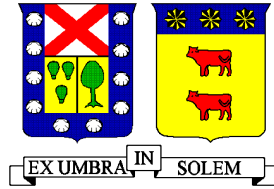


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE



EXPERIENCIAS CON SISTEMAS DE BASES DE DATOS NoSQL PARA SU
IMPLEMENTACIÓN EN LA DOCENCIA

Memoria de titulación presentada por
EDMUNDO JOSÉ ASENJO BONZI

Como requisito para optar al título de
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

Profesor Guía
JOSÉ LUIS MARTÍ LARA

AGOSTO de 2017

Resumen

Este trabajo presenta el análisis de diferentes motores de bases de datos NoSQL, con el fin de identificar los más adecuados para su uso en un ambiente académico de introducción a esta tecnología. Se presenta un análisis comparativo usando criterios como barrera de entrada, facilidad de uso, curva de aprendizaje e importancia de conocer esta tecnología y los problemas a los cuales se enfoca. Este documento concluye con reflexiones acerca de las diferencias más relevantes entre estos sistemas en un ambiente docente, además de proponer un conjunto de experiencias académicas para realizar en el aula, enfocadas en la manera más eficiente de acercar al alumnado a esta tecnología y el cómo se diferencia de los sistemas relacionales convencionales.

Abstract

This paper presents the analysis of different NoSQL database engines in order to identify the most suitable ones for use in an academic environment of introduction to this technology. A comparative analysis using criteria entrance barrier, ease of use, learning curve and importance of knowing this technology and the problems to which it is focused is presented. This document concludes with reflections on the most relevant differences of the systems in a teaching environment as well as proposing a set of academic experiences for the teachers, focused on the most efficient way to bring students to this technology and how it is distinguished from conventional relational systems.

Glosario

ACID: acrónimo en base de datos para Atomicity, Consistency, Isolation y Durability. Cualidades características de las bases de datos relacionales.

Apache: ‘The Apache Software Foundation’ es una fundación que da soporte a diversos softwares de código libre bajo su tutela.

API: acrónimo de Application Programming Interface. Corresponde a un conjunto de componentes que establecen la comunicación entre softwares como una capa de abstracción.

BASE: acrónimo en base de datos para Basically Available, Soft-state y Eventual consistency. Cualidades características de las bases de datos no relacionales.

BSON: formato de datos utilizado por la base de datos MongoDB para su almacenamiento y transferencia. El nombre referencia a Binary JSON (Binary JavaScript Object Notation).

CAP: acrónimo de Consistency, Availability, Partition tolerance. También conocido como Teorema de Brewer, enuncia que para un sistema de cómputo distribuido, es imposible garantizar estas tres características de manera simultánea.

CRUD: acrónimo en base de datos de Create, Read, Update, Delete que hace referencia a las operaciones habituales de crea, leer, actualizar y borrar.

DB/BD: database; base de datos.

ERP: acrónimo de Enterprise Resource Planning.

Escalamiento Horizontal: migración de un sistema a un modelo distribuido, sumando múltiples equipos de mediana gama.

Escalamiento Vertical: migración de un sistema a nuevo hardware más potente para mejorar el desempeño. Se encuentra limitado por el avance tecnológico.

JDK: acrónimo de Java Development Kit. Herramientas de desarrollo para programas en Java.

JSON: acrónimo de JavaScript Object Notation. Formato de texto para la transferencia de datos.

NoSQL: acrónimo de Not Only Structured Query Language. Sistemas de gestión de bases de datos no relacionales de diversas clases. Se caracterizan por suplir las necesidades que los sistemas relacionales no abarcan.

OLAP: acrónimo de On-Line Analytical Processing. Solución empleada en inteligencia de negocios que contiene datos resumidos en estructuras multidimensionales, llamadas cubos, para agilizar la consulta a grandes cantidades de datos.

RDBMS/SABDR: acrónimo de Relational Database Management System; Sistema de Administración de Base de Datos Relacional.

SAP: conjunto de herramientas enfocadas para que las empresas puedan optimizar diferentes aspectos como ventas, finanzas, recursos humanos, entre otros.

SQL: acrónimo de Structured Query Language. Lenguaje que utilizan las bases de datos relacionales para la manipulación de datos.

Índice de Contenidos

Índice de Contenidos	- 1 -
Índice de Tablas	- 2 -
Índice de Figuras	- 3 -
Introducción	- 6 -
Capítulo 1: Definición del Problema.....	- 8 -
1.1 Bases de Datos NoSQL.....	- 8 -
1.2 Objetivos del Trabajo.....	- 9 -
1.3 Alcances del Trabajo.....	- 10 -
Capítulo 2: Estado del Arte	- 11 -
Capítulo 3: Diseño del Trabajo	- 15 -
3.1 Bases de Datos Columnares	- 15 -
3.2 Bases de Datos Documentales.....	- 16 -
3.3 Bases de Datos de Grafos.....	- 16 -
3.4 Criterios de Comparación.....	- 16 -
Capítulo 4: Desarrollo del Trabajo.....	- 18 -
4.1 Bases de Datos Columnares	- 18 -
4.1.1 Apache Cassandra v3.0.9	- 18 -
4.1.2. Hypertable v0.9.8.11	- 23 -
4.1.3 Comparación Preliminar	- 30 -
4.1.4 Propuesta Actividad Académica	- 31 -
4.2 Bases de Datos Documentales.....	- 32 -
4.2.1 CouchDB v2.0.0.....	- 32 -
4.2.2 MongoDB v3.4.4.....	- 39 -
4.2.3 Comparación Preliminar	- 45 -
4.2.4 Propuesta Actividad Académica	- 47 -
4.3 Bases de Datos de Grafos.....	- 48 -
4.3.1 ArangoDB v2.8.11	- 48 -
4.3.2 Neo4j v3.2.0.....	- 64 -
4.3.3 Comparación Preliminar	- 75 -
4.3.4 Propuesta Actividad Académica	- 77 -
Conclusiones	- 79 -
Referencias.....	- 83 -

Índice de Tablas

Tabla 1. Criterios de comparación.	- 17 -
Tabla 2. Estructura lógica de almacenamiento en Hypertable, para el <i>set</i> de datos de ejemplo. ...	- 27 -
Tabla 3. Comparativa entre Cassandra y Hypertable.	- 31 -
Tabla 4. Comparativa entre CouchDB y MongoDB.	- 46 -
Tabla 5. Comparativa entre ArangoDB y Neo4j.	- 76 -

Índice de Figuras

Figura 1. Instalación de Cassandra (1).	- 19 -
Figura 2. Instalación de Cassandra (2).	- 19 -
Figura 3. Cassandra CQL Shell.	- 19 -
Figura 4. Ejemplo de comando <i>create keyspace</i> y <i>create table</i> de Cassandra.	- 20 -
Figura 5. Ejemplo de comando <i>insert</i> de Cassandra.	- 21 -
Figura 6. Ejemplo de comando <i>select</i> de Cassandra.	- 21 -
Figura 7. Ejemplo de comando <i>select</i> con <i>allow filtering</i> de Cassandra.	- 22 -
Figura 8. Ejemplo de comando <i>source</i> de Cassandra.	- 22 -
Figura 9. Ejemplo de comando <i>copy</i> de Cassandra.	- 23 -
Figura 10. Instalador ht4w para Hypertable (1).	- 24 -
Figura 11. Instalador ht4w para Hypertable (2).	- 24 -
Figura 12. Shell de Hypertable.	- 24 -
Figura 13. Ejemplo de comando <i>namespace</i> de Hypertable.	- 25 -
Figura 14. Ejemplo de comando <i>insert</i> de Hypertable.	- 26 -
Figura 15. Ejemplo de comando <i>select</i> de Hypertable.	- 26 -
Figura 16. Fragmento del documento <i>data.txt</i> para Hypertable.	- 27 -
Figura 17. Ejemplo de comando <i>load data infile</i> de Hypertable.	- 28 -
Figura 18. Ejemplo de comando <i>select limit</i> de Hypertable.	- 28 -
Figura 19. Ejemplo de comando <i>insert</i> con datos ya existente en Hypertable.	- 29 -
Figura 20. Ejemplo de comando <i>select</i> con opción <i>display_timestamps</i> de Hypertable.	- 29 -
Figura 21. Ejemplo de comando <i>dump</i> table de Hypertable.	- 29 -
Figura 22. Instalador de CouchDB para Windows (1).	- 33 -
Figura 23. Instalador de CouchDB para Windows (2).	- 33 -
Figura 24. Información de la instalación de CouchDB.	- 33 -
Figura 25. Panel principal de CouchDB.	- 34 -
Figura 26. Ejemplo de creación de base de datos en CouchDB.	- 34 -
Figura 27. Panel de administración de la base de datos de CouchDB.	- 35 -
Figura 28. Opciones de creación de documento, vista e índice de CouchDB.	- 35 -
Figura 29. Ejemplo de creación de un documento en CouchDB.	- 36 -
Figura 30. Visualización de la base de datos en CouchDB.	- 36 -
Figura 31. Visualización de las claves <i>_id</i> y <i>_rev</i> .	- 37 -
Figura 32. Pantalla guía para adjuntar archivos al documento de couchDB.	- 37 -
Figura 33. Ejemplo de imagen adjunta a un documento de CouchDB.	- 38 -
Figura 34. Ejemplo de búsqueda por herramienta Mango de CouchDB.	- 39 -
Figura 35. Instalador de MongoDB para Windows (1).	- 40 -
Figura 36. Instalador de MongoDB para Windows (2).	- 40 -
Figura 37. Instalador de MongoDB para Windows (3).	- 40 -
Figura 38. Programas <i>mongod.exe</i> y <i>mongo.exe</i> ejecutados.	- 41 -
Figura 39. Ejemplo de comando <i>show dbs</i> y comando <i>use</i> .	- 41 -
Figura 40. Ejemplo de comando <i>db.createCollection()</i> y comando <i>show collections</i> .	- 42 -
Figura 41. Ejemplo de comando <i>db.collection_name.insert()</i> .	- 42 -

Figura 42. Ejemplo del comando <i>db.collection_name.find()</i> con y sin opción <i>pretty()</i> .	- 43 -
Figura 43. Ejemplo de documento con arreglo de valores.	- 43 -
Figura 44. Ejemplo de creación de documento de estructura compleja.	- 44 -
Figura 45. Ejemplo de visualización de documento de estructura compleja.	- 44 -
Figura 46. Ejemplo de omisión de campos en comando <i>db.collection_name.find()</i> .	- 45 -
Figura 47. Instalador de ArangoDB para Windows (1).	- 49 -
Figura 48. Instalador de ArangoDB para Windows (2).	- 49 -
Figura 49. Instalador de ArangoDB para Windows (1).	- 50 -
Figura 50. Pantalla de ingreso de credenciales de ArangoDB.	- 50 -
Figura 51. Selección de base de datos de ArangoDB.	- 51 -
Figura 52. <i>Dashboard</i> de monitoreo de ArangoDB.	- 51 -
Figura 53. Creación de bases de datos de ArangoDB.	- 52 -
Figura 54. Ejemplo de creación de base de datos de ArangoDB.	- 52 -
Figura 55. Selección de base de datos de ArangoDB.	- 53 -
Figura 56. Creación de colecciones de Arango DB.	- 53 -
Figura 57. Ejemplo de creación de colección de ArangoDB.	- 54 -
Figura 58. Ejemplo de creación colecciones de aristas y documentos de ArangoDB.	- 54 -
Figura 59. Creación de documentos de ArangoDB.	- 54 -
Figura 60. Ejemplo de definición de <i>_key</i> para documento de ArangoDB.	- 55 -
Figura 61. Ejemplo de creación de documento de ArangoDB.	- 55 -
Figura 62. Visualización de documentos en una colección de ArangoDB.	- 56 -
Figura 63. Ejemplo de filtro en colección de ArangoDB (1).	- 56 -
Figura 64. Ejemplo de filtro en colección de ArangoDB (2).	- 57 -
Figura 65. Ejemplo de múltiples filtros en colección de ArangoDB.	- 57 -
Figura 66. Creación de aristas de ArangoDB.	- 57 -
Figura 67. Ejemplo de definición de <i>_key</i> , <i>_from</i> y <i>_to</i> para arista de ArangoDB.	- 58 -
Figura 68. Ejemplo de parámetros <i>_from</i> y <i>_to</i> de arista de ArangoDB.	- 58 -
Figura 69. Ejemplo de creación de arista de ArangoDB.	- 59 -
Figura 70. Visualización de aristas en una colección de ArangoDB.	- 59 -
Figura 71. Creación de grafos de ArangoDB.	- 60 -
Figura 72. Ejemplo de creación de grafo de ArangoDB.	- 60 -
Figura 73. Visualización de grafo de ArangoDB.	- 61 -
Figura 74. Menú de opciones de un nodo de ArangoDB.	- 61 -
Figura 75. Creación de nueva arista directamente en grafo de ArangoDB (1).	- 62 -
Figura 76. Creación de nueva arista directamente en grafo de ArangoDB (2).	- 62 -
Figura 77. Opciones de arista en grafo de ArangoDB.	- 62 -
Figura 78. Menú de configuración de grafo de ArangoDB.	- 63 -
Figura 79. Ejemplo de uso de configuración de grafo de ArangoDB.	- 64 -
Figura 80. Instalador de Neo4j para Windows (1).	- 65 -
Figura 81. Instalador de Neo4j para Windows (2).	- 65 -
Figura 82. Instalador de Neo4j para Windows (3).	- 65 -
Figura 83. Iniciando base de datos de Neo4j.	- 66 -
Figura 84. Acceso a la base de datos de Neo4j.	- 66 -
Figura 85. Panel principal de Neo4j.	- 67 -

Figura 86. Ejemplo de creación de nodos en Neo4j.....	- 67 -
Figura 87. Ejemplo de creación de múltiples nodos en Neo4j.....	- 68 -
Figura 88. Búsqueda de nodos con la etiqueta Masculino o Femenino.	- 68 -
Figura 89. Búsqueda de todos los nodos menos aquellos con etiqueta Vehiculo.	- 68 -
Figura 90. Visualización de grafo de Neo4j.....	- 68 -
Figura 91. Visualización de tabla de Neo4j.	- 69 -
Figura 92. Visualización de texto de Neo4j.	- 69 -
Figura 93. Búsqueda de las personas y el automovil. Resultado antes de crear las relaciones.	- 69 -
Figura 94. Ejemplo de creación de relación en Neo4j.	- 70 -
Figura 95. Ejemplo de creación de relación Conducen desde nodos con Edad mayor a 18.	- 71 -
Figura 96. Actualización de relación en Neo4j.	- 72 -
Figura 97. Alerta de eliminar un nodo con relaciones asociadas a él.....	- 72 -
Figura 98. Eliminación de un nodo y sus relaciones asociadas.....	- 73 -
Figura 99. Visualización de toda la malla de nodos en Neo4j.	- 73 -
Figura 100. Ejemplo de despliegue de nodos.....	- 74 -
Figura 101. Opciones de color, ancho de linea y alias para las relaciones.....	- 74 -
Figura 102. Opciones de color, tamaño de la esfera y alias para los nodos.	- 74 -
Figura 103. Indicadores de cantidad de nodos y relaciones en Neo4j.....	- 74 -

Introducción

El presente trabajo de memoria expone la implementación, manipulación y análisis de diferentes motores de bases de datos NoSQL, con el propósito de identificar el más adecuado para iniciar un aprendizaje en éstos por parte de estudiantes de carreras de Informática. Para alcanzar este propósito se realizó un análisis comparativo usando criterios como barrera de entrada, facilidad de uso, curva de aprendizaje y relevancia en el mercado de diferentes motores de bases de datos NoSQL. Se consideraron tres grupos principales de bases de datos NoSQL, que corresponden a bases de datos columnares, documentales y de grafos, escogiendo dos motores por grupo.

Como apoyo al estudio de las bases de datos NoSQL en un ambiente académico, además de presentar el análisis de los motores y el más adecuado para enseñar como primera aproximación a los sistemas NoSQL, se presentan observaciones finales de cada trabajo que incluyen tres trabajos prácticos para aplicar en un ambiente docente. Estos trabajos prácticos contienen los elementos esenciales para el correcto entendimiento del tipo de base de datos que se ha enseñado.

Este documento concluye con reflexiones acerca de las diferencias más relevantes de los sistemas estudiados, enfocadas en la manera más eficiente de acercar al alumnado a esta tecnología y el cómo se distingue de los sistemas relacionales convencionales.

El desarrollo de este trabajo se presenta en 4 capítulos:

El Capítulo 1 define el problema que dio origen a las bases de datos NoSQL, se describen las bases de datos NoSQL, se plantean los objetivos de este trabajo y su alcance.

El Capítulo 2 describe la historia de las bases de datos NoSQL y cómo comenzaron a formar parte del mercado. Se describen las características principales de los sistemas de bases de datos NoSQL, sus diferencias principales con los SABDR y los diferentes tipos de modelamiento de datos que se consolidaron.

El Capítulo 3 describe la metodología aplicada en el estudio de los sistemas de bases de datos NoSQL columnares, documentales y de grafos. Se definen los escenarios en los cuales fueron estudiados cada tipo de base de datos NoSQL. Se presentan los criterios de comparación considerados, alineados con los objetivos del trabajo.

El Capítulo 4 presenta el análisis realizado. El estudio se divide en 3 subcapítulos, cada uno dedicado a un tipo de base de datos NoSQL: columnar, documental y de grafos. Para cada sistema se hace una presentación del motor de base de datos, un método de instalación, creación y manejo de datos, terminando con un análisis comparativo entre las bases de datos estudiadas para un mismo paradigma. El capítulo finaliza con una propuesta de actividad académica para la introducción a cada tipo de sistema de base de datos NoSQL.

El trabajo de memoria concluye con un análisis comparativo de las bases de datos NoSQL, señalando sus virtudes, defectos y criterios a considerar al momento de su selección como sistema primario de almacenamiento de datos. Se menciona uno de los dos motores de base de datos por tipo de sistema NoSQL como mejor elección para su uso en la docencia, junto a una propuesta para trabajar a nivel educacional junto a notas de cada tarea que consideran elementos como facilidad de uso, similitud al sistema SQL, una alta curva de aprendizaje y la claridad del concepto NoSQL que evoca.

Capítulo 1: Definición del Problema

En este capítulo se define el problema que dio origen a las bases de datos NoSQL, se describen las bases de datos NoSQL, se plantean los objetivos de este trabajo y su alcance.

1.1 Bases de Datos NoSQL

Las bases de datos NoSQL nacieron como un movimiento de sistemas de bases de datos no relacionales, distribuidos, *open-source* y de escalabilidad horizontal. Este movimiento tomó fuerza debido a la necesidad de sistemas de almacenamiento que permitieran una definición de estructuras flexibles para los datos (semi-estructurados), un manejo de grandes volúmenes de datos y una mejora en el rendimiento por sobre lo que podría entregar la tecnología del momento (sistema distribuido) (NoSQL, 2009).

El almacenamiento de datos es cada vez más relevante en un mundo donde el avance de las tecnologías va acompañado de la generación de miles y millones de datos que bien estudiados y analizados permiten extraer información esencial en la toma de decisiones. Los sistemas de administración de bases de datos relacionales (SABDR, por sus siglas) toman el rol principal en este campo, siendo las herramientas de las tecnologías de la información capaces de almacenar grandes cantidades de datos en un formato simple y estructurado. Los SABDR ofrecen al usuario cualidades indispensables en un entorno transaccional: atomicidad en operaciones agrupadas; consistencia en los datos; aislamiento entre transacciones y persistencia de los datos en el tiempo (Referencia 8). Las bases de datos relacionales han definido la manera en que los datos han sido almacenados desde finales de los años '70.

A medida que la tecnología y los datos se asemejan cada vez más a la realidad que nos define, han surgido a flote las falencias de los sistemas relacionales. En un escenario donde los datos tienden a ser más diferentes que similares entre sí, la estructura rígida de los sistemas relacionales dificulta enormemente su modelamiento, sobre todo en escenarios donde esta estructura puede ser cambiante en el tiempo. Otra falencia que surgió fue en el manejo de grandes volúmenes de datos, sobre todo en la lectura. Los sistemas relacionales definen grados de vulnerabilidad de una tabla a inconsistencias y anomalías lógicas, estos grados reciben el nombre de formas normales, siendo la 3NF la más aplicada. La 3NF, entre sus características, define que, a excepción de la clave primaria para hacer referencias, ningún dato esté repetido en la base de datos (redundancia). Lo anterior, pudiendo ser práctico cuando se lee un dato (evitando tener el dato dos veces con valores diferentes) y

teniendo menor consumo de espacio de almacenamiento, resulta tener altos tiempos de respuesta debido al costoso trabajo lógico que implica el unir las tablas (*join*).

No siendo suficiente lo anterior, el rendimiento de los SABDR se encuentra limitado a la potencia que le pueda entregar la tecnología de punta a la fecha (escalamiento vertical), lo que no permite distribuir la carga del sistema en múltiples máquinas (Microsoft Developer Network, 2017; Junta de Andalucía, Conceptos sobre la escalabilidad) La dificultad de los sistemas relacionales en adaptarse a escenarios donde los datos no se limitan a una estructura fija, y donde su flujo fuera intenso y masivo mostró la necesidad de un sistema de almacenamiento diferente.

Es así que surgen los sistemas de bases de datos NoSQL, sistemas que cubren todas las limitantes de los sistemas relacionales convencionales. Poseen manejo de datos semi-estructurada, permiten el trabajo y almacenamiento en un ambiente distribuido sin reducir el rendimiento, y las operaciones de lectura resultan ser mucho más rápidas gracias a la duplicidad de los datos al estar desnormalizados.

La aparición más temprana de los SABDR como tecnología de almacenamiento de datos los ha fijado como el estándar en este aspecto, estando la industria familiarizada con ellos y confeccionando software que tengan siempre compatibilidad con este tipo de motores de bases de datos. Debido a que los sistemas NoSQL aparecieron posterior a los SABDR les ha resultado complejo entrar al mercado, ya que migrar de un sistema de almacenamiento a otro de estructura diferente resulta muy costoso e incluso inviable en algunos escenarios. Junto a esto, las tecnologías NoSQL no son impartidas de forma regular en los planes de carrera para formar profesionales TI debido a su reducida presencia en el mercado laboral.

1.2 Objetivos del Trabajo

Como objetivo general se tiene:

Analizar los diferentes sistemas de bases de datos NoSQL, exponiendo la eficiencia, flexibilidad y nivel de especialización alcanzado por ellos, con el propósito de determinar el más adecuado para aplicar en las aulas de clases como primer acercamiento a las tecnologías NoSQL.

Dentro de los objetivos específicos se encuentran:

- Definir criterios cualitativos que favorezcan la comprensión de cada uno de los tipos de bases de datos y que sirvan para comparar diversas bases de datos dentro de un paradigma NoSQL, con el propósito de escoger la más adecuada para comenzar su aprendizaje.
- Definir una estructura de tarea de laboratorio para el aprendizaje de cada uno de los tipos de bases NoSQL, que incluya los elementos esenciales para su comprensión de modo que permita su diferenciación con otros tipos de bases de datos NoSQL y relacionales.

1.3 Alcances del Trabajo

Este documento asume un conocimiento básico y general de bases de datos relacionales por parte del lector, por lo que no profundizará en esta área.

Para alcanzar los objetivos planteados se limitó el alcance de este trabajo al estudio de tres tipos de bases de datos NoSQL: columnar, documental y de grafos. Los tres tipos anteriores fueron escogidos por su popularidad, presencia en el mercado actual y diversas áreas de aplicación. Se limitó el estudio al uso de dos motores de bases de datos columnares, dos bases de datos documentales y dos bases de datos de grafos. Debido a que algunos motores de bases de datos pueden desempeñarse en más de una categoría (columnar, documental o de grafos), se limitó su uso a una única categoría con el propósito de utilizar diferentes herramientas y presentar una mayor gama de opciones NoSQL.

Para complementar el estudio de las bases de datos NoSQL, hubo casos en los que se utilizaron *software* de inteligencia de negocios y *software* de análisis y minería de texto. No se profundizó en el manejo de estos *softwares* de apoyo, ya que sólo cumplen un rol secundario en el estudio de los sistemas NoSQL y en este trabajo de memoria.

Capítulo 2: Estado del Arte

El término NoSQL (Not Only SQL), definido en 1998, describe un conjunto de sistemas de administración de bases de datos que suplen las falencias que los sistemas relacionales tienen. Sus principales características son menores tiempos de respuesta al manejar grandes volúmenes de datos, facilidad de trabajar con datos semi-estructurados, gran capacidad de escalabilidad horizontal y su implementación en un ambiente distribuido. Los sistemas NoSQL utilizan la desnormalización y la duplicidad de los datos en distintas ubicaciones con la finalidad de acceder a ellos de una manera más directa, teniendo un elemento completo en un único lugar y no repartido en múltiples tablas que necesitarían la utilización de *join* para entregar el elemento completo (Pandorafms Monitoring Bloc, 2016; Rackspace Cloud University; Santi Pérez, 2015).

La presencia de datos semi-estructurados dio origen a una multitud de formatos de almacenamiento. Con el tiempo fueron siendo evidentes los modelos más utilizados, lo que dio origen a que se definieran diferentes tipos de bases de datos NoSQL, siendo los más populares (Santi Pérez, 2015; Dan Sullivan, 2015):

- **Clave-Valor:** conocida también como ‘diccionario’, es la base de todos los demás modelos NoSQL e incluso de los SABDR. Se diferencian de estos últimos en que no se define una estructura rígida en un comienzo, pudiendo en cualquier momento definir y almacenar nuevas *keys*.
- **Columnar:** también llamadas tabulares, están orientadas al almacenamiento de los datos en ‘columnas’ por sobre el almacenamiento basado en filas. Permiten el almacenamiento de un registro con múltiples valores para una o varias de sus *keys* (por ejemplo, un individuo que tenga varias direcciones de correo) y la definición de ‘familias de columnas’, útiles para análisis OLAP (Ari Amster, 2016; DB-Engines, 2017). Las bases de datos columnares son una variante de las bases de datos clave-valor, siendo su principal diferencia que estas últimas no permiten más de un valor por clave ni sub-estructuras.
- **Documental:** orientadas al almacenamiento de ‘documentos’ en formato JSON y otros derivados de éste. Se basan en la estructura interna de los pares clave-valor para extraer información descriptiva del documento, facilitando su acceso, análisis y manipulación (Dan Gorst, 2014).
- **de Grafos:** orientadas al almacenamiento de los datos y las relaciones que existan entre ellos. Para casos en que la relación entre los datos es tan o más relevante que el dato propiamente tal.

Los distintos tipos de bases de datos NoSQL logran cubrir muchas de las necesidades que los sistemas relacionales no suplen, aunque no dejan de tener sus propias desventajas. Opuestamente a las características ACID que definen a un sistema relacional, los sistemas NoSQL se rigen por el acrónimo BASE (John D. Cook, 2009), que implica una disponibilidad en la mayoría del tiempo (*Basic Availability*), no asegura la durabilidad de los datos en el tiempo (*Soft-state*) y presenta una consistencia parcial de los datos (*Eventual Consistency*). Uno de los mayores defectos de los sistemas NoSQL, con respecto a sus contrapartes relacionales, es que no existe un estándar para este tipo de sistemas, lo que hace limitada las API's existentes para su interacción con otros *softwares* y por ende su implementación. Aún con estos defectos, los sistemas NoSQL se han hecho su espacio en el mercado, debido a que frente a su flexibilidad en el manejo de datos semi-estructurados, su capacidad de mejorar el rendimiento sólo agregando más módulos al sistema (equipos físicos) y sus rápidos tiempos de respuesta gracias a la ausencia de *join*, BASE termina siendo un *trade-off* despreciable y más que aceptable (John D. Cook, 2009).

Bajo la premisa de un rediseño completo de los SABDR, las bases de datos NoSQL consideran tres pilares fundamentales sobre los cuales trabajar, llamado el teorema CAP: consistencia, alta disponibilidad y escalabilidad (Julian Browne, 2009). La consistencia garantiza que a partir de un cambio en el estado de un sistema, todo cliente que acceda posteriormente, obtendrá el nuevo valor del estado. La disponibilidad es la cualidad de un sistema para mantenerse operativo principalmente ante contingencias tales como pueden ser fallas de discos, actualizaciones de bases de datos, actualizaciones de *software*, actualizaciones de sistema operativo, cortes de energía, cortes en la red o movimiento físico del equipo. La escalabilidad es la cualidad de un sistema para reaccionar y adaptarse a nuevos requerimientos sin perder calidad, manejar el crecimiento continuo de trabajo de manera fluida, o bien estar preparado para crecer sin perder calidad en los servicios ofrecidos (Rederencia 16). Existe un cuarto pilar que se conoce como tolerancia a fallos, la cual garantiza que un sistema continúe funcionando a pesar de pérdidas en los mensajes por particiones. Si bien es deseable contar con estas características, no es posible hacerlo plenamente y en forma simultánea. Es necesario renunciar o bien sacrificar parcialmente una característica para obtener las otras.

En el caso particular de las bases de datos columnares, estas están diseñadas para realizar consultas de agregación sobre grandes volúmenes de datos. Su comportamiento es similar a las bases de datos relacionales pero en el almacenamiento radica su gran diferencia, ya que los sistemas columnares almacenan columnas de datos en lugar de registros (DB-Engines, 2017). Entre las más populares se encuentran Cassandra, HBase, BigTable y SimpleDB (Rick Grehan, 2014).

Las bases de datos documentales están diseñadas para el almacenamiento de datos semi-estructurados. Son las más versátiles ya que permiten definir la estructura de almacenamiento de manera muy fiel a la realidad. Estos documentos son almacenados en formatos tales como JSON o XML. Entre las más populares se encuentran MongoDB, DynamoDB, CouchDB y RavenDB (DB-Enginers, 2017).

Las bases de datos de grafos están diseñadas para almacenar información en modelos de datos con muchas relaciones entre ellos, como pueden ser las redes sociales. Se basan en la teoría de grafos, utilizando nodos y aristas para representar los datos almacenados (Juan F. Cía, 2015). Entre las más populares se encuentran Neo4j, InfinityGraph y GraphDB (DB-Enginers, 2017).

Existe una discusión sobre si los beneficios que ofrecen las bases de datos NoSQL son reales o sólo pueden ser alcanzados bajo situaciones ideales y si estas bases de datos son una propuesta seria o bien una moda influenciada por grandes empresas de internet como Facebook, LinkedIn y Amazon.com, entre otros.

La gran mayoría de los sistemas actuales tienen implementadas bases de datos relacionales, por ser las más tradicionales y más conocidas. Sin embargo, la actual dinámica del mundo globalizado está poniendo a prueba la capacidad de estos sistemas: las personas e instituciones utilizan redes sociales para interactuar; las universidades implementan sistemas de *e-learning*; los bancos apoyan el comercio electrónico, el mercado financiero y muchas actividades más que son producto del cambio hacia el acceso a todos los servicios a través de internet, sobre todo con el auge del internet de las cosas (Genbetadev, 2014). Los sistemas NoSQL poseen las capacidades necesarias para apoyar este cambio y avance tecnológico. Varios servicios empiezan a utilizar NoSQL aunque otros son dependientes de SQL y el cambio implicaría un gran costo operacional. Pese a lo anterior, grandes empresas se encuentran incursionando en esta tecnología. Una de estas empresas es Twitter, que emplea tanto SQL como NoSQL en sus sistemas, demostrando que ambas pueden convivir perfectamente (JoeSek, 2010). Netflix, Cisco, HP, Facebook y Adobe, por nombrar algunos, son empresas que están comensando a utilizar Cassandra, base de datos NoSQL del tipo columnar (JoSek, 2010). Google, Square Enix, Disney, IGN, The New York Times y Forbes son otros ejemplos de que el movimiento NoSQL está ganando adeptos, en este caso con el uso de MongoDB, base de datos NoSQL del tipo documental (CIO, 2015). NASA, Walmart, ebay, LinkedIn también se han sumado a este movimiento, mejorando sus servicios al utilizar Neo4J, base de datos NoSQL del tipo de grafos (Neo Technology, Inc., 2017).

Desde la perspectiva de los adeptos a los SABDR se pueden mencionar las siguientes críticas a las bases de datos NoSQL:

- El mercado NoSQL está muy fragmentado (Alan Downin, 2011).
- Cada uno de los motores de bases de datos NoSQL posee su propia interfaz, haciendo de su adaptación una inversión significativa. Debido al nivel de especialización, una compañía podría tener que instalar más de una de estas bases de datos (Yury Izrailevsky, 2011).
- Utilizar una base datos NoSQL implica usar un modelo de desarrollo de aplicaciones diferente a la tradicional arquitectura de 3 capas, por lo cual, una aplicación existente de 3 capas no puede ser simplemente convertida para bases de datos NoSQL, requiere una reestructuración del modelo de desarrollo (Alan Downing, 2011).
- Con frecuencia el teorema CAP se reduce a elegir dos de tres: consistencia, disponibilidad o tolerancia a particiones (Michael Stonebraker, 2010).
- Los modelos de datos sin una estructura rígida podrían ser una mala decisión de diseño. Si bien los modelos de datos semi-estructurados son flexibles desde el punto de vista del diseñador, son difíciles para consultar sus datos. En este tipo de modelado de datos el manejo es delegado a la capa de aplicación (Alan Downing, 2011).

Por otro lado, desde la perspectiva de los adeptos a los sistemas NoSQL se pueden mencionar las siguientes razones para desarrollar y utilizar estos almacenamientos:

- El alto rendimiento que los sistemas NoSQL proveen es de varios órdenes de magnitud superior. Cassandra puede escribir en un almacenamiento de datos más de 50 GB en sólo 0.12 milisegundos, mientras que MySQL tardaría 300 milisegundos para la misma tarea (Lakshman Avinash & Malik Prashant, 2009).
- Incremento del volumen de información no estructurada y empleo de hardware más económico. En contraste con los SABDR, la mayoría de las bases de datos NoSQL son diseñadas para poder escalar horizontalmente y no tener que depender de hardware de última generación (Javier Soltero, 2009).
- Evitar el costoso mapeo objeto-relacional. La mayoría de las bases de datos NoSQL son diseñadas para almacenar estructuras de datos más simples o más similares a las utilizadas en los lenguajes de programación orientados a objetos. De esta forma, se evita costosos mapeos objeto-relacional (Christof Strauch, 2010).

Capítulo 3: Diseño del Trabajo

En este capítulo se describe la metodología aplicada en el estudio de los sistemas de bases de datos NoSQL columnares, documentales y de grafos. Se definen los escenarios en los cuales fueron estudiados cada tipo de base de datos NoSQL. Se presentan los criterios de comparación considerados, alineados con los objetivos del trabajo.

Para el estudio de los sistemas de bases de datos NoSQL columnar, documental y de grafos fue definido un escenario para cada sistema de gestión. Cada escenario fue diseñado de modo tal de resaltar las virtudes que caracterizan a dicho tipo de base de datos NoSQL, mostrando el nivel de especialización de cada motor.

3.1 Bases de Datos Columnares

A la hora de elegir, existen varias bases de datos del tipo columnar, entre las que se pueden nombrar BigTable, Cassandra, Hana, HBase, Hypertable y SimpleDB. Google's BigTable y Amazon's SimpleDB son grandes opciones, con muchas herramientas y características, a tal punto que pueden llegar a ser atemorizantes para nuevos usuarios, perdiendo el objetivo de este trabajo de memoria de acercar a las tecnologías NoSQL, por lo que no fueron escogidos (Apache Cassandra, 2016; The Apache Software Foundation, 2007; Hypertable Inc., 2014). SAP Hana es la base de datos que utiliza SAP, un *software* para gestión de recursos empresariales (ERP por sus siglas en inglés); para trabajar con Hana es necesario tener una licencia para SAP y, además, saber utilizar SAP, lo que se escapa del alcance de este trabajo (Ref: SAP, 2017). Cassandra y HBase son ambas bases de datos codificadas en Java y licenciadas por Apache; son parte de un proyecto *open-source* llamado Apache Hadoop (Ref: Ari Amster, 2016). Tras probar las dos, se decidió utilizar Cassandra por su mayor similitud a SQL y su implementación más directa en sistema Windows. Buscando cubrir alternativas diferentes, no se utilizó HBase como segunda opción por la similitud de arquitectura con Cassandra, quedando así Hypertable, quien logró cumplir los objetivos de facilidad de uso y rápida familiaridad con el sistema (Ref: ht4w, Hypertable for Windows).

3.2 Bases de Datos Documentales

Cuando se habla de las bases de datos del tipo documental se reduce a unas pocas las que verdaderamente resaltan, entre ellas Apache CouchDB, Azure DocumentDB y MongoDB. Debido a su gran popularidad y presencia en el mercado se escogieron Apache CouchDB y MongoDB (Ref: The Apache Software Foundation, 2007). Ambas se encuentran cabeza a cabeza como los mejores sistemas de bases de datos documentales lo que hace más que interesante desde un enfoque de desarrollo profesional el aprender sobre ellas.

3.3 Bases de Datos de Grafos

Las bases de de datos del tipo grafo son reducidas debido a la particularidad de necesidades que cubren, y entre ellas, es Neo4j la que siempre lidera la elección de los consumidores. Como segunda opción, se evaluaron dos bases de datos más, GraphDB y ArangoDB. Tras utilizar ambos sistemas se optó por continuar con ArangoDB ya que cumplía de mejor manera las expectativas de este trabajo, destacándose sobre GraphDB tanto en la instalación y configuración como en la manipulación y simplicidad del sistema (Ref: Neo Technology, Inc., 2017; ArangoDB Company, 2017).

3.4 Criterios de Comparación

Al comenzar el aprendizaje de bases de datos, la decisión de ‘por dónde empezar’ se reduce a escoger MySQL, SQLServer u Oracle, por nombrar algunos; sin embargo, son todas plataformas de bases de datos relacionales. Cuando se habla de comenzar el aprendizaje en sistemas NoSQL la decisión no es sólo de la plataforma, sino también del tipo de paradigma, lo que complejiza la definición de criterios a considerar para escoger la herramienta que mejor se acomode a las necesidades de un principiante.

Cuando se habla de aprender una nueva tecnología de la información, los lenguajes de programación son los que mayor variedad de opciones ofrecen en su rubro (Osmary Guevara, 2015), lo que ha permitido que se fijen puntos claves donde centrar la atención al momento de elegir con cuál comenzar. Es deseable un sistema de fácil instalación y puesta en marcha, con una corta curva de aprendizaje que se complemente con la documentación adecuada para consultar (Takub, 2014). A lo anterior se suma el estilo o enfoque que se desea aprender (orientado a objeto, desarrollo web, aplicaciones móviles, por ejemplo) y la popularidad y aplicación en el mercado, ya que esto es clave cuando se habla de oportunidades laborales y salariales (Noé Cruz, 2017).

Extrapolando los puntos nombrados en el párrafo anterior, se definieron los criterios que se emplearon para evaluar a los motores de bases de datos. Los criterios van enfocados en la búsqueda de la mejor experiencia posible para una persona que inicia en el área de motores de bases de datos NoSQL, donde la facilidad de uso, documentación disponible y curva de aprendizaje están entre los aspectos más relevantes (ver tabla 1).

Criterio	Descripción
Instalación	Proceso de instalación, configuración y puesta en marcha de la base de datos.
Manejo de Datos CRUD	Complejidad de las sentencias para la creación, lectura, búsqueda, actualización y eliminación de datos.
Documentación y Soporte	Cantidad, calidad y facilidad de acceso a la documentación y soporte técnico.
Curva de Aprendizaje	Similitud en la confección de sentencias con SQL u otros sistemas afines. Grado de dificultad de introducción y mejoramiento.
Claridad del Paradigma	La herramienta deja en clara evidencia el paradigma que abarca, diferenciándose de los demás.
Interfaz Gráfica	Facilidad de uso, ubicación de mensajes de error/alerta, íconos y ubicación de funcionalidades bajo etiquetas con nombres intuitivos.
Relevancia en el Mercado	Presencia en la industria. Importancia de conocer la herramienta y el valor que entrega al profesional en su carrera.

Tabla 1. Criterios de comparación.

Capítulo 4: Desarrollo del Trabajo

Este capítulo se presenta dividido en 3 secciones, cada una centrada en uno de los tipos de bases de datos NoSQL: columnar, documental y de grafos. En cada sección se presentan los motores de bases de datos seleccionados en el capítulo anterior correspondientes al paradigma anunciado. Para cada base de datos se hace una presentación del motor de base de datos, su instalación, creación y manejo de datos, terminando con un análisis comparativo entre las bases de datos estudiadas. El capítulo finaliza con una propuesta de trabajo académico para la introducción a cada tipo de sistema de base de datos NoSQL y el motor de base de datos ideal para este propósito.

4.1 Bases de Datos Columnares

En esta sección se presentan los motores de bases de datos columnares utilizados en el trabajo: Cassandra e Hypertable

4.1.1 Apache Cassandra v3.0.9

Estrenada en 2008, Cassandra es una base de datos columnar basada en ideas extraídas de Google's BigTable y Amazon's DynamoDB. Escrita en Java y licenciada por Apache, Cassandra ofrece escalabilidad y alta disponibilidad sin comprometer rendimiento (Referencia 22). Posee gran tolerancia a fallos gracias a su eficiente característica de duplicidad de nodos. Cassandra prioriza la disponibilidad de los datos y la flexibilidad en el uso de particiones por sobre la consistencia de los datos (Ref: Julian Browne, 2009).

Instalación

Como primer paso se debe instalar Java Development Kit (JDK). Este componente es de libre acceso desde el sitio de Oracle (Ref: Oracle, 2017) y será necesario para el correcto funcionamiento de varias de las bases de datos de este trabajo de memoria.

Para instalar Cassandra en Windows se descarga desde su distribuidor certificado DataStax (Ref: Datastax, 2017). Siguiendo los pasos del instalador, tras pocas ventanas de configuración y ubicación de la instalación, Cassandra ya está instalado en el sistema, listo para operar utilizando su terminal CQL Shell (ver figuras 1, 2 y 3).

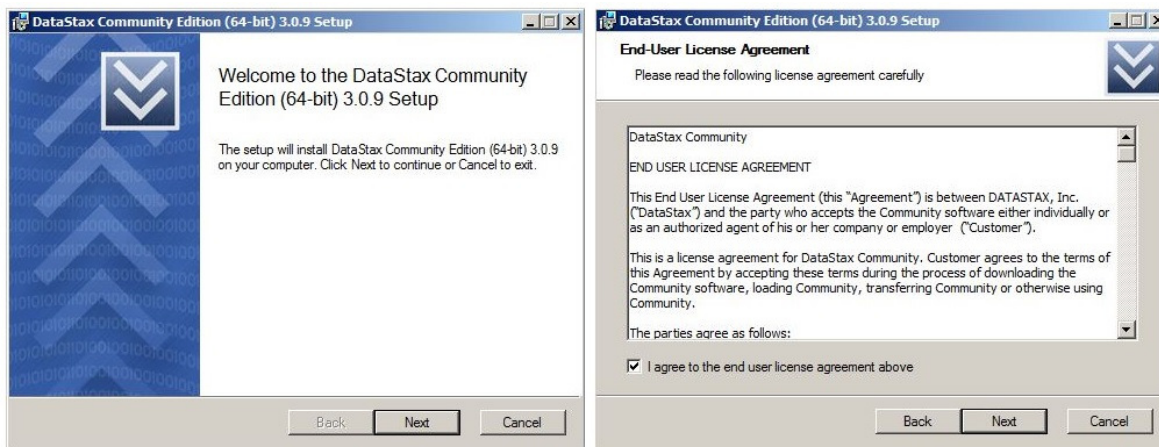


Figura 1. Instalación de Cassandra (1).

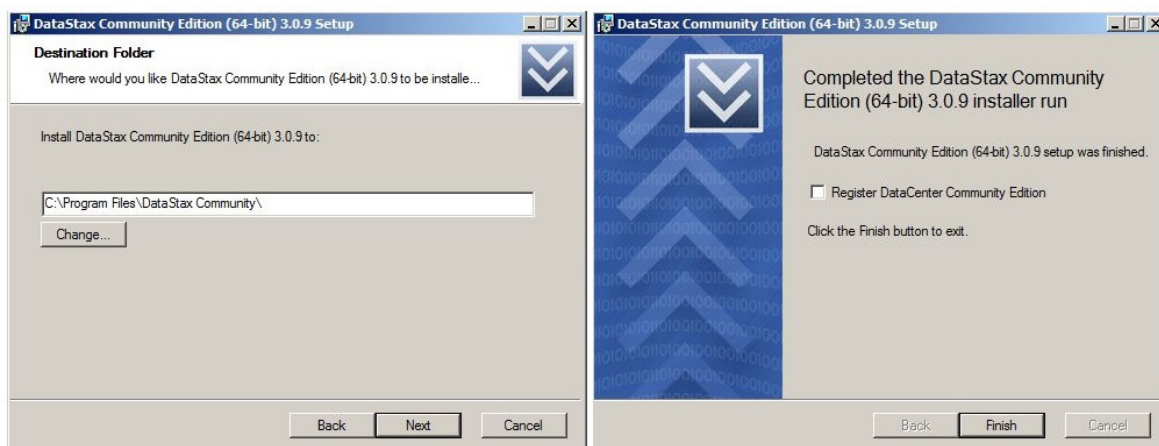


Figura 2. Instalación de Cassandra (2).

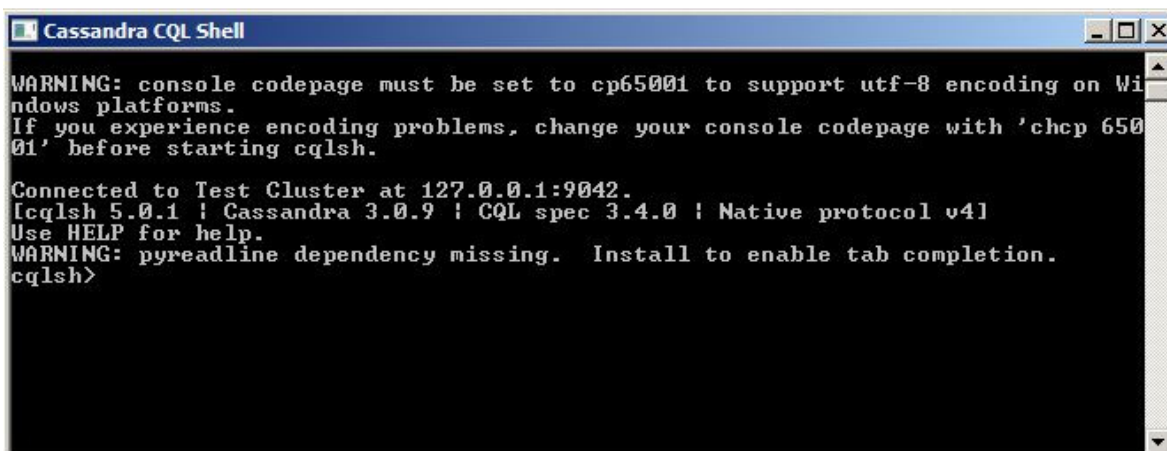
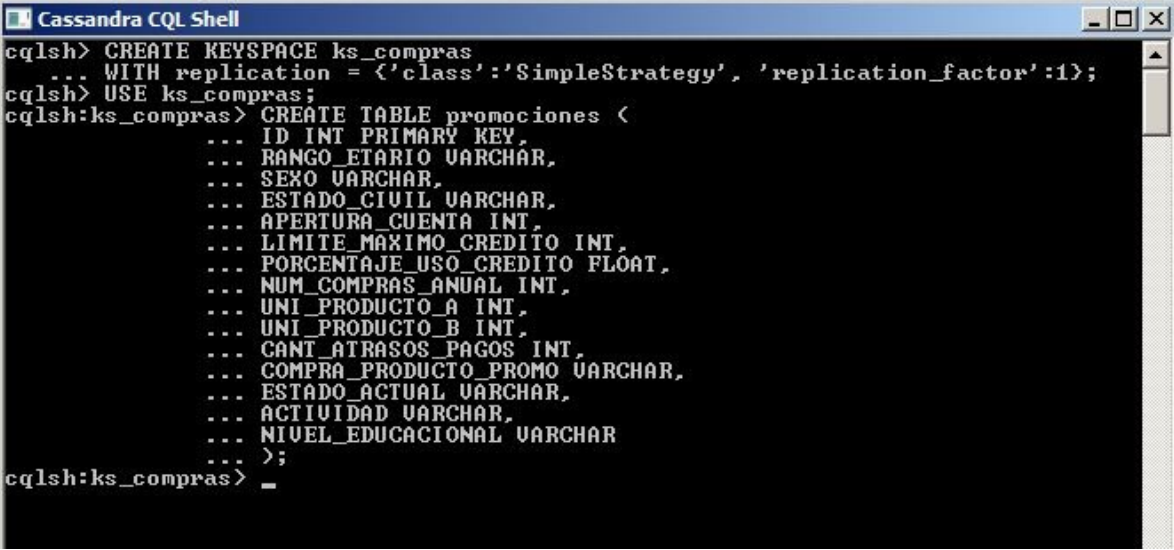


Figura 3. Cassandra CQL Shell.

Existe una herramienta gráfica llamada DataStax DevCenter para trabajar con Cassandra (Ref: Datastax, 2017); sin embargo se prescindió de su uso debido a que la interacción con la base de datos por medio del terminal CQL Shell cumple de mejor manera los objetivos propuestos en esta memoria.

Creación y Manejo de Datos

Para trabajar con Cassandra se utiliza CQL (Ref: Apache Cassandra, 2016), similar en sintaxis con SQL, por lo que resulta familiar su uso y aplicación. En Cassandra se debe definir un *keyspace* donde se deben crear sus tablas. Los *keyspaces* deben crearse según la arquitectura de la implementación. Para el caso de una única máquina, se indica ‘SimpleStrategy’ y un factor de replicación de 1, que representa la redundancia de los datos. Definiendo el *keyspace*, se utiliza y luego se crea una tabla del mismo modo que se haría en un sistema SQL (ver figura 4).



```
Cassandra CQL Shell
cqlsh> CREATE KEYSPACE ks_compras
... WITH replication = {'class':'SimpleStrategy', 'replication_factor':1};
cqlsh> USE ks_compras;
cqlsh:ks_compras> CREATE TABLE promociones <
... ID INT PRIMARY KEY,
... RANGO_ETARIO VARCHAR,
... SEXO VARCHAR,
... ESTADO_CIVIL VARCHAR,
... APERTURA_CUENTA INT,
... LIMITE_MAXIMO_CREDITO INT,
... PORCENTAJE_USO_CREDITO FLOAT,
... NUM_COMPRAS_ANUAL INT,
... UNI_PRODUCTO_A INT,
... UNI_PRODUCTO_B INT,
... CANT_ATRASOS_PAGOS INT,
... COMPRA_PRODUCTO_PROMO VARCHAR,
... ESTADO_ACTUAL VARCHAR,
... ACTIVIDAD VARCHAR,
... NIVEL_EDUCACIONAL VARCHAR
... >;
cqlsh:ks_compras> _
```

Figura 4. Ejemplo de comando *create keyspace* y *create table* de Cassandra.

La sintaxis para crear nuevos registros es idéntica a SQL. Utilizando el comando *insert* se podemos poblar la tabla. Un ejemplo se muestra en la figura 5:

```
Cassandra CQL Shell
cqlsh:ks_compras> INSERT INTO promociones <
... ID,
... RANGO_ETARIO,
... SEXO,
... ESTADO_CIVIL,
... APERTURA_CUENTA,
... LIMITE_MAXIMO_CREDITO,
... PORCENTAJE_USO_CREDITO,
... NUM_COMPRAS_ANUAL,
... UNI_PRODUCTO_A,
... UNI_PRODUCTO_B,
... CANT_ATRASOS_PAGOS,
... COMPRA_PRODUCTO_PROMO,
... ESTADO_ACTUAL,ACTIVIDAD,
... NIVEL_EDUCACIONAL>
... VALUES <
... 1,
... 'MENOR QUE 30',
... 'MASCULINO',
... 'SOLTERO',
... 2014,
... 301500,
... 100,
... 6,
... 0,
... 5,
... 16,
... 'SI',
... 'DOS MESES',
... 'DEPENDIENTE',
... 'ESTUDIANTE UNIVERSITARIO'
... >;
cqlsh:ks_compras>
```

Figura 5. Ejemplo de comando *insert* de Cassandra.

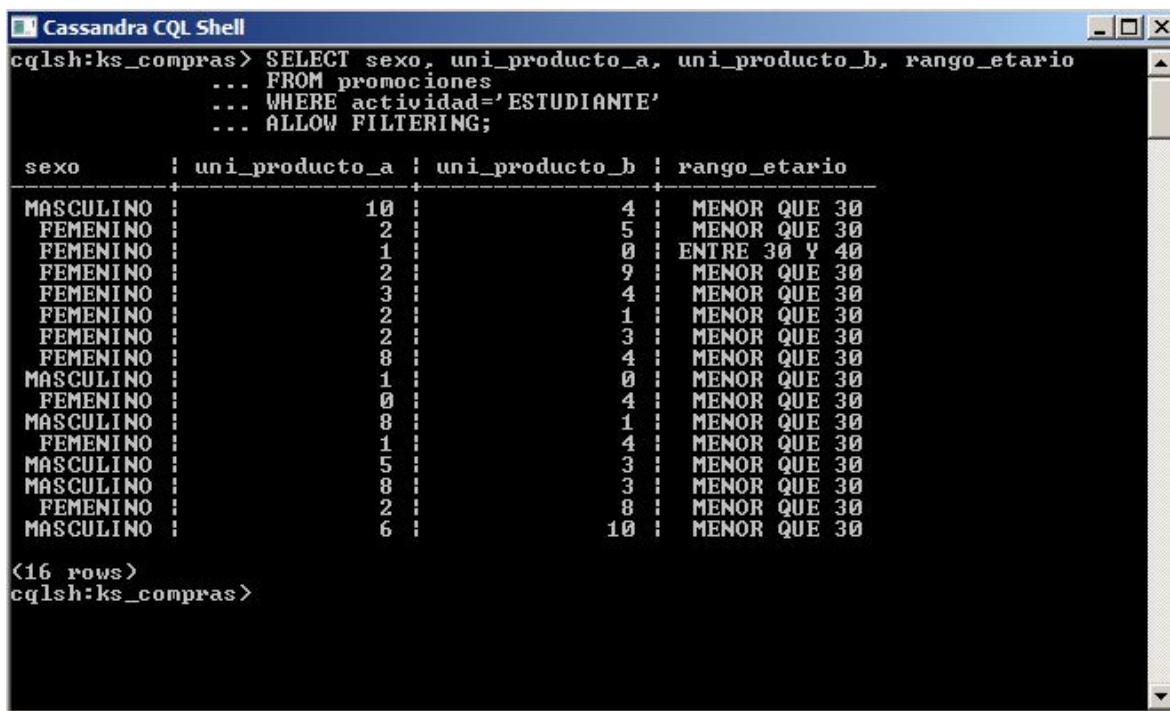
Consultar una tabla resulta igual de familiar. Utilizando el comando *select* se puede obtener información de los datos contenidos en la tabla, como se ejemplifica en la figura 6.

```
Cassandra CQL Shell
cqlsh:ks_compras> SELECT sexo, uni_producto_a, uni_producto_b, rango_etario
... FROM promociones
... WHERE actividad='ESTUDIANTE'
... ;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
cqlsh:ks_compras>
```

Figura 6. Ejemplo de comando *select* de Cassandra.

Si se intenta una consulta con cláusula *where* sobre una clave que no sea primaria o no esté indexada, Cassandra mostrará una advertencia debido al alto costo de búsqueda. La opción *allow filtering* permite saltarse esta condición y hacer de todas formas la consulta. Para grandes volúmenes de datos, se puede usar *allow filtering* junto a *limit* de modo que el

set de datos consultado esté acotado; sin embargo, esto no asegura un buen rendimiento ni fiabilidad de la respuesta retornada. Lo mejor es tener mucha información sobre lo que se busca, definiendo varias cláusulas *where*, lo que resultaría en una consulta de costo relativamente bajo, incluso cuando ninguna de las claves en las cláusulas *where* esté indexada (ver figura 7).



```

cqlsh:ks_compras> SELECT sexo, uni_producto_a, uni_producto_b, rango_etario
... FROM promociones
... WHERE actividad='ESTUDIANTE'
... ALLOW FILTERING;

```

sexo	uni_producto_a	uni_producto_b	rango_etario
MASCULINO	10	4	MENOR QUE 30
FEMENINO	2	5	MENOR QUE 30
FEMENINO	1	0	ENTRE 30 Y 40
FEMENINO	2	9	MENOR QUE 30
FEMENINO	3	4	MENOR QUE 30
FEMENINO	2	1	MENOR QUE 30
FEMENINO	2	3	MENOR QUE 30
FEMENINO	8	4	MENOR QUE 30
MASCULINO	1	0	MENOR QUE 30
FEMENINO	0	4	MENOR QUE 30
MASCULINO	8	1	MENOR QUE 30
FEMENINO	1	4	MENOR QUE 30
MASCULINO	5	3	MENOR QUE 30
MASCULINO	8	3	MENOR QUE 30
FEMENINO	2	8	MENOR QUE 30
MASCULINO	6	10	MENOR QUE 30

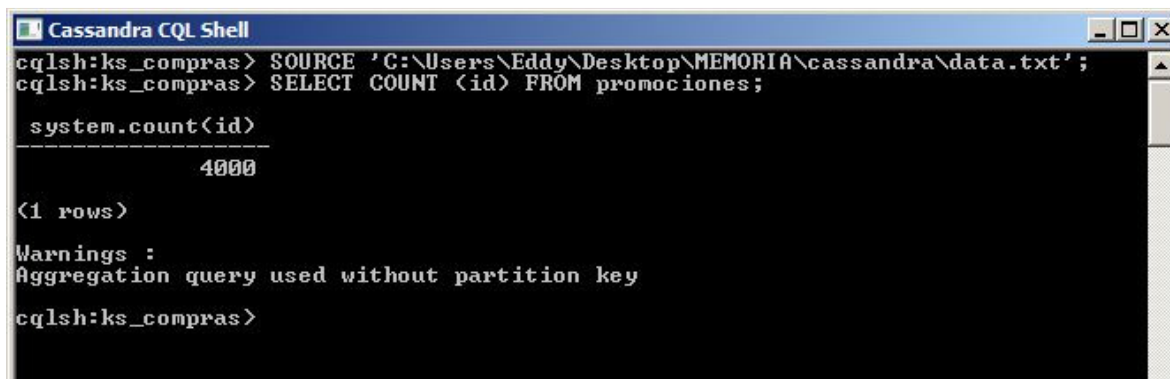
```

<16 rows>
cqlsh:ks_compras>

```

Figura 7. Ejemplo de comando *select* con *allow filtering* de Cassandra.

Para ingresar varios registros, se utiliza el comando *source*. En la figura 8 se ejemplifica el uso de este comando al insertar 4000 desde un archivo de origen *txt*. Terminado este proceso, se verificó que los 4000 registros fueran cargados exitosamente.



```

cqlsh:ks_compras> SOURCE 'C:\Users\Eddy\Desktop\MEMORIA\cassandra\data.txt';
cqlsh:ks_compras> SELECT COUNT (<id>) FROM promociones;

```

system.count(<id>)
4000

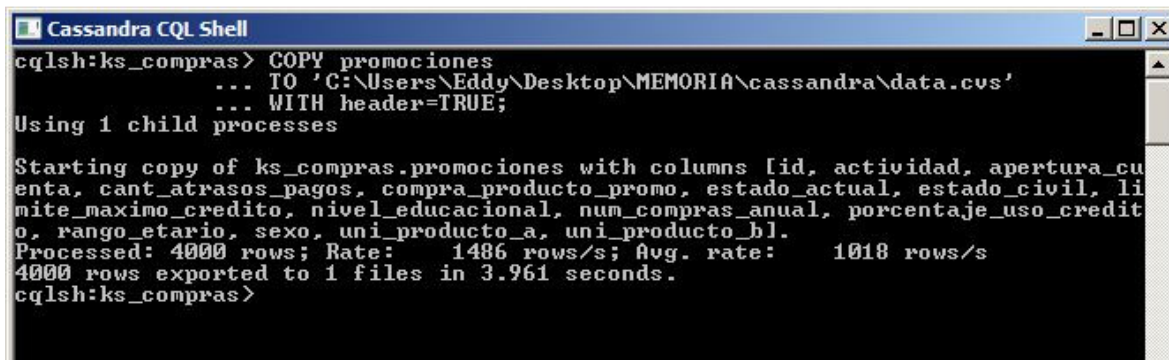
```

<1 rows>
Warnings :
Aggregation query used without partition key
cqlsh:ks_compras>

```

Figura 8. Ejemplo de comando *source* de Cassandra.

Para exportar los registros desde Cassandra se utiliza el comando *copy*, el que genera un archivo en formato *cvs* (Comma-Separated Values) en la dirección indicada (ver figura 9).



```
Cassandra CQL Shell
cqlsh:ks_compras> COPY promociones
... TO 'C:\Users\Eddy\Desktop\MEMORIA\cassandra\data.csv'
... WITH header=TRUE;
Using 1 child processes

Starting copy of ks_compras.promociones with columns [id, actividad, apertura_cu
enta, cant_atrasos_pagos, compra_producto_promo, estado_actual, estado_civil, li
mite_maximo_credito, nivel_educacional, num_compras_anual, porcentaje_uso_credit
o, rango_etario, sexo, uni_producto_a, uni_producto_b].
Processed: 4000 rows; Rate: 1486 rows/s; Avg. rate: 1018 rows/s
4000 rows exported to 1 files in 3.961 seconds.
cqlsh:ks_compras>
```

Figura 9. Ejemplo de comando *copy* de Cassandra.

4.1.2. Hypertable v0.9.8.11

Hypertable es un proyecto *open-source* para la implementación de un sistema de administración de base de datos inspirado en el diseño de Google's BigTable. Su desarrollo inició el 2008 y está escrito mayoritariamente en C++; fue diseñado expresamente con el propósito de resolver el problema de escalabilidad que las tradicionales SABDR no manejan (Referencia 27). Hypertable es considerada a sí misma una versión mejorada de HBase, con una semántica más simplista, mejor rendimiento, menor consumo de recursos y con una estructura de almacenamiento de datos que le permite ajustarse a una amplia gama de aplicaciones.

Instalación

La instalación de Hypertable en Windows se puede realizar utilizando alguno de sus instaladores para Windows que se han desarrollado bajo la licencia “*GNU General Public License, version 3*” (Ref: Free Software Foundation, 2017). En este caso, se utilizó el instalador del proveedor HT4W (Ref: ht4w, Hypertable for Windows). Finalizada la instalación (ver figuras 10 y 11) se ejecuta el archivo *Hypertable.exe* para acceder al terminal de la base de datos, como se muestra en la figura 12.

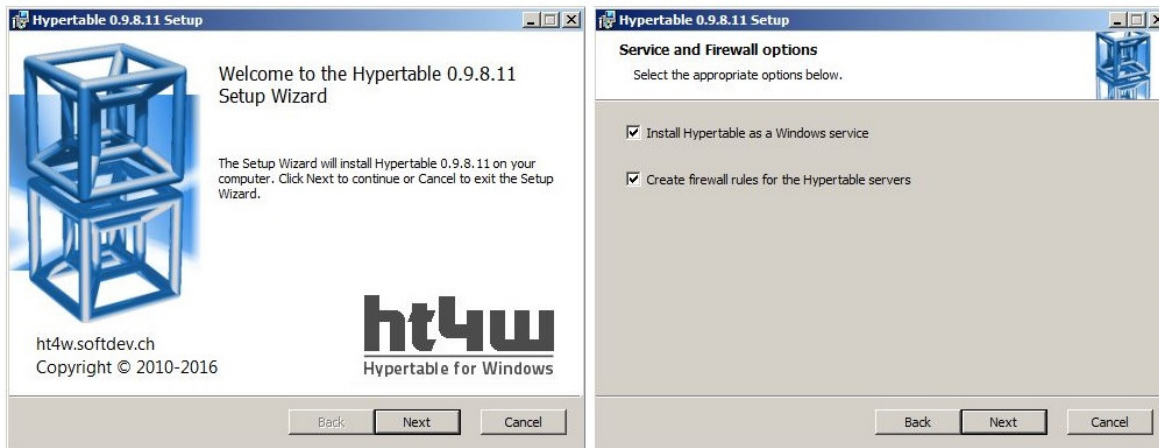


Figura 10. Instalador ht4w para Hypertable (1).

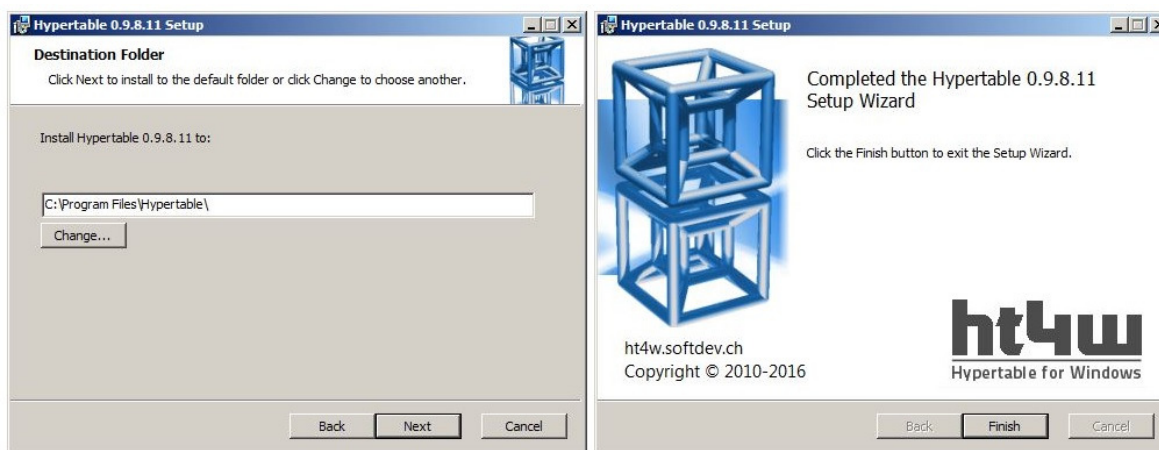


Figura 11. Instalador ht4w para Hypertable (2).

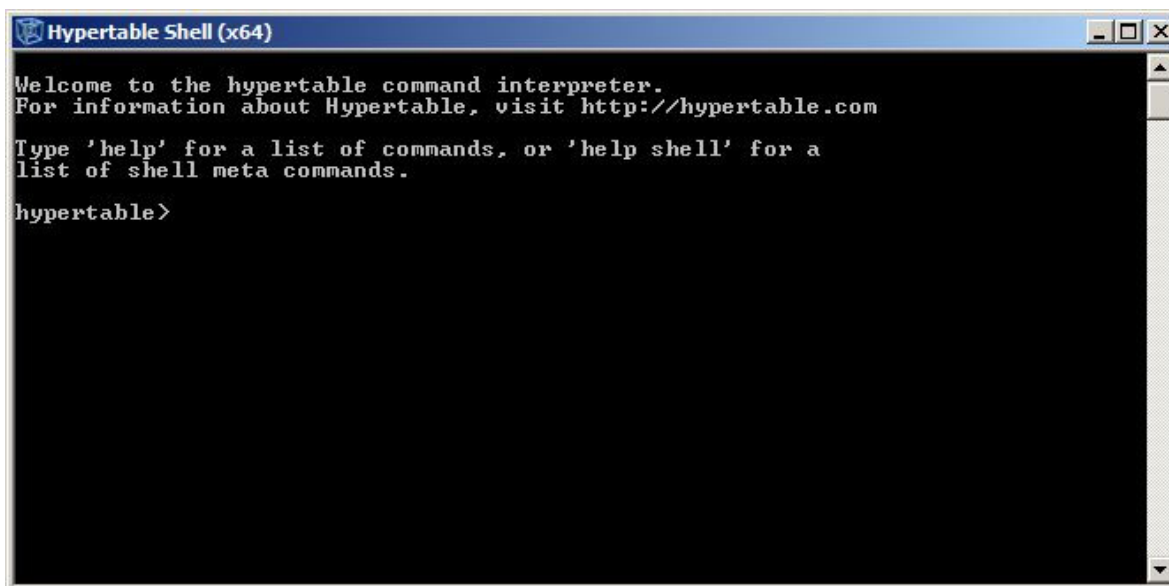
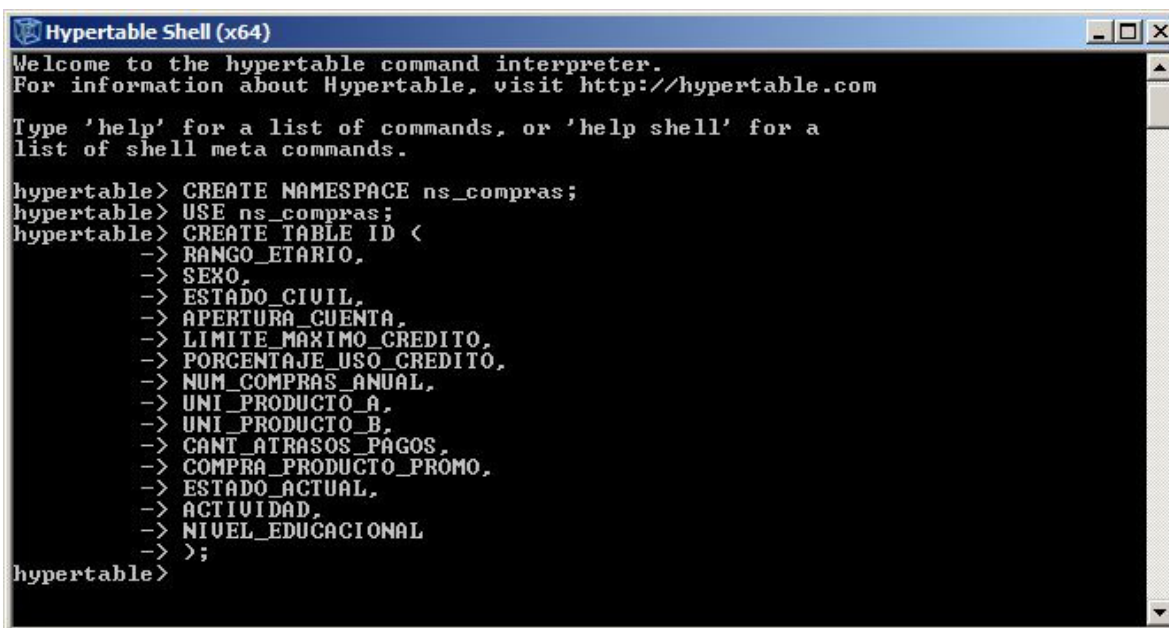


Figura 12. Shell de Hypertable.

Creación y Manejo de Datos

Hypertable cuenta con su lenguaje propio para trabajar, llamado Hypertable Query Language (HQL), el cual comparte gran similitud con la sintaxis de SQL, por lo que para partir se define un *namespace* (*keyspace* en Cassandra, *database* en SQL). Dentro del *namespace* se define una tabla, como se muestra en la figura 13.



```
Hypertable Shell (x64)
Welcome to the hypertable command interpreter.
For information about Hypertable, visit http://hypertable.com

Type 'help' for a list of commands, or 'help shell' for a
list of shell meta commands.

hypertable> CREATE NAMESPACE ns_compras;
hypertable> USE ns_compras;
hypertable> CREATE TABLE ID <
-> RANGO_ETARIO,
-> SEXO,
-> ESTADO_CIVIL,
-> APERTURA_CUENTA,
-> LIMITE_MAXIMO_CREDITO,
-> PORCENTAJE_USO_CREDITO,
-> NUM_COMPRAS_ANUAL,
-> UNI_PRODUCTO_A,
-> UNI_PRODUCTO_B,
-> CANT_ATRASOS_PAGOS,
-> COMPRA_PRODUCTO_PROMO,
-> ESTADO_ACTUAL,
-> ACTIVIDAD,
-> NIVEL_EDUCACIONAL
-> >;
hypertable>
```

Figura 13. Ejemplo de comando *namespace* de Hypertable.

Llama la atención de que en la definición de la tabla no se describa el tipo de dato de cada columna, como que tampoco se describa una clave primaria. Hypertable no soporta tipos de datos, siendo todos los tratados como un conjunto de bytes consecutivos. Hypertable permite apreciar el concepto de *Key-Value* de almacenamiento, ya que la clave que define al valor a almacenar se crea utilizando el nombre de la tabla y la columna, que se aprecia de mejor manera en la inserción de un registro, ejemplificado en la figura 14.

```
Hypertable Shell (x64)
hypertable> INSERT INTO ID VALUES
-> <"1","RANGO_ETARIO","MENOR QUE 30">,
-> <"1","SEXO","MASCULINO">,
-> <"1","ESTADO_CIVIL","SOLTERO">,
-> <"1","APERTURA_CUENTA","2014">,
-> <"1","LIMITE_MAXIMO_CREDITO","301500">,
-> <"1","PORCENTAJE_USO_CREDITO","100">,
-> <"1","NUM_COMPRAS_ANUAL","6">,
-> <"1","UNI_PRODUCTO_A","0">,
-> <"1","UNI_PRODUCTO_B","5">,
-> <"1","CANT_ATRASOS_PAGOS","16">,
-> <"1","COMPRA_PRODUCTO_PROMO","SI">,
-> <"1","ESTADO_ACTUAL","DOS MESES">,
-> <"1","ACTIVIDAD","DEPENDIENTE">,
-> <"1","NIVEL_EDUCACIONAL","ESTUDIANTE UNIVERSITARIO">;

Elapsed time: 0.07 s
Avg value size: 6.57 bytes
Total cells: 14
Throughput: 213.59 cells/s
Resends: 0
hypertable>
```

Figura 14. Ejemplo de comando *insert* de Hypertable.

Hypertable define el nombre de la tabla como una clave más, al que se refiere como *row. row-key* en conjunto definen la clave de una tabla. Una búsqueda de todos los elementos en la tabla (ver figura 15).

```
Hypertable Shell (x64)
hypertable> SELECT * FROM ID;
1      RANGO_ETARIO      MENOR QUE 30
1      SEXO              MASCULINO
1      ESTADO_CIVIL      SOLTERO
1      APERTURA_CUENTA  2014
1      LIMITE_MAXIMO_CREDITO  301500
1      PORCENTAJE_USO_CREDITO  100
1      NUM_COMPRAS_ANUAL    6
1      UNI_PRODUCTO_A      0
1      UNI_PRODUCTO_B      5
1      CANT_ATRASOS_PAGOS   16
1      COMPRA_PRODUCTO_PROMO  SI
1      ESTADO_ACTUAL        DOS MESES
1      ACTIVIDAD            DEPENDIENTE
1      NIVEL_EDUCACIONAL    ESTUDIANTE UNIVERSITARIO
hypertable>
```

Figura 15. Ejemplo de comando *select* de Hypertable.

KEY		VALUE
row	column	
1	RANGO_ETARIO	MENOR QUE 30
1	SEXO	MASCULINO
1	ESTADO_CIVIL	SOLTERO
1	APERTURA_CUENTA	2014
1	LIMITE_MAXIMO_CREDITO	301500
1	PORCENTAJE_USO_CREDITO	100
1	NUM_COMPRAS_ANUAL	6
1	UNI_PRODUCTO_A	0
1	UNI_PRODUCTO_B	5
1	CANT_ATRASOS_PAGOS	16
1	COMPRA_PRODUCTO_PROMO	SI
1	ESTADO_ACTUAL	DOS MESES
1	ACTIVIDAD	DEPENDIENTE
1	NIVEL_EDUCACIONAL	ESTUDIANTE UNIVERSITARIO

Tabla 2. Estructura lógica de almacenamiento en Hypertable, para el *set* de datos de ejemplo.

Si la implementación de Hypertable dependiera de su facilidad en la carga de datos, Hypertable no sería utilizada, ya que la sintaxis particular del comando *insert* haría sumamente trabajosa la carga incluso de pequeños volúmenes de datos como son 4000 registros. El comando *load data infile* permite cargar registros desde un archivo *.txt* si éstos se encuentran en formato *tsv* (*Tab-Separated Value*). Se debe definir las claves correspondientes a la tabla destino, lo que se conoce como encabezado o *header*. El comando *load data infile* permite obtener el *header* de un archivo diferente del que se leen los datos, pero en este caso se define todo en un único archivo, lo que se indica en la primera fila del archivo anteponiendo el “signo gato” (#) en aquella clave que corresponde al *row* (por *default* es la primera leída en el archivo), como se muestra en la figura 16.

#ID	RANGO_ETARIO	SEXO	ESTADO_CIVIL	APERTURA_CUENTA	LIMIT
1	MENOR QUE 30	MASCULINO	SOLTERO	2014	301500 100
2	MENOR QUE 30	FEMENINO	SOLTERO	2006	301500 0
3	ENTRE 30 Y 40	FEMENINO	VIUDO	2011	4711251 100
4	ENTRE 40 Y 50	FEMENINO	SOLTERO	2006	301500 0
5	MAYOR QUE 50	MASCULINO	SOLTERO	2012	301500 35
6	ENTRE 30 Y 40	FEMENINO	CASADO	2005	1219411 100
7	ENTRE 30 Y 40	FEMENINO	SOLTERO	2007	301500 0

Figura 16. Fragmento del documento *data.txt* para Hypertable.

```

Hypertable Shell (x64)
hypertable> LOAD DATA INFILE "C:\Users\Eddy\Desktop\MEMORIA\hypertable\data.txt"
-> INTO TABLE "ID";

Loading 392,864 bytes of input data...

0% 10 20 30 40 50 60 70 80 90 100%
|---|---|---|---|---|---|---|---|---|---|
*****
Load complete.

Elapsed time: 0.38 s
Avg value size: 5.98 bytes
Avg key size: 3.72 bytes
Throughput: 1367966.98 bytes/s (1042212.05 bytes/s)
Total cells: 53130
Throughput: 140746.30 cells/s
Resends: 0
hypertable>

```

Figura 17. Ejemplo de comando *load data infile* de Hypertable.

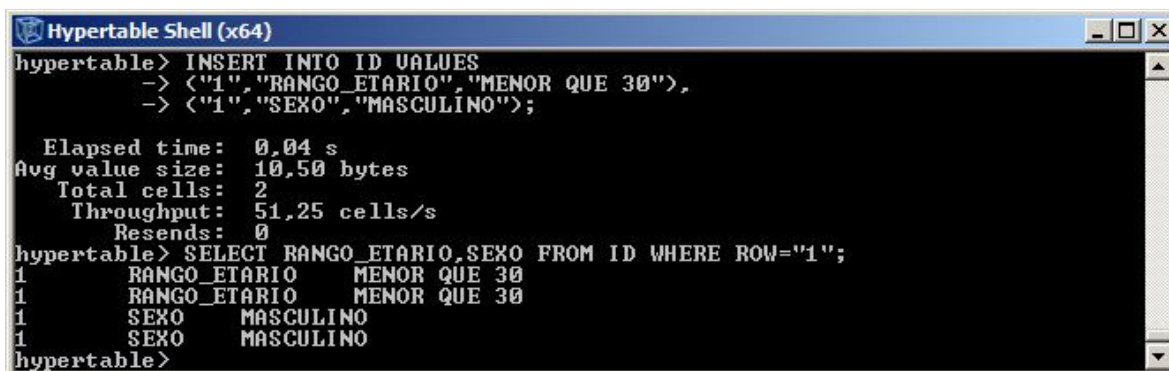
```

Hypertable Shell (x64)
hypertable> SELECT * FROM ID LIMIT 3;
1      RANGO_ETARIO      MENOR QUE 30
1      SEXO      MASCULINO
1      ESTADO_CIVIL      SOLTERO
1      APERTURA_CUENTA 2014
1      LIMITE_MAXIMO_CREDITO      301500
1      PORCENTAJE_USO_CREDITO      100
1      NUM_COMPRAS_ANUAL      6
1      UNI_PRODUCTO_A      0
1      UNI_PRODUCTO_B      5
1      CANT_ATRASOS_PAGOS      16
1      COMPRA_PRODUCTO_PROMO      SI
1      ESTADO_ACTUAL      DOS MESES
1      ACTIVIDAD      DEPENDIENTE
1      NIVEL_EDUCACIONAL      ESTUDIANTE UNIVERSITARIO
10     RANGO_ETARIO      ENTRE 30 Y 40
10     SEXO      MASCULINO
10     ESTADO_CIVIL      CASADO
10     APERTURA_CUENTA 2013
10     LIMITE_MAXIMO_CREDITO      301500
10     PORCENTAJE_USO_CREDITO      0
10     NUM_COMPRAS_ANUAL      5
10     UNI_PRODUCTO_A      5
10     UNI_PRODUCTO_B      4
10     CANT_ATRASOS_PAGOS      15
10     COMPRA_PRODUCTO_PROMO      SI
10     ESTADO_ACTUAL      SIN DEUDA
10     ACTIVIDAD      DEPENDIENTE
10     NIVEL_EDUCACIONAL      ESTUDIANTE UNIVERSITARIO
100    RANGO_ETARIO      ENTRE 30 Y 40
100    SEXO      MASCULINO
100    ESTADO_CIVIL      SOLTERO
100    APERTURA_CUENTA 2005
100    LIMITE_MAXIMO_CREDITO      301500
100    PORCENTAJE_USO_CREDITO      0
100    NUM_COMPRAS_ANUAL      9
100    UNI_PRODUCTO_A      3
100    UNI_PRODUCTO_B      2
100    CANT_ATRASOS_PAGOS      13
100    COMPRA_PRODUCTO_PROMO      NO
100    ESTADO_ACTUAL      SIN DEUDA
100    ACTIVIDAD      DEPENDIENTE
100    NIVEL_EDUCACIONAL      EDUC. TECNICA
hypertable>

```

Figura 18. Ejemplo de comando *select limit* de Hypertable.

Se comentó lo curioso que resulta el no tener que definir una clave primaria en la tabla, lo que hace pensar que el conjunto *row-key* supliría esta necesidad de distinguir dos registros de forma única. Entonces, si se intenta ingresar nuevamente una dupla *row-key* ya existente deberá indicar que no es posible debido a que no pueden existir elementos duplicados (ver figura 19).



```

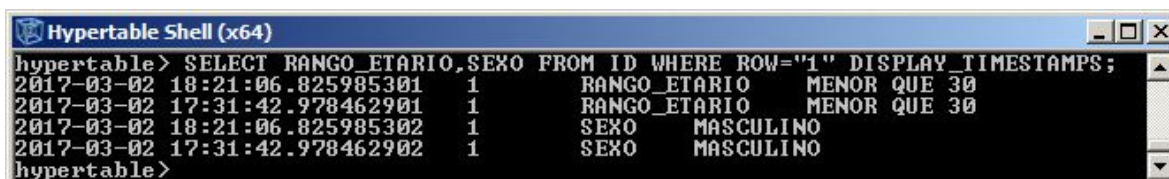
Hypertable Shell (x64)
hypertable> INSERT INTO ID VALUES
-> <"1", "RANGO_ETARIO", "MENOR QUE 30">,
-> <"1", "SEXO", "MASCULINO">;

Elapsed time: 0.04 s
Avg value size: 10.50 bytes
Total cells: 2
Throughput: 51.25 cells/s
Resends: 0
hypertable> SELECT RANGO_ETARIO, SEXO FROM ID WHERE ROW="1";
1      RANGO_ETARIO      MENOR QUE 30
1      RANGO_ETARIO      MENOR QUE 30
1      SEXO              MASCULINO
1      SEXO              MASCULINO
hypertable>

```

Figura 19. Ejemplo de comando *insert* con datos ya existente en Hypertable.

Resulta que Hypertable genera su propia clave primaria según el momento en que se hizo el *insert*. La cláusula *timestamps* permite ver la fecha y hora en que se ingresó un dato, y es la forma de diferenciar de forma unívoca dos registros (ver figura 20). Por *default* el sistema utiliza las variables de sistema para crea esta estampilla de tiempo pero puede ser ingresada manualmente.



```

Hypertable Shell (x64)
hypertable> SELECT RANGO_ETARIO, SEXO FROM ID WHERE ROW="1" DISPLAY TIMESTAMPS;
2017-03-02 18:21:06.825985301 1      RANGO_ETARIO      MENOR QUE 30
2017-03-02 17:31:42.978462901 1      RANGO_ETARIO      MENOR QUE 30
2017-03-02 18:21:06.825985302 1      SEXO              MASCULINO
2017-03-02 17:31:42.978462902 1      SEXO              MASCULINO
hypertable>

```

Figura 20. Ejemplo de comando *select* con opción *display_timestamps* de Hypertable.

Ante la necesidad de extraer los datos, el comando *dump* permite realizar un *backup* de una tabla de la base de datos a un archivo *txt* (ver figura 21).



```

Hypertable Shell (x64)
hypertable> DUMP TABLE ID
-> INTO FILE "C:\Users\Eddy\Desktop\MEMORIA\hypertable\backup.txt";
hypertable>

```

Figura 21. Ejemplo de comando *dump* table de Hypertable.

4.1.3 Comparación Preliminar

En la tabla 3 se presentan los sistemas de bases de datos tratados sometidos a los criterios de comparación, destacando cuál se desempeña mejor en cada punto.

Criterio	Cassandra	Hypertable	Argumento
Instalación	✓	✗	Si bien ninguno tuvo inconvenientes, Cassandra permite una instalación limpia tanto en sistemas Windows como Linux. Para el caso de Hypertable, hubo que buscar una versión ejecutable para Windows.
Manejo de Datos CRUD	---	---	Ambos sistemas utilizan una sintaxis similar a SQL, lo que resulta en instrucciones CRUD fáciles de aprender e integrar.
Documentación y Soporte	✓	✗	Ambos sistemas cuentan con documentación de fácil acceso en sus canales oficiales de comunicación. Sin embargo, Cassandra, al ser más conocida (Referencia 45), posee un mayor número de usuarios y adeptos, lo que resulta en un mayor número comunidades y foros.
Curva de Aprendizaje	---	---	Ambos sistemas entregan las herramientas necesarias para el auto-aprendizaje. Si bien Cassandra posee una mayor comunidad de apoyo, esto ya fue considerado en el punto anterior.
Claridad del Paradigma	✓	✗	Este punto es debatible, ya que el paradigma columnar es de los más difíciles de entender cuando se viene de la estructura rígida de filas y registros de SQL. Los grupos de familia de columnas de Cassandra ayudan a este entendimiento.
Interfaz Gráfica	✓	✗	Ambos sistemas usan como interfaz básica una consola con líneas de comando. En esta consola, Cassandra despliega de mejor manera los datos y otro tipo de información relevante, además de contar con un <i>add-on</i> gráfico dedicado.

Relevancia en el Mercado			Importantes industrias como Facebook, Netflix, Apple, Reddit y Twitter ya están usando Cassandra o migrando a ella.
--------------------------	---	---	---

Tabla 3. Comparativa entre Cassandra y Hypertable.

Al trabajar con las bases de datos columnares, Cassandra y Hypertable, se apreciaron diferencias tanto en la configuración del motor de bases de datos como la estructura de almacenamiento de las mismas. Utilizando los criterios definidos en este trabajo, Cassandra resultó ser un sistema más maduro. La instalación y puesta en marcha es sumamente sencilla, tanto para sistemas Windows como Linux; por el otro lado, Hypertable le debe su presencia en Windows a la comunidad de la base de datos, ya que fue ésta quien desarrolló un instalador con instrucciones para su implementación en este ambiente. El manejo de los datos fue similar en ambos sistemas, ninguno destacándose sobre el otro, ambos con gran similitud a la sintaxis SQL familiar en el rubro. Cassandra fue una de las primeras bases de datos columnares en hacerse un espacio en la comunidad, lo que le asentó un puesto fijo al momento de evaluar sistemas columnares. Desde entonces esta comunidad no ha hecho otra cosa de nutrir al sistema, lo que ha hecho de Cassandra un sistema más completo y limpio, con mayor soporte y documentación para nuevos adeptos. Sin embargo, lo anterior no significa que Hypertable sea un sistema difícil de asimilar pero sí es cierto que Cassandra resulta más llevadero. Ambos sistemas trabajan el paradigma columnar, paradigma que ambos dejan en claro al trabajar con ellos, pero es Cassandra quien lo refina y lleva a otro nivel con la definición de grupos de familias, ideales para análisis OLAP e Inteligencia de Negocios. Al hablar de la visualización de los datos, ambos sistemas trabajan en principio con una terminal propia de cada uno, mas Cassandra cuenta con mejoras de interfáz gráfico dedicado, soportado de forma oficial.

4.1.4 Propuesta Actividad Académica

A continuación se expone la propuesta académica a realizar por los alumnos, definiendo los elementos esenciales para el correcto aprendizaje del paradigma a estudiar.

Una experiencia académica para acercarse a este tipo de sistema sería el levantamiento de una base de datos Cassandra, poblarla con datos que permitan el modelamiento por columnas y un análisis OLAP. La definición de familias de columnas y análisis en la totalidad de los datos serían claves para cumplir los objetivos de aprendizaje del sistema. A continuación se presenta el encabezado y propósitos de la actividad docente:

“ Los alumnos deberán instalar la base de datos Cassandra y poblarla con un set de datos correspondiente a registros de personas con datos relacionados a la compra de diferentes productos (cantidad de unidades, compra en promoción, rango etario, sexo, crédito) para posteriormente realizar consultas sobre todos ellos en la búsqueda de generar información del conjunto como puede ser el producto más vendido, el promedio de crédito entregado a personas según sexo o rango etario, entre otros.

Se espera que durante esta experiencia el alumno:

- *Asimile los conceptos de Keyspace y Familia de Columnas.*
- *Defina un keyspace y una familia.*
- *Realizar operaciones de creación, eliminación y actualización.*
- *Realizar operaciones de búsqueda.*

El trabajo incluye la redacción de un documento en el que el alumno demuestre el aprendizaje de los puntos anteriormente señalados, el que incluirá las diferentes consultas ejecutadas, acompañadas de capturas de imagen. ”

4.2 Bases de Datos Documentales

En esta sección se presentan los motores de bases de datos documentales utilizados en el trabajo: CouchDB y MongoDB.

4.2.1 CouchDB v2.0.0

CouchDB es un sistema de base de datos *open-source* fundado en el año 2005, formando el año 2008 parte de la fundación Apache. Couch es el acrónimo para *Cluster Of Unreliable Commodity Hardware*. CouchDB es una base de datos enfocada a documentos que busca optimizar el análisis del metadato; trabaja con archivos en formato *json* para el almacenamiento de los datos y *JavaScript* como lenguaje de consulta (Referencia 30). Se rige por las características BASE, dejando a la aplicación manejar cualquier clase de conflicto por inconsistencia (Ref: John D. Cook, 2009).

Instalación

Los diferentes instaladores de CouchDB se encuentran disponibles en su sitio web <http://couchdb.apache.org/>. El instalador de Windows resulta simple de entender y utilizar, entregando información relevante respecto a la instalación. La única salvedad en este

proceso es no instalar CouchDB en un directorio que contenga ‘espacios’ en el nombre, ya que esto podría generarle conflictos a la base de datos en un futuro.

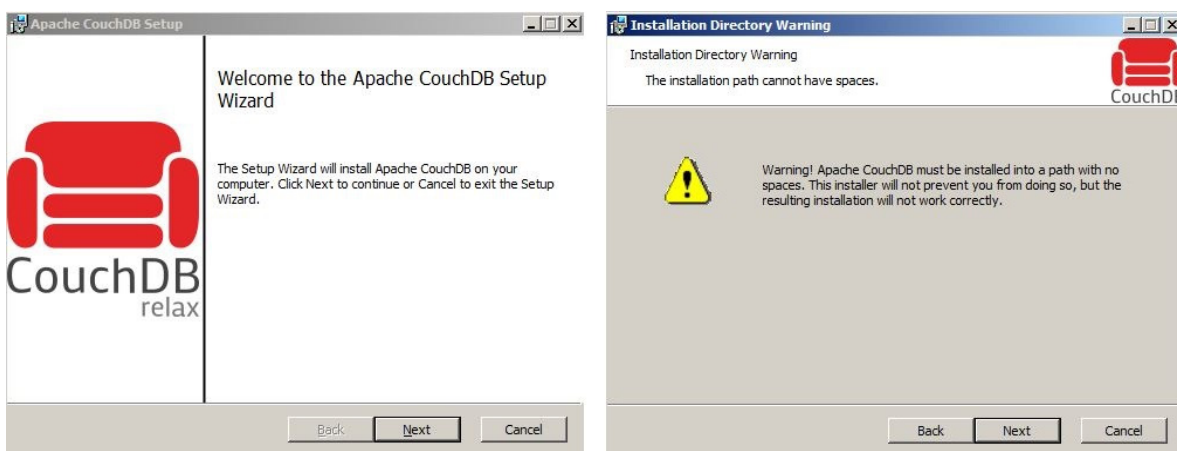


Figura 22. Instalador de CouchDB para Windows (1).

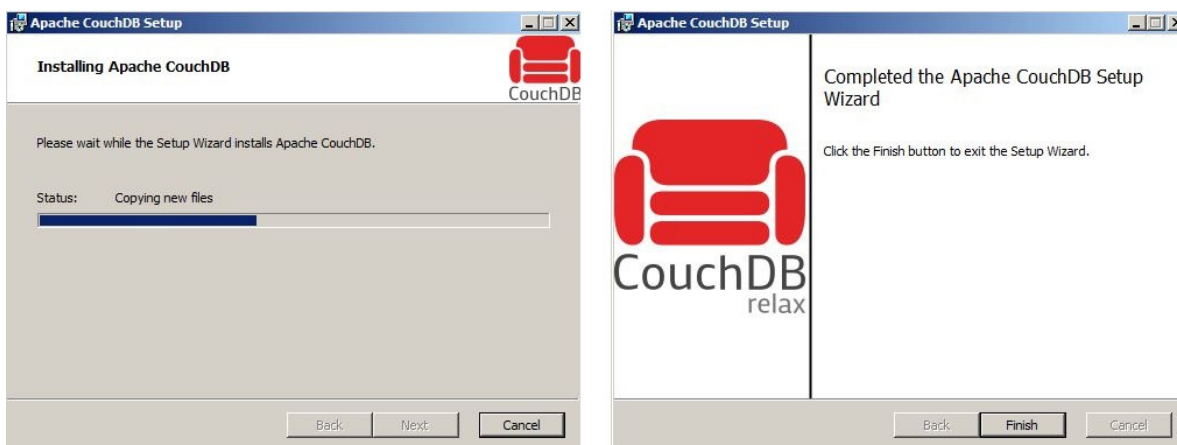


Figura 23. Instalador de CouchDB para Windows (2).

Tras terminada la instalación se puede acceder en un navegador a la dirección *127.0.0.1:5984* donde se indicará la versión de CouchDB instalada (ver figura 24).



Figura 24. Información de la instalación de CouchDB.

Para visualizar el área de trabajo y todas sus herramientas se debe acceder por la dirección `127.0.0.1:5984/_utils/`, conducto principal para trabajar con la base de datos (ver figura 25).

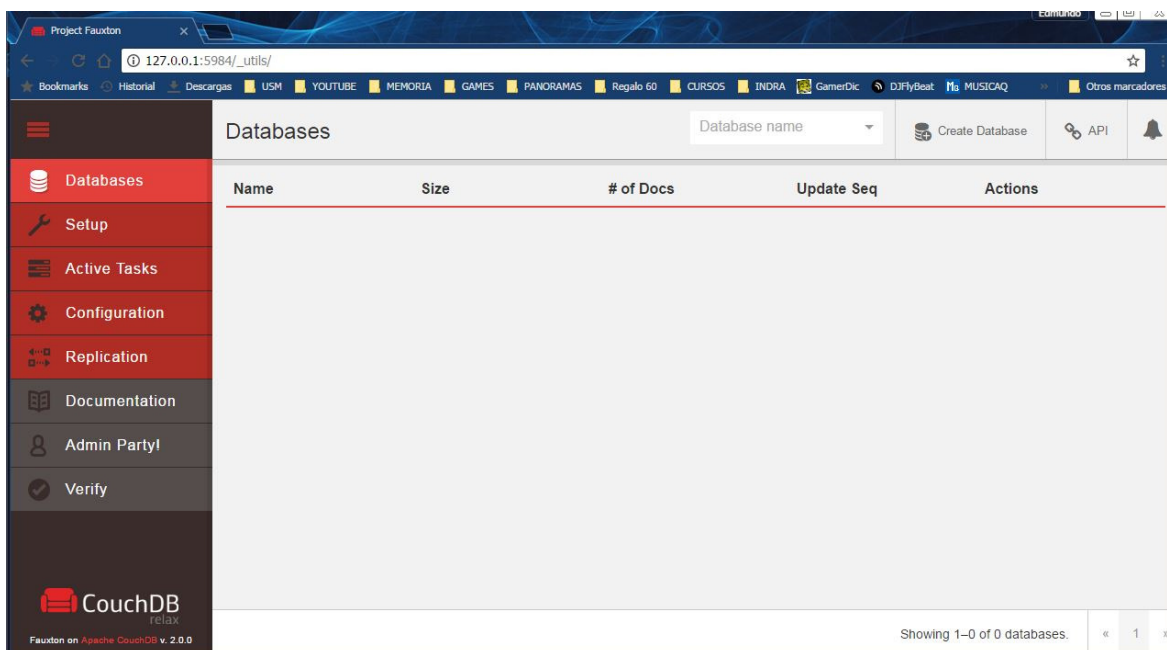


Figura 25. Panel principal de CouchDB.

Creación y Manejo de Datos

La navegación por CouchDB es intuitiva y simple; a su vez permite agregar más nivel de detalle a medida que uno profundiza en las diferentes opciones que brinda. Para crear una base de datos se debe ubicar en la opción con dicho nombre y definir un nuevo espacio de trabajo. Tras aceptar, la base de datos se encuentra lista para poblarla con documentos (ver figuras 26 y 27).

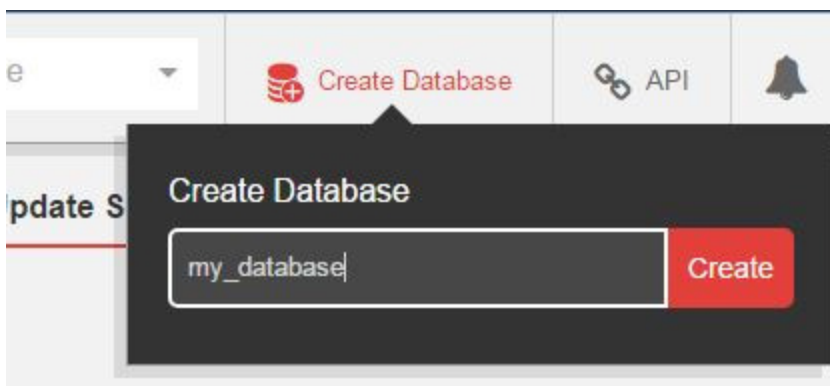


Figura 26. Ejemplo de creación de base de datos en CouchDB.

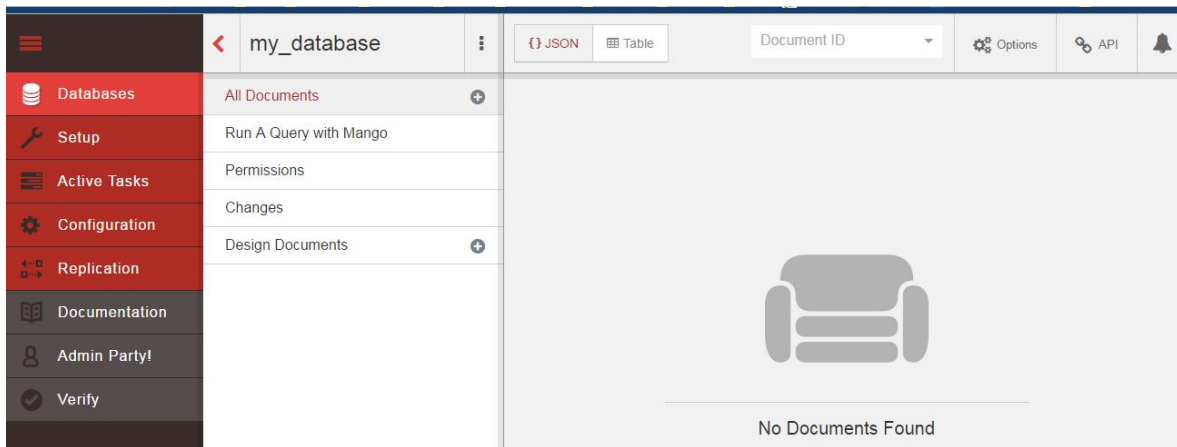


Figura 27. Panel de administración de la base de datos de CouchDB.

All Documents permite poblar la base de datos creando nuevos documentos; a su vez, permite configurar herramientas para la navegación por los documentos con el uso de vistas e índices.

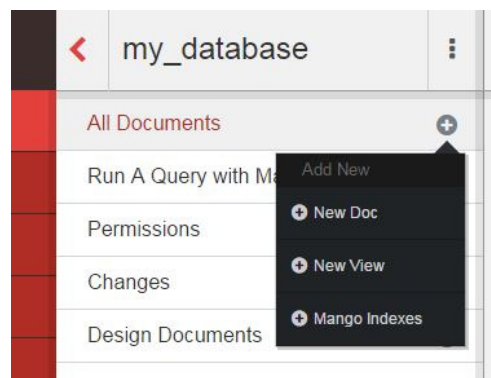


Figura 28. Opciones de creación de documento, vista e índice de CouchDB.

Para crear un documento se selecciona la opción *New Doc*. Allí se puede definir la estructura del documento, ya sea ésta pares clave-valor, clave con arreglo de valores, o incluso un nuevo documento dentro del mismo. Esto se ve reflejado en la figura 29 donde se define la estructura de una ficha personal, siendo el nombre un ejemplo de clave-valor, los *email* un ejemplo de clave con arreglo de valores, y la formación académica un sub-documento.

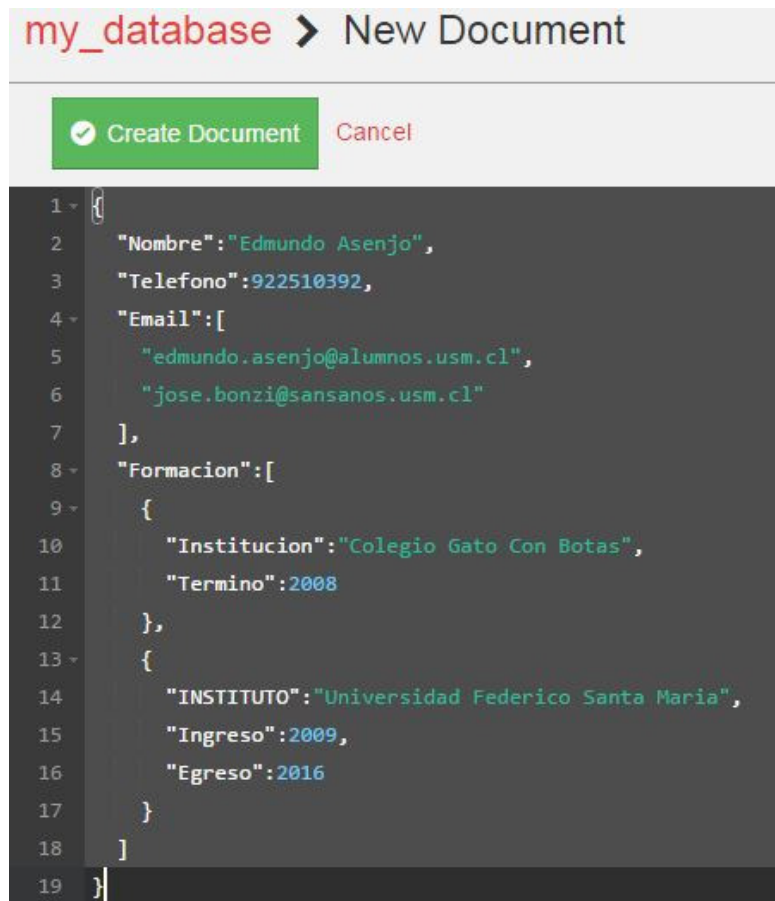


Figura 29. Ejemplo de creación de un documento en CouchDB.

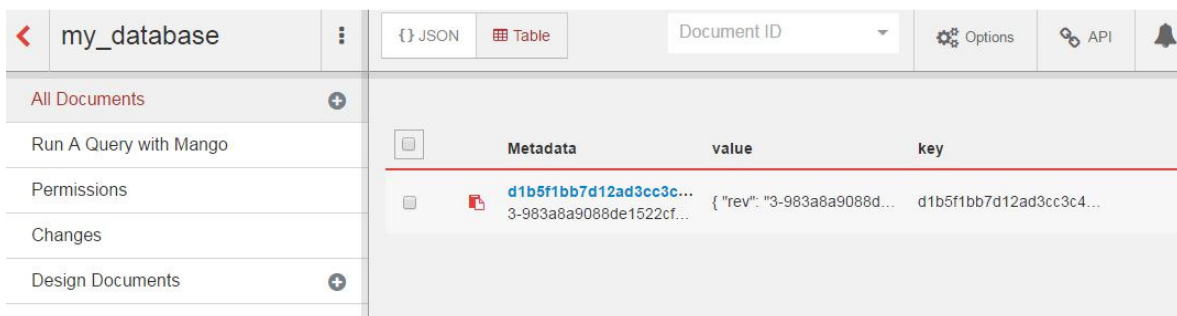


Figura 30. Visualización de la base de datos en CouchDB.

Al seleccionar el documento que fue creado se puede notar la inclusión de dos atributos adicionales: *_id* y *_rev* (ver figura 31). El primero puede ser definido manualmente, pero de no estar presente en el documento, CouchDB le asigna uno de forma automática. El segundo, *_rev*, es un parámetro que define CouchDB para mantener las diferentes versiones del documento.

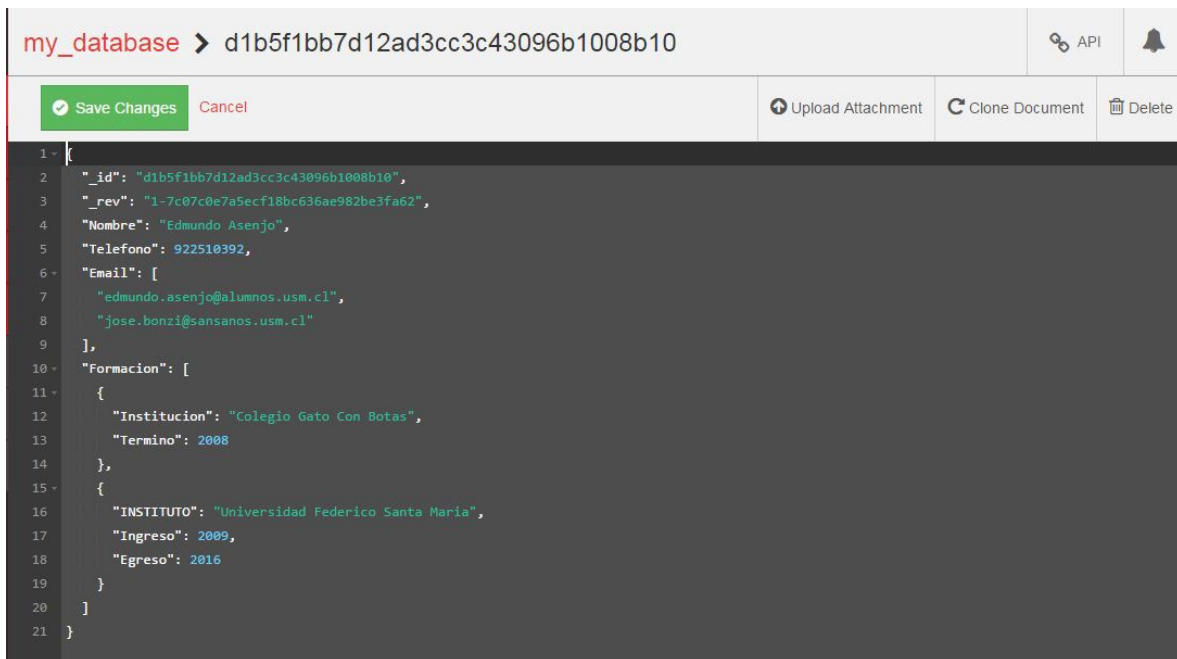


Figura 31. Visualización de las claves `_id` y `_rev`.

En general, las bases de datos NoSQL permiten almacenar en el mismo objeto elementos como imágenes, sonidos y videos. CouchDB permite hacer esto de manera muy simple tras seleccionar dentro del documento la opción *Upload Attachment*. Se desplegará una ventana de búsqueda de archivo, para ingresar una imagen. CouchDB agregará el elemento al documento bajo la clave “`_attachment`” (ver figura 32).

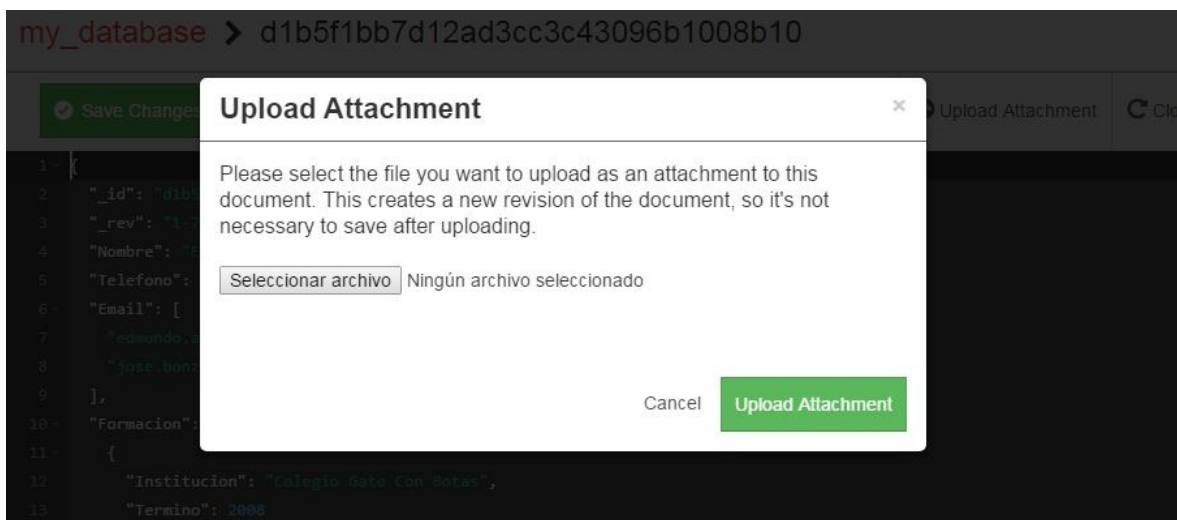


Figura 32. Pantalla guía para adjuntar archivos al documento de couchDB.

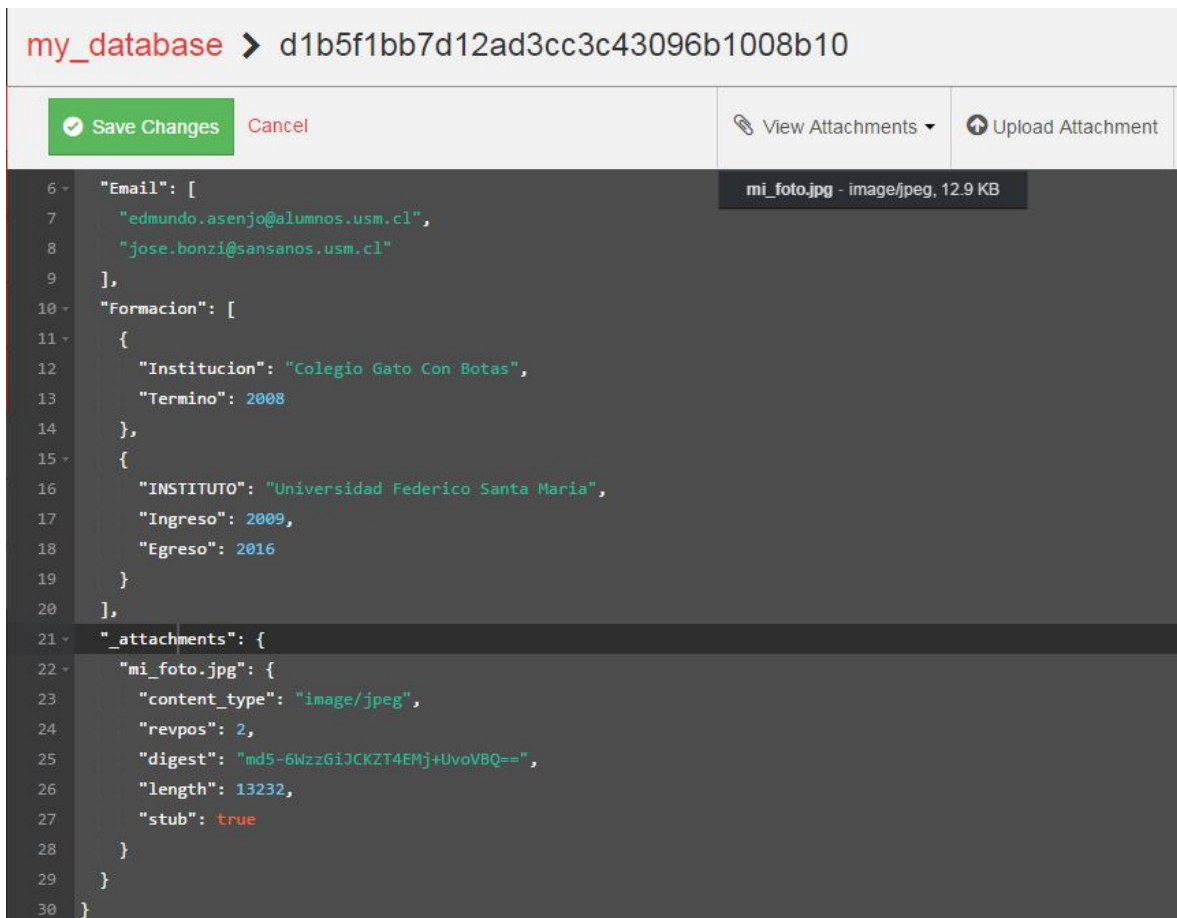


Figura 33. Ejemplo de imagen adjunta a un documento de CouchDB.

La búsqueda de documentos en CouchDB puede hacerse de dos maneras. La primera sería navegar por el listado de documentos que se despliega en el panel administrativo de la base de datos (Figura 34), opción que no es factible cuando el volumen de documentos es muy grande, o bien busca alguna característica en particular, ya que se debería entrar uno a uno en los documentos para visualizar si cumplen o no con dicha característica.

La segunda manera es utilizando la herramienta Mango, la cual permite describir, de manera similar a como se hace con un documento, los valores que se desee buscar. Esta herramienta resulta ser la principal para búsquedas puntuales bajo demanda. Esta herramienta posee la capacidad de complejizarse tanto como uno lo desee, soportando operaciones *and/or* entre otras (ver figura 34).

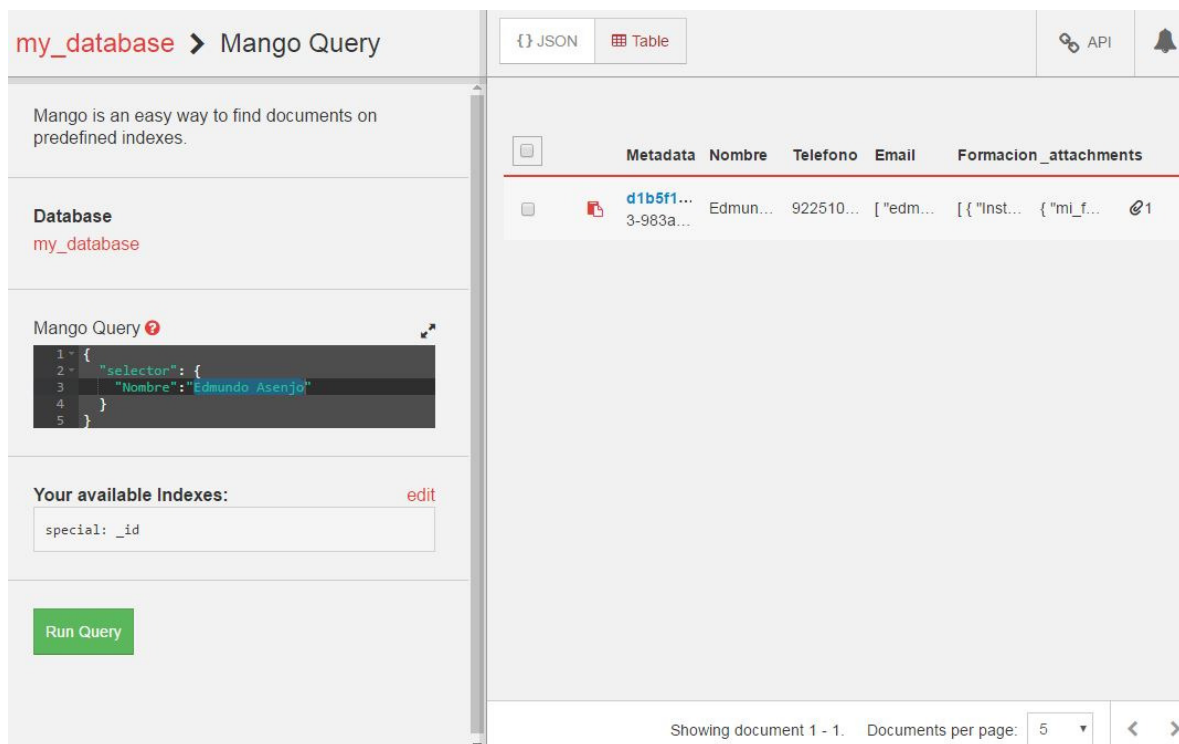


Figura 34. Ejemplo de búsqueda por herramienta Mango de CouchDB.

4.2.2 MongoDB v3.4.4

MongoDB es un software de licencia libre de base de datos orientado a documentos creado en 2007 y lanzado en 2009. Está escrito en C++ lo que le confiere cierta cercanía a los recursos de hardware de la máquina, permitiendo una rápida ejecución de sus tareas. Como toda base de datos documental, permite registros con esquemas de datos diferentes, variando sus atributos sin la necesidad de que éstos tengan que repetirse en otros registros. MongoDB se encuentra entre los sistemas documentales más populares gracias a su destacada velocidad y sencillo sistema de consultas (DB-Engines, 2017).

Instalación

En el sitio web de MongoDB se puede encontrar instaladores para Windows, Linux, OSX y Solaris. El instalador permite poner en marcha lo necesario para utilizar y disponer de este motor de base de datos documental (ver figuras 35, 36 y 37).

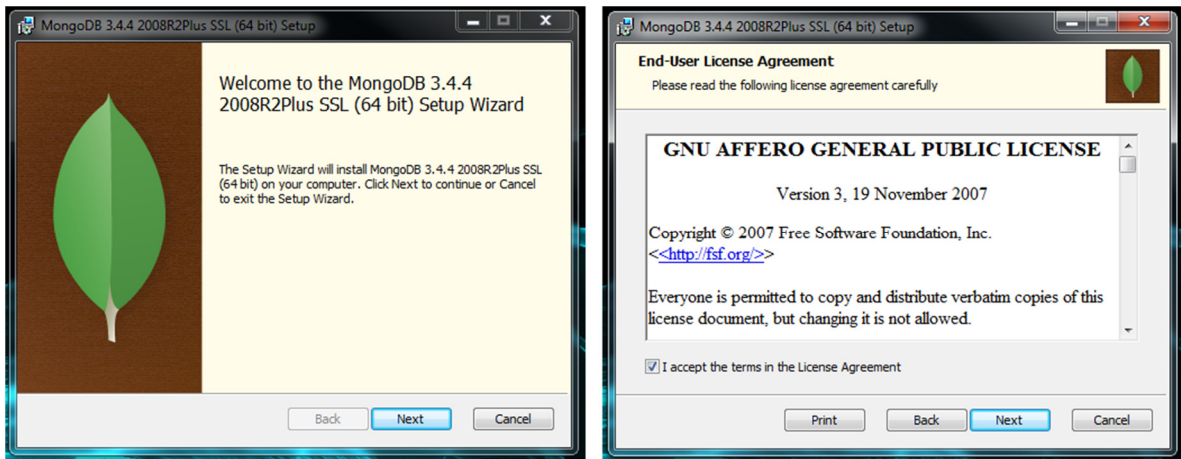


Figura 35. Instalador de MongoDB para Windows (1).

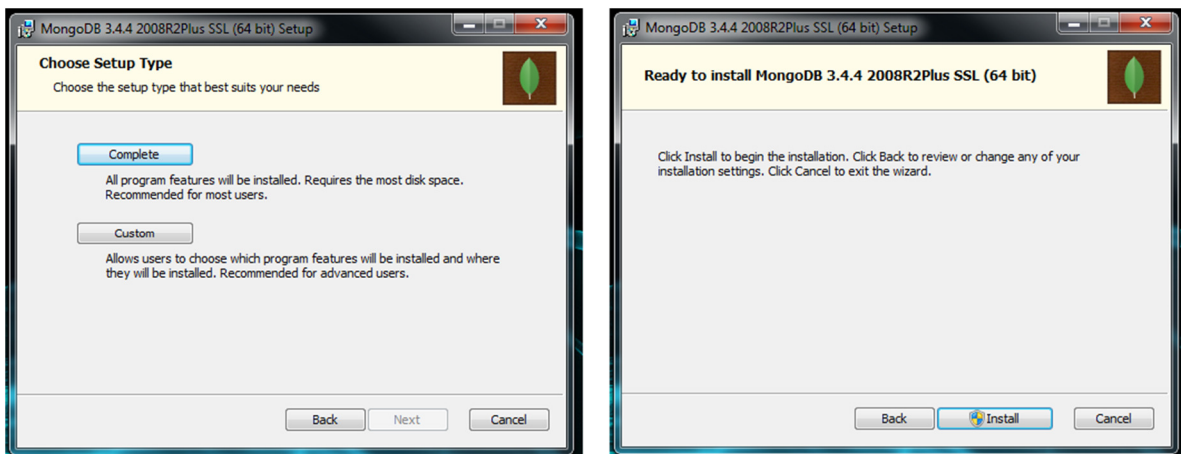


Figura 36. Instalador de MongoDB para Windows (2).

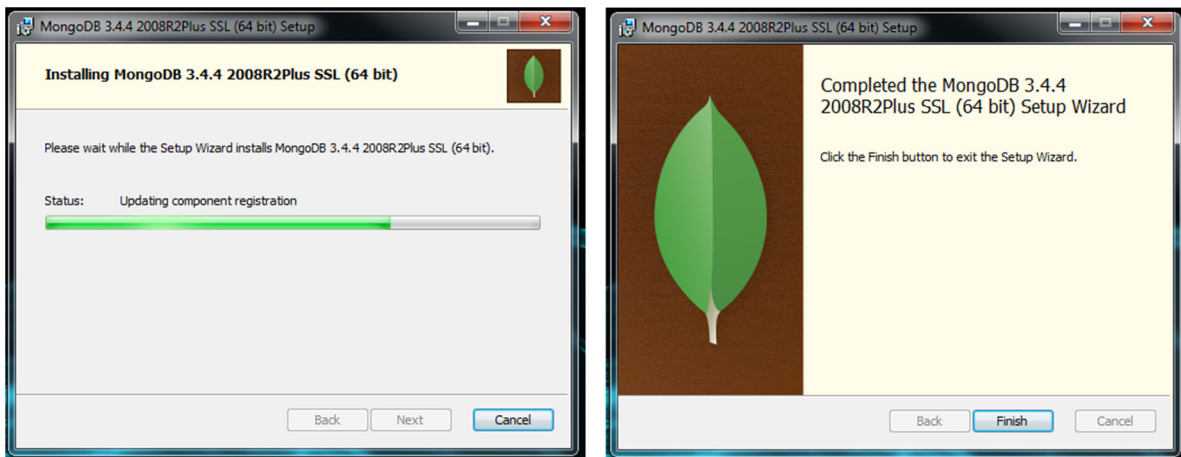
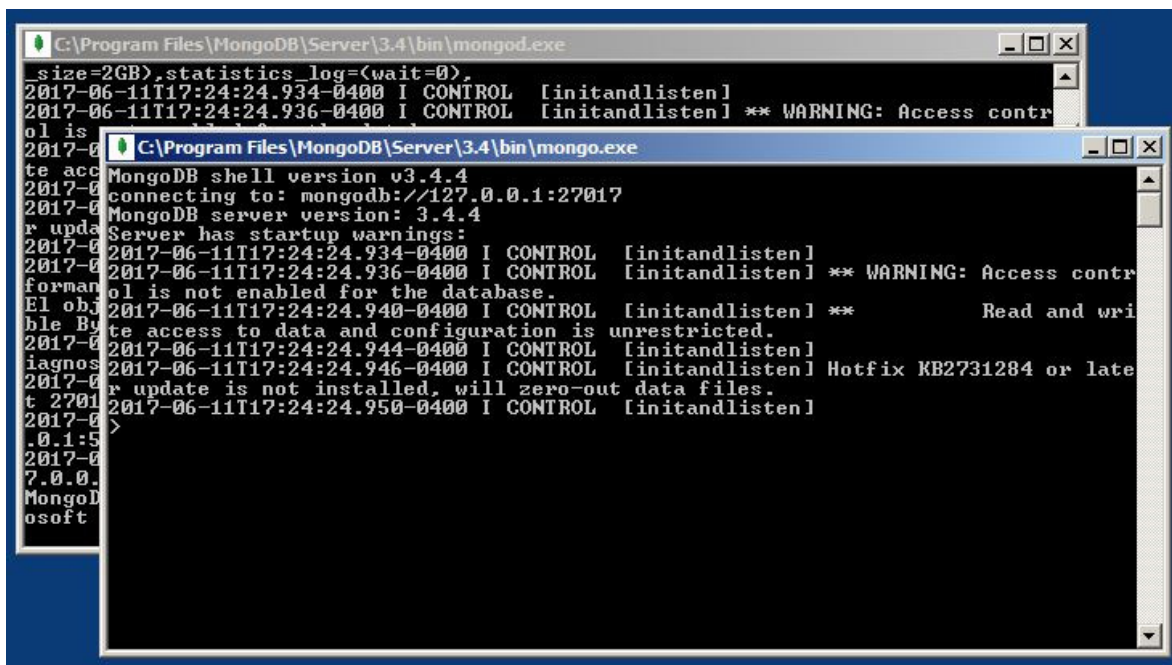


Figura 37. Instalador de MongoDB para Windows (3).

Al finalizar la instalación se deben ejecutar los programas *mongod.exe* y *mongo.exe*, ambos ubicados en la ruta por defecto *C:\Program_Files\MongoDB\Server\3.4\bin*. El primer programa, *mongod.exe*, inicia el servicio de la base de datos mientras que el segundo, *mongo.exe*, corresponde a la terminal que permite trabajar en la base de datos (ver figura 38).



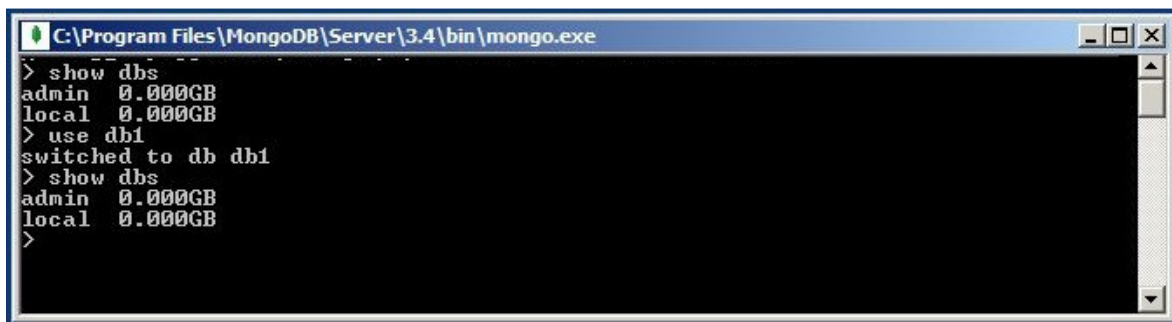
```
C:\Program Files\MongoDB\Server\3.4\bin>mongod.exe
size=2GB),statistics_log=(wait=0).
2017-06-11T17:24:24.934-0400 I CONTROL [initandlisten]
2017-06-11T17:24:24.936-0400 I CONTROL [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2017-06-11T17:24:24.940-0400 I CONTROL [initandlisten] **
Read and write access to data and configuration is unrestricted.
2017-06-11T17:24:24.944-0400 I CONTROL [initandlisten]
2017-06-11T17:24:24.946-0400 I CONTROL [initandlisten] Hotfix KB2731284 or late
r update is not installed, will zero-out data files.
2017-06-11T17:24:24.950-0400 I CONTROL [initandlisten]

C:\Program Files\MongoDB\Server\3.4\bin>mongo.exe
MongoDB shell version v3.4.4
connecting to: mongod://127.0.0.1:27017
MongoDB server version: 3.4.4
Server has startup warnings:
2017-06-11T17:24:24.934-0400 I CONTROL [initandlisten]
2017-06-11T17:24:24.936-0400 I CONTROL [initandlisten] ** WARNING: Access contr
ol is not enabled for the database.
2017-06-11T17:24:24.940-0400 I CONTROL [initandlisten] **
Read and write access to data and configuration is unrestricted.
2017-06-11T17:24:24.944-0400 I CONTROL [initandlisten]
2017-06-11T17:24:24.946-0400 I CONTROL [initandlisten] Hotfix KB2731284 or late
r update is not installed, will zero-out data files.
2017-06-11T17:24:24.950-0400 I CONTROL [initandlisten]
```

Figura 38. Programas *mongod.exe* y *mongo.exe* ejecutados.

Creación y Manejo de Datos

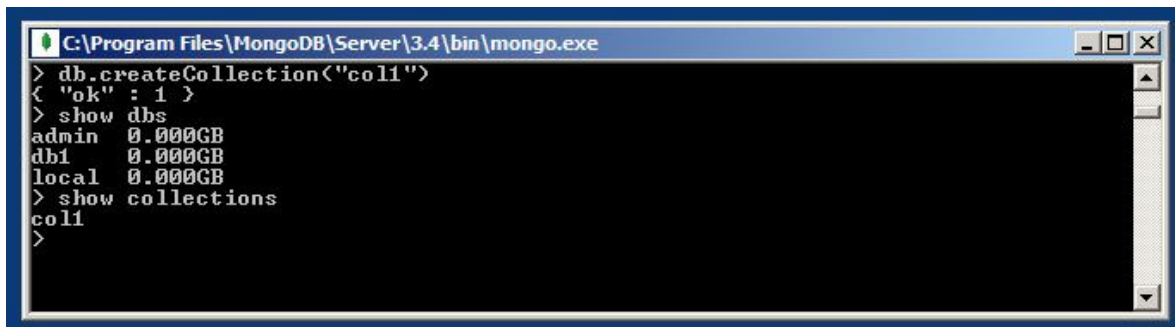
Se utiliza el comando *show dbs* para visualizar las bases de datos existentes. Para crear y utilizar una base de datos se utiliza el comando *use* (ver figura 39).



```
C:\Program Files\MongoDB\Server\3.4\bin>mongo.exe
> show dbs
admin 0.000GB
local 0.000GB
> use db1
switched to db db1
> show dbs
admin 0.000GB
local 0.000GB
>
```

Figura 39. Ejemplo de comando *show dbs* y comando *use*.

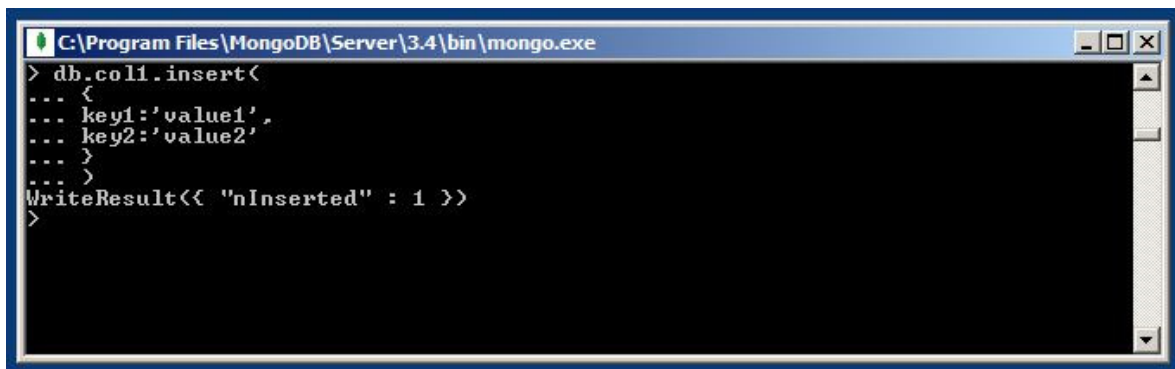
En la figura 39 se puede apreciar que la base de datos recién creada (*db1*) no se visualiza. Esto ocurre porque aún no se ha creado una colección en ella. Las colecciones son a MongoDB como las tablas son a SQL. Para crear una colección se utiliza el comando *db.createCollection()* y para visualizar todas las colecciones, el comando *show collections*. Tras crear la colección *coll* en la base de datos *db1*, la base de datos pasa a estar creada y visible con el comando *show dbs*, lo cual se ejemplifica en la figura 40.



```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
> db.createCollection("coll")
{ "ok" : 1 }
> show dbs
admin  0.000GB
db1    0.000GB
local  0.000GB
> show collections
coll
>
```

Figura 40. Ejemplo de comando *db.createCollection()* y comando *show collections*.

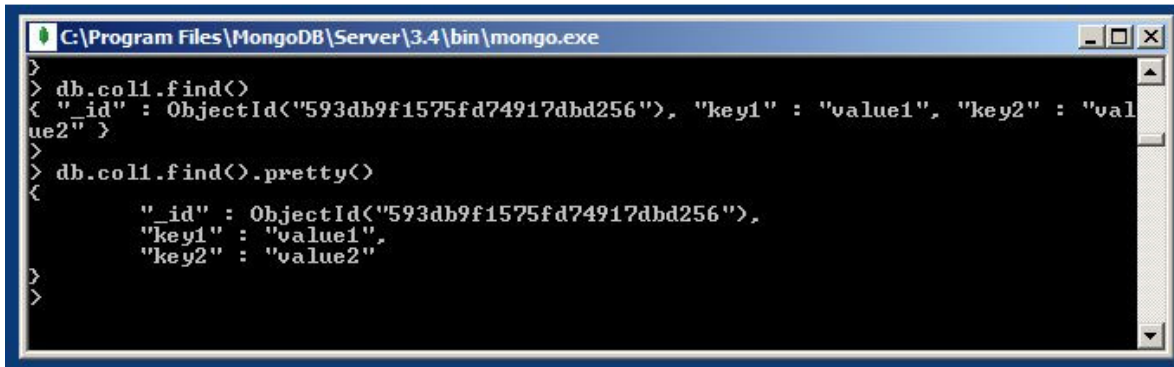
Para la creación de documentos se utiliza el comando *db.collection_name.insert()*. Este comando ingresará los valores indicados a la colección indicada y desplegará un mensaje de éxito o fallo de la transacción (ver imagen 41).



```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
> db.coll.insert(
... {
...   key1:'value1',
...   key2:'value2'
... }
... )
WriteResult<< "nInserted" : 1 >>
>
```

Figura 41. Ejemplo de comando *db.collection_name.insert()*.

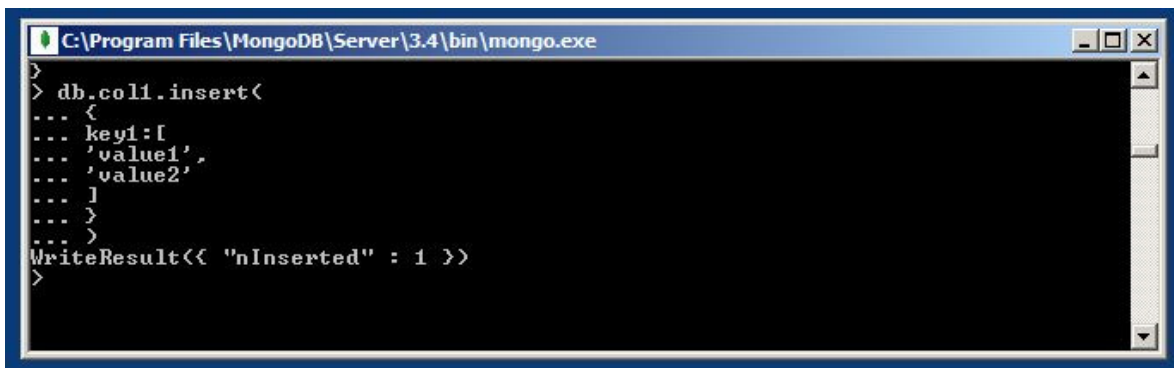
Para consultar los documentos existentes se utiliza el comando *db.collection_name.find()*, el cual buscará y retornará todos los documentos que cumplan las condiciones indicadas en los parámetros. Sin parámetros, retorna todos los documentos de la colección. El comando permite la opción *pretty()*, una opción para visualizar en forma estructurada el documento. En la figura 42 se ejemplifica el comando *db.collection_name.find()* sin y con opción *pretty()*, respectivamente.



```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
>
> db.col1.find()
{ "_id" : ObjectId<"593db9f1575fd74917dbd256">, "key1" : "value1", "key2" : "value2" }
>
> db.col1.find().pretty()
{
  "_id" : ObjectId<"593db9f1575fd74917dbd256">,
  "key1" : "value1",
  "key2" : "value2"
}
```

Figura 42. Ejemplo del comando `db.collection_name.find()` con y sin opción `pretty()`.

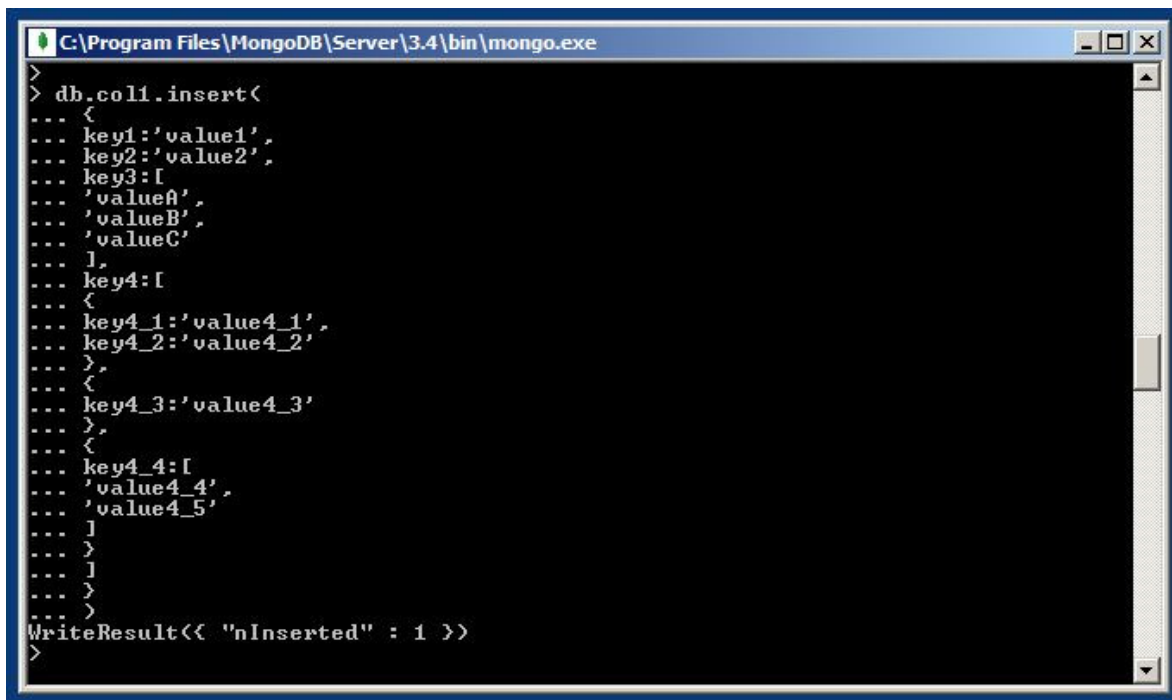
Como base de datos documental que es, MongoDB permite múltiples niveles de estructuración en los documentos. En la figura 43 se ejemplifica la inserción de un documento con un arreglo de valores.



```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
>
> db.col1.insert(
... {
...   key1:[
...     'value1',
...     'value2'
...   ]
... }
... )
WriteResult<< "nInserted" : 1 >>
>
```

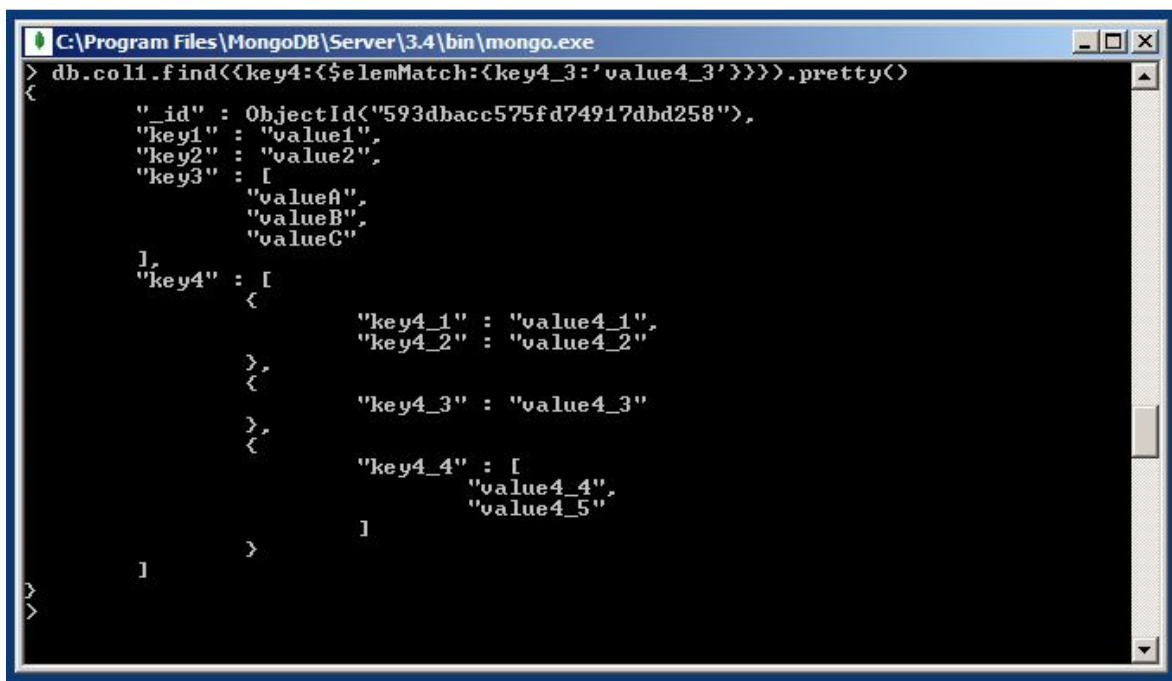
Figura 43. Ejemplo de documento con arreglo de valores.

El potencial de las bases de datos documentales radica en la flexibilidad que se tiene para definir la estructura del documento, lo que permite almacenar todo un conjunto de datos en una única unidad de forma esquematisada. En las figura 44, 45 y 46 se ejemplifica lo anterior con la creación de un documento con pares clave-valor, arreglos y sub-documentos.



```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
>
> db.col1.insert(
...   {
...     key1:'value1',
...     key2:'value2',
...     key3:[
...       'valueA',
...       'valueB',
...       'valueC'
...     ],
...     key4:[
...       {
...         key4_1:'value4_1',
...         key4_2:'value4_2'
...       },
...       {
...         key4_3:'value4_3'
...       },
...       {
...         key4_4:[
...           'value4_4',
...           'value4_5'
...         ]
...       }
...     ]
...   }
... )
WriteResult<< "nInserted" : 1 >>
>
```

Figura 44. Ejemplo de creación de documento de estructura compleja.

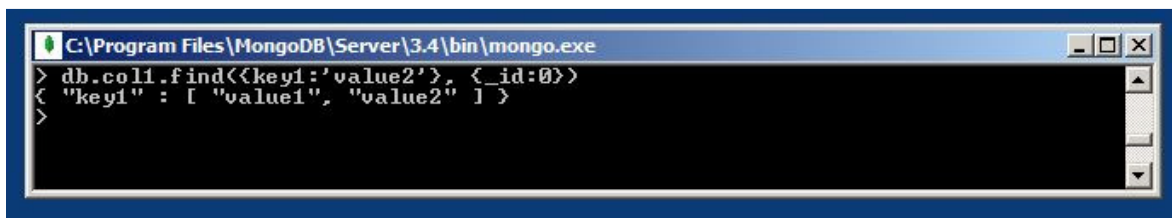


```
C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
> db.col1.find(<<key4:<{$elemMatch:<key4_3:'value4_3'}}>>).pretty()
{
  "_id" : ObjectId("593dbacc575fd74917dbd258"),
  "key1" : "value1",
  "key2" : "value2",
  "key3" : [
    "valueA",
    "valueB",
    "valueC"
  ],
  "key4" : [
    {
      "key4_1" : "value4_1",
      "key4_2" : "value4_2"
    },
    {
      "key4_3" : "value4_3"
    },
    {
      "key4_4" : [
        "value4_4",
        "value4_5"
      ]
    }
  ]
}
1
>
>
```

Figura 45. Ejemplo de visualización de documento de estructura compleja.

Para la búsqueda en documentos con mayor cantidad de niveles en su estructura se utiliza el comando `db.collection_name.find()` con la opción `$elemMatch:{}`, que permite navegar en los diferentes niveles del documento (ver figura 45).

Debido a la complejidad de los documentos que se pueden almacenar, habrá veces en que éstos tengan mucha información para mostrar. Para limitar los campos que se desean mostrar se puede ampliar los parámetros del comando `db.collection_name.find()`. En la figura 46 se ejemplifica lo anterior, realizando una búsqueda, omitiendo el campo `_id`.



```

C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe
> db.coll.find(<<key1:'value2'>>, {_id:0})
{ "key1" : [ "value1", "value2" ] }
>

```

Figura 46. Ejemplo de omisión de campos en comando `db.collection_name.find()`.

4.2.3 Comparación Preliminar

En la tabla 4 se presentan los sistemas de bases de datos tratados, sometidos a los criterios de comparación, destacando cuál se desempeña mejor en cada punto.

Criterio	CouchDB	MongoDB	Argumento
Instalación	---	---	Ambas bases de datos tienen instaladores dedicados para múltiples sistemas operativos, lo que hace fácil e intuitiva su instalación.
Manejo de Datos CRUD	✓	✗	La exploración nativa de CouchDB desde el navegador web facilita la manipulación de la base de datos y la definición de índices y búsquedas. Aunque MongoDB maneja de mejor manera grandes volúmenes de datos cambiantes, CouchDB es muy superior si el <i>set</i> de datos es poco cambiante.
Documentación y Soporte	---	---	Ambos sistemas cuentan con documentación de fácil acceso en sus canales oficiales de comunicación, además de tener diversos foros y comunidades de desarrolladores (Referencias 30 y 43).

Curva de Aprendizaje			Gracias a su interfaz web, CouchDB es sumamente intuitivo y de fácil iniciación en él, radicando su complejidad únicamente en la definición y redacción de vistas.
Claridad del Paradigma	---	---	Ambas bases de datos permiten almacenar, definir, manipular y estructurar los documentos de igual manera. El formato JSON en el que trabajan permite apreciar claramente el paradigma.
Interfaz Gráfica			CouchDB posee su interfaz nativa a través del navegador web. MongoDB, aunque no los tiene de forma nativa, posee complementos que permiten trabajar con la base de datos de forma más gráfica y no sólo por un terminal (Referencia 50).
Relevancia en el Mercado			CouchDB se desempeña mejor en ambientes donde la estructura de los datos espoco cambiante, destacándose en ésto. Sin embargo, debido a la generalidad de los ambientes que requieren de un sistema de almacenamiento documental, es MongoDB quien posee mayor presencia y áreas de aplicación.

Tabla 4. Comparativa entre CouchDB y MongoDB.

Al trabajar con las bases de datos documentales CouchDB y MongoDB, no se apreciaron mayores diferencias en la estructura de almacenamiento pero sí en lo referente a la manera de presentar las diferentes opciones de configuración del motor de bases de datos y definiciones de criterios de búsqueda. Utilizando los criterios definidos en este trabajo, CouchDB resultó ser el sistema que mejor cumplía las condiciones para aplicarse en el ámbito académico. La instalación de ambos sistemas no tiene complejidad, pero CouchDB presenta en su panel de configuración de la base de datos, todo tipo de opciones y configuraciones fácilmente reconocibles y ajustables. Del mismo modo, la presentación del contenido de las bases de datos es más claro y apreciable en CouchDB, todo gracias a su interfaz a través del navegador web de manera nativa. Es cierto que MongoDB posee más de un *add-on* que mejore la forma de manejar la base de datos y los documentos, agregándole un panel de control, pero ya que no es un elemento nativo del motor, no fue evaluado (pero sí los hay oficiales y con su respectivo soporte técnico). En lo que es el aprendizaje de los sistemas, CouchDB resulta intuitivo, con paneles de ayuda al situar el

cursos en los diferentes elementos, no así la terminal de MongoDB, que sólo entrega un listado de operaciones a consultar al utilizar el comando de ayuda. Las dos herramientas logran proyectar en el usuario la naturaleza del paradigma enfocado a documento y poseen basta documentación y soporte técnico de apoyo al paradigma y su entendimiento.

4.2.4 Propuesta Actividad Académica

En el siguiente apartado se expone la propuesta académica a realizar por los alumnos, definiendo los elementos esenciales para el correcto aprendizaje del paradigma a estudiar.

Una experiencia académica para acercarse a este tipo de sistema sería el levantamiento de una base de datos CouchDB, poblarla con datos que permitan el modelamiento en documentos y múltiples operaciones de lectura y escritura. La redundancia (desnormalización), inconsistencia, variedad de estructuras entre documentos y definición de sub-documentos serían claves para cumplir los objetivos de aprendizaje del sistema. A continuación se presenta el encabezado y propósitos de la actividad docente:

“ Los alumnos deberán instalar la base de datos CouchDB y poblarla con diferentes curriculums para posteriormente realizar consultas sobre todos ellos en la búsqueda de generar información del conjunto, como puede ser los intereses más comunes, la cantidad de herramientas manejadas por cada individuo, trabajos realizados, entre otros.

Se espera que durante esta experiencia el alumno:

Asimile los conceptos de Colección y Documento.

- *Defina una base de datos y una colección donde almacenar los documentos.*
 - *USE db_name*
 - *db.createCollection(coll_name)*
- *Importe los Curriculims Vitaes que se recopilarán para este propósito.*
 - *mongoimport --db db_name --collection coll_name --file path*
- *Realizar operaciones de creación, eliminación y actualización de documentos.*
 - *db.coll_name.insert(document)*
 - *db.coll_name.update(criteria)*
 - *db.coll_name.remove(criteria)*
- *Realizar operaciones de búsqueda de documentos.*
 - *db.coll_name.find(criteria)*

Cada alumno deberá confeccionar dos curriculums de diferentes personas y subirlos al espacio que se designará para tal propósito en la web del ramo. Estos curriculums (dos archivos en formato json por alumno) formarán el total de documentos a importar en la base de datos.

Cada curriculum deberá contar con los siguientes campos, sin estar limitado a agregar otros:

- *Resumen: texto que englobe todo lo contenido en el currículum. Un párrafo de presentación.*
- *Datos Personales: Nombre, Dirección, Teléfonos, Email, Redes Sociales.*
- *Educación: Enseñansa Básica, Enseñansa Media, Educación Superior, Otros.*
- *Laboral: Prácticas, Trabajos (freelancer, empleado, emprendedor).*
- *Conocimientos Técnicos: OS, Lenguajes de Desarrollo, Bases de Datos, Móviles (app), Otros Softwares.*
- *Intereses Personales: Deportes, Música, Animales, Entretenimiento.*

Los alumnos tienen la libertad de agregar los campos que estimen pertinentes y estructurarlos de la manera que deseen pero cumpliendo con los siguientes requerimientos mínimo:

- *Un resumen que corresponda a un campo de texto.*
- *Una clave con múltiples valores (key1:[val1, val2....])*
- *Una clave de documentos (key:[(key1:val1,key2:val2....),(key3:val3....),...])*

El trabajo incluye la redacción de un documento en el que el alumno demuestre el aprendizaje de los puntos anteriormente señalados, el que incluirá las diferentes consultas ejecutadas, acompañadas de capturas de imagen. ”

4.3 Bases de Datos de Grafos

En esta sección se presentan los motores de bases de datos de grafos utilizados en el trabajo: ArangoDB y Neo4j.

4.3.1 ArangoDB v2.8.11

ArangoDB es una base de datos multi-modelo. Originalmente llamada AvocadoDB, se conoce como la base de datos universal debido a su diseño que permite de forma nativa el almacenamiento de datos en formato clave-valor, documental o de grafo, todos éstos manejados bajo un mismo lenguaje de consulta (Referencia 35). ArangoDB unifica en un

único sistema la flexibilidad estructural del almacenamiento documental con la representación gráfica de un almacenamiento de grafos y, aunque no es la mejor herramienta en ninguna de estas categorías, el hecho de que permita tener ambos ambientes en un único software hace relevante su análisis. Sin embargo, para estar alineados con los objetivos de este trabajo, ArangoDB se evaluó en base a sus competencias para un almacenamiento en grafo.

Instalación

ArangoDB se encuentra disponible para diversos sistemas operativos. Luego de descargar el instalador para Windows desde su sitio web (Ref: ArangoDB Company, 2017), éste permite rápidamente establecer la configuración mínima necesaria para poner en marcha este motor de base de datos multi-modelo (ver figuras 47, 48 y 49).

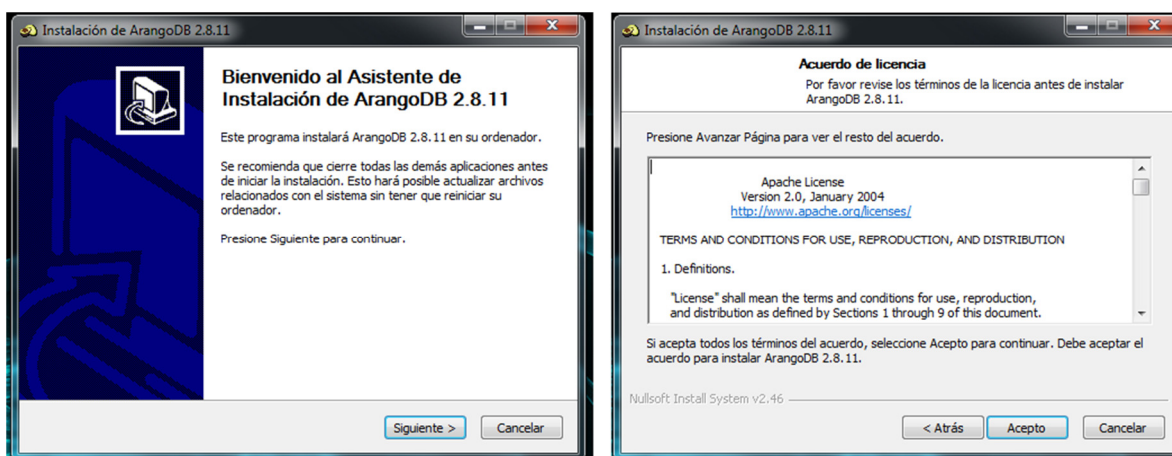


Figura 47. Instalador de ArangoDB para Windows (1).

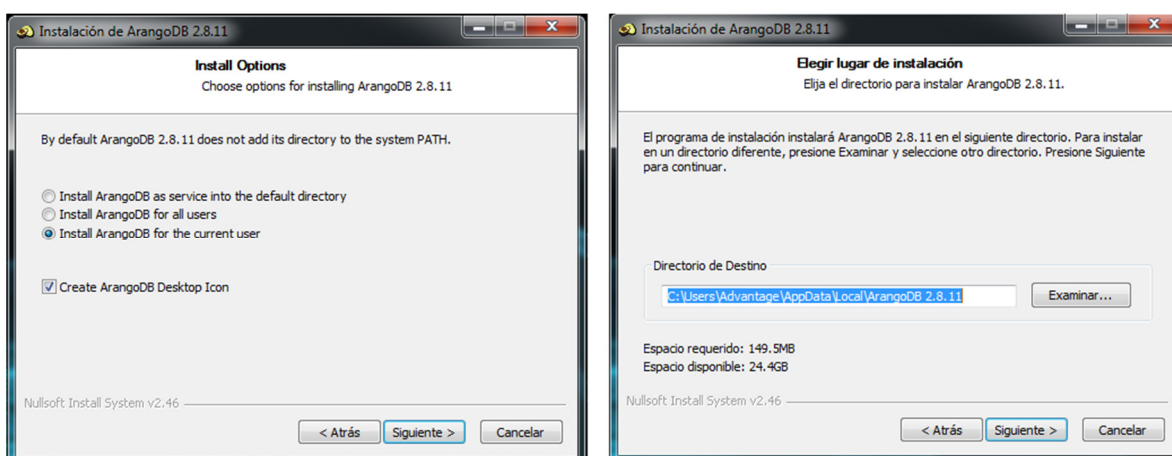


Figura 48. Instalador de ArangoDB para Windows (2).

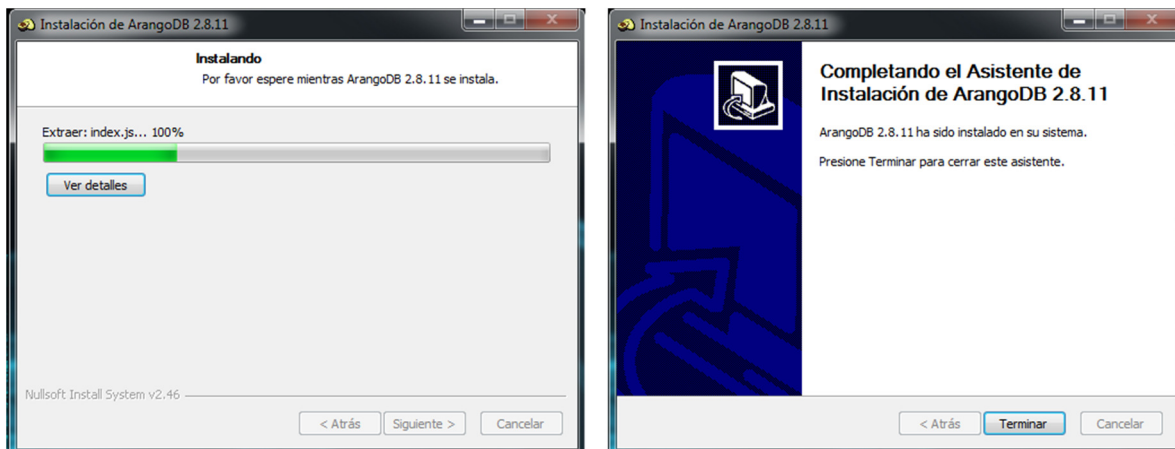


Figura 49. Instalador de ArangoDB para Windows (1).

Terminada la instalación se ejecuta el programa Arango management Interface, el cual abrirá el navegador para acceder a la base de datos (ver figura 50).

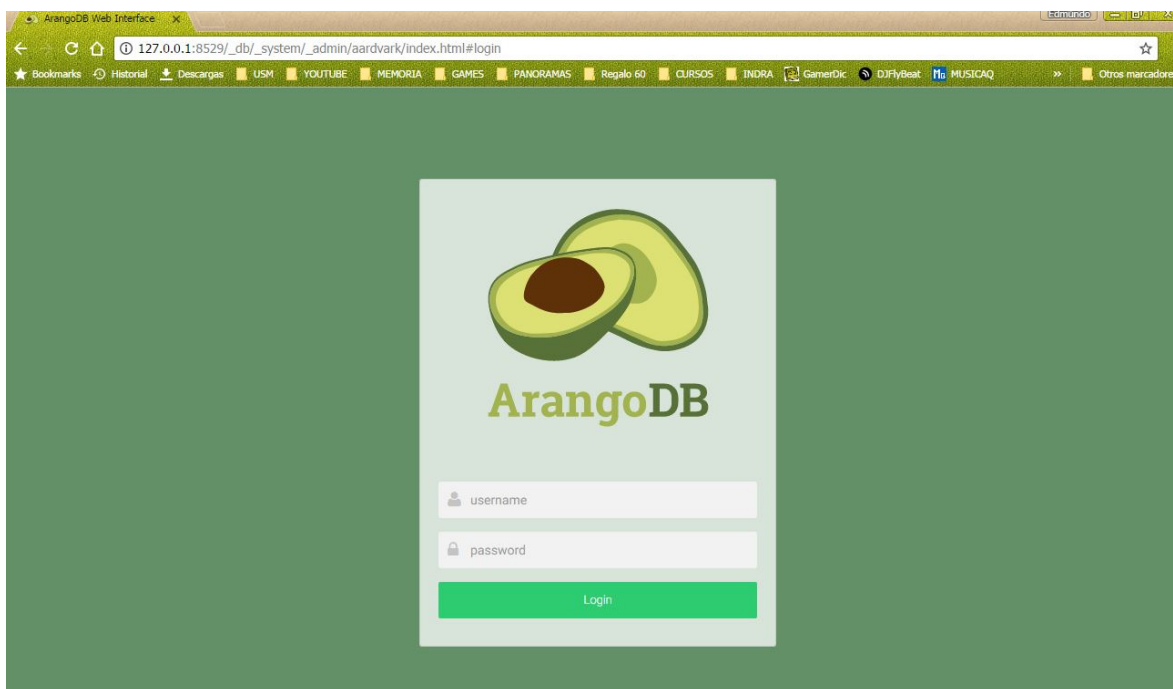


Figura 50. Pantalla de ingreso de credenciales de ArangoDB.

Tras ingresar las credenciales (el usuario por defecto es *root*) el sistema pide indicar a qué base de datos se quiere acceder (ver figura 51).

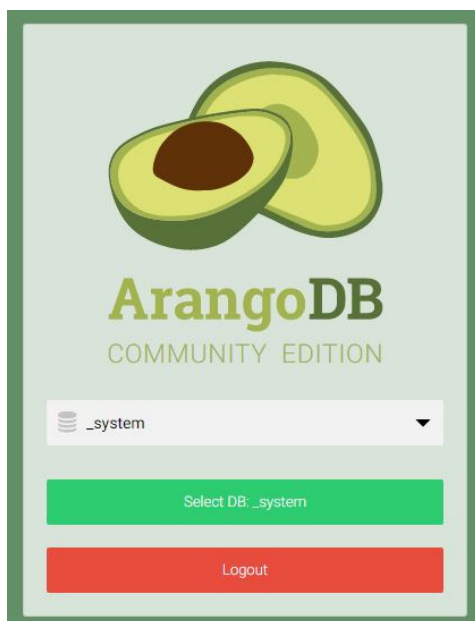


Figura 51. Selección de base de datos de ArangoDB.

La base de datos `_system` se utiliza como centro de mando, entregando diversas herramientas para el monitoreo y control del comportamiento de las bases de datos en el sistema. En la figura 52 se presenta el *dashboard* que entrega la base de datos `_system`.

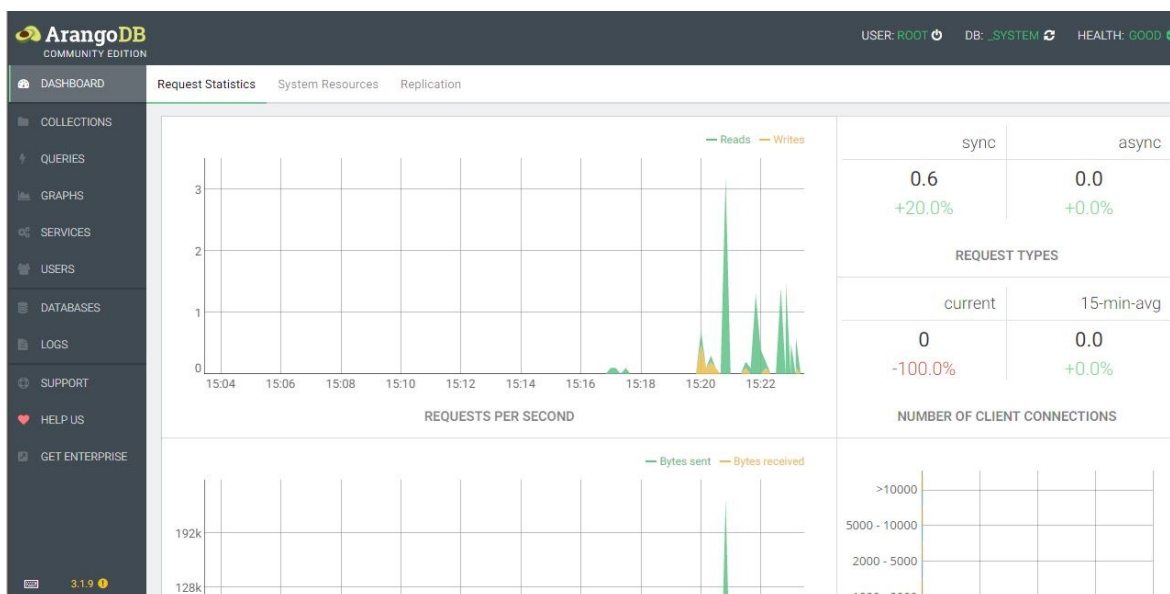


Figura 52. Dashboard de monitoreo de ArangoDB.

Creación y Manejo de Datos

La interfaz por navegador web de ArangoDB es clara y limpia, indicando a cada momento las diferentes opciones y herramientas que tiene el usuario para desempeñar sus tareas.

Para crear una base de datos se selecciona dicha opción del menú lateral de ArangoDB (ver figuras 53 y 54).

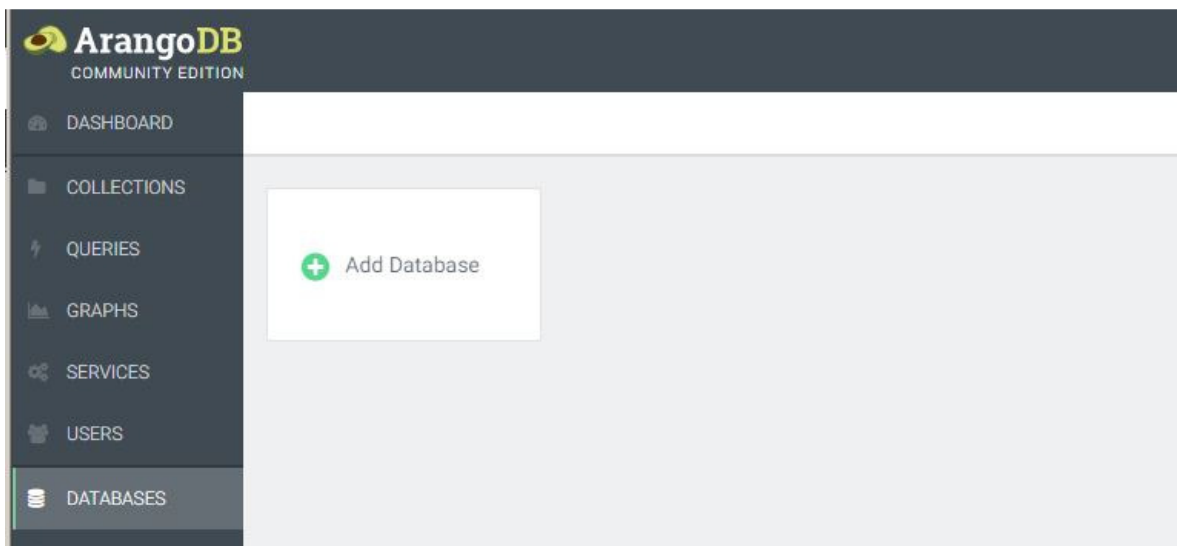


Figura 53. Creación de bases de datos de ArangoDB.

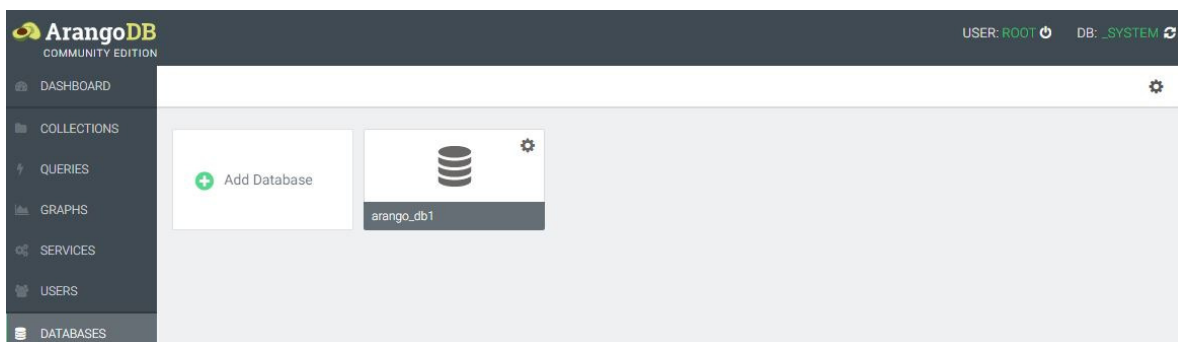


Figura 54. Ejemplo de creación de base de datos de ArangoDB.

Tras crear la base de datos, ésta se podrá seleccionar en la ventana de selección de bases de datos presentada en la figura 51, lo cual se puede apreciar en la figura 55.

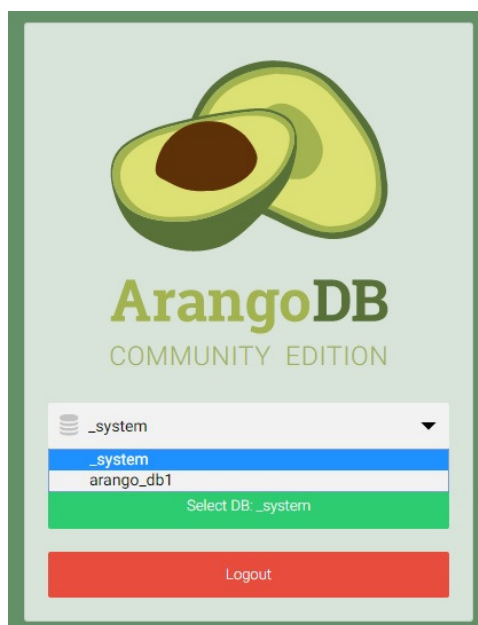


Figura 55. Selección de base de datos de ArangoDB.

El siguiente paso es crear las tablas que contendrán los datos. En ArangoDB se refieren a éstas como colecciones, similar a como lo hace MongoDB. Para crear una nueva colección se selecciona esta opción desde el menú lateral (ver figura 56).

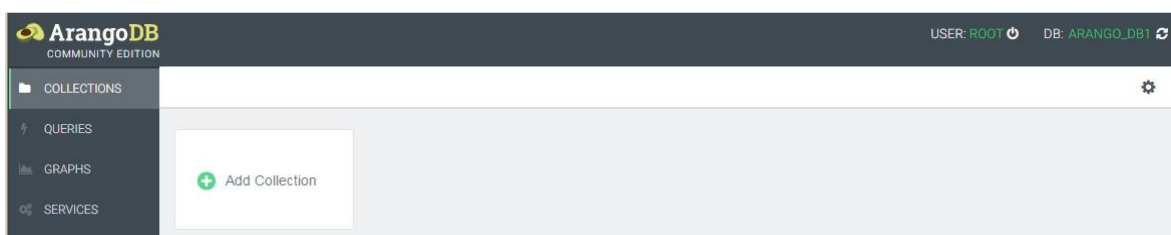


Figura 56. Creación de colecciones de Arango DB.

ArangoDB define dos tipos de colecciones: documentos y aristas. Los documentos serán los nodos cuando se utiliza la herramienta para graficar los datos de ArangoDB, mientras que las aristas serán las reglas que uniran los nodos (ver figura 57). A modo de ejemplo, en la figura 58 se crea la colección de documentos *persona*, y la colección de aristas *conoce* como ejemplo.

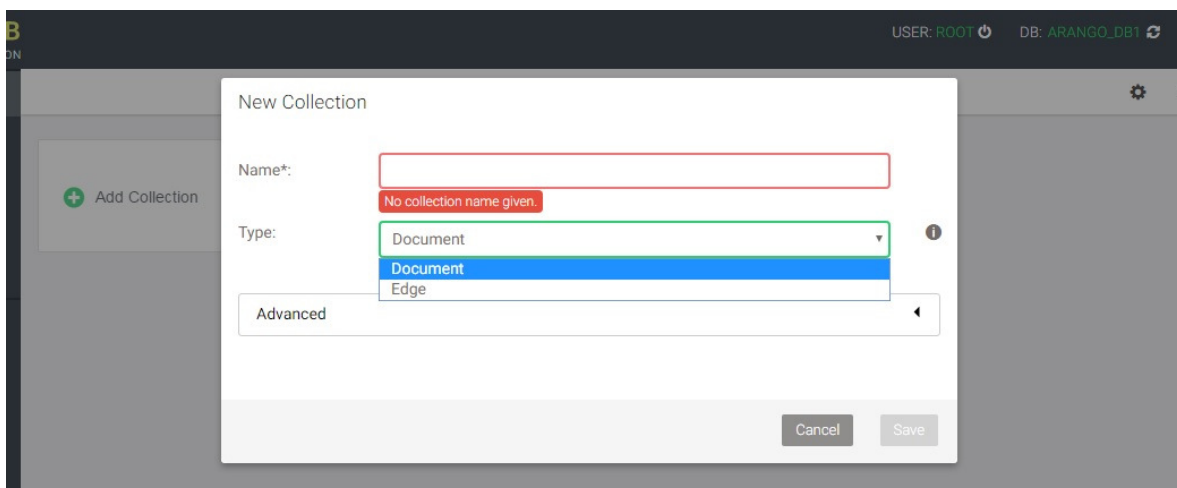


Figura 57. Ejemplo de creación de colección de ArangoDB.

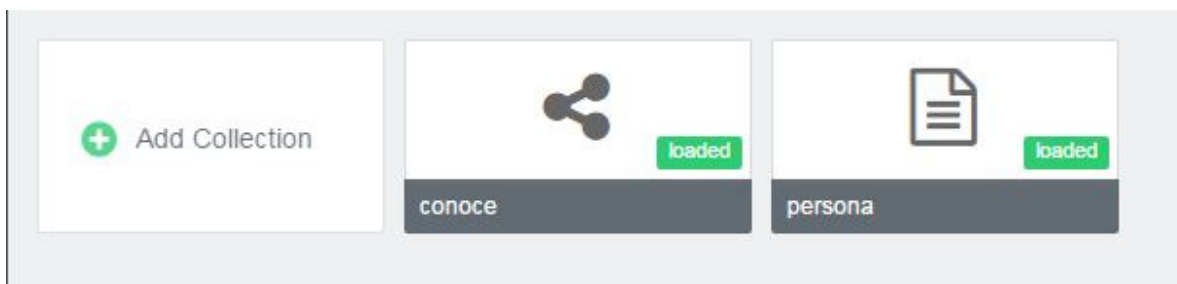


Figura 58. Ejemplo de creación colecciones de aristas y documentos de ArangoDB.

Al seleccionar la colección *persona* se presenta un listado de los documentos en ella, inicialmente vacía (ver figura 59). Para crear un nuevo documento sólo debe seleccionarse la opción para tal motivo, indicada por un ícono '+'. Se puede definir una clave primaria *_key* o bien dejarla vacía para que el sistema genere una de manera automática (ver figura 60).

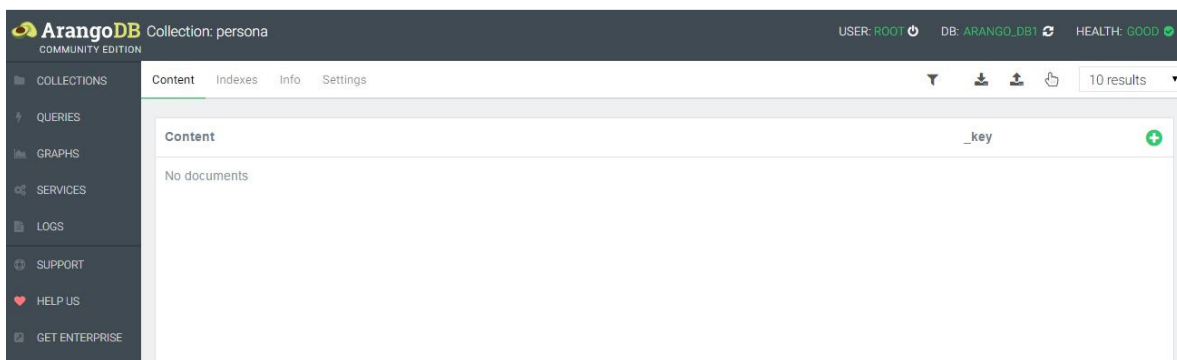


Figura 59. Creación de documentos de ArangoDB.

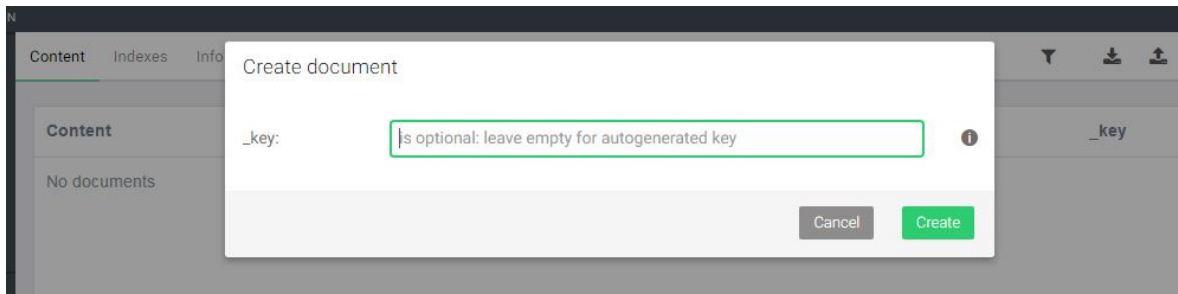


Figura 60. Ejemplo de definición de `_key` para documento de ArangoDB.

Al crear un documento en ArangoDB, además del parámetro `_key` se crean dos parámetros más de forma automática: `_id`, como identificador global en la base de datos, y `_rev`, como controlador de versiones del documento (ver figura 61).

ArangoDB comparte la definición de documentos de las bases de datos del tipo documental, permitiendo pares clave-valor, arreglo de valores y sub-documentos (ver figura 61).



Figura 61. Ejemplo de creación de documento de ArangoDB.

Content	_key
{ "Educacion" : { ("Colegio" : "instituto1" , "Año" : 1990) , ("Colegio" : "instituto2" , "Año" : 2000) } , "Gustos" : { "musica" , "deporte" } , "Nom. ...	1761718
{ "Educacion" : { ("Colegio" : "instituto1" , "Año" : 1990) , ("Colegio" : "instituto2" , "Año" : 2000) } , "Gustos" : { "arte" , "moda" } , "Nombre" ...	1761542
{ "Educacion" : { ("Colegio" : "instituto1" , "Año" : 1990) , ("Colegio" : "instituto2" , "Año" : 2000) } , "Gustos" : "musica" , "Nombre" : "Pam...	1761757
{ "Educacion" : { ("Colegio" : "instituto1" , "Año" : 1990) , ("Colegio" : "instituto2" , "Año" : 2000) } , "Gustos" : "arte" , "Nombre" : "Clauida" }	1761819

Figura 62. Visualización de documentos en una colección de ArangoDB.

Para buscar en la colección de documentos, ArangoDB cuenta con filtros de búsqueda que pueden ser aplicados y mezclados sobre la colección según las necesidades del usuario (ver figuras 63, 64 y 65).

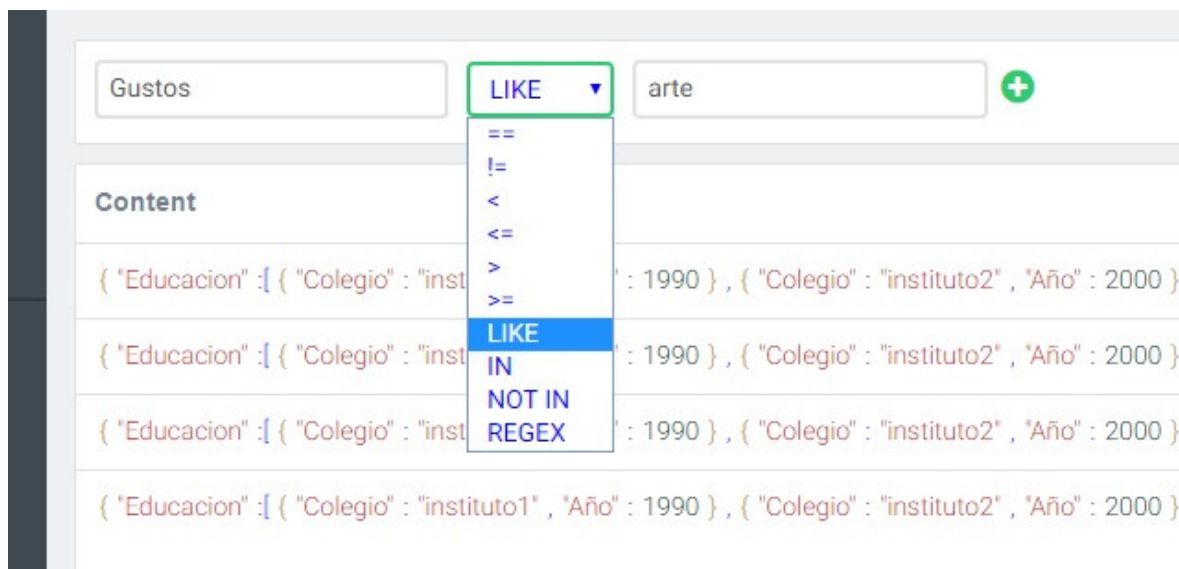


Figura 63. Ejemplo de filtro en colección de ArangoDB (1).

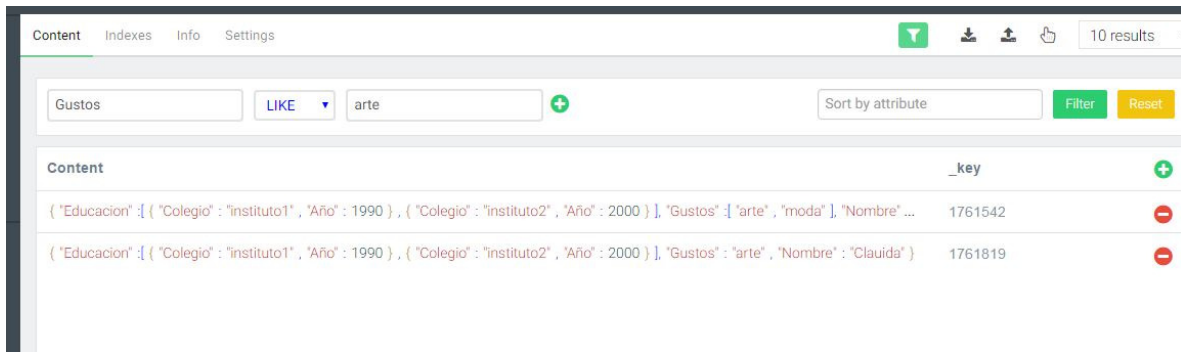


Figura 64. Ejemplo de filtro en colección de ArangoDB (2).

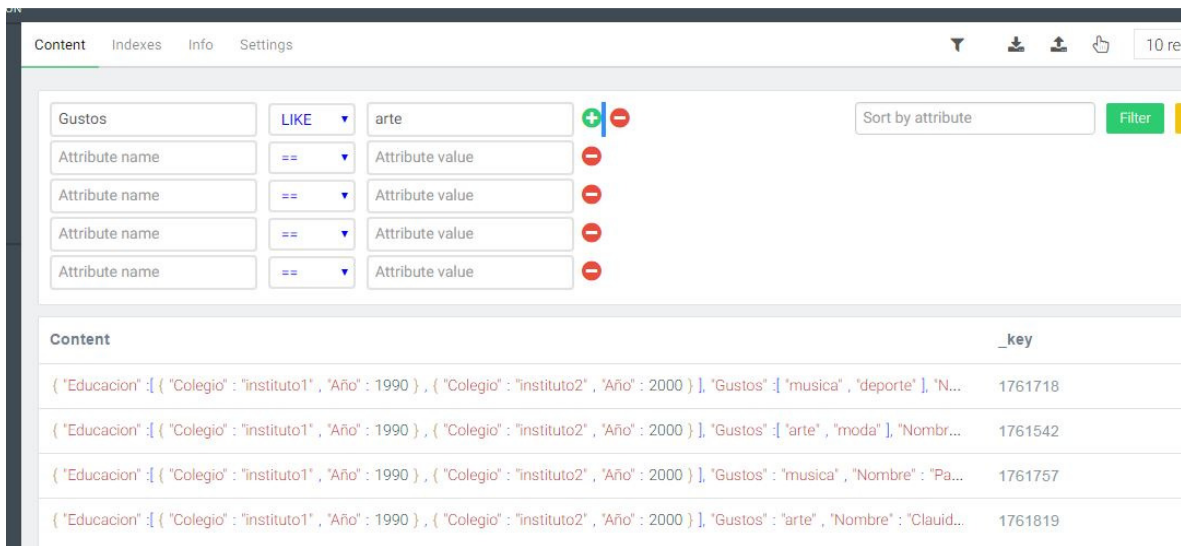


Figura 65. Ejemplo de múltiples filtros en colección de ArangoDB.

Al seleccionar la colección *conoce* (ver figura 58) se presenta un listado de aristas en ella, inicialmente vacía (ver figura 66). Para crear una nueva arista sólo debe seleccionarse la opción para tal motivo, indicada por un ícono '+'. De igual modo que los documentos, se puede definir una clave primaria *_key* o bien dejarla vacía para que el sistema genere una de manera automática (ver figura 67).

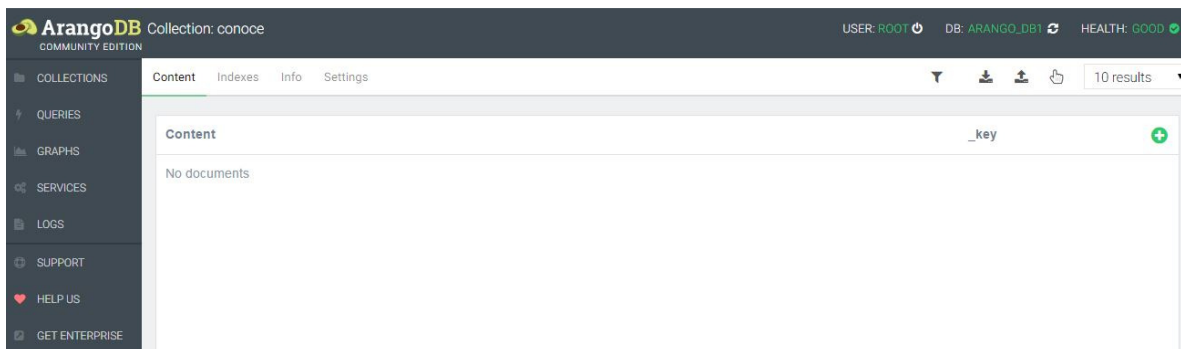


Figura 66. Creación de aristas de ArangoDB.

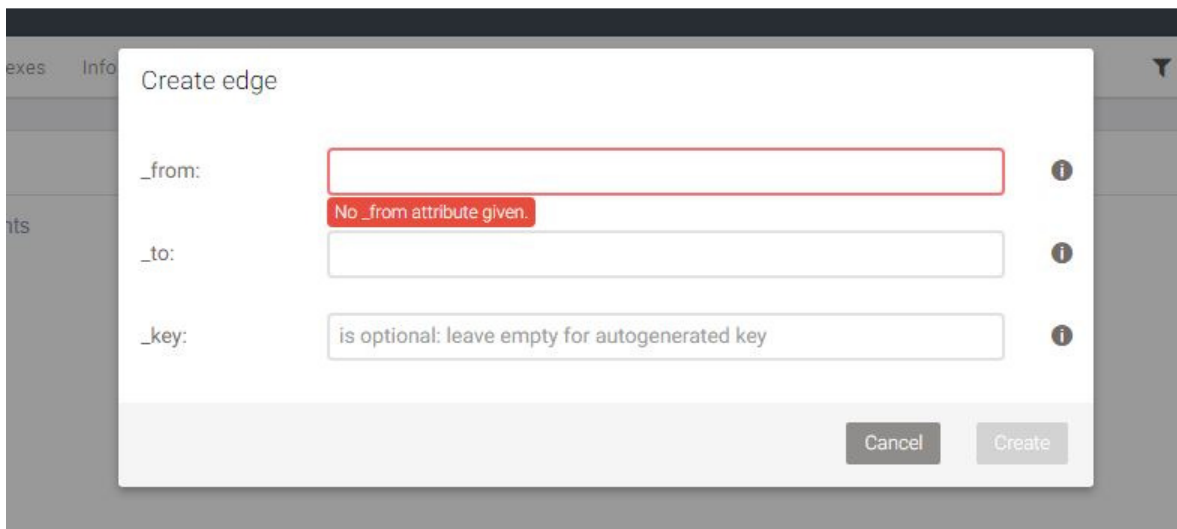


Figura 67. Ejemplo de definición de `_key`, `_from` y `_to` para arista de ArangoDB.

Las aristas tienen los parámetros `_from` y `_to`, los cuales indican el sentido de la relación entre los documentos. Para estos campos se utiliza el parámetro `_id` de los documentos, ya que es éste el que los define de forma inequívoca en toda la base de datos (ver figura 68). Los documentos pueden proceder de distintas colecciones.

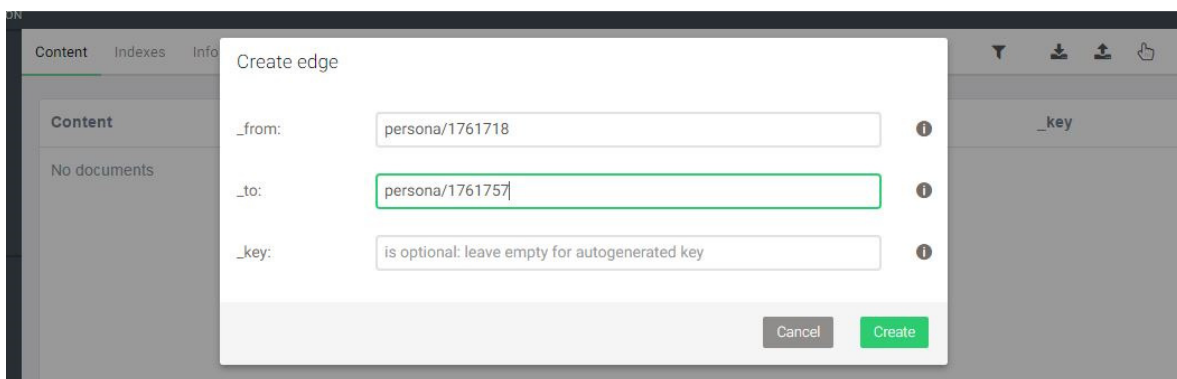


Figura 68. Ejemplo de parámetros `_from` y `_to` de arista de ArangoDB.

ArangoDB trata de igual modo a las aristas como a los documentos. Ambos elementos son documentos, compartiendo las mismas atribuciones automáticas de sistema (parámetros `_id`, `_rev` y `_key`) y la definición de pares clave-valor, arreglo de valores y sub-documentos en la estructura interna de la arista (ver figura 69). La diferenciación entre aristas y documentos radica únicamente en los parámetros `_from` y `_to`.

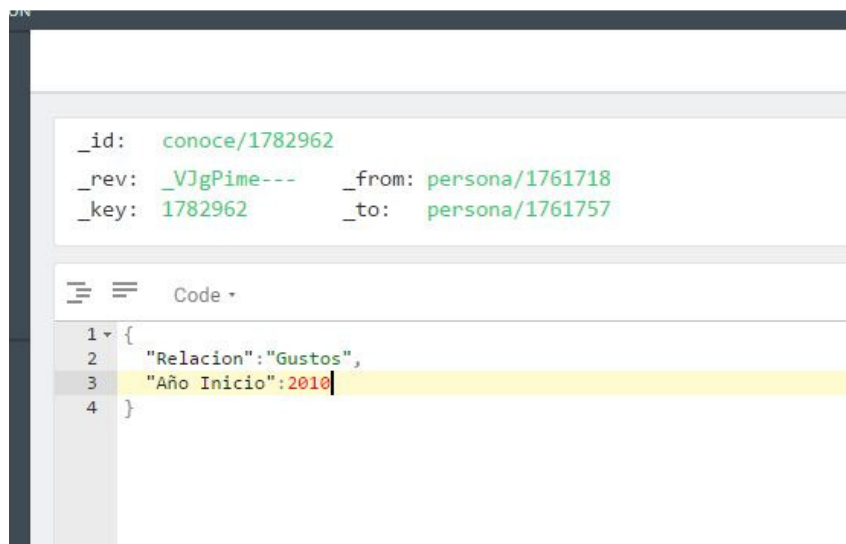


Figura 69. Ejemplo de creación de arista de ArangoDB.

La colección de aristas comparte las mismas funcionalidades de presentación de contenido y búsqueda que la colección de documentos (ver figura 63 y 65).

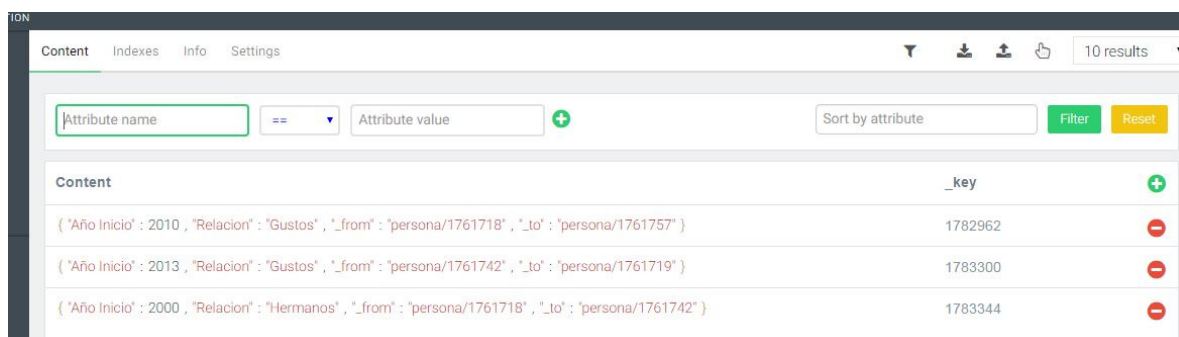


Figura 70. Visualización de aristas en una colección de ArangoDB.

Visualización Gráfica de Datos

Para visualizar los datos en forma de grafo se debe seleccionar dicha opción desde el menú lateral (figura 71). Al crear un nuevo grafo se debe definir, además de su nombre, la colección de aristas que utilizará y las colecciones de inicio y fin de estas mismas aristas (ver figura 72).

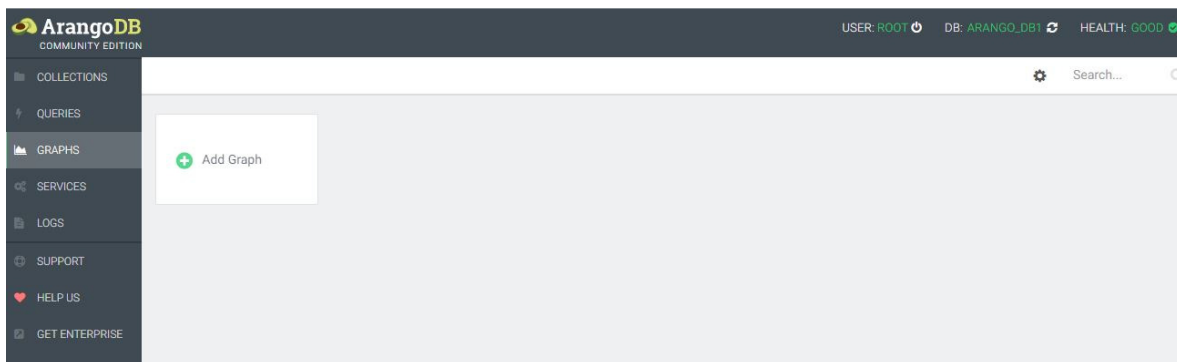


Figura 71. Creación de grafos de ArangoDB.

The image shows a 'Create Graph' dialog box. At the top, there are two tabs: 'Example Graphs' and 'Create Graph'. Below the tabs are five input fields, each with an information icon (i) on the right. The fields are: 'Name*' with the value 'grafo1'; 'Edge definitions*' with a dropdown showing 'x conoce'; 'fromCollections*' with a dropdown showing 'x persona'; 'toCollections*' with a dropdown showing 'x persona'; and 'Vertex collections' with the text 'Vertex Collections'. At the bottom right of the dialog are two buttons: 'Cancel' and 'Create'.

Figura 72. Ejemplo de creación de grafo de ArangoDB.

La interacción y despliegue gráfico de la malla es simple, con diferentes opciones para mover, ocultar y desplegar más nodos con sólo unos *clicks*. Los nodos pueden ser desplazados y ubicados en diferentes posiciones, a su vez que pueden ser ocultados de ser deseado. La cantidad total de nodos y aristas presentes es siempre visible (ver figura 73).

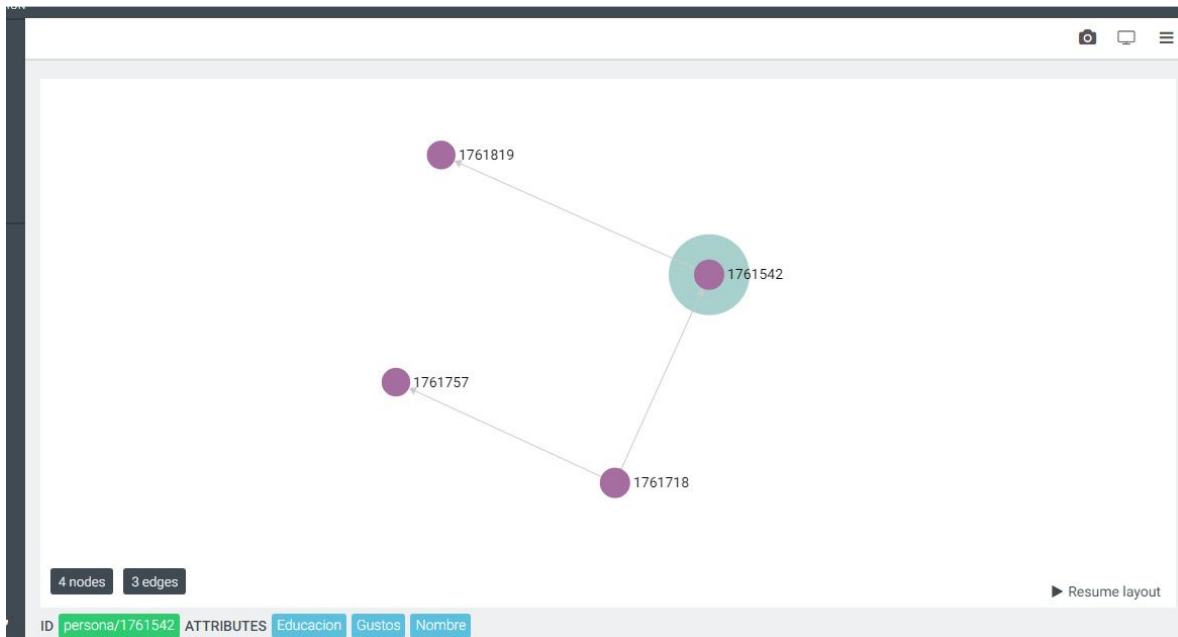


Figura 73. Visualización de grafo de ArangoDB.

ArangoDB permite trabajar sobre nodos y aristas directamente en el grafo. Al seleccionar un nodo se despliegan opciones como editar y eliminar, que impactan directamente a ese documento en su colección (ver figura 74). Adicionalmente, existe la opción de poder crear aristas nuevas directamente en el grafo (ver figuras 75, 76, y 77).



Figura 74. Menú de opciones de un nodo de ArangoDB.

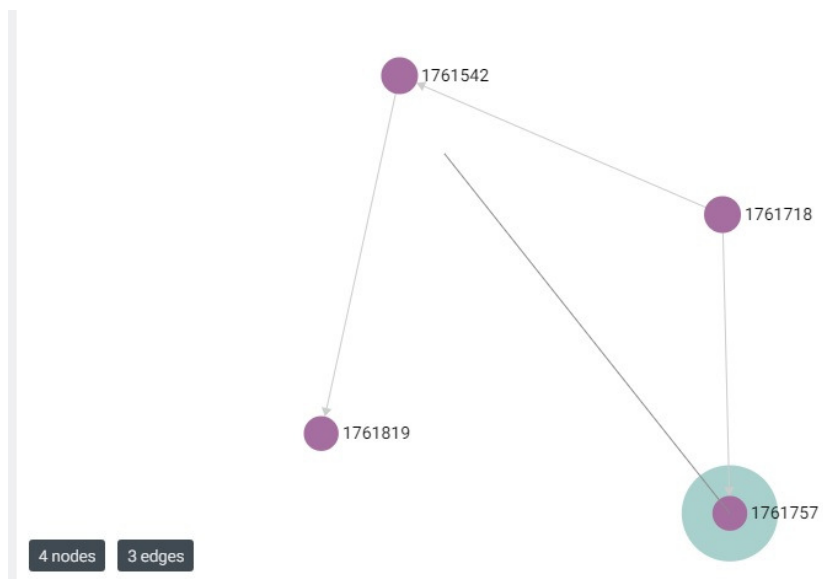


Figura 75. Creación de nueva arista directamente en grafo de ArangoDB (1).

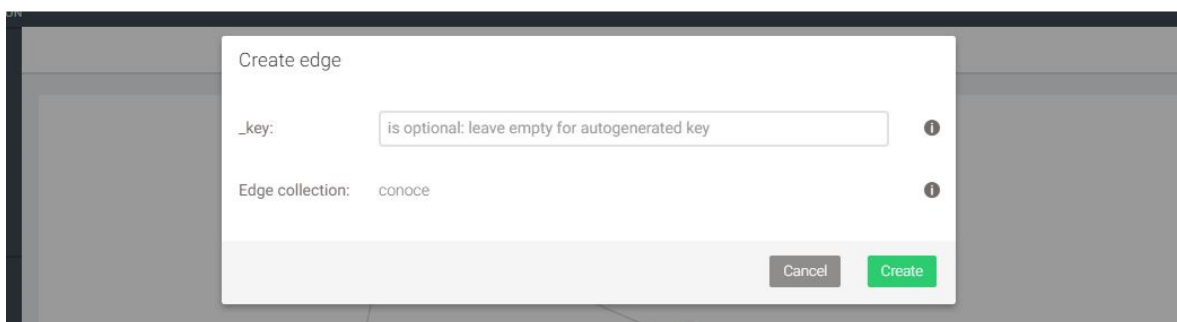


Figura 76. Creación de nueva arista directamente en grafo de ArangoDB (2).

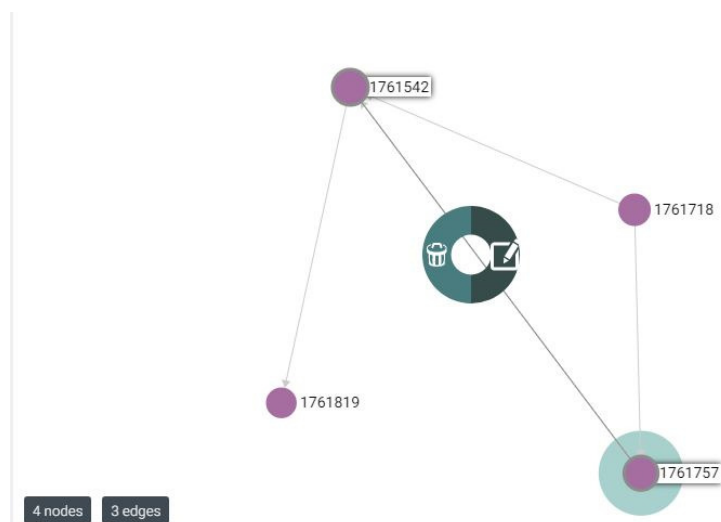


Figura 77. Opciones de arista en grafo de ArangoDB.

ArangoDB proporciona un panel lateral con una variedad de opciones para configurar los nodos y aristas, permitiendo cambiar su tamaño o color, o mostrar como nombre en el nodo o arista un atributo a elección (ver figuras 78 y 79).

USER: ROOT DB: ARANGO_DB1 HEALTH: GOOD

Graph

Startnode: persona/1761718

Layout: Force

Renderer: Canvas

Search Depth: 2

Limit: 250

Nodes

Label: _key

Add Collection Name: No

Color By Collections: Yes

Color: [Green Bar]

Color Attribute: attribute

Size By Connections: Yes

Sizing Attribute: attribute

Figura 78. Menú de configuración de grafo de ArangoDB.

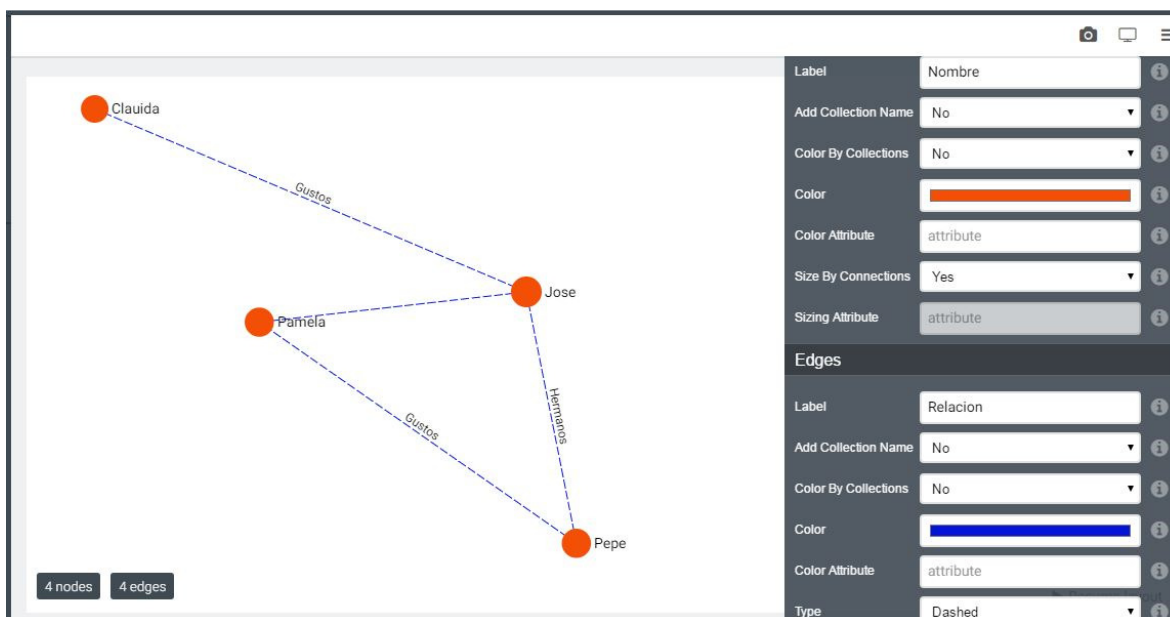


Figura 79. Ejemplo de uso de configuración de grafo de ArangoDB.

4.3.2 Neo4j v3.2.0

Neo4J es una base de datos de grafos programada en Java y lanzada en 2007. Su desarrolladora es la compañía Neo Technology, una *startup* sueca (Referencias 2 y 33). Neo4j utiliza su propio lenguaje de consultas llamado Cypher. Las bases de datos de grafos como Neo4j le dan un papel protagonista a la relación entre los datos, permitiendo su modelado gráfico a través de una malla de nodos y aristas. Grandes empresas como Walmart e eBay ya lo están utilizando para mejorar sus servicios y la atención a sus clientes.

Instalación

En el sitio web de Neo4j (Referencia 33) se encuentra el instalador del motor de base de datos, el que guía paso a paso durante el proceso. En este proceso, lo poco a configurar será el directorio donde se ubicará la base de datos.

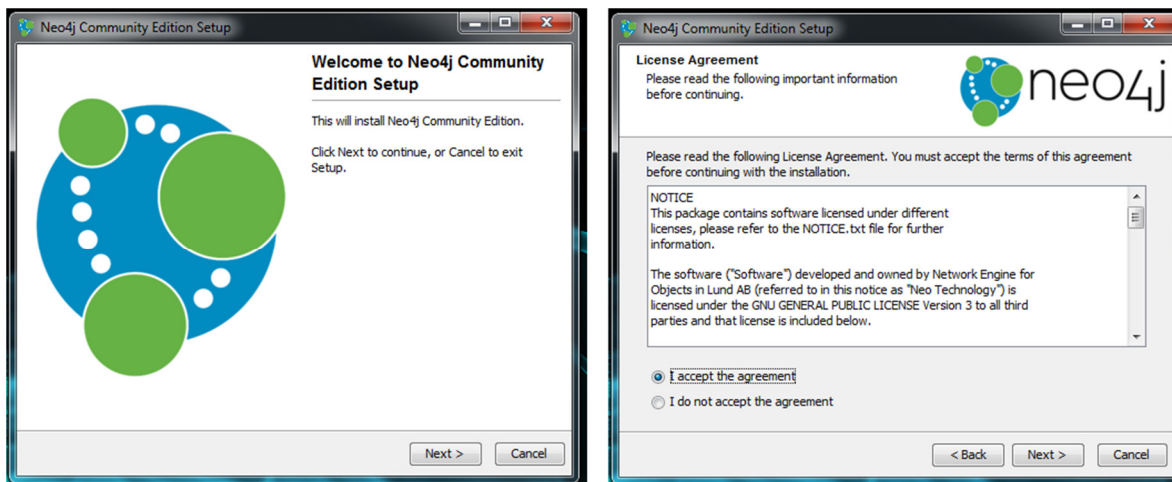


Figura 80. Instalador de Neo4j para Windows (1).

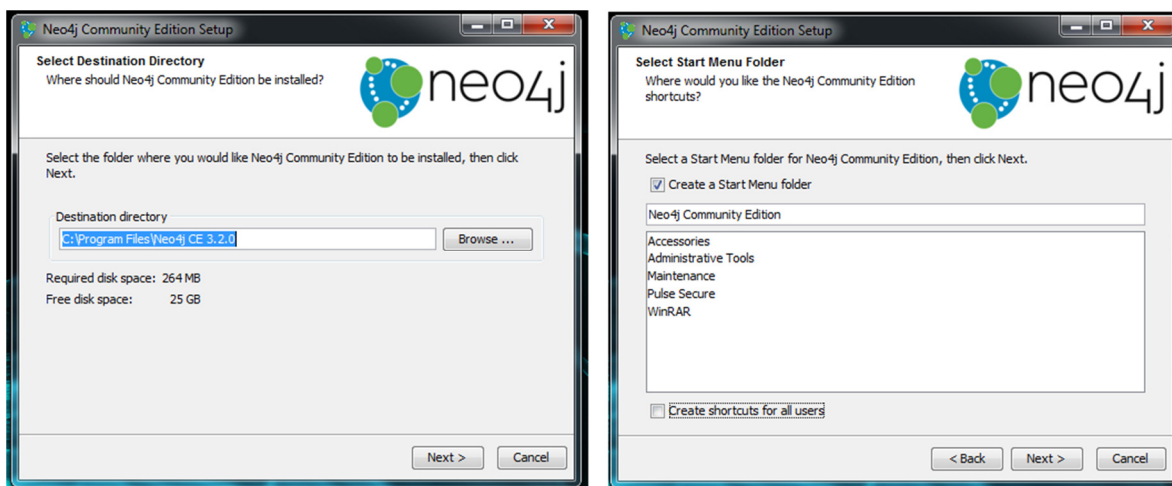


Figura 81. Instalador de Neo4j para Windows (2).

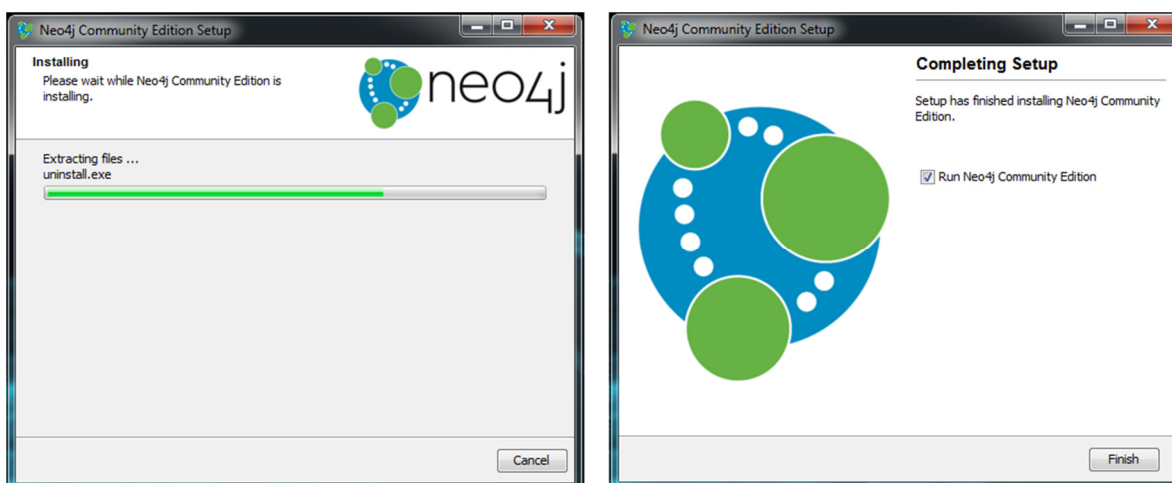


Figura 82. Instalador de Neo4j para Windows (3).

Terminando la instalación, se puede acceder a las bases de datos ubicando el directorio en el cual fueron creadas; en caso de no existir una base de datos, se crea una nueva. Todo esto en el panel provisto para tal fin (ver figura 83).

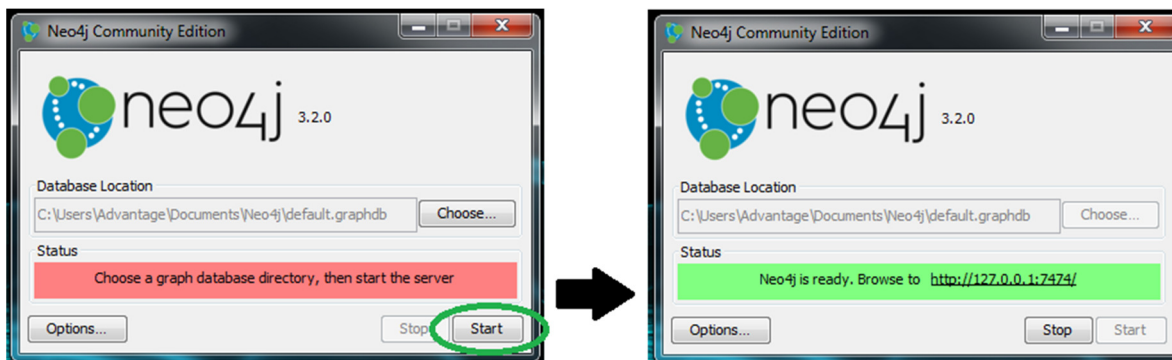


Figura 83. Iniciando base de datos de Neo4j.

En el panel se dará una dirección web la cual se debe ingresar en el navegador web de preferencia para tener acceso a la base de datos. Al acceder se despliega una ventana de usuario y contraseña para tener diferentes perfiles. En este caso se utilizó la sesión por defecto: Usuario: neo4j, Contraseña:neo4j (ver figura 84).

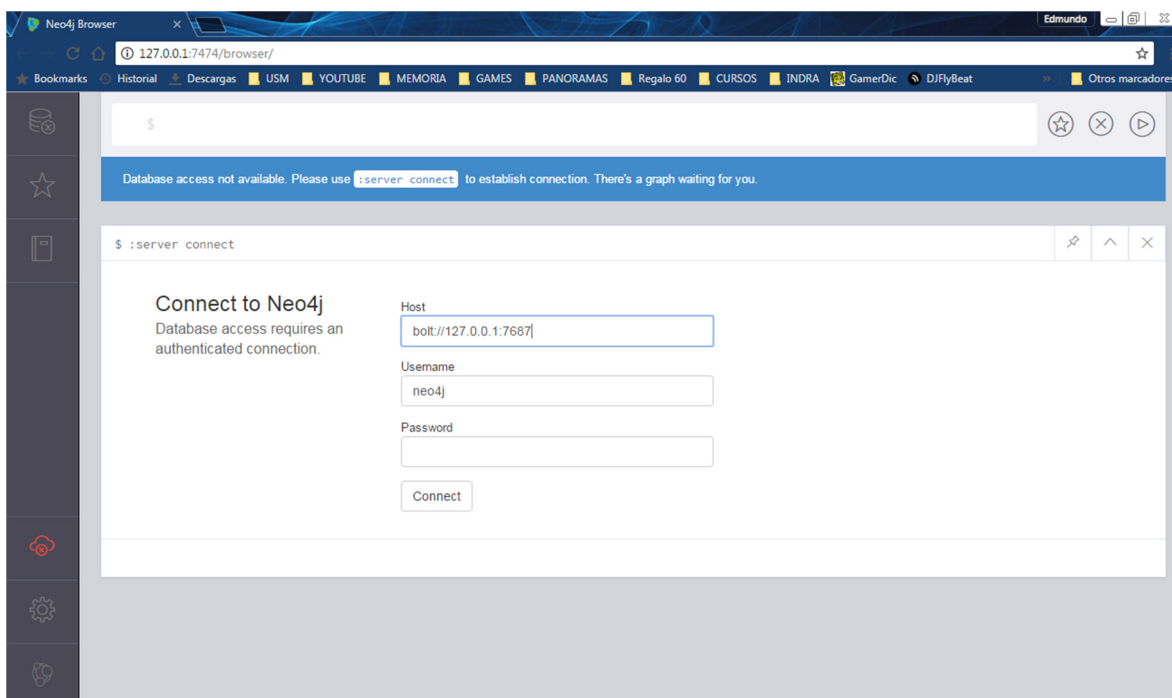


Figura 84. Acceso a la base de datos de Neo4j.

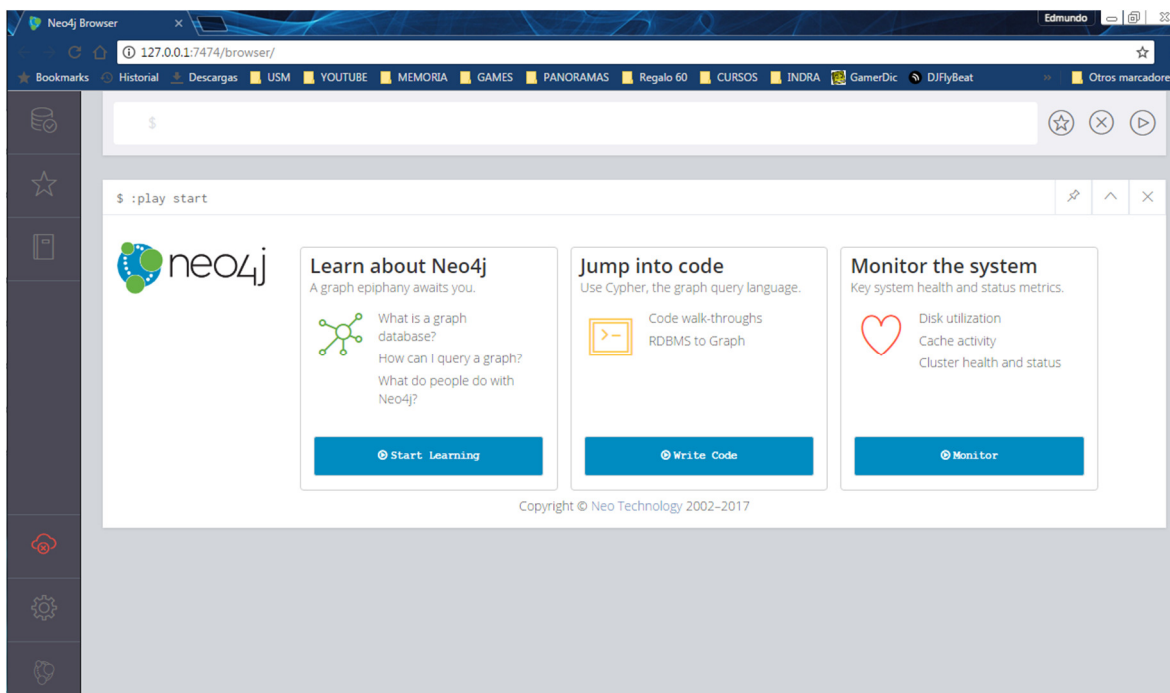


Figura 85. Panel principal de Neo4j.

Creación y Manejo de Datos

La unidad elemental de Neo4j son los nodos. Éstos son entidades con diferentes atributos y que se relacionan de distinta manera con sus pares. Cypher es el lenguaje que utiliza Neo4j para las diferentes operaciones CRUD. Para crear un nodo, se debe definir una etiqueta bajo la cual se identifique su naturaleza, por decirlo de cierto modo, y sus atributos (ver figura 86).

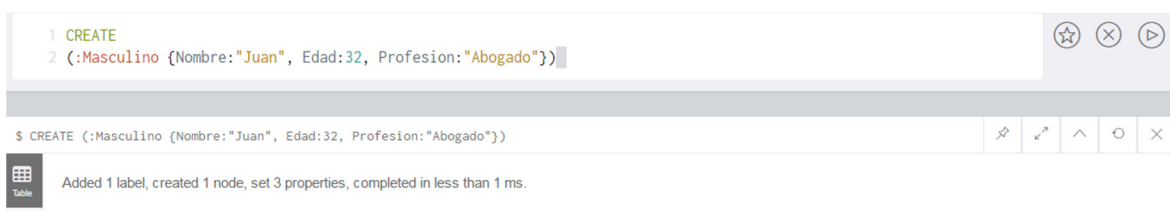


Figura 86. Ejemplo de creación de nodos en Neo4j.

Por su puesto, se pueden crear múltiples nodos con diferentes etiquetas y con distintos atributos, como se muestra en la figura 87.

```
1 CREATE
2 (:Masculino {Nombre:"Pedro", Edad:16, Profesion:"Estudiante", Curso:"2do Medio"}),
3 (:Femenino {Nombre:"Angelica", Edad:28, Profesion:"Enfermera", Rut:"12.123.853-5"}),
4 (:Femenino {Nombre:"Beatriz", Edad:15, Profesion:"Estudiante", Colegio:"Los Pinos"}),
5 (:Vehiculo {Tipo:"Automovil", Marca:"Honda", Modelo:"Civic", Año:1993})
```

\$ CREATE (:Masculino {Nombre:"Pedro", Edad:16, Profesion:"Estudiante", Curso:"2do Medio"}), (:Femenino {Nombre:"Angelica", Edad:28, Profesion:"Enfermera", Rut:"12.123.853-5"}), (:Femenino {Nombre:"Beatriz", Edad:15, Profesion:"Estudiante", Colegio:"Los Pinos"}), (:Vehiculo {Tipo:"Automovil", Marca:"Honda", Modelo:"Civic", Año:1993})

Added 4 labels, created 4 nodes, set 16 properties, completed after 121 ms.

Figura 87. Ejemplo de creación de múltiples nodos en Neo4j.

Para realizar búsquedas se debe definir la etiqueta del nodo o de la relación a buscar y sus atributos. Esto es necesario para crear relaciones en nodos existentes, ya que se ubican los nodos a ser conectados y luego se crea la relación sobre ellos.

A modo de ejemplo, en la figura 87 se buscan a todas las personas para relacionarlas con el nodo Vehículo. Esto se puede hacer ya sea buscando las etiquetas Masculino y Femenino, o bien excluyendo la etiqueta Vehículo. Los nodos resultantes se les asigna el alias 'consulta' (ver figuras 88 y 89).

```
$ Match (consulta) WHERE (consulta:Masculino) OR (consulta:Femenino) return consulta
```

Figura 88. Búsqueda de nodos con la etiqueta Masculino o Femenino.

```
$ Match (consulta) WHERE NOT (consulta:Vehiculo) return consulta
```

Figura 89. Búsqueda de todos los nodos menos aquellos con etiqueta Vehiculo.

Sobre el mismo caso de ejemplo, el resultado para ambas consultas será el mismo. Saber cuál será la mejor manera de conseguir el resultado esperado dependerá de la malla de datos que se tenga. El resultado obtenido puede ser visualizado de tres maneras diferentes: grafo, tabla y texto (ver figuras 90, 91 y 92).

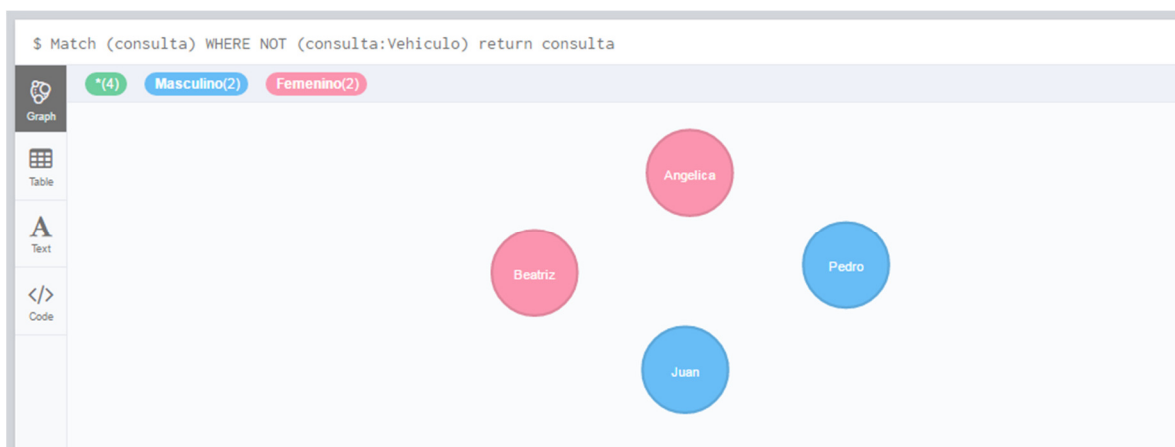


Figura 90. Visualización de grafo de Neo4j.

\$ Match (consulta) WHERE NOT (consulta:Vehiculo) return consulta

"consulta"
{ "Nombre": "Juan", "Edad": 32, "Profesion": "Abogado" }
{ "Nombre": "Pedro", "Edad": 16, "Curso": "2do Medio", "Profesion": "Estudiante" }
{ "Rut": "12.123.853-5", "Nombre": "Angelica", "Edad": 28, "Profesion": "Enfermera" }
{ "Nombre": "Beatriz", "Edad": 15, "Colegio": "Los Pinos", "Profesion": "Estudiante" }

Figura 91. Visualización de tabla de Neo4j.

\$ Match (consulta) WHERE NOT (consulta:Vehiculo) return consulta

"consulta"
{ "Nombre": "Juan", "Edad": 32, "Profesion": "Abogado" }
{ "Nombre": "Pedro", "Edad": 16, "Curso": "2do Medio", "Profesion": "Estudiante" }
{ "Rut": "12.123.853-5", "Nombre": "Angelica", "Edad": 28, "Profesion": "Enfermera" }
{ "Nombre": "Beatriz", "Edad": 15, "Colegio": "Los Pinos", "Profesion": "Estudiante" }

Figura 92. Visualización de texto de Neo4j.

En el mismo contexto del ejemplo de la figura 87 y tras encontrar los nodos de las personas, se debe ubicar el nodo Vehículo con el que queremos relacionarlo (ver figura 93).

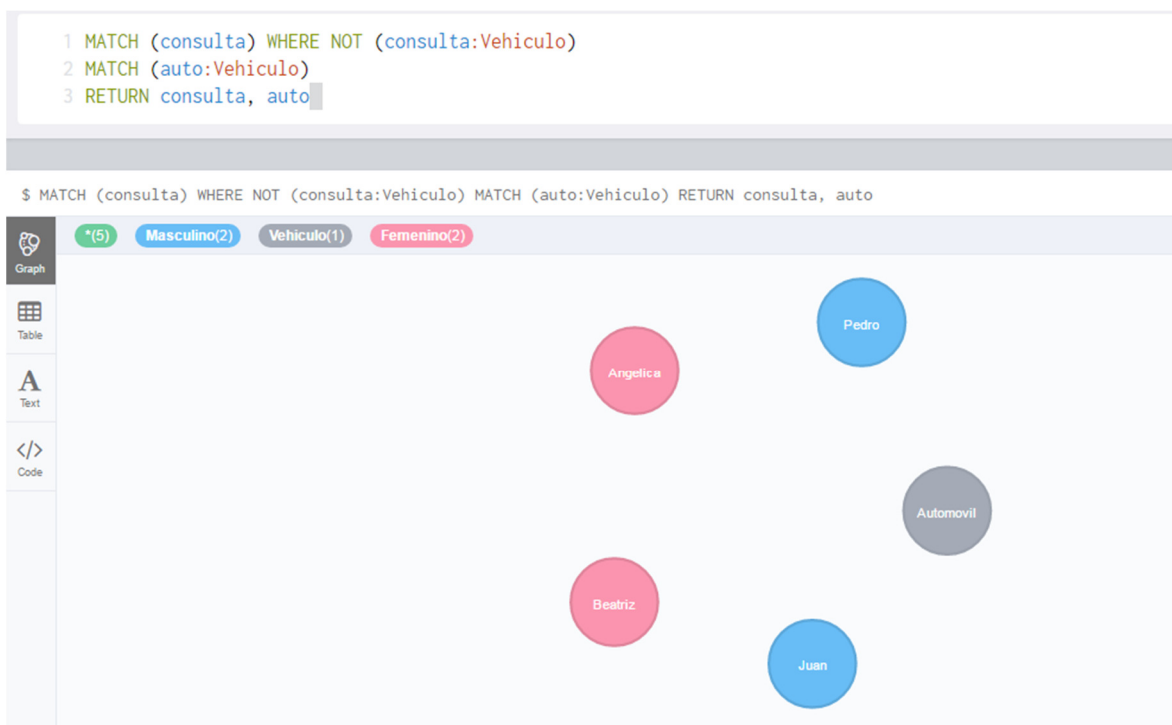


Figura 93. Búsqueda de las personas y el automovil. Resultado antes de crear las relaciones.

Definiendo la relación que se quiere establecer entre ellos y el nodo Vehículo, se crea la relación utilizando los alias asignados a los resultados de las búsquedas. La relación a crear será Viajan (ver figura 94).

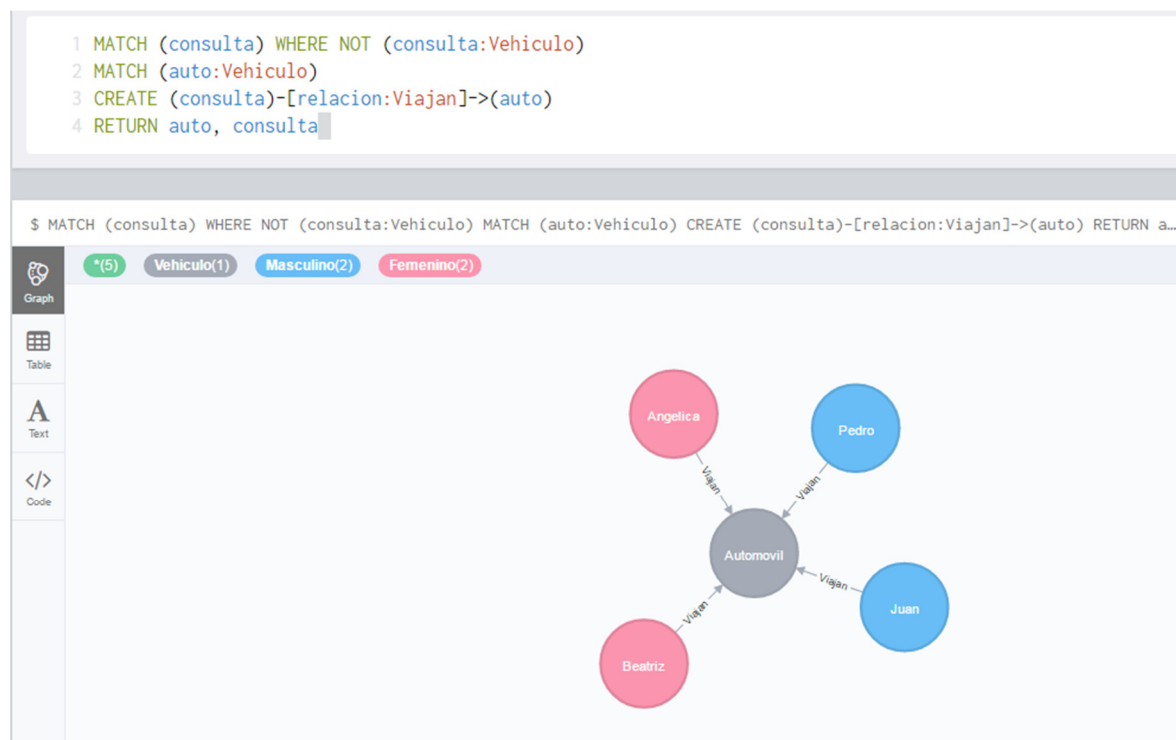


Figura 94. Ejemplo de creación de relación en Neo4j.

La consulta anterior permite apreciar que las relaciones también pueden llevar alias, los cuales son muy recomendables al momento de ejecutar consultas de búsqueda complejas y crear relaciones. Otro aspecto a nombrar, es que Neo4j no trabaja relaciones bidireccionales; para representar una relación en ambos sentidos es necesario crear dos relaciones diferentes, apuntando en sentidos opuestos. Si bien es completamente factible, no se recomienda hacerlo ya que sobrecarga de manera innecesaria la base de datos y la malla de nodos. Esta situación se puede abordar haciendo una correcta interpretación de las relaciones unidireccionales.

Una consulta más compleja sobre el ejemplo de la figura 87 sería el crear una relación Conduce entre los nodos con cláusula Edad mayor a 18 años. Para esto hay que precisar esta información en la consulta y crear la nueva relación (ver figura 95).

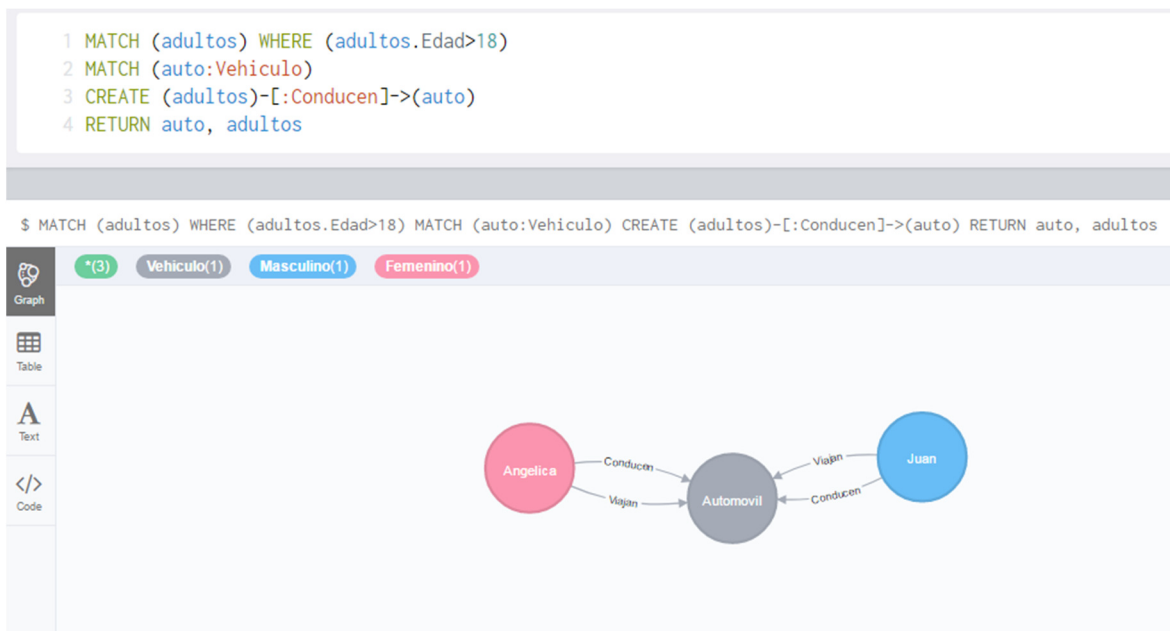


Figura 95. Ejemplo de creación de relación Conducen desde nodos con Edad mayor a 18.

En la consulta anterior no fue necesario precisar la etiqueta de los nodos ‘adultos’ ya que Neo4j busca si el nodo tiene la cláusula Edad; de no tenerlo lo descarta. Al no tener el nodo Vehículo la cláusula Edad no fue necesario especificar que debían tener etiqueta Femenino o Masculino (sobre el ejemplo de la figura 87).

Las relaciones son muy útiles pues entregan información entre los diferentes nodos, por lo que es práctico el que contengan atributos. Para agregarle a la relación Conduce el año desde el cual la relación comenzó, sobre el ejemplo de la figura 87, es necesario situarse en la relación y ejecutar una actualización (ver figura 96).

```

1 MATCH (:Masculino)-[inicio_el:Conducen]->(auto:Vehiculo)
2 MATCH (:Femenino)-[inicio_ella:Conducen]->(auto)
3 SET inicio_el = {Desde:2009}, inicio_ella = {Desde:2012, Hasta:2015}
4 RETURN auto

```

\$ MATCH (:Masculino)-[inicio_el:Conducen]->(auto:Vehiculo) MATCH (:Femenino)-[inicio_ella:Conducen]->(auto) SET inicio_el = {...

Graph

* (5) Vehiculo(1) Masculino(2) Femenino(2)

* (6) Conducen(2) Viajan(4)

Table

Text

Code

Conducen <id>: 307 Hasta: 2015 Desde: 2012

Figura 96. Actualización de relación en Neo4j.

Para eliminar un nodo basta con utilizar el comando *delete* sobre él. Sin embargo, Neo4j no permite eliminar un nodo si éste posee alguna relación con otro, por lo cual primero se debe eliminar todas las relaciones asociadas al nodo que se desee eliminar. En la figura 97 se aprecia la alerta al momento de intentar eliminar un nodo con relaciones asociadas a él. En la figura 98 se muestra cómo eliminar una relaciones asociadas a un nodo y, posteriormente, se elimina el nodo.

```

1 MATCH (eliminar:Masculino) WHERE eliminar.Nombre="Juan"
2 DELETE eliminar

```

\$ MATCH (eliminar:Masculino) WHERE eliminar.Nombre="Juan" DELETE eliminar

Error

ERROR Neo.ClientError.Schema.ConstraintValidationFailed

Cannot delete node<210>, because it still has relationships. To delete this node, you must first delete its relationships.

Figura 97. Alerta de eliminar un nodo con relaciones asociadas a él.

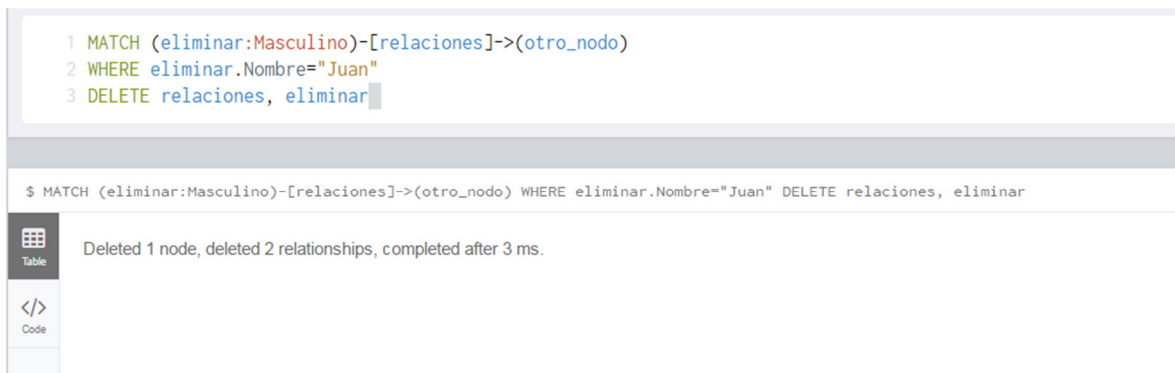


Figura 98. Eliminación de un nodo y sus relaciones asociadas.

El resultado restante lo podemos apreciar haciendo una consulta que retorne toda la malla, como muestra la figura 99.

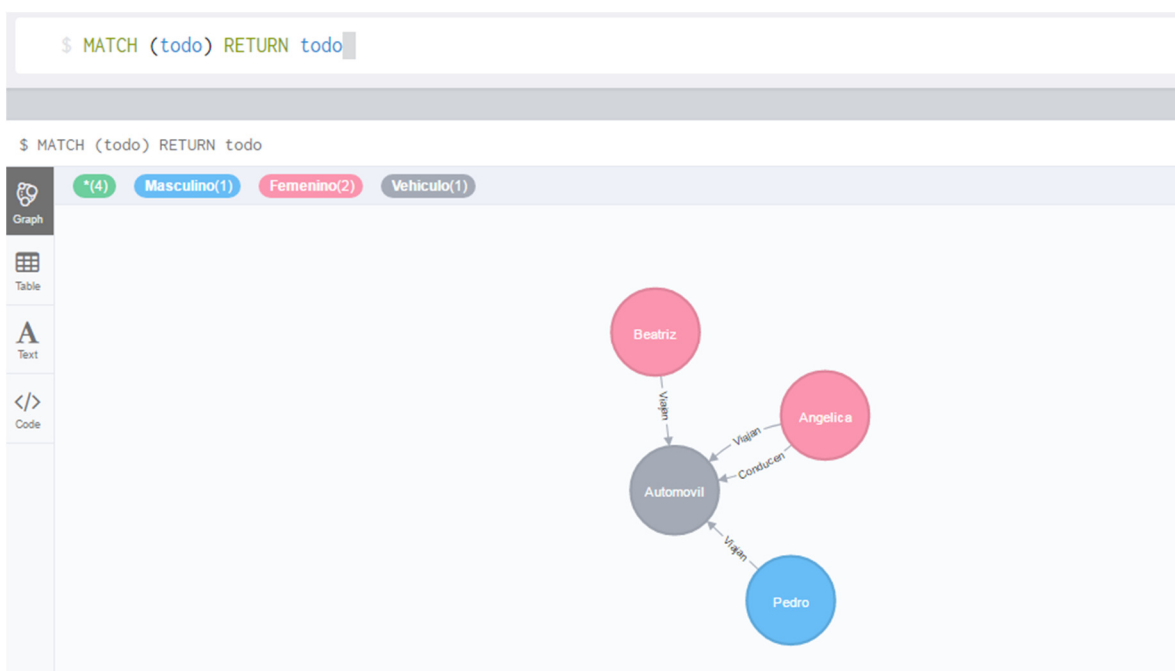


Figura 99. Visualización de toda la malla de nodos en Neo4j.

Todo el trabajo en Neo4j, tanto la manipulación de nodos como la manipulación de relaciones, se basan por las mismas operaciones de búsqueda, creación actualización y eliminación, siendo fundamental la correcta identificación de los nodos y relaciones a manipular.

Visualización Gráfica de Datos

La interacción y despliegue gráfico de la malla es claro, con diferentes opciones para mover, ocultar y desplegar más nodos con sólo unos *clicks* (ver figura 100).

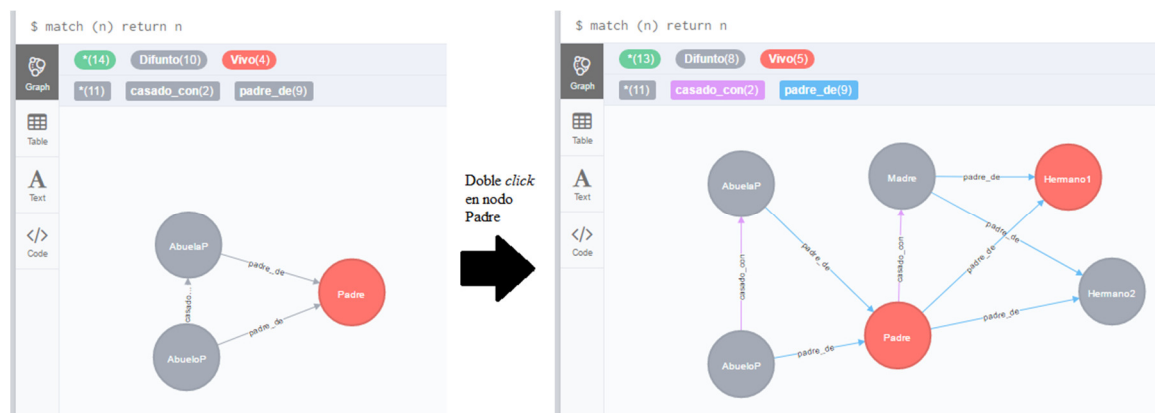


Figura 100. Ejemplo de despliegue de nodos.

Los nodos pueden ser desplazados y ubicados en diferentes posiciones, y pueden ser ocultados de ser deseado. Neo4j permite diferenciar los nodos y relaciones según su etiqueta, ya sea cambiando su tamaño o color o mostrando como nombre en el nodo o relación un atributo a elección (ver figuras 101 y 102).

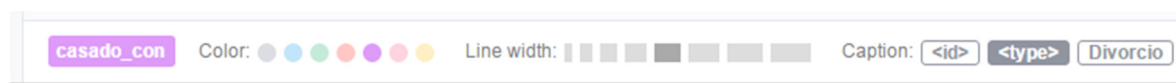


Figura 101. Opciones de color, ancho de línea y alias para las relaciones



Figura 102. Opciones de color, tamaño de la esfera y alias para los nodos.

Además, un pequeño panel muestra en todo momento la cantidad total de elementos desplegados según su etiqueta, práctico al momento de analizar los resultados obtenidos y la presencia o no de ciertos nodos y relaciones. Esto se puede apreciar en la figura 103.



Figura 103. Indicadores de cantidad de nodos y relaciones en Neo4j.

4.3.3 Comparación Preliminar

En la tabla 5 se presentan los sistemas de bases de datos sometidos a los criterios de comparación, destacando cuál se desempeña mejor en cada punto.

Criterio	ArangoDB	Neo4j	Argumento
Instalación	---	---	Neo4j tiene una instalación muy simple, pues tras unos pocos <i>clicks</i> ya se puede manejar la base de datos. ArangoDB, si bien con unas cuantas opciones más al momento de instalar, sigue siendo sencilla de implementar.
Manejo de Datos CRUD			La definición de nodos y aristas en Neo4j es intuitiva, con ejemplos como $A \Rightarrow B$, rápidamente se definen las relaciones entre los diferentes nodos. ArangoDB separa en dos ambientes los nodos y las aristas, teniéndose que cambiar entre ambientes para poder armar la malla.
Documentación y Soporte	---	---	Ambos sistemas cuentan con documentación de fácil acceso en sus canales oficiales de comunicación (Referencias 34 y 36).
Curva de Aprendizaje			ArangoDB cuenta con un lenguaje de consulta similar al de MongoDB, llamado AQL, que no resulta complicado de aprender, pero Cypher de Neo4j termina siendo superior por la similitud entre lo que se define en la consulta versus el resultado creado. Además, al trabajar en espacios separados, los nodos de las aristas puede resultar confuso en un comienzo.
Claridad del Paradigma			ArangoDB es una base de datos que abarca tanto al modelo documental como al de grafo, permitiendo uno trabajar con ambos de manera conjunta, incluso teniendo una gama de diferentes opciones de modelado gráfico (grafos, columnas, torta). Pese a esto, al analizar exclusivamente el módulo de grafos, Neo4j termina siendo una herramienta más refinada y especializada.

Interfaz Gráfica			Tanto ArangoDB como Neo4j se manipulan utilizando el navegador web. Ambos poseen indicaciones y etiquetas claras de las distintas opciones disponibles pero es fácil perderse en ArangoDB por la gran gama de opciones que engloba, tanto para el modelamiento documental como el de grafos.
Relevancia en el Mercado			Actualmente Neo4j es utilizado por un pequeño grupo del comercio para suplir necesidades muy puntuales. ArangoDB resulta ser una herramienta más completa, misma razón que le resta relevancia, ya que para una necesidad puntual se optará por una herramienta especializada.

Tabla 5. Comparativa entre ArangoDB y Neo4j.

Al trabajar con las bases de datos de grafos ArangoDB y Neo4j, se apreciaron claras diferencias debido a la naturaleza de ambas. Mientras ArangoDb es un sistema multi-modelo (clave-valor, documental y de grafos), Neo4j es un sistema refinado para el almacenamiento, modelado y análisis en grafos. Las herramientas no se diferencian al momento de instalarse y montarse, siendo ambos procesos guiados por sus respectivos instaladores. Ya en la base de datos propiamente tal, ArangoDB se destaca por su gran arsenal de herramientas y opciones para definir, estructurar, indexar y un sin fin más de variables para definir tanto en la base de datos como en los elementos a almacenar. Neo4j, si bien permite la definición de nodos y relaciones más complejas, no está optimizado para el análisis y la búsqueda dentro de estos complejos elementos. Por otro lado, ArangoDB maneja un balance entre la complejidad de sus nodos (enfocados al paradigma documental) y el análisis tanto del nodo como de su relación con la red completa de ellos. Neo4j presenta una alta curva de aprendizaje, lo que permite a nuevas personas adoptar rápidamente el enfoque que este paradigma presenta. ArangoDB resulta más completo y, por lo mismo, más complejo al momento de presentarle al usuario los múltiples enfoques que pueden adoptarse para modelar la información y su almacenamiento. Ambos motores de bases de datos se manipulan desde sus interfaces accesibles a través de un navegador web y ambos cuentan con vasta documentación para su entendimiento. Si bien ArangoDB es una herramienta más completa, que otorga una mayor selección de almacenamiento, búsqueda, análisis y manipulación de los datos, la poca especialización que tiene producto de abarcar una mayor gama de posibilidades le juega en contra; por su parte, Neo4j se ha especializado a tal punto de ser la herramienta de preferencia en el universo de las bases de datos de grafos, al ser un sistema limpio, refinado y altamente optimizado en la búsqueda y análisis de relaciones.

4.3.4 Propuesta Actividad Académica

En éste apartado se expone la propuesta académica a realizar por los alumnos, definiendo los elementos esenciales para el correcto aprendizaje del paradigma a estudiar.

Una experiencia académica para acercarse a este tipo de sistema sería el levantamiento de una base de datos Neo4j, poblarla con datos que permitan el modelamiento de una red de nodos y la manera cómo éstos se relacionan los unos a los otros, una red social, un árbol familiar, rutas de comercio entre ciudades. El hecho de que la relación entre los datos tome un papel igual de protagónico que los datos mismos sería clave para cumplir los objetivos de aprendizaje del sistema. A continuación se presenta el encabezado y propósitos de la actividad docente:

“ Los alumnos deberán instalar la base de datos Neo4j y confeccionar un árbol genealógico de al menos 3 niveles. Cada alumno deberá aportar su árbol familiar a su grupo y unirlos con la relación “amigo_de” los nodos de los alumnos, formando así una malla de nodos.

A continuación se detallan las características que deben cumplir tanto los nodos como las relaciones:

Desarrollo del Trabajo:

- *Cada árbol debe tener al menos 25 nodos.*
- *Las relaciones y atributos mínimos de las mismas, utilizadas para unir los nodos deberán incluir al menos las siguientes:*
 - *amigo_de Atributos: fecha_ini*
 - *padre_de Atributos: fecha_ini*
 - *casado_con/conviviente Atributos: fecha_ini, fecha_fin(si corresponde)*
- *Los tipos de nodos y atributos mínimos de los mismos deben ser al menos:*
 - *hombre Atributos: nombre, edad, actividad, gusto1, gusto2, disgusto, defunsion(si corresponde)*
 - *mujer Atributos: nombre, edad, actividad, gusto1, gusto2, disgusto, defunsion(si corresponde)*
- *La lista de gusto/disgusto a utilizar es la siguiente:*
 - *musica*
 - *futbol*
 - *basquetball*
 - *yoga*

- *películas*
 - *tecnología*
 - *viajar*
 - *juegos*
 - *leer*
 - *autos*
 - *motos*
 - *gatos*
 - *perros*
 - *naturaleza*
 - *moda*
- *La lista de actividad a utilizar es la siguiente:*
 - *profesor*
 - *ingeniero*
 - *militar*
 - *psicologo*
 - *estudiante*
 - *abogado*
 - *farmaceutico*
 - *medico*
 - *veterinario*
 - *otros (no especificar)*

Los alumnos deberán experimentar con la malla de nodos, realizando consultas en base a las relaciones que existen entre los nodos y sus atributos. Algunas consultas que pueden poner en práctica son:

- *Todas las personas que sean estudiante.*
- *Primos que poseen - los amigos del alumnoA*
- *Persona a menos saltos que comparta algún gusto con el padre de alumnoA*
- *Persona a mayor distancia de alumnoA que comparta algun gusto pero no le disguste algo de alumnosA*
- *Parientes que alumnosA podria presentarle a alumnoB según gustos y rango etario.*
- *Tios masculinos de un amigo que le disgusten los gatos y sean veterinarios*
- *Pariente vivo de mayor edad de cada alumno ”*

Conclusiones

En los últimos años las empresas se han visto limitadas a las capacidades que los sistemas de bases de datos relacionales tradicionales poseen, lo que las ha llevado a considerar otras alternativas como las bases de datos NoSQL, las cuales se destacan por: ser sistemas más económicos y menos complejos, reduciendo costos de licencias, infraestructura y desarrollo; sistemas altamente escalables, práctico por sobre todo en el manejo de *big data*; utilizan modelos de datos semi-estructurados que ofrecen mayor flexibilidad que los sistemas relacionales. En resumen, los sistemas NoSQL ofrecen una clase de sistemas de gestión de bases de datos no relacionales, distribuidas, horizontalmente escalables, de código abierto y más veloces, ya que no soportan las propiedades ACID.

En la actualidad se ha hablado hasta de 150 sistemas de bases de datos NoSQL distintas, mas la mayoría pueden agruparse en los tres grandes grupos que se han estudiado: columnar, documental y de grafos.

En este trabajo se definieron criterios tales que permitieron evaluar diferentes motores de bases de datos NoSQL en cada una de las categorías con el propósito de seleccionar la mejor alternativa a usar en un ambiente docente de introducción a este nuevo tipo de sistema de almacenamiento.

Las bases de datos columnares poseen un alto nivel de rendimiento y eficiencia, en especial en lo referente al análisis OLAP sobre el conjunto. Su alta curva de aprendizaje, alto rendimiento, almacenamiento de datos no estructurados, diseño para manejar grandes volúmenes de datos, rendimiento de lectura y escritura y alta disponibilidad las hacen una herramienta ideal para su aplicación en inteligencia de negocios y el análisis de datos. Estos sistemas de gestión de bases de datos se ejecutan en clústeres de varios servidores distribuidos geográficamente a través de múltiples centros de datos, lo que le da una gran tolerancia a fallos pero a coste de tener que tolerar alguna inconsistencia a corto plazo en las réplicas producto de la sincronización entre los nodos. Diversas áreas pueden utilizar este tipo de capacidad de procesamiento de datos, tales como:

- Análisis de seguridad utilizando el tráfico de red.
- *Big Science*, como la bioinformática utilizando datos genéticos.
- Análisis bursátil con datos comerciales.
- Aplicaciones a escala web, como la búsqueda.
- Servicios de redes sociales.

Las bases de datos documentales poseen una manera natural de modelar los datos, cercana a la programación orientada a objeto. Son flexibles, con esquemas dinámicos, reducen la complejidad de acceso a los datos, diseñadas para ofrecer flexibilidad. Si una aplicación requiere la capacidad de almacenar diversos atributos junto con grandes cantidades de datos, las bases de datos de documentos son una buena opción. Estas bases de datos proporcionan documentos incrustados, que son útiles para desnormalizar. En lugar de almacenar datos en tablas diferentes, los datos que se consultan frecuentemente juntos, se almacenan juntos en el mismo documento donde la agrupación de los datos en un único elemento (documento) resulta natural: un curriculum, una boleta de compras, un diario mural. Además, las bases de datos documentales mejoran las capacidades de consulta con la indexación y la capacidad de filtrar documentos basados en los atributos que éstos posean, proporcionando la posibilidad de ejecutar consultas en base a cualquier tipo de campo dentro del documento. Las bases de datos orientadas a documentos también permiten hacer consultas apoyadas en índices secundarios, lo que facilita actualizar registros incluyendo uno o más campos dentro del documento. Probablemente, las bases de datos documentales sean las más populares de las bases de datos NoSQL debido a su flexibilidad, rendimiento y facilidad de uso. Estas bases de datos se adaptan bien a una serie de situaciones, entre las cuales se pueden nombrar:

- Soporte *back-end* para sitios web con altos volúmenes de lecturas y escrituras.
- Gestión de tipos de datos con atributos variables.
- Seguimiento de tipos de metadatos variables.
- Aplicaciones que utilizan estructuras de datos *json*.
- Aplicaciones que se benefician de la desnormalización, incorporando estructuras dentro de estructuras.

Las bases de datos de grafos son adecuadas para problemas que se prestan a las representaciones de redes de entidades conectadas, donde las conexiones tienen relación y relevancia con otras conexiones. El almacenamiento en este tipo de bases de datos está optimizado para ejecutar la navegación entre nodos; por este motivo, las bases de datos orientadas a grafos son eficientes para realizar consultas en las que existan relaciones de proximidad entre datos, y no para ejecutar consultas globales. El procesamiento y representación gráfica a gran escala, por ejemplo de grandes redes sociales, puede utilizar este tipo de almacenamiento. Destacan en ambientes donde la relación entre las entidades es objeto de estudio, como podría ser el tráfico entre diferentes puntos de una ciudad, la relación familiar entre diferentes personas o la relación que tiene una persona en Twitter con sus seguidores (*followers*). Otros ejemplos incluyen:

- Gestión de la infraestructura de redes y TI.
- Gestión de identidades y accesos.
- Gestión de Procesos de Negocio.
- Recomendar productos y servicios.
- Redes sociales

A partir de estos ejemplos, está claro que cuando hay una necesidad de modelar relaciones explícitas entre entidades y recorrer rápidamente caminos entre entidades, las bases de datos de grafos son una buena opción de base de datos.

Utilizando los criterios definidos en este trabajo, Cassandra resulta ser un sistema más maduro. El hecho de ser una de las primeras bases de datos columnares en hacerse un espacio en la comunidad, le permitió asentarse un puesto fijo al momento de evaluar sistemas columnares. Cassandra se ha encargado de refinar su sistema de almacenamiento columnar, llevándolo a otro nivel con la definición de grupos de familias, ideales para análisis OLAP e Inteligencia de Negocios. Cassandra resulta ser una mejor elección como sistema de base de datos columnar para utilizar en el ámbito docente.

Utilizando los criterios definidos en este trabajo, CouchDB resultó ser el sistema que mejor cumplía las condiciones para aplicarse en el ámbito académico. CouchDB presenta en su panel de configuración de la base de datos, todo tipo de opciones y configuraciones fácilmente reconocibles y ajustables. La presentación del contenido de las bases de datos es más claro y apreciable en CouchDB gracias a su interfaz a través del navegador web, resulta intuitivo, con paneles de ayuda al situar el cursos en los diferentes elementos. CouchDB resulta ser una mejor elección como sistema de base de datos documental para utilizar en el ámbito docente.

Utilizando los criterios definidos en este trabajo, Neo4j resultó ser un sistema más refinado para el almacenamiento, modelamiento y análisis en grafos. Neo4j presenta una alta curva de aprendizaje, lo que permite a nuevas personas adoptar rápidamente el enfoque que este paradigma de grafos presenta; además, se ha especializado a tal punto de ser la herramienta de preferencia en el universo de las bases de datos de grafos, al ser un sistema limpio, refinado y altamente optimizado en la búsqueda y análisis de relaciones. Neo4j resulta ser una mejor elección como sistema de Base de Datos de Grafos para utilizar en el ámbito docente.

El trabajo desarrollado con los diferentes sistemas de bases de datos, el análisis realizado, la definición de criterios para su evaluación, conclusión sobre su implementación y uso en el área docente, y la confección de experiencias prácticas para principiantes en los sistemas NoSQL, han sido actos que se han alineado con los objetivos de este trabajo, cumpliéndolos de manera satisfactoria.

Este trabajo puede extender su estudio a la comparación entre sistemas relacionales con cada uno de los paradigmas (columnar, documental, de grafos), analizando virtudes y defectos; comparación entre los diferentes sistemas NoSQL y los criterios que se debe tener presente al momento de escoger cuál implementar en un determinado escenario, considerando aspectos como las necesidades del negocio y las proyecciones de crecimiento; los sistemas NoSQL como complemento de los sistemas relacionales, una implementación conjunta que suma valor al negocio, como puede ser tener un sistema NoSQL *front-end* para responder a necesidades en tiempo real y un sistema SQL *back-end* que responda bajo demanda.

Este trabajo permite ampliar el área de estudio que actualmente se presenta a los estudiantes de las ciencias de las tecnologías de la información, siendo un aporte en las carreras afines. Las bases de datos NoSQL ya se están haciendo su espacio en las diferentes empresas, lo que hace fundamental el tener los conocimientos necesarios para trabajar con ellas.

Este trabajo presenta un apoyo a la carrera de Ingeniería de Ejecución en Informática y a otras carreras del área TI, en cursos o talleres de especialización para aquellos estudiantes enfocados en el área de las bases de datos y el análisis de datos, permitiéndoles estar más capacitados y competentes ante los nuevos desafíos que se les presenten, otorgándoles un mayor nivel profesional en el mercado laboral y aumentando el prestigio de la institución como entidad formadora de profesionales TI.

Referencias

Genbetadev, 2014. Bases de datos NoSQL. Elige la opción que mejor se adapte a tus necesidades. Recuperado de <https://www.genbetadev.com/bases-de-datos/bases-de-datos-nosql-elige-la-opcion-que-mejor-se-adapte-a-tus-necesidades>

Amazon Web Services, Inc., 2017. ¿Qué es NoSQL?. Recuperado de <https://aws.amazon.com/es/nosql/>

Pandorafms Monitoring Bloc, 2016. NOSQL vs. SQL. Key differences and when to choose each. Recuperado de <https://blog.pandorafms.org/nosql-vs-sql-key-differences/>

Microsoft Developer Network, 2017. Información general sobre escalabilidad. Recuperado de [https://msdn.microsoft.com/es-cl/library/aa292203\(v=vs.71\).aspx](https://msdn.microsoft.com/es-cl/library/aa292203(v=vs.71).aspx)

Junta de Andalucía. Conceptos sobre la escalabilidad. Recuperado de <http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/220>

John D. Cook, 2009. ACID versus BASE for database transactions. Recuperado de <https://www.johndcook.com/blog/2009/07/06/brewer-cap-theorem-base/>

Kristof Kovacs, 2017. Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase vs Couchbase vs OrientDB vs Aerospike vs Neo4j vs Hypertable vs ElasticSearch vs Accumulo vs VoltDB vs Scalaris vs RethinkDB comparison. Recuperado de <https://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis>

Rackspace Cloud University, 2014. Relational vs NoSQL Databases. Recuperado de <https://www.youtube.com/watch?v=XPqrY7YEs0A>

Santi Pérez, 2015. Claves para elegir tu base de datos NoSQL. Recuperado de <https://www.ondho.com/claves-para-elegir-tu-base-de-datos-nosql/>

NoSQL, 2009. Your Ultimate Guide to the Non-Relational Universe!. Recuperado de <http://nosql-database.org/>

Ari Amster, 2016. Cassandra vs. Hadoop: A Comparative Look. Recuperado de <https://www.qubole.com/blog/big-data/cassandra-vs-hadoop/>

Dan Sullivan, 2015. Types of NoSQL databases and key criteria for choosing them. Recuperado de <http://searchdatamanagement.techtarget.com/feature/Key-criteria-for-choosing-different-types-of-NoSQL-databases>

Julian Browne, 2009. Brewer's CAP Theorem. Recuperado de <http://www.julianbrowne.com/article/viewer/brewers-cap-theorem>

Yohan Graterol, 2014. NoSQL como el futuro de las bases de datos. Recuperado de <http://www.maestrosdelweb.com/nosql-como-el-futuro-de-las-bases-de-datos/>

Database Journal, 2010. 10 things you Need to Know About NoSQL Databases. Recuperado de <http://www.databasejournal.com/features/article.php/3905531/10-things-you-Need-to-Know-About-NoSQL-Databases.htm>

Max Ivak Personal Site, 2001. RDBMSs vs. NoSQL Databases: Overview. Recuperado de <http://maxivak.com/rdbms-vs-nosql-databases/>

Apache Cassandra, 2016. Manage massive amounts of data, fast, without losing sleep. Recuperado de <http://cassandra.apache.org/>

Datastax, 2017. About Apache Cassandra. Recuperado de http://docs.datastax.com/en/archived/cassandra_win/3.0/cassandra/cassandraAbout.html

Dan Gorst, 2014. MongoDB vs CouchDB. Recuperado de <http://blog.scottlogic.com/2014/08/04/mongodb-vs-couchdb.html>

Róbert Novotný, 2012. Setting up HBase on Windows. Recuperado de <http://ics.upjs.sk/~novotnyr/blog/334/setting-up-hbase-on-windows>

Rick Grehan, 2014. Big data showdown: Cassandra vs. HBase. Recuperado de <http://www.infoworld.com/article/2610656/database/big-data-showdown--cassandra-vs--hbase.html>

ht4w. Hypertable for Windows. Recuperado de <http://ht4w.softdev.ch/>

DB-Engines, 2017. System Properties Comparison Cassandra vs. HBase. Recuperado de <http://db-engines.com/en/system/Cassandra%3BHBase>

Hypertable Inc., 2014. Big Data. Big Performance. Recuperado de <http://www.hypertable.com/home/>

The Apache Software Foundation, 2017. Apache CouchDB. Recuperado de <http://couchdb.apache.org/>

Elad Israeli, 2010. In-memory BI is nt the future. It's the past. Recuperado de <http://elasticube.blogspot.cl/2010/09/in-memory-bi-is-not-future-its-past.html>

Juan F. Cía., 2015. Neo4j: Qué es y para qué sirve una base de datos orientada a grafos. Recuperado de <https://bbvaopen4u.com/es/actualidad/neo4j-que-es-y-para-que-sirve-una-base-de-datos-orientada-grafos>

Neo Technology, Inc., 2017. Neo4j, the world's leading graph database. Recuperado de <https://neo4j.com/>

ArangoDB Company, 2017. ArangoDB – Highly available multi-model NoSQL database. Recuperado de <https://www.arangodb.com/>

JoSek, 2010. Cassandra se perfila como “la reina” del NoSQL. Recuperado de <http://josek.net/2010/03/cassandra-nsql/>

Yohan Graterol, 2014. NoSQL como el futuro de las bases de datos. Recuperado de <http://www.maestrosdelweb.com/nosql-como-el-futuro-de-las-bases-de-datos/>

William Díaz Sepúlveda, 2013. Bases de datos NoSQL: Llegaron para quedarse. Recuperado de <http://basesdedatosnosql.blogspot.cl/>

CIO., 2015. 9 historias de éxito de MongoDB. Recuperado de <http://cioperu.pe/articulo/20112/9-historias-de-exito-de-mongodb/?p=2>

MongoDB, Inc., 2017. Flexible enough to fit any industry. Recuperado de <https://www.mongodb.com/who-uses-mongodb>

DB-Enginers, 2017. Complete Ranking. Recuperado de <https://db-engines.com/en/ranking>

SAP, 2017. SAP HANA and Databases. Recuperado de <https://www.sap.com/latinamerica/products/technology-platforms/database-management-system.html>

Oracle, 2017. Java SE Development Kit 8 Downloads. Recuperado de <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html?ssSourceSiteId=otnes>

Free Software Foundation, 2017. GNU General Public License. Recuperado de <https://www.gnu.org/licenses/gpl-3.0.en.html>

3T Software Labs, 2017. Robomongo is now Robo 3T. Recuperado de <https://robomongo.org/>

Christof Strauch, 2010. NoSQL Databases. Recuperado de <http://www.christof-strauch.de/nosql dbs.pdf> - Pags. 3-17

Alan Downing, 2011. Debunking the NoSQL Hype. Recuperado de Oracle White Paper.

Yury Izrailevsky, 2011. NoSQL at Netflix. Recuperado de <http://techblog.netflix.com/2011/01/nosql-at-netflix.html>

Michael Stonebraker, 2010. Clarifications on the CAP Theorem and Data-Related Errors. Recuperado de <http://voltdb.com/clarifications-cap-theorem-and-data-related-errors/>

Lakshman Avinash & Malik Prashant, 2009. Cassandra - Structured Storage System over a P2P Network. Recuperado de http://static.last.fm/johan/nosql-20090611/cassandra_nosql.pdf

Javier Soltero, 2009. No to SQL? Anti-database movement gains steam. Recuperado de http://www.computerworld.com/s/article/9135086/No_to_SQL_Anti_database_movement_gains_steam?taxonomyId=173&pageNumber=1

Noé Cruz, 2017. 4 criterios para elegir tu primer language de programación. Recuperado de <http://www.1000tipsinformaticos.com/2015/09/4-criterios-para-elegir-tu-primer.html>

Takub, 2014. ¿Qué language de programación debes aprender?. Recuperado de <https://www.takub.com/es/blogspanish/item/205-que-lenguajes-de-programacion-debes-aprender>

Osmar Guevara, 2015. ¿Qué language de programación deberías aprender?. Recuperado de <https://geekytheory.com/que-lenguaje-de-programacion-deberias-aprender>