP	Variable characters (250)

15.10 Tabla DIM_ITEM_ADICIONAL

- **Descripción de la tabla DIM_ITEM_ADICIONAL**

  Almacena los datos de los ítems adicionales que se cargan a los pacientes.

- **Lista de columnas de la tabla DIM_ITEM_ADICIONAL**

Nombre	Tipo de dato
IAD_CODIGO	Character(2)
IAD_DESCRIP	Variable characters (100)
IAD_TIPO	Variable characters (19)

15.11 Tabla DIM_CASO_AUGE

- **Descripción de la tabla DIM_CASO_AUGE**

  Almacena un correlativo y una descripción de los casos auge.

- **Lista de columnas de la tabla DIM_CASO_AUGE**

Nombre	Tipo de dato
CAU_CORREL	Numeric(10,0)
CAU_OBSERV	Character(250)

15.12 Tabla DIM_CANASTA_FINAN

- **Descripción de la tabla DIM_CANASTA_FINAN**

  Almacena datos de la canasta

- **Lista de columnas de la tabla DIM_CANASTA_FINAN**

Nombre	Tipo de dato
CAU_CORREL	Numeric(10,0)
CAF_CORREL	Numeric(10,0)
CAF_DESCRIP	Variable characters (250)

15.13 Tabla DIM_PAQUETE_CONVENIO

- **Descripción de la tabla DIM_PAQUETE_CONVENIO**

  Almacena datos de paquete de convenios y una breve descripción

- **Lista de columnas de la tabla DIM_PAQUETE_CONVENIO**

Nombre	Tipo de dato
PAQ_CODIGO	Variable characters (15)
PAQ_DESCRIP	Variable characters (250)

15.14 Tabla DIM_INGRESO_CLASIFICACION

- **Descripción de la tabla DIM_INGRESO_CLASIFICACION**

  Almacena datos con la clasificación de los pacientes, su identificador es la concatenación de Ingreso + Categorización + Atención + Alta.

Ingreso:	UR(Urgencia), AD(Admisión)
Categorización:	C0-Sin categorización, C1-Gravemente enfermo, C2-Enfermo, C3-Portador de una afección aguda, C4-Afecciones electiva, SA-
Atención:	AM(Ambulatoria), HO(Hospitalaria), FC(Ficha cerrada), PI(Pre-ingreso), RN(Recién Nacido)
Alta:	0(Sin alta), 1(Egreso Normal), 2(Traslado a Red Pública), 3(Fallecido), 4(Hospitalizar), 5(Alta Voluntaria), 6(Pabellón suspendido), 7(Frecuenta 30 días), 8(Traslado a otra Clínica Privada), 9(Fuga del Paciente).

- **Lista de columnas de la tabla DIM_INGRESO_CLASIFICACION**

Nombre	Tipo de dato
INC_CLASIFICACION	Variable characters (11)
INC_TIPO_INGRESO	Variable characters (15)
INC_CATEGORIZACION	Variable characters (40)
INC_TIPO_ATENCION	Variable characters (40)
INC_MOTIVO_ALTA	Variable characters (40)

15.15 Tabla DIM_CIE_10

- **Descripción de la tabla DIM_CIE_10**

  Almacena datos de correspondiente a la Clasificación Internacional de Enfermedades.

- **Lista de columnas de la tabla DIM_CIE_10**

Nombre	Tipo de dato
CIE_CODIGO	Variable characters (10)
CIE_AGRUPADO	Variable characters (30)
CIE_DESCRIP	Variable characters (100)
CIE_DIAGNOS	Variable characters (255)

15.16 Tabla FACT_INGRESO

- **Descripción de la tabla FACT_INGRESO**

  Almacena datos de pacientes con cada atributos asociado, la cantidad de cada prestación y producto cargado, valor unitario, neto, iva y total de cada uno.

- **Lista de columnas de la tabla FACT_INGRESO**

Nombre	Tipo de dato
PAC_NUMFICHA	Numeric(10,0)
ING_CORREL	Numeric(4,0)
PAC_CORREL	Numeric(10,0)
PTD_RUT_TRAT	Variable characters (10)
PTD_RUT_ALTA	Variable characters (10)
ISA_CODIGO	Numeric(5,0)
PAQ_CODIGO	Variable characters (15)
SER_CODIGO	Variable characters (3)
PAB_CODIGO	Variable characters (5)
UPA_CORREL	Numeric(5,0)

EME_CORREL	Numeric(4,0)
CNV_CODIGO	Variable characters (15)
CCO_CORREL	Numeric(5,0)
CAU_CORREL	Numeric(10,0)
CAF_CORREL	Numeric(10,0)
INC_CLASIFICACION	Variable characters (11)
CRG_TIPO	Variable characters (3)
CRG_CODIGO	Numeric(10)
IAD_CODIGO	Character(2)
CIE_CODIGO	Variable characters (10)
ING_FECING	DateTime
ING_FECALTA	DateTime
ING_FECVAL	DateTime
ING_FECATEN_PAB	DateTime
CRG_CANTIDAD	Numeric(38,2)
CRG_VALOR_UNI	Numeric(38,0)
CRG_EXENTO	Numeric(38,0)
CRG_AFECTO	Numeric(38,0)
CRG_IVA	Numeric(38,0)
CRG_TOTAL	Numeric(38,0)

15.17 Tabla FACT_INGRESO_DURACION

- **Descripción de la tabla FACT_INGRESO_DURACION**

  Almacena datos de pacientes, la cantidad de días en la clínica y la cantidad de minutos en el pabellón.

- **Lista de columnas de la tabla FACT_INGRESO_DURACION**

Nombre	Tipo de dato
PAC_NUMFICHA	Numeric(10,0)
ING_CORREL	Numeric(4,0)

PAC_CORREL	Numeric(10,0)
PTD_RUT_TRAT	Variable characters (10)
PTD_RUT_ALTA	Variable characters (10)
ISA_CODIGO	Numeric(5,0)
PAQ_CODIGO	Variable characters (15)
CNV_CODIGO	Variable characters (15)
CAU_CORREL	Numeric(10,0)
CAF_CORREL	Numeric(10,0)
INC_CLASIFICACION	Variable characters (11)
ING_FECING	DateTime
ING_FECALTA	DateTime
ING_FECVAL	DateTime
ING_DURACION_DIAS	Int
PAB_DURACION_MINUTOS	Int

15.18 Tabla FACT_INGRESO_UTILI_HAB

- **Descripción de la tabla FACT_INGRESO_UTILI_HAB**

  Almacena datos de pacientes, la cantidad de días en la habitación

- **Lista de columnas de la tabla FACT_INGRESO_UTILI_HAB**

Nombre	Tipo de dato
PAC_NUMFICHA	Numeric(10,0)
ING_CORREL	Numeric(4,0)
PAC_CORREL	Numeric(10,0)
CRG_TIPO	Variable characters (3)
CRG_CODIGO	Numeric(10,0)
FEC_INICIO	DateTime
FEC_TERMINO	DateTime
CRG_CANTIDAD	Int

16 ANEXO E: PRUEBAS SEGUNDO MODELO DE CASSANDRA

16.1 Modelo 2

Este modelo de datos contempla todos los valores del modelo relacional, más 4 columnas por tuplas, creadas para poder realizar las consultas en Cassandra, que corresponden a clave_paciente, ing_year, ing_mes, ing_dia.

Por lo tanto está compuesto por un total de 98.359.314 columnas con registros, lo que equivale a un 13% más del modelo relacional. Ver Tabla 13: Cantidad de registro modelo 2 de Cassandra

- **Dominio**

Tabla de Hechos	Cantidad Tuplas	Cantidad Columnas	Total columnas
Fact_ingreso	**2.892.921.-**	**30**	**86.787.630.-**
Clave_paciente, ing_year, ing_mes, ing_dia	**2.892.921.-**	**4**	**11.571.684.-**
Total columnas			**98.359.314.-**

Tabla 13: Cantidad de registro modelo 2 de Cassandra

16.2 Resultado Pruebas

M1	Modelo de Cassandra con disminución de datos
M2	Modelo de Cassandra con todos los datos
M3	Modelo de SqlServer 2008

	M1	M2	M3
	C1	C1	C1
Q1	3111	4948	13946
Q2	5,1	9,2	12746
Q3	2580	5780	14823
Q4	5164	10700	29553

Tabla 14: Resultados de los tres modelos

En la Tabla 14: Resultados de los tres modelos, están los resultados obtenidos en las consultas de los tres modelos a comparar, los tiempos se encuentran expresados en milisegundos.

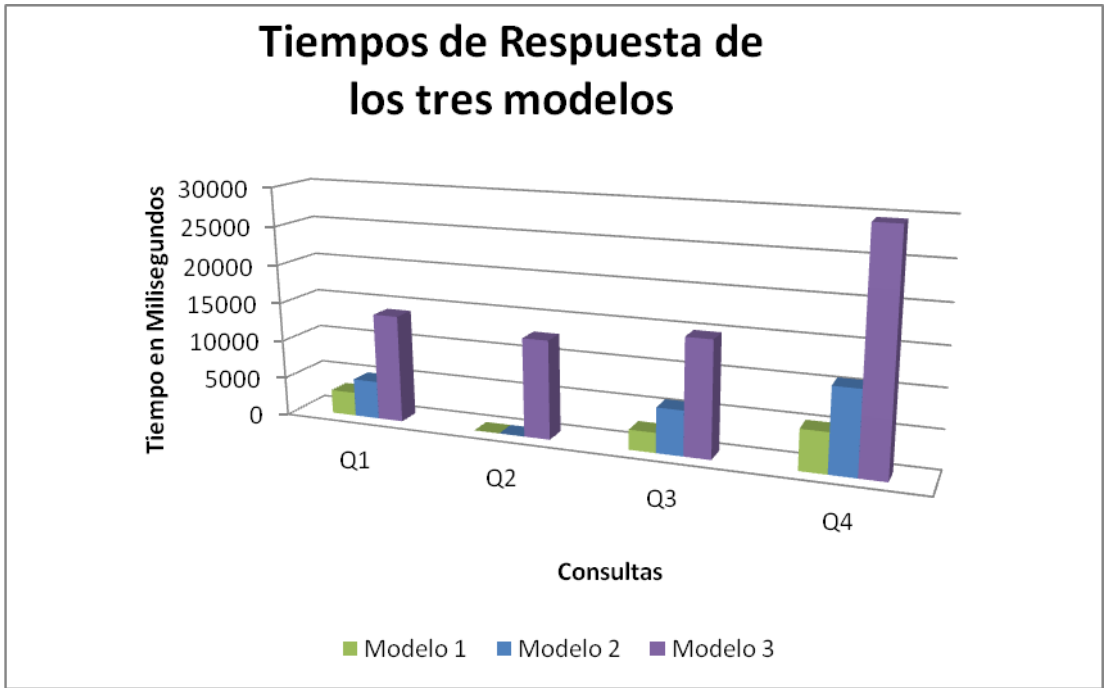


Ilustración 75: Gráfico de Barra comparativa de los tres modelos.

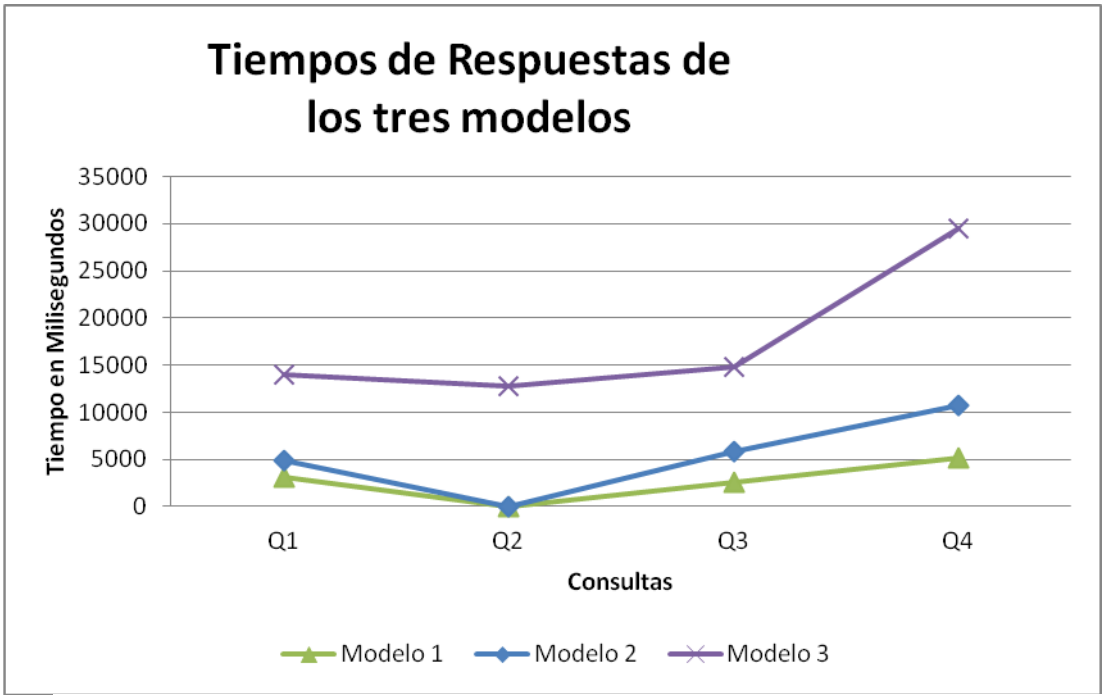


Ilustración 76: Gráfico de línea comparativo de los tres modelos.

Interpretación:

En la Ilustración 75: Gráfico de Barra comparativa de los tres modelos. e Ilustración 76: Gráfico de línea comparativo de los tres modelos. se grafican los valores obtenidos al realizar las consultas en los tres modelos.

En ambos gráficos que puede visualizar que la base de datos no relacional responde con tiempos bastante menores en comparación a los entregados por el motor de base de datos relacional de SQLServer, lo cual ocurre sin importar que el modelo 2 consta un 13% más de la cantidad de registro que el modelo implantado en el SQLServer, por lo tanto, se puede comprobar que sin importar la cantidad de datos, Cassandra sigue entregando resultados mayormente favorables que SQLServer y que la disminución de datos en el primer modelo no fue el factor fundamental para el que el este estudio tuviese éxito, sino que el almacenamiento utilizado por Cassandra fue el que logró que sus búsquedas y tiempos de respuestas sean más rápidos.