TRABAJO FINAL PARA LA ESPECIALIZACIÓN EN INGENIERÍA DE SOFTWARE

UN ESTUDIO COMPARATIVO DE BASES DE DATOS NOSQL

Autor: Lic. Marrero Luciano (UNLP)

Director: Mg. Thomas Pablo Javier (UNLP)

FACULTAD DE INFORMATICA

UNIVERSIDAD NACIONAL DE LA PLATA

Índice Temático

Capítulo 1

1.1 Introducción

Los motores de Bases de Datos tradicionales, de arquitectura relacional, han sido utilizados durante las últimas décadas para resolver cualquier problema asociado al almacenamiento y consulta de datos. Sin embargo, con el avance de las tecnologías informáticas y la comunicación, se presentan nuevos desafíos que estas Bases de Datos no han podido resolver de manera eficiente. Los modelos y arquitecturas que utilizan las Bases de Datos relacionales parecen no ser suficientes ante la generación masiva de datos, proveniente de aplicaciones que son mundialmente utilizadas. En la actualidad, la relevancia en la utilización de tecnología móvil y el acceso a internet ha ido aumentando debido a los cambios de hábitos con relación al estudio y trabajo a distancia, consumo electrónico, relaciones personales, entretenimiento, entre otras actividades. La generación de información se encuentra en constante crecimiento y la tecnología móvil y el acceso a internet se han convertido en una parte esencial de la vida cotidiana de las personas. Las redes sociales, el comercio electrónico y las plataformas de transmisión de contenidos, son los ejemplos más conocidos de los tipos de aplicaciones que han mostrado un incremento notorio en su utilización, en estos últimos años. Se estima que el 59,5% de la población mundial, utiliza o ha utilizado internet. El 66,6% utiliza, o ha utilizado, un dispositivo móvil, y más del 50% de la población mundial hace uso de alguna red social [19, 20, 21].

En este contexto, los principales inconvenientes que enfrentan las Bases de Datos relacionales son los siguientes:

- No son lo suficientemente flexibles y por lo tanto no están diseñadas para posibles cambios.
- Poseen inconvenientes para manejar datos heterogéneos.
- No están optimizadas para realizar tareas operativas y de análisis, en ciertas ocasiones son ineficientes (por ejemplo, Big Data).
- No están preparadas o pensadas para el desarrollo de aplicaciones utilizadas globalmente por miles de usuarios. En la actualidad, el alcance de ciertas aplicaciones de software era impensado hace varias décadas atrás (redes sociales, plataformas de contenido, aplicaciones móviles, entre otras).

Sin embargo, estos problemas a los que se enfrentan las Bases de Datos relacionales no implican que dejarán de ser utilizadas. En realidad, estas problemáticas han contribuido con la necesidad de utilizar y combinar nuevas alternativas para el almacenamiento y tratamiento de los datos. En algunos de los sistemas y aplicaciones que deben afrontar la realidad de un mundo conectado, es más importante la flexibilidad, la velocidad de respuesta y la capacidad de escalar horizontalmente que otras cuestiones como la consistencia o disponer de una estructura

completamente definida para los datos. Surgen así, nuevas tecnologías como alternativa a los motores de Bases de Datos relacionales, éstas se denominan Bases de Datos no relacionales o Bases de Datos NoSQL [13, 18].

Si bien el término NoSQL tiene sus orígenes aproximadamente en el año 1998, recién una década después, hacia el año 2009, aparecen un conjunto de implementaciones de motores de Bases de Datos que no se basan en el modelo relacional. Las características comunes en estos motores son principalmente 3: ausencia de un esquema riguroso para los datos (schema-less), escalabilidad horizontal sencilla, y agilidad de respuesta.

La ausencia de esquema riguroso permite no tener que contar con un esquema rigurosamente definido para la representación de los atributos. La información que se almacena puede tener diferente cantidad de atributos, según la necesidad del usuario. De esta manera aumenta la claridad, se optimiza el espacio físico (sólo se almacena lo necesario) y el rendimiento (evitar uniones de tablas para obtener los datos) [20, 22].

La escalabilidad horizontal, hace referencia a que los motores de Bases de Datos no relacionales están pensados principalmente para facilitar esta escalabilidad. Generalmente, los motores de Bases de Datos relacionales escalan verticalmente, es decir, aumentando la capacidad de procesamiento de sus servidores. Por el contrario, los motores de Bases de Datos no relacionales están diseñados para aumentar la cantidad de nodos o servidores y no su potencial, es decir, escalar horizontalmente.

La tercera característica vinculada al tiempo de respuesta hace referencia a que en general, en los motores de Base de Datos no relacionales, las operaciones ocurren en memoria principal y persisten los datos cada cierto tiempo, lo que permite alta velocidad de respuesta al usuario final. Operar de esta manera, posee el riesgo de que puedan perderse operaciones de escritura o perder la consistencia. Esto se resuelve permitiendo que una operación de escritura se realice en más de un nodo antes de darla por válida, o disminuyendo el tiempo entre la actualización en memoria principal y su correspondiente persistencia en disco (consistencia eventual) [2, 6].

El objetivo de este trabajo final consiste en elaborar un análisis comparativo entre las diferentes categorías de almacenamiento no relacional de datos.

A partir de la siguiente sección se detallan los conceptos generales de Bases de Datos; en el capítulo 2 se describe la evolución tecnológica para el almacenamiento de información y las Bases de Datos y; en el capítulo 3 se explican distintos tipos para el almacenamiento de datos; en el capítulo 4 se profundiza en conceptos sobre el modelo relacional de datos y los Sistemas de Gestión de Bases de Bases de Datos Relacionales (SGBDR), que desde su postulación en el año 1970 por E. F. Codd [34], se ha consolidado como el tipo de almacenamiento de datos más utilizado del mercado [43]. Se continúa en el capítulo 5, donde se explican los conceptos fundamentales para el almacenamiento no estructurado de datos y se detallan los motores de Bases de Datos no relacionales estudiados. Finalmente, en el capítulo 6 se presentan los casos de estudio, y en el capítulo 7 el análisis de resultados obtenidos, conclusiones y trabajo futuro.

1.2 Conceptos generales de Bases de Datos

1.2.1 Bases de Datos

Una Base de Datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En ella, se representan aspectos del mundo real, aquellos que son de interés del usuario, es decir, cada dato almacenado tiene un propósito específico [34, 39].

La palabra dato hace referencia a una característica que puede registrarse y que no tiene capacidad de comunicar un significado, por lo tanto, no altera el comportamiento de quien lo recibe. La importancia de los datos radica en la capacidad de asociarse bajo un contexto para convertirse en información y poder ofrecer un significado, conocimiento, idea o conclusión. En una Base de Datos los datos almacenados se integran tratando de evitar redundancia, motivo por el cual los datos son compartidos por una o más aplicaciones [34].

Cada Base de Datos es diseñada y pensada para satisfacer una serie de requerimientos específicos y con el simple objetivo de integrar toda la información que sea de interés. Generalmente, existen cuatro perfiles o visiones de usuarios característicos que intervienen en el entorno de una Base de Datos. Los administradores, los diseñadores, los desarrolladores de las aplicaciones que necesitan de los datos contenidos en la Base de Datos y los usuarios finales que la utilizan con el objetivo para la que fue creada.

La administración hace referencia a la implementación y configuración de una Base de Datos. Cuestiones de seguridad, acceso concurrente y disponibilidad son características importantes que se deben cumplir para lograr un funcionamiento aceptable. El diseño corresponde a la abstracción de los datos y cómo éstos deben combinarse o relacionarse para llegar a un resultado razonable. En el proceso de diseño se deben tener en cuenta las restricciones de los datos y las reglas de negocio, como también tener la capacidad de integrar las opiniones del resto de los usuarios. Las aplicaciones son las que permiten interactuar a los usuarios finales con la Base de Datos, permitiendo consultar, insertar, modificar y eliminar datos [32, 37].

La inversión de tiempo y/o dinero para diseñar, crear y administrar una Base de Datos, se realiza con el objetivo de reemplazar los sistemas de archivos tradicionales, los cuales presentaban los siguientes inconvenientes:

- **Redundancia e inconsistencia de los datos**: el mismo dato puede ser almacenado más de una vez, lo que puede provocar inconsistencia. Es decir, el mismo dato con posibles valores distintos, dependiendo del archivo consultado.
- **Dificultad en el acceso a los datos:** a medida que los archivos aumentan su tamaño, se hace más lento el acceso a un dato.
- **Aislamiento de los datos:** datos dispersos en varios archivos y en distinto formato. Esto es costoso para las aplicaciones al momento de la recuperación del dato apropiado.

En los sistemas de archivos convencionales, además, existe una dificultad real para lograr restricciones de integridad, para lograr un acceso concurrente optimizado y para lograr seguridad en el acceso a la información. Todas estas dificultades e inconvenientes motivan el desarrollo e implementación de los Sistemas de Gestión de Bases de Datos (SGBD) [34, 36].

1.2.2 Sistema de Gestión de Bases de Datos (SGBD)

Un Sistema de Gestión de Bases de Datos (SGBD) es una aplicación con funcionalidades específicas para la creación y administración de Bases de Datos, que brinda una visión abstracta y se ocupa de cuestiones de implementación interna de los datos. De esta manera, un SGBD, convierte la administración de los datos y su gestión en una "caja negra" que se interpone entre el usuario y las Bases de Datos. Su propósito es lograr una definición y manipulación de forma sencilla y ordenada de los datos que se encuentran en una Base de Datos. La organización de SGBD se puede especificar básicamente en tres niveles de abstracción [24, 32]:

- **Un nivel bajo o interno:** donde se manipulan físicamente los datos. Es el SGBD quien se ocupa del almacenamiento de los datos liberando al usuario de esta tarea.
- **Un nivel alto o externo:** representa la visión del usuario final. Es la administración de la Base de Datos y la abstracción de su implementación interna. Se realizan múltiples tareas mediante opciones de menú, generalmente intuitivas y autoguiadas, no hay necesidad de tener conocimiento sobre la estructura o composición interna de los datos.
- **Un nivel intermedio o de conocimiento medio:** donde se describe a la Base de Datos mediante un lenguaje conceptual brindando una visión abstracta del sistema. Un conjunto de palabras o comandos que permiten efectuar diversas tareas de acceso, definición y manipulación de los datos, así como también tareas de mantenimiento y administración de la Base de Datos.

  A través de este tipo de lenguajes se pueden abordar tareas de creación, modificación o eliminación de cualquier elemento que integre la Base de Datos, crear relaciones, validaciones, privilegios de acceso y uso, entre otras.

Existen diversos aspectos y características inherentes a un SGBD que le permiten lograr una gestión adecuada de una Base de Datos. Entre ellas, se describen [38, 39]:

- **Abstracción de datos:** el usuario no tiene la necesidad de conocer el detalle físico del almacenamiento de los datos en una Base de Datos. Al momento de la creación de una Base de Datos, no es necesario conocer el número de archivos que se necesitan.
- **Flexibilidad e independencia:** la complejidad de las Bases de Datos y la necesidad de ir adaptándolas a los cambios del contexto, hacen que una característica importante de los SGBD sea la facilidad para adaptarse a estos posibles cambios. No es necesario conocer aspectos de su implementación física. De esta manera, se pueden realizar

cambios en aspectos físicos o aspectos tecnológicos y el usuario no se verá afectado. Esto se denomina, independencia física de los datos.

- **Consistencia de los datos:** una Base de Datos representa el fragmento de una realidad que posee determinadas condiciones (o reglas de negocio) y en ciertas ocasiones se necesita de redundancia de información. Será necesario contar con mecanismos o herramientas para mantener la consistencia de aquellos valores que aparecen representados más de una vez en la Base de Datos. Los SGBD suelen permitir la programación de ciertas situaciones bajo condiciones específicas para mantener la consistencia.
- **Seguridad sobre los datos:** los datos persistidos o almacenados, seguramente, sea el activo más valioso para quién ha elegido administrar su información en una Base de Datos. Se debe garantizar la seguridad y confidencialidad de los datos contenidos en una Base de Datos. Un SGBD posee mecanismos para definir autorizaciones o derecho de acceso en diferentes niveles. De esta manera, se puede prohibir el acceso frente a usuarios malintencionados o restringirlo para usuarios de menor experiencia. Además, se pueden definir políticas de copias de seguridad con el fin de restaurar la Base de Datos en caso de existir un problema que implique la pérdida de información.
- **Integridad de los datos:** es necesario tener mecanismos para garantizar la validez de los datos almacenados. Un SGBD permite definir reglas que protegen a los datos de posibles introducciones o alteraciones descuidadas que puedan corromper su veracidad. Si el SGBD detecta que una operación va en contra de las reglas que se han establecido, podrá ser rechazada. Estas reglas se denominan reglas de integridad del usuario. Existen otras reglas que son inherentes al modelo, es decir, que el SGBD siempre las tendrá, éstas se denominan reglas de integridad del modelo. Por ejemplo, un SGBD relacional nunca aceptará que una tabla posea filas duplicadas o un SGBD jerárquico no aceptará que una entidad sea definida como hija de dos entidades diferentes.
- **Control de concurrencia**: en cualquier contexto en el que se encuentre una Base de Datos, seguramente, una numerosa cantidad y variedad de perfiles de usuarios necesitarán acceder de forma simultánea con el objetivo de recuperar y manipular datos. Un SGBD permite coordinar y sincronizar estos accesos de modo de que no se produzcan inconsistencias. Cuando los accesos concurrentes son solamente para recuperar datos con el solo fin de consulta (solo lecturas), el único problema esperable es el rendimiento causado por las limitaciones de los soportes tecnológicos que se disponen. Cuando los accesos requieren realizar cambios, pueden surgir problemas de interferencia, llevando a la obtención de datos erróneos. Motivo por el cual, es fundamental, que el SGBD posea mecanismos para monitorear estas situaciones.
- **Manejo de transacciones:** una transacción es un conjunto de acciones, ejecutadas como una unidad lógica de trabajo, que producen un cambio consistente en la Base de Datos, es decir, que no se corrompe ninguna regla de integridad que se haya establecido previamente o que derive del propio modelo. Un SGBD posee mecanismos para llevar

adelante determinadas acciones para aquellos casos en que las transacciones no se comporten de manera esperada.

- **Tiempo de respuesta**: un SGBD minimiza u optimiza las solicitudes de los usuarios con el objetivo de responder en el menor tiempo posible.

Un SGBD provee facilidades para gestionar un gran volumen de datos, maneja políticas de respaldo, para salvaguardar cualquier problema de pérdida de información. Poseen reglas de consistencia y cuentan con mecanismos para administrar el acceso simultáneo, evitando así, conflictos o alteraciones no permitidas sobre un dato. Logran un manejo centralizado de la seguridad mediante la restricción en el acceso, de esta manera, protegen los datos contra accesos malintencionados y otorgan permisos dependiendo del tipo de usuario que acceda.

Además, la centralización de la información permite compartir los datos entre distintas aplicaciones de usuario, ahorrando de esta manera, espacio de almacenamiento y la confianza de que todos trabajan con la misma versión de la información [18, 19].

A pesar de estas ventajas, existen cuestiones o aspectos importantes que deben ser atendidos y que pueden ser considerados un costo o desventaja, algunos de estos aspectos son:

- **Complejidad:** un SGBD es un software grande y complejo que posee gran cantidad de funcionalidades. Es importante comprender toda su funcionalidad para obtener todo su potencial.
- **Consumo de recursos:** un SGBD puede llegar a requerir una cantidad considerable de memoria. Los requerimientos que exige un SGBD deben ser considerados.
- **Costo económico:** en este aspecto, existe una gran variedad de alternativas. Algunos SGBD poseen licencia dual, es decir, una versión libre o gratuita y otra versión, con determinadas funcionalidades extras, pero con un costo económico. Otros SGBD, son completamente pagos o completamente libres u “open source”.
- **Equipamiento adicional:** la implantación de un SGBD puede requerir equipamiento adicional, desde la adquisición de grandes equipos informáticos, exclusivos para el SGBD, hasta el acondicionamiento de un lugar físico adecuado para su funcionamiento.
- **Capacitación:** en algunas ocasiones, incorporar un SGBD, requiere de la capacitación y preparación del personal. Inicialmente, suele haber cierta resistencia al cambio y este es un desafío que no debe ser pasado por alto. No obstante, esto no debería ser un impedimento para mejorar la infraestructura del almacenamiento de los datos.
- **Prestaciones:** un SGBD puede ser accedido por múltiples aplicaciones con diversos objetivos. En ciertas ocasiones, son sometidos a una gran demanda de operaciones, algo que puede derivar en que los tiempos de respuesta no sean óptimos.
- **Vulnerable a disponibilidad:** generalmente la información se encuentra centralizada en el SGBD y el sistema es vulnerable ante un fallo que pueda producirse. En la actualidad, muchos de los SGBD del mercado poseen mecanismos o estrategias para salvaguardar la disponibilidad de la información ante algún inconveniente.

1.2.3 Arquitectura de un SGBD

Existen tres características inherentes a un SGBD:

- La separación entre las aplicaciones y sus datos.
- El manejo de múltiples vistas por parte de los usuarios.
- El uso de un catálogo para almacenar el esquema de la Base de Datos.

En el año 1975, el comité *ANSI-SPARC* (*American National Standard Institute - Standards Planning and Requirements Committee*) propuso una arquitectura de tres niveles para un SGBD, la cual resulta muy útil a la hora de conseguir estas tres características. Esta arquitectura está basada en tres niveles o esquemas: el nivel interno, el nivel externo (o de usuario) y el nivel conceptual [40].

En la figura 1 se representa gráficamente la arquitectura basada en tres niveles.

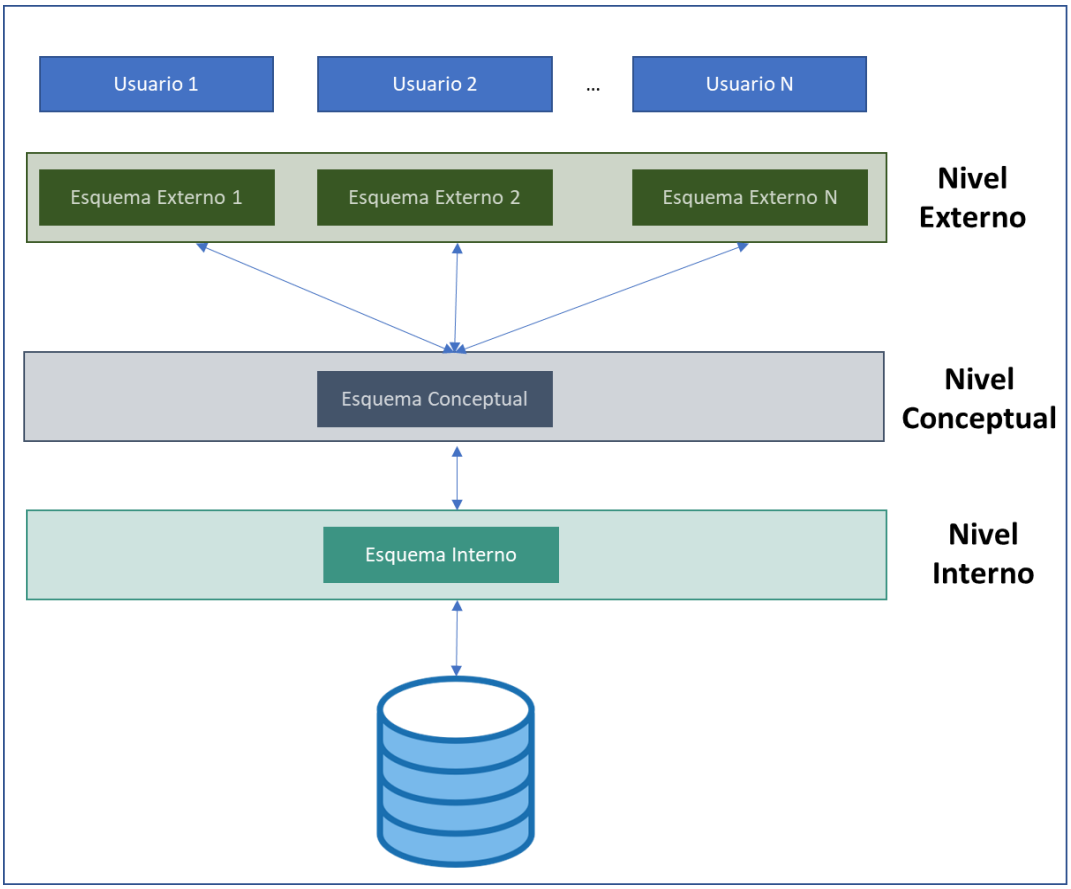

Figura 1. Arquitectura basada en tres niveles.

Así mismo, se describió las interacciones entre estos tres niveles y todos los elementos que conforman cada uno de ellos, estas son:

- **Nivel interno:** contiene la descripción de la organización física de la Base de Datos. Formas de accesos a los datos (índices, hashing, apuntadores, etc.), codificación de los datos, gestión del espacio, tamaño de página, etc. Los únicos datos que existen se encuentran en este nivel.

- **Nivel externo:** contiene distintas vistas de usuario. Cada esquema describe la visión que tiene de la Base de Datos un grupo de usuarios, ocultándose del resto. En este nivel se sitúan diferentes visiones lógicas que los procesos externos tendrán de la Base de Datos.
- **Nivel conceptual:** define un esquema conceptual. Describe una única estructura lógica y global de la Base de Datos ocultando los aspectos físicos de almacenamiento. Se representa con elementos conceptuales, como entidades o conjuntos, atributos o propiedades y las relaciones definidas entre estos conceptos.

El objetivo central de esta arquitectura en tres niveles es separar las aplicaciones de usuarios de la información almacenada físicamente. Sin embargo, no siempre se logra. En algunos SGBD se incluyen detalles del nivel físico en el esquema conceptual y cuando se manejan vistas de usuarios, en los esquemas externos se utilizan modelos similares a los conceptuales.

Los tres esquemas son solamente descripciones de los mismos datos, pero en distinto nivel de abstracción. Los datos solo existen en el nivel físico, almacenados en algún dispositivo de memoria secundaria. Un SGBD debe ser capaz de transformar la solicitud del usuario, expresada en términos de un esquema externo, a un esquema conceptual, y luego, expresarla en el esquema interno, que posteriormente se procesa en la Base de Datos [32, 34].

La arquitectura de tres niveles proporciona dos tipos de independencia respecto a los datos, la lógica y la física.

- **Independencia lógica respecto de los datos**: capacidad de alterar el esquema conceptual sin tener que alterar los esquemas externos ni las aplicaciones de usuario.
- **Independencia física respecto de los datos:** capacidad de modificar el esquema interno sin tener que alterar el esquema conceptual o los externos. Se refiere a la separación entre las aplicaciones de usuario y las estructuras físicas de almacenamiento.

A continuación, se detallan algunos componentes importantes, que, por lo general, conforman la arquitectura de un SGBD [18, 39]:

- **Gestor de archivos:** un SGBD realiza su propia gestión de archivos manteniendo información sobre los mismos. Se encarga de administrar los espacios de memoria temporal de entrada y salida entre la memoria secundaria y la memoria primaria. También se encarga de gestionar el espacio en la memoria secundaria, añadiendo o quitando bloques en los archivos conforme a las necesidades.
- **Gestor de definición y manipulación de datos:** ayuda al SGBD a simplificar y facilitar el acceso a los datos. Es habitual que un SGBD brinde la posibilidad de contar con un lenguaje de alto nivel (SQL, CQL, OQL, etc.) que le permite al usuario escribir determinadas sentencias para la definición y manipulación de los datos en una Base de Datos. Estas sentencias, al estar expresadas en un lenguaje de alto nivel necesitan ser analizadas, optimizadas y con ellas generar un plan de ejecución eficiente para el SGBD. Para ello, este gestor posee distintos componentes:

    - **El intérprete del Lenguaje de Definición de Datos (DDL):** interpreta las instrucciones que corresponden a la definición de los datos de una Base de Datos y las registra en un diccionario de datos interno.
    - **El compilador del Lenguaje de Manipulación de Datos (DML):** traduce las sentencias que manipulan los datos a nuevas sentencias de bajo nivel, más eficientes y que entiende el motor de evaluación de consultas.
    - **Motor de evaluación de consultas:** lleva adelante el plan de ejecución de sentencias de bajo nivel que han sido generadas por el DML.

- **Gestor de transacciones:** garantiza llevar adelante las solicitudes de acceso a los datos, bloqueando y liberando los mismos según las necesidades. En entornos concurrentes, un SGBD debe ser capaz de soportar múltiples peticiones de forma simultánea, esto requiere de una coordinación y planificación constante para no perder información. Realizar una buena administración de bloqueos y liberación de los datos, es fundamental para el éxito de las transacciones que ocurren en un SGBD.
- **Gestor de recuperación:** un SGBD es capaz de monitorear la concurrencia llevando un riguroso control sobre las peticiones que realizan los usuarios. Se lleva un registro minucioso de todos los cambios que van sucediendo sobre los datos en la Base de Datos, con el fin de poder recuperar cualquier información ante cualquier tipo de fallo. Este gestor es el encargado de mantener el registro de cambios y restablecer el sistema a un estado consistente ante cualquier problema.
- **Gestor de seguridad:** un SGBD permite establecer niveles de privilegios a los usuarios y se encarga de restringir el acceso completo o parcial a una Base de Datos.

1.2.4 Escalabilidad

La escalabilidad es la habilidad que tiene un sistema de adaptarse a un crecimiento de carga sin perder calidad en los servicios ofrecidos. Generalmente, la escalabilidad se realiza hacia arriba, es decir, aumentando el tamaño y la potencia del sistema; pero también es posible realizarla hacia abajo, utilizando solo los recursos necesarios en situaciones de baja demanda. Su efectividad debe responder a los requisitos específicos para los que sería útil dentro del proyecto. Existen 3 dimensiones principales al hablar de escalabilidad, estas son [22, 26]:

- **Escalabilidad en carga**: se refiere a la facilidad de ampliar o reducir los recursos para que el sistema acepte diferentes tipos de cargas según su necesidad.

- **Escalabilidad geográfica**: se refiere a que un sistema es geográficamente escalable cuando mantiene íntegra su utilidad y usabilidad sin considerar la distancia de los usuarios y los recursos.
- **Escalabilidad administrativa**: facilidad de usabilidad y manejo de un solo sistema distribuido por varios actores.

Existen dos maneras de escalar un sistema, escalamiento horizontal y escalamiento vertical. A continuación, detallan ambos escalamientos [4, 13].

- **Escalamiento vertical**: es el incremento de recursos en un solo nodo, ya sea añadiendo memoria, capacidad de procesamiento, etc. Su crecimiento depende de la ampliación o cambio del hardware por una versión adicional o más potente. La figura 2 representa gráficamente el escalamiento vertical.

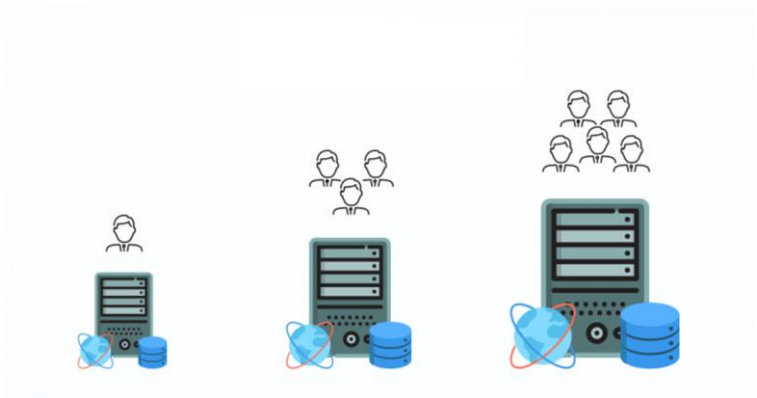

Figura 2. Escalamiento vertical.

Este tipo de escalamiento no tiene mayor incidencia en el rendimiento ya que con el tiempo, el almacenamiento y otras capacidades vuelven a demandar ampliación y renovación. Este escalamiento puede ser realizado cada cierto tiempo para ir pasos delante de los límites lógicos que impone el hardware e ir incorporando nuevas tecnologías.

El escalamiento horizontal se produce sobre el hardware, lo que hace que las aplicaciones no se vean alteradas. Además, es sencillo de implementar con un plan de migración ordenado. Puede ser realizado de forma rápida en términos de tiempo.

También conlleva una serie de cuestiones importantes que se deben tener en cuenta. Esta técnica de escalabilidad tiene un crecimiento limitado, impuesto por el hardware. Si ocurre un fallo, significa una interrupción total de las aplicaciones y además, seguramente represente un alto costo de inversión en equipos.

- **Escalamiento Horizontal**: consiste en el incremento de los nodos en un sistema, de tal manera que, al aumentar la demanda de procesamiento, se añaden más nodos al sistema, sin tener que ampliar los recursos de estos. La figura 3 representa gráficamente el escalamiento horizontal.

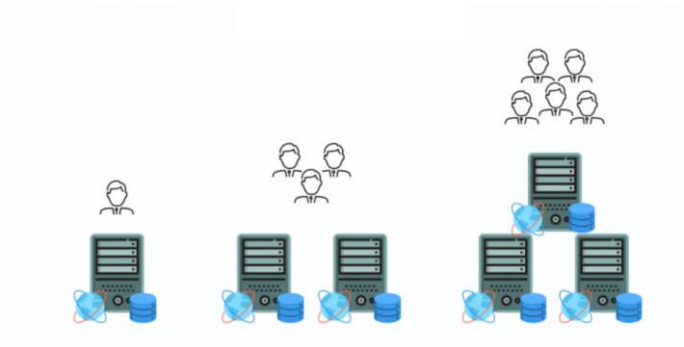

Figura 3. Escalamiento horizontal.

Cuando el rendimiento del sistema se ve afectado por el incremento de usuarios u operaciones, se amplían los nodos para equilibrar la demanda de trabajo en cada nodo. El concepto de escalabilidad horizontal está basado en la idea de que, cuando la cantidad de datos en un nodo y las peticiones hechas a este crecen linealmente, el tiempo de respuesta de cada petición crece exponencialmente.

La escalabilidad horizontal es una característica importante en una Base de Datos. Esto se debe a que el crecimiento es sumamente extenso y técnicamente no tiene límites. Es posible añadir nodos en la medida que se necesiten.

En el caso de las Bases de Datos, la escalabilidad horizontal se ha de llevar a cabo separando la información en diferentes instancias o nodos. La replicación de la información es importante cuando se administran grandes volúmenes de datos. Uno de los objetivos principales de la replicación es obtener el mejor cuidado y provecho de los datos, es decir, que permita solucionar problemas tales como:

- Distribución de los datos en varios servidores.
- Mantener la seguridad e integridad de los datos.
- Alta disponibilidad y tolerancia a fallos.

Capítulo 2

2.1 Historia de Bases de Datos

2.1.1 Evolución de los medios de almacenamiento

A lo largo de la historia, siempre ha existido interés por mantener y resguardar información a través del tiempo, desde la antigüedad en donde existían grandes bibliotecas y otros tipos de registros, hasta la actualidad que con el avance de la tecnología se puede mantener la información digitalizada de millones de usuarios. Cada día, la tecnología se vuelve más accesible, hay cada vez más usuarios y, por ende, se generan cada vez más datos.

El concepto de almacenamiento de datos relacionado con un tratamiento automático de la información se remonta al siglo XIX (1884) con la máquina tabuladora electromagnética (ideada por Herman Hollerith [44]). Esta máquina procesaba una cinta de papel o tarjeta perforada, con el objetivo de colaborar en el tratamiento de la información. En la figura 4, se presenta una imagen correspondiente a una de las versiones de la máquina tabuladora electromagnética.

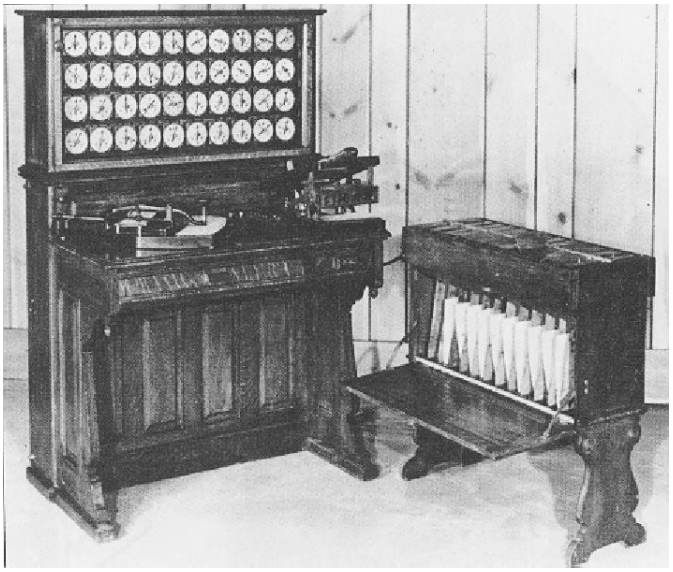

Figura 4. Máquina tabuladora electromagnética.

Durante años, las tarjetas perforadas fueron un medio importante en el ingreso y almacenamiento de datos, su procesamiento se fue perfeccionando y se diseñaron equipos electrónicos complejos capaces de procesar miles de tarjetas diariamente [39].

Para mediados del siglo XX, se comienza a aplicar una nueva tecnología para el almacenamiento de datos, las cintas magnéticas. En estas, se graba y lee información de forma secuencial sobre un soporte magnético. Las cintas magnéticas cubrieron por muchos años la demanda del mercado industrial en el procesamiento secuencial de información [37].

La figura 5 presenta una imagen en la que se pueden ver distintos formatos en los que se tiene almacenamiento en cinta magnética.

Figura 5. Dispositivos de almacenamiento en cintas magnéticas.

Posteriormente, surgen los discos magnéticos como una nueva alternativa en el almacenamiento de datos. Este es un dispositivo electrónico de acceso aleatorio que permite almacenar una gran cantidad de información. La figura 6 presenta una imagen en la que se pueden ver distintos formatos que poseen almacenamiento magnético.

Figura 6. Dispositivos de almacenamiento magnético.

El disco magnético aparece, por primera vez en una computadora de la firma IBM (IBM RAMAC 305), en el año 1956. En la figura 7 se presenta una foto de dicha computadora.

Con el pasar de los años, fue evolucionando su performance, aumentando la capacidad de almacenamiento, mejorando la velocidad en los tiempos de respuesta y siendo cada vez más pequeño en cuanto a su tamaño físico. Comúnmente es conocido como el “disco duro” y se

transformó en el dispositivo de almacenamiento no volátil indispensable en casi todas las computadoras [37, 39. 50].

Figura 7. IBM RAMAC 305.

Actualmente, se está empleando una nueva tecnología para el almacenamiento no volátil de datos, el Almacenamiento de Estado Sólido (SSD, por sus siglas en inglés, Solid State Drive). Es un método de almacenamiento creado mediante dispositivos de circuitos integrados que no utiliza soportes magnéticos. En la figura 8 se presenta una foto de algunos de los formatos en los que se puede presentar este tipo de almacenamiento.

Figura 8. Dispositivos de almacenamiento de Estado Sólido (SSD).

Seguramente en un futuro cercano sean estos dispositivos los que ocupen el lugar de los discos magnéticos. Por el momento, ambas tecnologías pueden coexistir, brindando cada una sus ventajas. Un disco económico que puede almacenar una gran cantidad de información, pero algo lento (disco magnético) y/o un disco mucho más rápido, de menor capacidad y a un costo económico más alto (SSD) [21, 22].

En esta constante evolución tecnológica, ninguna de las tecnologías ha sido reemplazada completamente de un momento a otro, siempre han existido procesos de transición en los cuales, algunas metodologías van quedando obsoletas.

2.1.2 Evolución de las Bases de Datos

Los predecesores de las Bases de Datos fueron los sistemas de archivos. Estos sistemas se conforman por un conjunto de archivos de datos y programas que permiten a los usuarios finales trabajar con ellos. Hasta hace algunas décadas atrás, el foco se centraba en la resolución de problemas particulares del mundo comercial y empresarial. Si se pensaba en automatizar información, se desarrollaba una aplicación específica para un sector o grupo cerrado de usuarios que trabajaban con la misma información. El objetivo principal consistía en bajar los costos de procesamiento de la información y había poca consideración con el almacenamiento de los datos. Generalmente, cada aplicación almacena y procesa sus datos bajo un formato y estructura específica, no estandarizada [37, 39].

Estos sistemas fueron adecuados mientras las aplicaciones y sus datos se podían operar de forma independizada, sin necesidad de compartir información. Este concepto cambió y fue necesario empezar a compartir información entre distintos sectores con sus propios datos, pero con distintos formatos. Inicialmente, este problema se afrontó exportando copias de los datos con el formato adecuado a la necesidad de cada sistema o aplicación. Esta tarea, generó inconvenientes, redundancia de la información, inconsistencia en los datos y problemas de seguridad.

En algunos proyectos era necesario procesar gran cantidad de información, pero no contaban con algún sistema adecuado para gestionar tal cantidad de datos. Estos proyectos se basaban en la división del trabajo en partes, en solucionar piezas más pequeñas, las cuales requerían menor cantidad de información y la unión de todas estas piezas, formaban la solución en su totalidad. Esta estructura se puede representar en forma de un árbol jerárquico, o lo que se denomina estructura jerárquica. A partir de aquí, se empezaron a utilizar estructuras jerárquicas de registros que se guardaban en dispositivos de almacenamiento en serie, más precisamente en cintas magnéticas. De esta manera surgen los sistemas de datos jerárquicos. Este modelo jerárquico carece de ciertas aptitudes para modelar y representar relaciones complejas entre los datos. En respuesta a las problemáticas existentes en los sistemas jerárquicos, surge un nuevo concepto, el sistema de red, el cual permitió representar múltiples relaciones entre los registros de información. Posteriormente, este sistema sirvió para definir un conjunto de especificaciones que permitieron la creación de Bases de Datos y la administración de sus datos [34, 37].

Ambos sistemas, jerárquico y de red, constituyen la primera generación de sistemas de gestión de Bases de Datos. Pero, a pesar de poder representar información compleja e interrelacionada, presentan algunas desventajas:

- Requieren de programas de aplicación complejos para responder a cualquier tipo de consulta de datos.
- La independencia de los datos es mínima.
- Una modificación en su estructura resulta compleja por su rigidez y exige un conocimiento sobre la forma en que se han almacenado los datos.
- Es necesario conocer las unidades de información y las relaciones que tienen estas entre sí (principalmente en el almacenamiento jerárquico).

En 1970 Edgar Frank Cood [45] presentó el artículo titulado: A Relational Model of data for Large Shared Data Banks (Un modelo relacional de datos para grandes bancos de datos compartidos) el cual dio origen al modelo relacional de datos. Las Bases de Datos relacionales, basadas en el modelo relacional, constituyen la segunda generación de sistemas para la gestión de Bases de Datos [37, 38].

El modelo relacional posee algunas debilidades e inconvenientes, por ejemplo, su limitada capacidad de modelar, con mayor valor semántico, objetos del mundo real o gestionar datos complejos, como contenido multimedia, sistemas gráficos o documentos. Para subsanar estas limitaciones, surgen algunas versiones "extendidas" para el modelo relacional de datos.

Para fines de 1980 y principios de 1990 la industria del software experimenta un importante crecimiento y junto a la programación estructurada surge un nuevo paradigma para el desarrollo de aplicaciones, la Programación Orientada a Objetos (POO). Los requerimientos y características de estas nuevas aplicaciones difieren en gran medida de las típicas aplicaciones de gestión. La estructura de un objeto es más compleja, las transacciones son de larga duración, se requiere de nuevos tipos de datos para almacenar imágenes, videos, textos, entre otros, y hace falta la definición de operaciones específicas [19, 22].

Los requerimientos de los sistemas de información se vuelven cada vez más complejos, se debe tener la capacidad de aplicar una gran cantidad de conceptos del mundo real y dar respuesta a un desarrollo más rápido y seguro. Con el objetivo de satisfacer las necesidades para estas nuevas aplicaciones, surge un nuevo tipo de Base de Datos, las Bases de Datos Orientadas a Objetos (DBOO) y las Bases de Datos Objeto-Relacional (DBOR), que son una extensión de las Bases de Datos Relacionales. Estos tipos de Bases de Datos constituyen la tercera generación en la evolución de las Bases de Datos.

El término Base de Datos Objeto-Relacional se utiliza para describir una Base de Datos que ha evolucionado desde el modelo relacional hasta una base de datos híbrida, es decir, que contiene ambas tecnologías, relacional y de objetos [30, 49].

Para fines del siglo XX y principios del siglo XXI, el impacto de los avances tecnológicos, la influencia de internet y las formas de comunicación han crecido de forma exponencial. Inicialmente, estos avances han contribuido en la gestión automatizada de información

distribuida en distintos puntos, lo que contrasta con la filosofía inicial del concepto de Base de Datos, la de mantener toda la información centralizada en un único repositorio de datos.

Surgen dos nuevos conceptos de Bases de Datos, las Bases de Datos Distribuidas (BDD) y las Bases de Datos Activas (BDA). Las primeras administran datos que pertenecen lógicamente a un solo sistema, pero se encuentran en nodos o servidores ubicados en lugares geográficamente distantes y que se conectan entre sí a través de una red de comunicación. Las segundas, satisfacen a aquellas aplicaciones que requieren de una respuesta inmediata en una situación puntual o crítica, sin esperar a la intervención del usuario. A diferencia de los sistemas pasivos, un sistema de Base de Datos activo debe responder ante determinados sucesos o reglas que el diseñador describe.

En general, estos nuevos paradigmas no son categorizados como una nueva generación, sino que son consideradas como nuevas extensiones de las ya existentes [30, 32].

Desde hace más de una década, el panorama de las Bases de Datos ha ido cambiando radicalmente. Los avances tecnológicos, el auge de la comunicación a través de internet y el desarrollo de dispositivos electrónicos, cada vez más pequeños y potentes que permiten compartir información con cualquier persona o entidad a través de aplicaciones de software mundialmente utilizadas, han cambiado la dinámica de la información. Actualmente, considerar a las Bases de Datos solamente como un repositorio de información, puede llevar a una administración incorrecta de los datos.

En respuesta a este avance, hace algunos años ha surgido un nuevo grupo o concepto de Bases de Datos denominado, Bases de Datos no Relacionales o Bases de Datos NoSQL (no solo SQL). Las Bases de Datos NoSQL responden a la evolución en el almacenamiento de la información, y no son exactamente un tipo de Bases de Datos, sino que son un conjunto de tipos de Bases de Datos [1, 18, 21].

No existe un momento exacto y concreto en que un sistema o paradigma de Bases de Datos ha dejado de ser utilizado para dar lugar a su sucesor, de hecho, es probable que actualmente existan usuarios que aún continúen utilizando sistemas de Bases de Datos antiguos, pero que funcionalmente son capaces de satisfacer sus necesidades y requerimientos de negocio.

Capítulo 3

3.1 Tipos de Bases de Datos

3.1.1 Bases de Datos Jerárquicas

Entre los primeros modelos de datos utilizados, se encuentran las estructuras jerárquicas de los árboles. La implementación del modelo jerárquico se lleva a cabo mediante el manejo de punteros o enlaces. Este modelo fue desarrollado para representar problemas en el que predominan las relaciones de uno a muchos y es un modelo muy rígido donde los diferentes conceptos de un determinado problema se organizan en múltiples niveles de acuerdo con la relación padre-hijo, es decir, que una entidad padre puede tener varios descendientes y una entidad hija tendrá una única entidad padre en el nivel superior inmediato.

La representación gráfica del modelo jerárquico se realiza mediante la estructura de un árbol invertido. La raíz determina el nivel superior y está ocupada por una única entidad, y a partir de ahí se van ramificando el resto de las entidades que se encuentran en los distintos niveles del árbol. Todo va quedando unido por las relaciones de padre-hijo. En la figura 9 se presenta gráficamente una estructura jerárquica [34, 36].

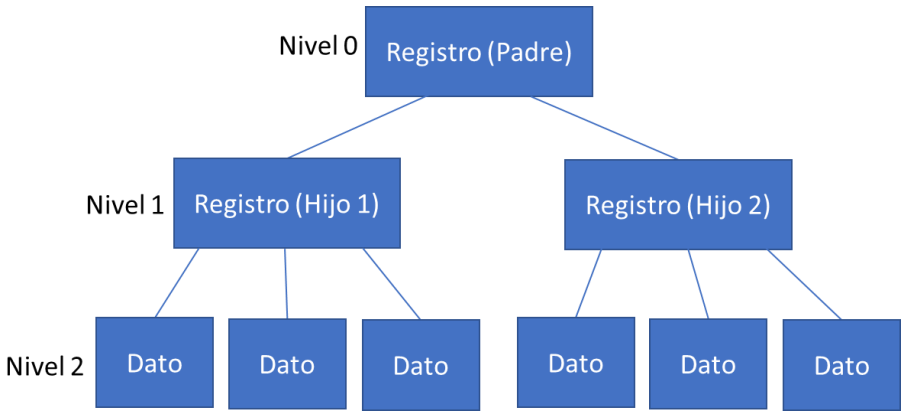

Figura 9. Representación gráfica de una estructura jerárquica.

En este tipo de Bases de Datos no hay diferencia entre la visión lógica y la visión física de la Base de Datos. Las relaciones se establecen siempre a nivel físico, es decir, mediante referencia a direcciones pertenecientes al medio de almacenamiento (sectores y pistas). Los datos se almacenan en forma de registros y cada registro posee un conjunto de campos.

El concepto de entidad, en el modelo jerárquico, se denomina “segmento” o “nodo” y todos los nodos que pertenecen a un mismo nivel dependen de un único nivel inmediato superior. De acuerdo con su ubicación en la estructura, un nodo puede ser:

- **Nodo raíz:** ocupa el nivel superior y es la entrada a la estructura.
- **Nodo hijo**: dependen de un nivel superior.
- **Nodo padre**: posee nodos dependientes.

La estructura en forma de árbol jerárquico determina que la búsqueda o el acceso a los datos se realiza a través de un camino único que siempre inicia en el nodo raíz, avanzando por los distintos nodos, de arriba hacia abajo y de izquierda a derecha [34, 37].

La estructura jerárquica posee las siguientes características:

- Se organiza en un conjunto de niveles.
- El nodo raíz, es el más alto en la estructura jerárquica y corresponde al nivel 0.
- Un nodo puede brindar acceso a un número ilimitado de otros nodos de nivel inferior que se denominan nodos “hijos” o nodos “descendientes”.
- Cada nodo, excepto la raíz, se corresponde con un solo nodo de nivel superior que se denomina nodo “padre”.
- Se denominan nodos “hojas” a todos los nodos que no poseen descendientes (nodos hijos).

Además de estas características, también existe una terminología asociada a una estructura jerárquica:

- **Altura**: es el número de niveles de la estructura.
- **Momento**: número de nodos de la estructura en determinado instante.
- **Peso**: número de nodos que no poseen descendientes (nodos hojas).
- **Subárbol**: conjunto de nodos que se obtienen a partir de un nodo descendiente (o hijo).

El modelo jerárquico posee restricciones propias de una estructura jerárquica de árboles, estas restricciones son:

- Solo se tiene un único punto de entrada a la estructura, la raíz.
- Solo se permiten relaciones uno a uno, o uno a muchos. No se permiten relaciones de muchos a muchos.
- No existen relaciones reflexivas.
- Para cualquier acceso a la información almacenada, es obligatorio el acceso desde la raíz.
- La estructura se debe recorrer siempre de acuerdo con un orden prefijado.
- Una vez creada la estructura no es posible su modificación.

Un ejemplo de este tipo de Base de Datos es la IBM Information Management System (IMS). IBM junto las empresas Rockwell y Caterpillar diseñó IMS en 1966 dentro del programa Apollo. El desafío era almacenar un extenso inventario proveniente de una lista de materiales del cohete

lunar Saturno V y de la nave Apolo. IMS apareció en un terminal IBM 2740 en Downey, California un 14 de agosto de 1968 y aún, en la actualidad, continúa vigente. Además, IMS soporta aplicaciones desarrolladas en Java, JDBC, XML y Servicios Web [42, 70].

El modelo jerárquico, debido a su rigidez, presenta inconvenientes o limitaciones, debido a que no implementa ningún control sobre sus datos. Es responsabilidad de las aplicaciones garantizar que se cumplen las condiciones invariantes que se requieran (por ejemplo, evitar la duplicidad de registros). Representar relaciones de muchos a muchos, o situaciones que no responden a una jerarquía obliga a introducir redundancia de los datos, con el riesgo de caer en una redundancia no controlada. El mantenimiento de estas Bases de Datos requiere de una operatoria costosa, debido a las restricciones inherentes del modelo. Toda inserción, excepto el nodo raíz, requiere de un nodo padre, y se deben considerar los accesos en la búsqueda de este nodo. Además, el modelo no ofrece la posibilidad de representar o definir restricciones de usuario [36, 37].

3.1.2 Bases de Datos en Red

El modelo de Bases de Datos basado en red representa las entidades y relaciones como nodos y aristas en una estructura de grafos. Las relaciones existentes entre los nodos quedan expresadas en términos de las aristas existentes entre éstos. En la figura 10 se presenta gráficamente una estructura de red.

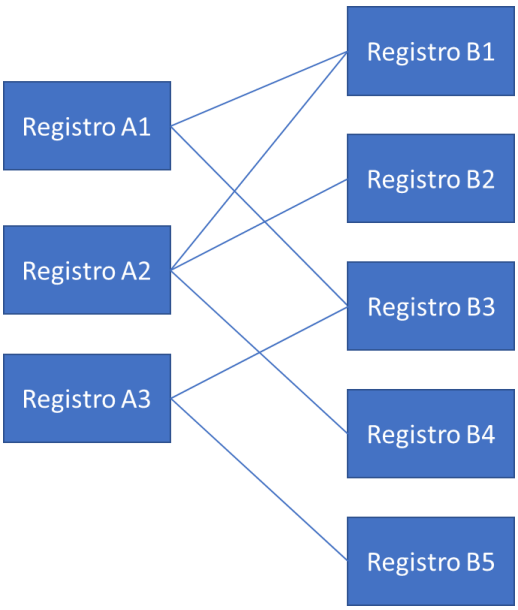

Figura 10. Representación gráfica del modelo de red.

No existen restricciones en el número de aristas que pueden existir entre los nodos, lo que supone que se puede llegar a modelar estructuras de datos complejas. Las Bases de Datos en red se conforman por una componente estática y una componente dinámica. La componente estática está representada por los objetos que componen el grafo, los nodos, los atributos, las

aristas y las restricciones. La componente dinámica, está relacionada con la navegación o los caminos que se recorren dentro del grafo [32, 38].

El modelo de red constituye una variación sobre el modelo jerárquico. Este modelo se construye sobre el concepto de múltiples ramas (estructuras de nivel inferior) que emanan de uno o varios nodos (estructuras de nivel alto). Estas Bases de Datos resultan más flexibles que las Bases de Datos jerárquicas, esto se debe a la ausencia de las restricciones inherentes al modelo. En este caso, es posible representar relaciones de todo tipo entre los nodos, uno a muchos, muchos a muchos o reflexivas. No obstante, la ausencia de restricciones hace que sea difícil al momento de su implementación física y en ciertas ocasiones puede ser ineficiente. Las Bases de Datos basadas en el modelo de red necesitan añadir una serie de restricciones con el fin de poder realizar una implementación a nivel físico y obtener el mejor rendimiento del sistema [34, 36].

El modelo de red surge en 1969 en una publicación estándar de la Conferencia de Lenguajes en Sistemas de Datos o CODASYL (Conference on Data Systems Languages). Un modelo de datos perteneciente a este paradigma es el modelo CODASYL (1969). Este es un modelo simplificado del modelo de red original que solo admite cierto tipo de relaciones y posee un conjunto de restricciones adicionales con el objetivo realizar una implementación eficiente sin limitar la flexibilidad del modelo original [38, 39].

Ambos enfoques, jerárquico y de red, utilizan punteros físicos para las estructuras de las Bases de Datos, es decir, direcciones de disco para relacionar los registros de distintos archivos. Para vincular dos registros, en uno de ellos hay que añadir la dirección del otro. Este tipo de organización limita en gran medida las operaciones que los usuarios pueden realizar con los datos y son vulnerables frente a cambios físicos. Por ejemplo, si los datos cambian su ubicación física se requerirá de un proceso de actualización (a nivel físico) para que los registros hagan referencia a la nueva posición.

3.1.3 Bases de Datos Relacionales

El modo de ver y entender a las Bases de Datos cambió radicalmente cuando surge el modelo relacional. Este modelo de datos se transformó en el modelo de datos más popular y exitoso para los proveedores de Sistemas de Gestión de Bases de Datos (SGBD) comerciales. Surgen así, los Sistemas de Gestión de Bases de Datos Relacionales, basados en el modelo relacional (SGBDR). Su popularidad se basa en la sencillez de la representación lógica de sus datos y en el lenguaje de definición y manipulación de datos, basado en álgebra de conjuntos y el cálculo de predicados, el cual, actualmente es un estándar y probablemente es el más reconocido y utilizado en el mundo de las Bases de Datos. Este lenguaje se denomina SQL (por sus siglas en inglés, Structured Query Language).

El principal objetivo del modelo de datos relacional es que la Base de Datos sea percibida como una estructura lógica con un conjunto de relaciones y no como una estructura meramente física

de implementación. De esta manera, se logra un alto grado de independencia de los datos. Además, esta percepción de estructura lógica de la Base de Datos debe ser simple y uniforme, utilizando conceptos que se interpretan de una única manera y sean de forma atómica [34, 37, 39].

El modelo relacional posee los siguientes objetivos principales:

- **Independencia física:** la forma en que los datos son almacenados no debe influir en su manipulación lógica.
- **Independencia lógica**: la modificación de los elementos de la Base de Datos no afecta a las aplicaciones que interactúen con ella.
- **Flexibilidad**: distintas visiones según la necesidad de los usuarios y aplicaciones.
- **Uniformidad**: las estructuras lógicas siempre tienen la forma conceptual de una tabla.

Una Base de Datos Relacional emplea a nivel físico archivos, pero la visión lógica son tablas, logrando una implementación transparente y uniforme para el usuario de la Base de Datos. En la figura 11 se presenta gráficamente la estructura tabular de una Base de Datos relacional.

	Columna 1	Columna 2	Columna 3	Columna 4
Fila 1	Valor	Valor	Valor	Valor
Fila 2	Valor	Valor	Valor	Valor
Fila 3	Valor	Valor	Valor	Valor
Fila 4	Valor	Valor	Valor	Valor

Figura 11. Representación lógica de una tabla en una Base de Datos relacional.

El modelo relacional posee un conjunto de conceptos importantes a tener presente, esto son [34, 36]:

- **Relación:** una relación es una tabla con columnas y filas. Una relación se utiliza para almacenar información sobre los objetos del mundo real que se representan en la Base de Datos. De esta manera se brinda al usuario una visión lógica de la Base de Datos (un conjunto de tablas). Cada fila de la tabla corresponde a un registro individual y contiene un conjunto de campos o “atributos” que poseen una coherencia lógica entre sí. El atributo representa el nombre de una columna de la relación [34, 36]. Las relaciones o tablas del modelo relacional poseen las siguientes propiedades o características:
    - Poseen un nombre único.
    - Los valores de los atributos son atómicos, es decir, en cada tupla o fila cada atributo posee un único valor.
    - No hay dos atributos con el mismo nombre.
    - No hay orden para la definición de los atributos.

- No hay tuplas o filas duplicadas.
- Las tuplas o filas no se encuentran ordenadas.

En este modelo, los datos, se almacenan en forma de valores explícitos, es decir, no existen punteros, sino que a través de estos valores se puede establecer un vínculo.

- **Dominio:** el dominio es un conjunto de valores atómicos o indivisibles. Son las unidades semánticas de datos más pequeñas y simples que no poseen una estructura interna. Los dominios conforman una poderosa característica del modelo relacional. Cada atributo de una Base de Datos relacional se define sobre un dominio, pudiendo existir varios atributos definidos sobre un mismo dominio. El concepto de dominio le permite al usuario definir el significado y la fuente de valores que un atributo puede tomar. Esto hace que exista más información disponible para el sistema al momento de realizar alguna operatoria, evitando así, operaciones semánticamente incorrectas [34, 37].

- **Clave Primaria**: en una relación o tabla, no pueden existir dos filas que posean el mismo valor en todas sus columnas, debe existir al menos un valor que las diferencie. Se denomina clave primaria al atributo, o conjunto de atributos, que satisface las siguientes propiedades:

  - **Única**: no existen dos filas en la relación que posean el mismo valor.
  - **Mínima**: no existe un subconjunto que posea la propiedad de único.

  Puede ocurrir que en una relación exista más de un atributo (o conjunto de atributos) que cumplan con las propiedades anteriormente mencionadas. Estos se denominan claves candidatas.

- **Clave Foránea:** en el modelo relacional existen conexiones o vínculos entre las filas de distintas relaciones e incluso con filas que pertenecen a la misma relación. El concepto proporcionado por el modelo relacional para expresar estos vínculos se denomina clave foránea. Una clave foránea es un atributo (o conjunto de atributos) que permite vincular filas entre relaciones. El valor que contiene una clave foránea se debe corresponder con un valor de clave primaria existente en una relación del modelo o bien, aún no poseer valor definido (nula).

- **Atributos Opcionales (o nulos)**: al momento de agregar o modificar una fila en una relación, puede suceder que el valor para ciertos atributos aún no se encuentre definido, estos valores se denominan nulos. Estos valores poseen un sentido semántico en la relación, pero no siempre deben tener definido un valor del dominio [34, 36].

- **Reglas o restricciones de integridad**: el modelo relacional establece que los datos contenidos en las relaciones deben cumplir con ciertas reglas para garantizar su correctitud. Estas reglas son:

    - **Integridad de dominio**: cuando se define un atributo, hay que definir su dominio, es decir, el conjunto de valores posibles que dicho atributo podrá tomar. El atributo no puede contener un valor por fuera de su dominio.
    - **Integridad de clave primaria:** esta regla establece que ninguno de los atributos que componen la clave primaria puede contener un valor nulo.
    - **Integridad referencial:** esta regla establece que, si una relación contiene una clave foránea, su valor debe coincidir con el valor de clave primaria con el cual se establece el vínculo, o bien, contener un valor nulo.
    - **Restricción o regla de negocio:** estas reglas no están definidas para el modelo relacional y dependen de SGBDR. A través de estas reglas se pueden establecer ciertas restricciones que son estratégicas y específicas para el funcionamiento del modelo.

Las Bases de Datos Relacionales han sido capaces de satisfacer con éxito las necesidades, en cuanto al almacenamiento de información, de las aplicaciones de gestión tradicionales. Sin embargo, a principios de la década del 90, surgen aplicaciones más complejas, como el diseño en Ingeniería, experimentos científicos, sistemas geográficos o sistemas multimedia. Las Bases de Datos tradicionales empiezan a mostrar deficiencias ante estos nuevos requerimientos y se tiene la necesidad de representar objetos de la vida real más complejos. Se necesitan nuevos tipos de datos para almacenar imágenes y textos, y, además, se necesitan operaciones no estándar y transacciones de larga duración.

3.1.4 Bases de Datos Orientadas a Objetos

En respuesta a esta problemática y a la popularidad adquirida de los lenguajes basados en la Programación Orientados a Objetos, surge un nuevo paradigma de Bases de Datos, las Bases de Datos Orientadas a Objetos (BDOO) basadas en el Modelo Orientado a Objetos. Este nuevo paradigma ofrece flexibilidad para la administración de los requerimientos de las nuevas aplicaciones y no se encuentra limitado por los tipos de datos y los lenguajes de consulta. Posee gran potencial, debido a que le proporciona al diseñador la posibilidad de especificar, tanto la estructura de los objetos, como el comportamiento u operatoria permitida para estos. Además, las BDOO han sido diseñadas para ser integradas fácilmente con los lenguajes Orientados a Objetos. En la figura 12, se representa un ejemplo de una Base de Datos Orientada a Objetos utilizando un Diagrama de Clases UML (Unified Modeling Language) [30, 36, 38, 41].

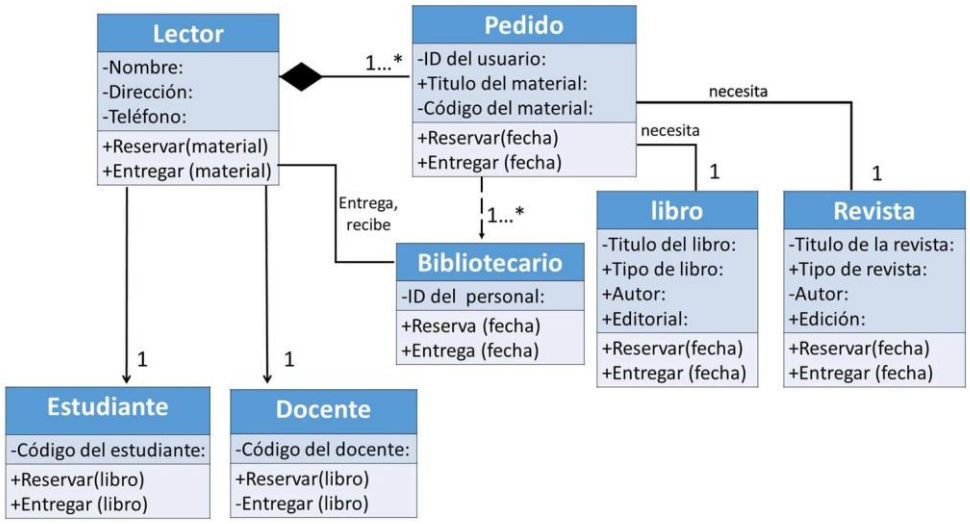

Figura 12. Representación de una Base de Datos Orientada Objetos mediante la utilización de un Diagrama de Clases que representa un sistema bibliotecario [41].

El modelo Orientado a Objetos ha aportado un gran cambio en el modo de ver los datos y la forma de tratarlos. En los paradigmas tradicionales, los datos y el comportamiento se almacenan de forma separada, en cambio en el paradigma de objetos se combinan en una misma entidad. Esta combinación es una ventaja, las entidades son unidades autocontenidas fáciles de reutilizar y su comportamiento no se encuentra ligado al programa de aplicación, por lo que su comportamiento es predecible y conocido. Las Bases de Datos Orientadas a Objetos, no han reemplazado a las Bases de Datos Relacionales, sino que se han adoptado como una alternativa para ciertas aplicaciones complejas [32, 36, 38].

En una BDOO las relaciones entre los objetos se representan mediante la inclusión del identificador del objeto con el cual se quiere vincular. Un identificador es un atributo interno que posee cada objeto, este atributo no es manipulable directamente, su valor los asigna y administra el sistema. Para que estas relaciones funcionen, los identificadores de los objetos deben corresponderse en ambos extremos de la relación. Este es un tipo de integridad de relaciones similar a la integridad referencial de las Bases de Datos relacionales, en donde se gestiona la verificación especificando relaciones inversas. De la misma forma que en las Bases de Datos relacionales se deben especificar las reglas de integridad, en las BDOO se deben especificar las relaciones inversas indicando el rol de cada relación, es decir, el significado o correspondencia de cada atributo para ambos extremos del vínculo [30, 34, 38].

Existen dos aspectos importantes a destacar en las relaciones en una BDOO:

- El identificador de un objeto no debe cambiar mientras pertenezca a la Base de Datos.
- Una BDOO es navegacional, como el modelo jerárquico y de red. Las únicas relaciones que se pueden utilizar en las consultas son las que se han definido previamente. Se limita la flexibilidad, pero los accesos presentan mejores prestaciones que las Bases de Datos

relacionales, debido a que es más rápido seguir el camino de los identificadores de los objetos que realizar operaciones entre cruces de tablas.

Una característica de las BDOO es la posibilidad de representar atributos multivaluados. De esta manera una relación de muchos a muchos no requiere de entidades intermedias. Cada participante de la relación contendrá un atributo con el conjunto de valores del otro participante con el que se relaciona. No obstante, se debe tener en cuenta que, si la relación a representar posee sus propios datos, entonces en ese caso se necesitará una entidad intermedia, como sucede en una Base de Datos relacional.

Capítulo 4

4.1 Bases de Datos Relacionales (BDR)

4.1.1 Introducción

En este capítulo se detallan algunos aspectos importantes sobre las Bases de Datos relacionales, las cuales, desde hace varias décadas, se mantienen como la tecnología predominante del mercado para el almacenamiento estructurado de datos. A pesar del paso del tiempo y el surgimiento de nuevas tecnologías para el almacenamiento de datos, el modelo relacional de Bases de Datos se mantiene vigente y los motores de Bases de Datos relacionales se ubican entre los más populares del mercado. Gran parte de su éxito en el mercado se debe a que brindan soporte tanto para la definición de los datos que se almacenan como para su manipulación posterior mediante la operatoria que brinda el modelo relacional de datos [18, 43].

4.1.2 Definición y generalidades

Una Base de Datos relacional se basa en el modelo relacional que ha sido explicado en el capítulo 3. El modelo relacional representa la Base de Datos como una colección de “relaciones”. La visión lógica de una relación es una “tabla” de valores. El nombre de la tabla y el de sus columnas, generalmente permiten interpretar el significado de los valores para cada fila y además, cada fila representa un hecho que normalmente se corresponde con una entidad u objeto del mundo real. En la terminología del modelo relacional, una fila se denomina “tupla”, la cabecera de la columna se llama “atributo” y la tabla se denomina “relación”.

Los datos almacenados en una Base de Datos relacional pueden agruparse en conjuntos que poseen la misma naturaleza. El conjunto de todos los datos de un mismo tipo se denomina “dominio” y los dominios representan conjuntos finitos de datos. Cada columna de una relación posee un tipo de dato asociado y la cantidad de columnas se denomina grado de la relación [30, 34, 39].

Matemáticamente, los elementos de un conjunto no están ordenados, por lo tanto, las tuplas de una relación no tienen un orden específico. Cuando se almacenan los datos físicamente, siempre existe un orden entre ellos, que es el orden en que fueron almacenados. De manera similar, cuando se presentan los datos de la relación, las tuplas se muestran en un cierto orden. Este orden no es parte de la definición formal de una relación, esto se debe a que la relación es una representación lógica o abstracta, pero sí es posible establecer diferentes órdenes lógicos sobre los distintos atributos de esta [34, 38].

Para que una tabla pueda considerarse relación de una Base de Datos relacional tiene que cumplir los siguientes requisitos:

- Debe existir un único valor en cada elemento de las filas (intersección fila y columna).
- Los valores de todos los elementos de una columna deben pertenecer al mismo dominio.
- No puede haber dos filas con todos sus valores iguales
- El nombre de cada columna (atributo) debe ser único dentro de la tabla.
- El valor de los elementos dentro de cada columna debe ser independiente del orden de las columnas.
- El valor de los elementos de las filas debe ser independiente del orden de las filas.

Como todas las tuplas de una tabla se tienen que diferenciar en al menos un valor para cualquiera de sus atributos, pueden identificarse por uno o varios campos. A este conjunto de campos se le llama “Clave Primaria”, que puede ser simple, si la constituye un solo atributo, o compuesta, si la constituyen varios. Por cuestiones de eficiencia, el grupo de campos que componen la clave primaria debe ser el mínimo posible. Obviamente una clave primaria nunca puede estar vacía (valor nulo) [34, 36].

Además, en una Base de Datos Relacional se pueden realizar operaciones utilizando un lenguaje denominado: Lenguaje de Consulta Estructurado o SQL (por sus siglas en inglés, Structured Query Language). SQL, es un lenguaje declarativo y permite especificar un conjunto de operaciones. Es capaz de combinar operaciones que se basan en álgebra y cálculo relacional con operaciones adicionales. Además, es un lenguaje definido por el estándar ISO/ANSI SQL que utilizan los principales fabricantes de Sistemas de Gestión de Bases de Datos Relacionales [30, 39].

Generalmente, en los lenguajes procedimentales se deben especificar todos los pasos que hay que dar para conseguir un resultado. Sin embargo, en SQL solo se debe indicar qué es lo que queremos obtener, y el sistema decidirá cómo obtenerlo. Es un lenguaje sencillo, potente y se utiliza en distintos niveles: usuarios, programadores y administradores de la Base de Datos. SQL se considera un lenguaje completo que permite crear y mantener objetos en una Base de Datos relacional. Además, de asegurar esos objetos y manipular la información dentro de estos. Las instrucciones que brinda SQL se dividen en tres categorías de acuerdo con las funciones que realizan, estas categorías son:

- Lenguaje de Definición de Datos (DDL, por sus siglas en inglés, Data Definition Language): las instrucciones DDL que pertenecen a este lenguaje se utilizan para crear, modificar o borrar objetos en una Base de Datos relacional, como tablas, vistas, esquemas, dominios, activadores, procedimientos, entre otros.
- Lenguaje de Control de Datos (DCL, por sus siglas en inglés, Data Control Language): las instrucciones DCL permiten controlar el acceso y permisos con respecto a objetos

específicos en la Base de Datos. Las instrucciones DCL también permiten controlar el tipo de acceso que cada usuario posee sobre la Base de Datos.

- Lenguaje de Manipulación de Datos (DML, por sus siglas en inglés, Data Manipulation Language): las instrucciones DML se utilizan para recuperar, agregar, modificar o borrar datos almacenados en la Base de Datos.

En algunos motores de Bases de Datos relacionales el lenguaje utilizado no es SQL puro. Cada proveedor de Base de Datos extiende su lenguaje con el fin de implementar algunas características propias para mejorar su funcionalidad. Además, algunas implementaciones de SQL son anteriores a que se transforme en un estándar. En consecuencia, cada proveedor posee una variación ligeramente diferente del estándar SQL [30, 34, 39].

4.1.3 Transacciones

Una Base de Datos es utilizada por diferentes tipos de usuarios y para distintos propósitos. Seguramente, gran cantidad de usuarios intentan acceder a los datos al mismo tiempo. Cuanto mayor sea el número de usuarios accediendo de forma simultánea a la Base de Datos, mayor será la probabilidad de que existan problemas cuando se intente ver o modificar los mismos datos en el mismo momento. Para evitar conflictos en los datos, SQL permite utilizar transacciones para controlar las acciones de los usuarios individuales. Una transacción es una unidad lógica de trabajo, compuesta por una o más instrucciones SQL, y que realizan un conjunto de acciones relacionadas [32, 34].

Si ocurre algún problema en cualquier momento durante la ejecución de una transacción, se regresa a su punto inicial y la Base de Datos regresa al estado en que se encontraba antes de que la transacción se inicie. Cualquier acción que haya modificado la Base de Datos se cancela y los datos se restauran a su estado original. En el caso de que la transacción se complete de forma exitosa, entonces todos los cambios son implantados. Durante todo el proceso, siempre se asegura la integridad de la Base de Datos [34, 37].

Para que un conjunto de instrucciones SQL sean consideradas como una transacción, debe cumplir con las propiedades de Atomicidad, Consistencia, Aislamiento y Durabilidad (ACID, por sus siglas en inglés, Atomicity, Consistency, Isolation, Durability). Estas propiedades se detallan a continuación.

4.1.4 Propiedades ACID (Atomicity, Consistency, Isolation, Durability)

En una Base de Datos relacional, el concepto de transacción representa una unidad lógica de procesamiento sobre la misma, la cual se debe ejecutar completamente para garantizar la correctitud de los datos. Debido a esto, es que las transacciones deben cumplir con las propiedades de ACID. En la figura 13 se presentan las propiedades ACID.

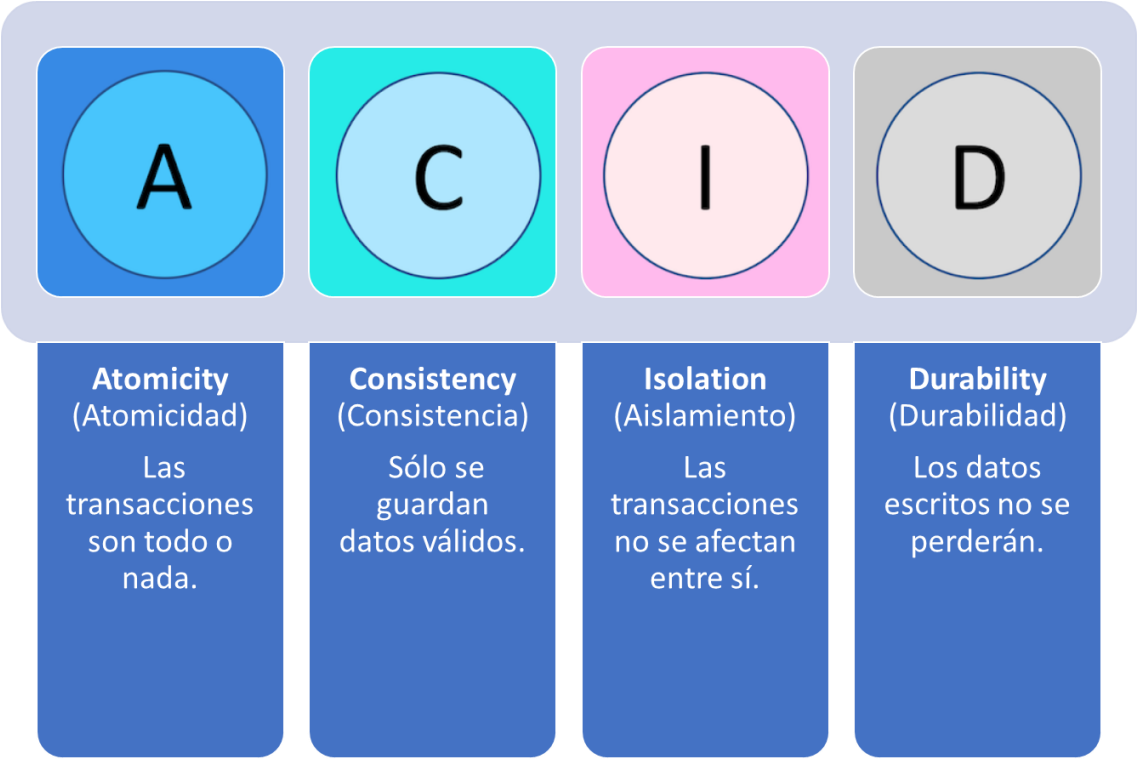

Figura 13. Propiedades ACID.

- **Atomicidad (Atomicity)**: cada transacción se opera como una unidad atómica de trabajo, es decir, se ejecuta completamente o ningún cambio afectará a la Base de Datos. El usuario no tiene el control cuando una transacción no puede completar su ejecución debido a un fallo. El SGBD relacional será el responsable de la recuperación ante algún fallo.
- **Consistencia (Consistency):** la ejecución completa de una transacción debe mantener la Base de Datos de forma consistente. Debe cumplir con las reglas de integridad definidas.
- **Aislamiento (Isolation):** cada transacción que ingrese al sistema se ejecutará de forma aislada, es decir, no se tiene en cuenta el efecto de otras transacciones que se encuentren ejecutando en ese instante. El SGBD relacional será responsable de intercambiar operaciones de las distintas transacciones con el objetivo de obtener mejores prestaciones.
- **Durabilidad (Durability):** cuando una transacción finaliza con éxito, los cambios realizados permanecerán, aún si el sistema falla.

4.1.5 Reglas de integridad

Una vez que es definida la estructura de los datos en la Base de Datos relacional, se deben establecer las reglas de integridad que los datos deberán cumplir para garantizar su correctitud. Cuando se define un atributo sobre un dominio se impone una restricción sobre el conjunto de valores que dicho atributo puede contener. Esto se denomina: restricción de dominio. En ciertas ocasiones, el valor de un atributo en una relación puede ser desconocido o no estar definido, en

este caso, se dice que el valor es nulo. El valor nulo representa ausencia de información y debe ser tratado de modo diferente [30, 39].

Existen dos reglas de integridad que son restricciones que se deben cumplir en una Base de Datos relacional:

- **Regla de integridad de entidades:** esta regla dice que ninguno de los atributos que componen la clave primaria puede ser nulo. Por definición, una clave primaria es irreducible, es decir, que ningún subconjunto de la clave primaria podría identificar de forma unívoca.
- **Reglas de integridad referencial:** esta regla se aplica a las claves foráneas. Si en una tabla se define una clave foránea, entonces, sus valores deben coincidir con valores de clave primaria a la que hace referencia o bien, deben ser nulos.

  Para respetar la regla de integridad referencial, existen tres alternativas para su definición:

  - **Regla de valores nulos**: se debe definir si la clave foránea permite valores nulos.
  - **Regla de eliminación (o modificación)**: se debe definir qué ocurre cuando se intenta eliminar (o modificar) el valor al que hace referencia una clave foránea. las posibilidades son:
    - **Restringir**: no es posible eliminar la fila o modificar el valor al que se está referenciando.
    - **Propagar**: en el caso de que se elimine o modifique la fila referenciada, la eliminación o modificación será propagada a las filas que la referencian.
    - **Anular**: en este caso, al eliminar o modificar la fila referenciada, los valores de las filas que la referencian se establecen a nulo. Esto es posible solamente si las claves foráneas fueron definidas para que acepte valores nulos.
    - **Valor por defecto**: en este caso, al eliminar o modificar la fila referenciada, los valores de las filas que la referencian se establecen en un valor por defecto previamente definido.

  Además, de la regla de integridad de entidades y de la regla de integridad referencial, es posible que sea necesario imponer ciertas restricciones específicas que definen la estrategia sobre el dominio del que forman parte los datos. Estás reglas se denominan: reglas de negocio. Estas reglas pueden involucrar varias tablas de la Base de Datos y su generación involucra descubrir sobre qué tablas serán aplicadas las acciones de forma autónoma para mantener la política que requiere el usuario final [36, 37].

4.1.6 Concepto de Normalización

Un concepto importante durante el diseño de una Base de Datos relacional es el concepto de normalización. La normalización es una técnica para diseñar la estructura lógica de los datos en el modelo relacional y este proceso también fue desarrollado por Codd en el año 1972 [34, 39, 45].
La normalización es una metodología de diseño que parte de una serie de atributos, propiedades o características de los datos, y éstos se van agrupando en relaciones o tablas según su afinidad. Los fundamentos de esta técnica es que un conjunto dado de tablas es reemplazado por otro conjunto de tablas con una estructura más simple con el objetivo principal de minimizar en lo posible la redundancia de información y evitar ciertas anomalías que pueden producirse en el uso y mantenimiento de la Base de Datos [34, 36].

Algunas de las ventajas principales que posee la normalización son:

- Evitar anomalías en la operatoria de la Base de Datos (altas, bajas y modificaciones).
- Mejora la independencia de los datos.
- No establece restricciones artificiales en la estructura de datos.

Para llevar a cabo el proceso de normalización existen reglas que se denomina formas normales. Las tres primeras formas normales estaban contempladas en la formulación original de la teoría relacional de Codd [45]. Después, se definió una versión modificada de la tercera forma normal (Boyce-Codd), y posteriormente fueron definidas la cuarta y la quinta forma normal originando un mayor refinamiento en el diseño de Bases de Datos. Cada paso en el proceso de normalización consiste en reducir sistemáticamente una relación o tabla en una colección de tablas más pequeñas (es decir, de menor grado) que son equivalentes a la de partida. Además, cada paso, posee correspondencia con una forma normal que satisface las reglas o propiedades determinadas. Conforme se avanza en este proceso, las nuevas tablas que se van generando tienen un formato más estricto y, por lo tanto, son menos vulnerables a los comportamientos anómalos que puedan darse en las operaciones de actualización de datos [36, 37, 38].

Un concepto fundamental para el proceso de normalización es el de dependencia funcional. Una dependencia funcional ocurre entre atributos de una misma tabla y se define de la siguiente manera [34, 36]:

- Si X e Y son atributos de la tabla R, se dice que Y es funcionalmente dependiente de X (se denota por X→Y) si cada valor de X tiene asociado un solo valor de Y sin ambigüedad. A X se le denomina determinante, ya que X determina el valor de Y. Se dice que el atributo Y es completamente dependiente de X si depende funcionalmente de X y no depende de ningún subconjunto de X.

Una dependencia funcional es una noción semántica. El reconocimiento de las dependencias funcionales es parte del proceso de entender lo que significan los datos en un contexto determinado. A continuación, se detallan las 5 principales formas normales [34]:

- **Primera Forma Normal (1FN)**: una tabla se encuentra en primera forma normal si, y sólo si, todos los dominios contienen valores atómicos.
- **Segunda Forma Normal (2FN)**: una tabla se encuentra en 2FN si, y sólo si, está en 1FN y, además, cada atributo que no es clave primaria depende completamente de la totalidad de la clave y no de un subconjunto de ella.
- **Tercera Forma Normal (3FN)**: una tabla se encuentra en 3FN si, y sólo si, ya se encuentra en 2FN y, además, cada atributo que no es clave primaria no depende transitivamente de la clave primaria.
- **Boyce-Codd Forma Normal (BCFN)**: una tabla está en la forma normal de Boyce-Codd si, y sólo si, todo determinante es una clave candidata.
- **Cuarta Forma Normal (4FN)**: una tabla está en cuarta forma normal (4FN) si cumple la 3FN y no tiene atributos multivaluados. Una dependencia multivaluada tiene lugar cuando el valor de un atributo determina un conjunto de valores múltiples.

La normalización es una técnica que se ha desarrollado para obtener estructuras de datos eficientes, garantizando un buen diseño lógico de la Base de Datos.

4.2 Motores de Bases de Datos Relacionales en el mercado actual

Actualmente existe una diversidad de motores de Bases de Datos relacionales. Hay motores de Bases de Datos relacionales que poseen Licencia GNU [71], la cual es ampliamente utilizada en el mundo del Software Libre y permite al usuario ejecutar, copiar, distribuir, estudiar, modificar o mejorar el producto con total libertad y sin costo económico alguno. Estos motores de Bases de Datos poseen la ventaja de que el usuario puede acceder a los detalles de implementación, de esta manera, se independiza del proveedor ya que la información es abierta y no existe una política específica de ocultamiento. En el caso de algún inconveniente u incompatibilidad en el funcionamiento del producto, el usuario posee la libertad de modificarlo o adaptarlo según sus necesidades, y, además puede distribuirlo con el objetivo de beneficiar al resto de los usuarios.

También, existen motores de Bases de Datos libres, pero que poseen funcionalidades adicionales a cambio de la adquisición de una licencia comercial, la cual posee un costo económico relativo al servicio ofrecido. Uno de los problemas principales de los motores de Bases de Datos libres suele ser el soporte técnico. Generalmente la información brindada no es clara, o ante una consulta, su respuesta suele demorar y en algunos casos es inexistente.

En el caso de los motores de Bases de Datos relacionales que poseen licencia comercial, el usuario debe adquirir la licencia, que implica un costo económico, para poder hacer uso de determinada versión y/o características del producto. Las inversiones en mejoras que realizan los proveedores en estos motores de Bases de Datos son notorias. Generalmente, incorporan

herramientas para pruebas de rendimiento, monitorización en tiempo real y diagnóstico, entre otras.

Además, estos motores de Bases de Datos suelen brindar soporte o asistencia constante ante cualquier problema, esto es una ventaja importante con respecto a otros motores de Bases de Datos gratuitos. Otro aspecto importante, es la seguridad. Generalmente, en los productos comerciales se mejoran los servicios de copias de seguridad o recuperación ante fallos importantes. Un motor de Base de Datos con licencia comercial, por su costo económico, debe responder a la exigencia del usuario afrontando ciertos desafíos como el rendimiento, administración y seguridad, entre otros.

En algunos casos, los motores de Bases de Datos con licencia comercial, para obtener sus mayores prestaciones o beneficios, requieren de configuraciones complejas que necesitan de una gran cantidad de recursos y en ciertas ocasiones pueden restringir su usabilidad a ciertas plataformas particulares.

En el mercado actual, algunos de los motores de Bases de Datos más populares que brindan almacenamiento relacional son [43]:

- **Oracle** [46]: es un Sistema de Gestión de Base de Datos (SGBD) que implementa almacenamiento relacional de datos. Es multiplataforma y puede desempeñarse en diferentes modelos de ejecución (cliente/servidor, distribuido, centralizado, entre otros). Sus limitaciones están determinadas por la plataforma en la cual se esté ejecutando. Responde a medianas y grandes demandas, por lo que cumple con las exigencias que necesita un SGBD. Actualmente, posee una infraestructura completa para trabajar en la nube denominada OCI (Oracle Cloud Infrastructure) [46].

  Numerosas empresas mundialmente conocidas se caracterizan por ser clientes y hacer uso de los servicios de Oracle, entre ellas podemos encontrar a FedEx [90], Spotify [80], Grupo BIMBO [91] y Gol Transportes Aéreos [92], entre otras.

- **MySQL** [47]: es el Sistema de Gestión de Bases de Datos relacional más extendido en la actualidad al estar basado en código abierto. Desarrollado originalmente por MySQL AB, fue adquirida por Sun MicroSystems en 2008 y esta su vez comprada por Oracle Corporation en 2010, la cual ya era dueña de un motor propio InnoDB para MySQL. MySQL es un sistema de gestión de Bases de Datos que cuenta con una doble licencia. Por una parte, es de código abierto, pero por otra, cuenta con una versión comercial gestionada por la compañía Oracle [46].

  Las versiones Enterprise, diseñadas para aquellas empresas que quieran incorporarlo en productos privativos, incluyen productos o servicios adicionales tales como herramientas de monitorización y asistencia técnica oficial. MySQL presenta la ventaja de estar basada en código abierto, lo que la hace muy interesante para algunos

desarrolladores, formando una ingente comunidad que brinda soporte a sus usuarios [47].

Este motor de Base de Datos es uno de los seleccionados para las experimentaciones que se presentan en este trabajo y será detallado en el siguiente.

- **Microsoft SQL Server** [48]: es uno de los principales Sistemas de Gestión de Bases de Datos relacional del mercado que presta servicio a un amplio abanico de aplicaciones de software destinadas a la inteligencia empresarial y análisis sobre entornos corporativos. Está basado en el lenguaje Transact-SQL, incorpora un conjunto de extensiones de programación propias de lenguaje estándar y su aplicación está disponible para usarse tanto a nivel on-premise o bajo una modalidad cloud. Algunas de las funciones principales que distinguen a Microsoft SQL Server, son su variedad de herramientas destinadas a la gestión y análisis de datos, así como la inteligencia empresarial con la que obtener conocimientos sobre tu negocio y clientes apoyadas en Machine Learning. Al estar basada en código abierto posee una gran comunidad que ofrece soporte a otros usuarios [48].

  Facebook [75], Uber [83] y Netflix [73], son algunas de las grandes empresas que hacen uso de Microsoft SQL Server. Sus ventajas lo posicionan como uno de los motores de Bases de Datos más utilizados del mercado.

- **PostgreSQL** [49]: es un sistema de código abierto de administración de Bases de Datos del tipo relacional, aunque también es posible ejecutar consultas que sean no relaciones. En este sistema, las consultas relacionales se basan en SQL, mientras que las no relacionales hacen uso de JSON. Su desarrollo es llevado adelante por una gran comunidad de colaboradores de todo el mundo que día a día brindan su aporte para hacer de este sistema una de las opciones más sólidas a nivel de Bases de Datos. Posee tipos de datos avanzados y permite ejecutar optimizaciones de rendimiento avanzadas, características que por lo general se ven en sistemas de Bases de Datos comerciales, como por ejemplo Oracle [46, 49].

  PostgreSQL nace a mediados de la década de 1980 a partir de Ingres, otro proyecto de Bases de Datos que tuvo su origen en la década anterior. La primera versión al público tuvo acceso limitado y estuvo disponible en el año 1989. El proyecto siguió creciendo y mejorando en los años anteriores, pero el equipo que lo llevaba adelante se separó en 1994. Dado que Postgres contaba con licencia libre el proyecto fue retomado y relanzado con soporte para SQL al año siguiente. Recibió su nombre actual en el año 1997 junto al lanzamiento de su versión 6.0. En los años siguientes, logró obtener una gran cantidad de usuarios que decidieron unirse al proyecto, dando origen a la gigantesca comunidad que hoy en día respalda a PostgreSQL [49].

Además, posee una serie de características que lo hacen apropiado para una amplia gama de aplicaciones, estas características se pueden resumir en:

- **Calidad del código**: cada línea de código que ingresa a PostgreSQL es revisada por varios expertos, y todo el proceso de desarrollo está impulsado por su comunidad, por lo que los informes de errores, las correcciones y la verificación se logran muy rápidamente.
- **Extensibilidad**: es flexible con extensiones que cubren las necesidades de uso. Si es necesario algo muy específico, se puede escribir una extensión propia o hacer que un proveedor de PostgreSQL lo haga.
- **SQL y NoSQL**: se puede utilizar como un sistema de gestión de Base de Datos relacional y/o como una solución NoSQL para almacenar documentos JSON.
- **Disponibilidad y resiliencia de datos**: para entornos de producción se brindan características adicionales de alta disponibilidad, resiliencia y seguridad.

Grandes empresas, en diferentes lugares del mundo, hacen uso de PostgreSQL. Algunos ejemplos son: Uber [83], Netflix [73], Instagram [84], entre otros.

- **IBM Db2** [50]: es una familia de productos de gestión de datos, incluyendo la Base de Datos relacional Db2. Esta Base de Datos proporciona una plataforma de datos para operaciones tanto transaccionales como analíticas, así como disponibilidad continua de datos para mantener los flujos de trabajo transaccionales y los análisis operando de manera eficiente.

  A partir de su versión 9, integra XML de manera nativa, lo que IBM ha denominado pureXML, esto permite almacenar documentos completos dentro del tipo de datos XML para realizar operaciones y búsquedas de manera jerárquica dentro de este, e integrarlo con búsquedas relacionales [50].

  Db2 ofrece funciones avanzadas para mejorar la gestión de datos, entre ellas:

  - Ahorro de espacio de almacenamiento sin sacrificar el rendimiento. Se pueden evaluar predicados de consulta sin tener que descomprimir los datos.
  - Posibilidad de tener una Base de Datos temporal para que los cambios del sistema y del negocio sean capturados, mantenidos y consultados.
  - Inteligencia Artificial (AI, por sus siglas en inglés, Artificial intelligence) y Aprendizaje Automático (ML, por sus siglas en inglés, Machine Learning) que ofrece capacidades de consulta en lenguaje natural, un optimizador de consultas ML y una plataforma de gestión de datos híbrida para permitir el intercambio fluido de datos estructurados, no estructurados y semiestructurados.
  - Elección del modelo de implementación, incluye implementación en la nube alojada en las instalaciones o la implementación en la nube administrada.

- Ofrece compatibilidad con Oracle SQL.
- Permite ejecutar cargas de trabajo mixtas con escalabilidad, alto rendimiento y disponibilidad permanente.

Además, de estos 5 motores de Bases de Datos relacionales, se pueden encontrar otros muy populares, como son: Microsoft Access [51], SQLite [52], MariaDB [53], Snowflake [54], entre otros. Actualmente, existe una lista de más de 150 motores que permiten almacenamiento relacional de datos [43].

Capítulo 5

5.1 Bases de Datos No Relacionales (NoSQL o No Solo SQL)

5.1.1 Introducción

Desde los inicios de la informática, el campo de las Bases de Datos ha sido un tema en constante estudio, cambio e innovación. Los Sistemas Gestores de Bases de Datos (SGBD), se han convertido en la pieza fundamental del manejo y administración de los datos que se generan y almacenan en cualquier sistema de información actual. Los SGBD relacionales abarcan adecuadamente gran parte de las necesidades en el almacenamiento de información. La variedad de productos existentes en el mercado, logran cubrir, en casi todas las ocasiones, las necesidades de almacenamiento, disponibilidad y consistencia de información. Sin embargo, desde hace más de una década, se viene experimentando un aumento en las problemáticas relacionadas con el almacenamiento digital referente a todo tipo de información, la cual es generada por millones de usuarios como consecuencia de actividades pertenecientes al ámbito personal y/o profesional [13, 20].

Desde el auge de Internet, el avance en las telecomunicaciones, el aumento en el uso de la tecnología móvil y el surgimiento de múltiples aplicaciones con diversas características (redes sociales, plataformas multimediales, plataformas de videoconferencias, etc.) ha cambiado el escenario de las necesidades de almacenamiento y procesamiento de información.

Esta nueva realidad, ha incrementado de manera exponencial el volumen de información digital que millones de usuarios generan a diario a través de la utilización de estas aplicaciones. La gran cantidad de información generada por aplicaciones que son mundialmente utilizadas han revelado ciertos inconvenientes que poseen los SGBD relacionales cuando son expuestos a un gran volumen de datos [13, 21].

Algunos de estos inconvenientes son: el tiempo de respuesta en los accesos, bloqueos innecesarios, dificultad para distribuir la Base de Datos geográficamente, necesidad de incorporar equipamiento cada vez más costoso, entre otros.

En la figura 14, se presentan algunos ejemplos del volumen de datos que ciertas aplicaciones utilizadas mundialmente deben administrar y procesar a cada minuto [55].

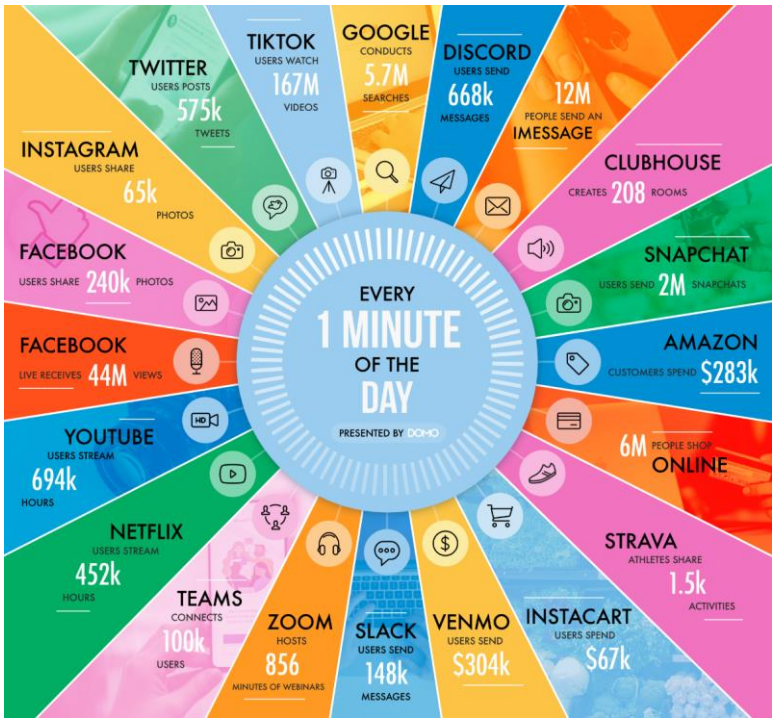

Figura 14. Los datos nunca duermen (Data never sleeps) [55]

Algunos valores para resaltar:

- 5.7 millones de búsquedas ocurren en Google [89] a cada minuto.
- 240 mil fotos comparten los usuarios de Facebook [75] a cada minuto.
- 65 mil fotos comparten los usuarios de Instagram [84] a cada minuto.
- 575 mil tweets en Twitter [76] a cada minuto.

Estos son algunos ejemplos de empresas y organizaciones que han tenido que afrontar esta problemática y se han visto en la necesidad de buscar alternativas para el almacenamiento digital de información con el objetivo de brindar mejores prestaciones.

Es así, que surgen un conjunto de soluciones como alternativa a los sistemas tradicionales de Bases de Datos relacionales. Estas nuevas alternativas plantean un cambio en el almacenamiento de la información digital y se conocen como Bases de Datos No Relacionales o NoSQL (No Solo SQL).

5.1.2 Bases de Datos NoSQL

El término NoSQL fue utilizado inicialmente a fines de la década de 1990 para hacer referencia a una Base de Datos Relacional que no utilizaba un Lenguaje de Consulta Estructurado (SQL) en su funcionamiento. Posteriormente, para el año 2009 vuelve a ser utilizado y se transforma en la principal alternativa a las Bases de Datos Relacionales. Esto se debe principalmente a que las necesidades del mercado, con respecto al almacenamiento digital de información, cambiaron radicalmente.

NoSQL en su origen, se inició como una combinación de dos palabras “No” y “SQL” para referenciar a las Bases de Datos que no utilizaban el modelo de datos normalizado de Bases de Datos relacionales y, por consiguiente, no aplican las mismas propiedades. En otras palabras, no organizan sus datos de forma estructurada (tablas, filas y columnas). Como su nombre sugiere no utilizan el lenguaje de consultas SQL para acceder a los datos, sino que utilizan lenguajes alternativos adecuados para sus modelos de datos [1, 13, 35].

Las Bases de Datos NoSQL surge ante la necesidad de tener Bases de Datos con capacidad de escalar en la Web y poder llegar a millones de usuarios de forma eficiente, así como a dispositivos móviles conectados desde diversos puntos geográficos.

Las Bases de Datos NoSQL han sido diseñadas para manipular grandes volúmenes de datos de manera muy rápida sin seguir el modelo de las Bases de Datos tradicionales. El término NoSQL se ha extendido cada vez más en los últimos años y ha pasado a ser un término más general, siendo No Sólo SQL (Not Only SQL).

Las clasificaciones de las Bases de Datos NoSQL no son rígidas y hay modelos que contemplan propiedades que abarcan más de una categoría de Bases de Datos. Actualmente, existe una diversidad de problemáticas con características distintas y es difícil pensar que un único tipo de Base de Datos sea el adecuado para aplicar en todas las situaciones posibles. Es necesario estudiar y evaluar la integración y/o combinación de distintos motores de Bases de Datos con el objetivo de satisfacer ciertos requerimientos, por ejemplo, alto rendimiento, escalabilidad horizontal a bajo costo, flexibilidad en los esquemas físicos, entre otros.

Su crecimiento y popularidad establecen una competencia con las Bases de Datos relacionales e incluso se ha alcanzado la integración de ambos Sistemas de Gestión de Bases de Datos. Las Bases de Datos NoSQL están diseñadas específicamente para manejar grandes volúmenes de datos en tiempo real y que requieren velocidad y facilidad para escalar. Además, estas Bases de Datos son importantes para la analítica de Big Data. Por esta razón son utilizadas por empresas mundialmente conocidas como, Google [89], Twitter [76], LinkedIn [76], Facebook [75], entre otras [2, 10, 18].

Entre las características principales de las Bases de Datos NoSQL, se pueden mencionar:

- Almacenamiento de gran cantidad de datos.
- Escalamiento lineal sin afectar el rendimiento.
- Acceso rápido.
- Distribución y manipulación de datos no estructurados.
- Funcionamiento con hardware estándar de bajo costo.

Algunos motores de Bases de Datos NoSQL no requieren de la definición de un esquema previo. La ausencia de este esquema establece que los datos no tienen una definición de atributos físicos, es decir, cada registro o documento puede contener información con diferente formato. Esto permite almacenar sólo aquellos atributos que interesen, facilitando el polimorfismo de

datos bajo una misma colección de información. Además, es posible almacenar estructuras de datos complejas en un solo documento, como puede ser el caso de almacenar la información de un blog (título, cuerpo del texto, fecha, autor/es, etc.) junto con los comentarios realizados a las diferentes entradas, y todo ello en un único registro [4, 12, 13].

La facilidad para escalar horizontalmente, le permite aumentar el rendimiento del sistema añadiendo, simplemente, más nodos, sin necesidad de realizar ninguna otra operación excepto indicar cuáles son los nodos disponibles.

La rapidez en respuesta a una consulta se logra porque muchos de los sistemas NoSQL realizan las operaciones directamente en memoria principal y sólo vuelcan los datos en disco cada cierto período. De esta forma, las operaciones de lectura y escritura son rápidas. Esta ventaja también posee riesgos por la durabilidad de los datos, ya que ante cualquier fallo se puede originar pérdida o inconsistencia de datos. Este riesgo se suele resolver permitiendo que una operación de escritura se realice en más de un nodo o bien disminuyendo el tiempo entre cada grabado de datos a disco.

Es importante remarcar algunos conceptos fundamentales antes de investigar y utilizar un motor de Base de Datos NoSQL. Estas Bases de Datos están pensadas para ser escalables y distribuidas. Debido a esto, es importante tener en cuenta el Teorema de CAP [13], el cual se detalla en la siguiente sección.

Además, existe una gran variedad de enfoques, requisitos y funcionalidades para las Bases de Datos NoSQL, lo que hace difícil mantener una visión general de todas ellas. NoSQL no es un tipo de Base de Datos, sino que es un conjunto de tipos de Bases de Datos en donde cada una presenta sus propias implementaciones [13, 18, 21, 29].

5.1.3 Teorema de CAP

El Teorema de CAP es el acrónimo para los siguientes conceptos:

- **Consistencia (Consistency):** garantiza que cualquier operación de lectura retornará la actualización más reciente de un registro dado. Cualquier cambio, debe reflejarse en todos los nodos existentes.
- **Disponibilidad (Availability):** especialmente alta disponibilidad. Esto significa que un sistema está diseñado e implementado para continuar operativo a pesar de que algún nodo falle o se encuentre fuera de servicio. Cuando se pierde comunicación, el sistema automáticamente debe tener la capacidad de seguir operando mientras la comunicación es restablecida y una vez que lo hace, deberá sincronizar con los demás nodos.
- **Tolerancia a la Partición (Partition Tolerance):** es la capacidad que posee el sistema para continuar la operación a pesar de que un nodo falle o quede aislado de la red.

También se entiende como la capacidad de agregar o sacar nodos dinámicamente y que el sistema no se vea afectado en cuanto a su operatoria.

El Teorema de CAP expone que un sistema que opera de forma distribuida no puede ponderar los tres conceptos mencionados previamente de forma simultánea (Consistencia, Disponibilidad y Tolerancia a Partición). Solamente se pueden satisfacer dos de ellas, relegando a un segundo plano la restante. Bajo estas restricciones, es importante considerar aquellos requerimientos que sean críticos para el sistema de información y sus reglas de negocio. Las Bases de Datos NoSQL siguen distintos métodos y poseen distintas características al momento de ser escalables y distribuidas, por lo cual no todas cumplen con los mismos puntos del Teorema de CAP [31, 33].

En la figura 15, se presenta gráficamente el Teorema de CAP y algunos ejemplos de motores de Bases de Datos según este teorema.

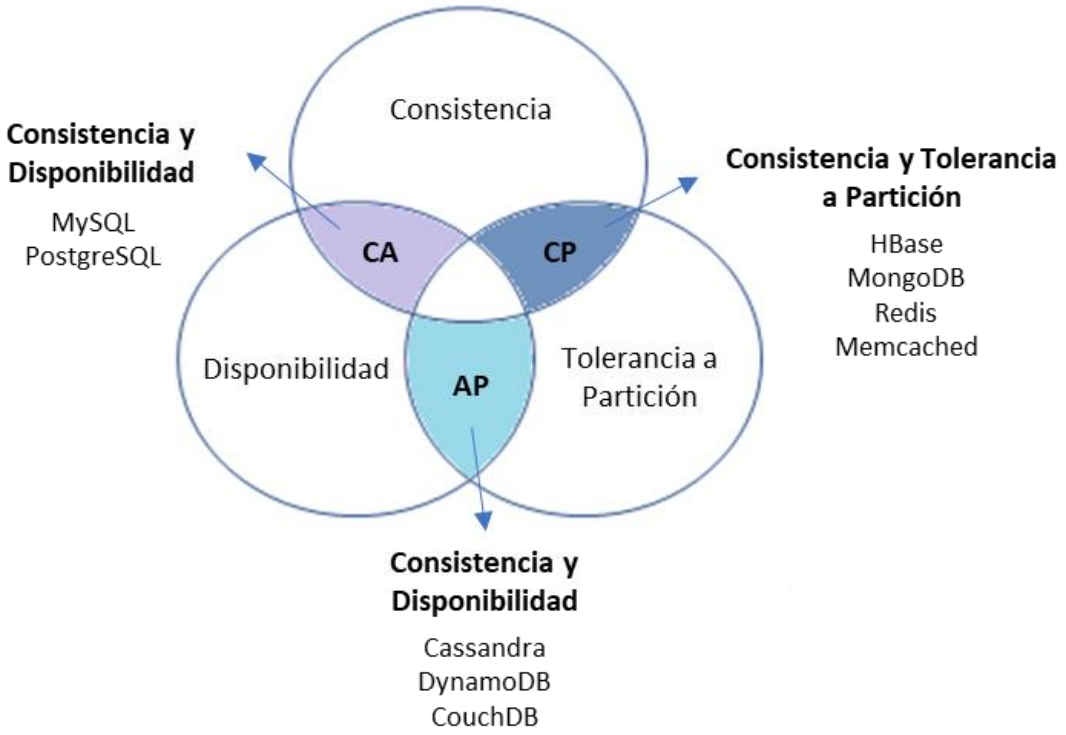

Figura 15. Teorema de CAP.

Las alternativas que surgen del Teorema de CAP pueden ser:

- **CA (Consistencia y Disponibilidad):** en este caso se sacrifica la tolerancia a partición. El sistema siempre estará disponible con información consistente. Debido a esto, no se admite la pérdida de comunicación entre los nodos (particiones de red). Al no soportar particionamiento, cuando hay un fallo de la red se presenta un mensaje para que se vuelva a intentar la operación en otro momento. Esto se debe a que no se está garantizada la consistencia en todos los nodos. Algunos de los motores de Bases de

Datos que hacen uso de esta alternativa son: MySQL [47], PostgreSQL [49], SQL Server [48], Neo4j [65], entre otros.

- **CP (Consistencia y Tolerancia a Partición):** en este caso no se asegura la disponibilidad. El sistema siempre garantizará consistencia, aunque existan problemas de conexión entre los nodos (particiones de red), pero no se asegura la disponibilidad constante. Al tolerar partición de red y priorizar la consistencia, es necesario que las solicitudes se apliquen en todos los nodos. En el caso de que algún nodo no se encuentre disponible las solicitudes no se podrán realizar. Algunos motores de Bases de Datos que hacen uso de esta alternativa son: MongoDB [59], Redis [56], HBase [63], entre otros.
- **AP (Disponibilidad y Tolerancia a la Partición):** En este caso se permiten presentar datos temporalmente inconsistentes. El sistema siempre estará disponible, aunque existan problemas de conexión entre los nodos (partición de red). Pueden existir datos temporalmente inconsistentes. Al tolerar partición de red, el sistema igualmente recibe y aplica una solicitud, como no se garantiza la consistencia, no es necesario confirmar el pedido en todos los nodos. Algunos de los motores de Bases de Datos que hacen uso de esta alternativa son: Cassandra [62], DynamoDB [57], CouchDB [93].

Esta clasificación no es definitiva, algunos motores de Bases de Datos poseen configuraciones que hacen cambiar su comportamiento por defecto y pueden pasar de una categoría a otra. En la elección de un motor de Bases de Datos es fundamental considerar el contexto y las ventajas y/o desventajas que cada uno ofrece.

A continuación, se presenta un ejemplo práctico en donde se aplican las distintas alternativas para el Teorema de CAP:

- Se considera un sistema distribuido compuesto por 3 nodos: A, B y C, ubicados geográficamente en distintos puntos y donde cada uno cuenta con una réplica exacta de la información. Estos nodos son los encargados de recibir las solicitudes, procesarlas y brindar las respuestas correspondientes. En un momento determinado, ante una petición de escritura el sistema falla debido a la pérdida de comunicación entre algunos de los nodos, quedando el sistema dividido en dos partes: {A, B} y {C}. A continuación, se explica que sucede en cada alternativa según el Teorema de CAP:

    - **Escenario 1 - El sistema garantiza la consistencia y es tolerante a particiones (CP)**: es necesario que la operación de escritura se confirme en los tres nodos para que todos queden consistentes. A pesar de que uno de los nodos ha quedado aislado, el sistema continúa operando debido a que se toleran particiones en la red, pero la operación será rechazada y se solicitará su reintento en otro momento. Hasta que los nodos no vuelvan a estar conectados nuevamente, no se podrá aplicar la operación de escritura, es decir, que el sistema no estará disponible.

- **Escenario 2 - El sistema garantiza la disponibilidad y es tolerante a particiones (AP):** la operación de escritura será confirmada. La actualización no se verá reflejada temporalmente en todos los nodos. En este caso, el sistema no garantiza la consistencia inmediata. La información procesada por un nodo puede no haber sido replicada al resto de los nodos de la red.
- **Escenario 3 - el sistema garantiza la disponibilidad y la consistencia (CA):** en este escenario no se admiten las particiones entre nodos. En ese caso la operación de escritura no será aplicada, debido a que no es posible garantizar la consistencia inmediata, pero, por otro lado, si la operación falla no se estaría garantizando la total disponibilidad, este caso es similar a CP. Este escenario es poco probable, ya que para que funcione se necesita que la red de comunicación entre los nodos sea perfecta. La forma de conseguir la tolerancia a partición es tener réplicas de nodos, de forma que, si un nodo no puede estar disponible, entonces hay otro capaz de tomar el relevo. Si se quiere tener consistencia inmediata y garantizar disponibilidad absoluta, se deben replicar los datos entre los nodos cada vez que se haga una operación de escritura.

5.1.4 Propiedades BASE (Basically Available, Soft State, Eventually Consistent)

Previamente, en el capítulo 4, se definieron las propiedades de ACID (Atomicity, Consistency, Isolation y Durability) para los Sistemas Gestores de Bases de Datos relacionales. Estas propiedades son necesarias para garantizar que un conjunto de instrucciones sea considerado una transacción en un entorno relacional. No obstante, poner en práctica estas propiedades no es una tarea sencilla, involucra un conjunto de acciones que consumen tiempo durante el procesamiento de la transacción.

Los Sistemas de Bases de Datos NoSQL, con el objetivo de garantizar ser escalables y distribuidos, proponen una visión más relajada o redefinida para las propiedades de ACID, sacrificando algunas características para obtener mejores prestaciones. NoSQL propone un modelo llamado BASE [13, 21, 23].

Estas propiedades proponen una visión más liviana y optimista del sistema, acepta que la consistencia en la Base de Datos se encuentre en un estado de flujo o que la disponibilidad se logre mediante el apoyo a fallos parciales sin necesidad de que el sistema completo se detenga. En la figura 16 se presentan las propiedades BASE.

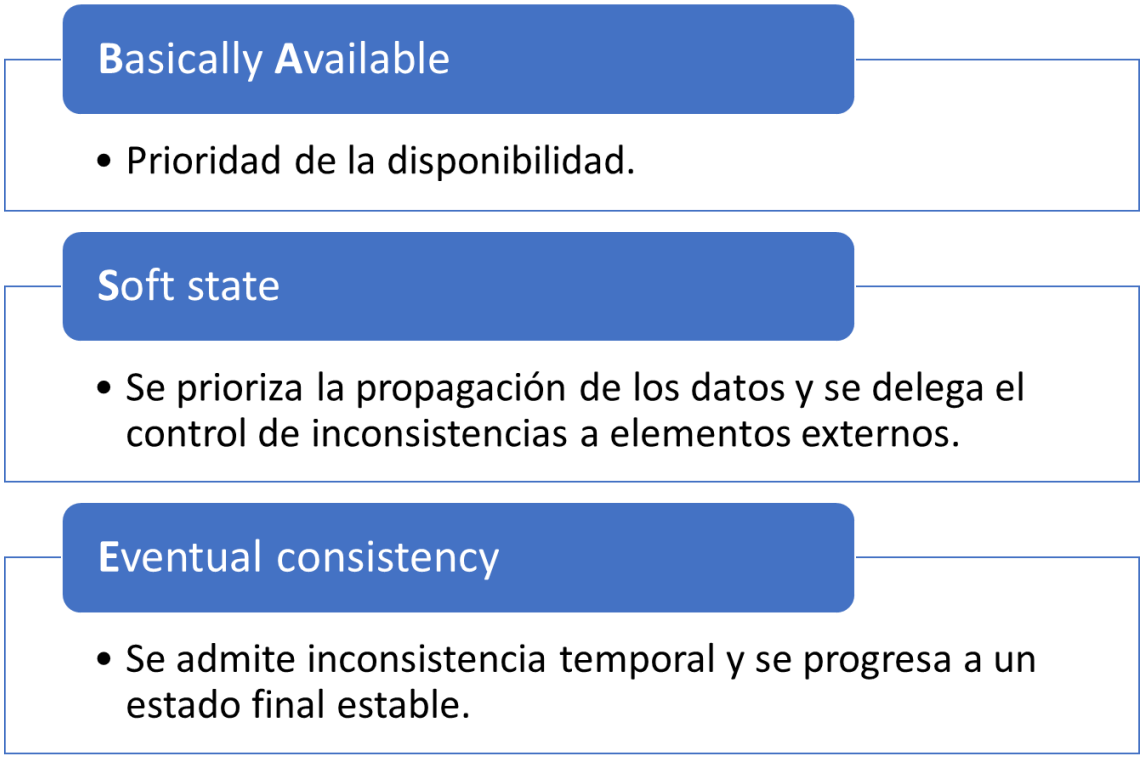

Figura 16. Propiedades BASE.

A continuación, se detallan las propiedades BASE (Basically Available, Soft State, Eventually Consistent):

- **Disponibilidad Básica (Base Availability)**: el sistema se encuentra disponible la mayoría del tiempo. Puede estar temporalmente inconsistente y habrá una respuesta ante cualquier solicitud, aunque podría ser una respuesta de que la operación no se ha podido realizar.
- **Estado Suave o Débil (Soft State)**: los datos en los diferentes nodos de la red no tienen que ser mutuamente consistentes en todo momento. Se acepta cierta inexactitud temporal. En momentos que no se ejecutan operaciones puede haber cambios debido a la "consistencia final".
- **Consistencia Eventual (Eventual Consistency)**: el sistema eventualmente se volverá consistente. En algún momento, los datos finalizarán su propagación a los nodos correspondientes. El sistema continuará recibiendo información y no verificará la consistencia para cada transacción antes de pasar a la siguiente.

5.1.5 Categorías de almacenamiento

Actualmente, existen una diversidad de productos NoSQL. Los más populares del mercado se pueden catalogar en una de las siguientes 4 categorías de almacenamiento de datos [3, 13, 43]:

- Almacenamiento Clave-Valor.
- Almacenamiento Documental.

- Almacenamiento en Familia de Columnas.
- Almacenamiento en Grafos.

5.1.5.1 Almacenamiento Clave-Valor

Las Bases de Datos que implementan este tipo de almacenamiento son vistas como una tabla de hash. Los usuarios agregan u obtienen elementos a través de una clave conocida [6, 19]. En la figura 17 se representa gráficamente este tipo de almacenamiento.

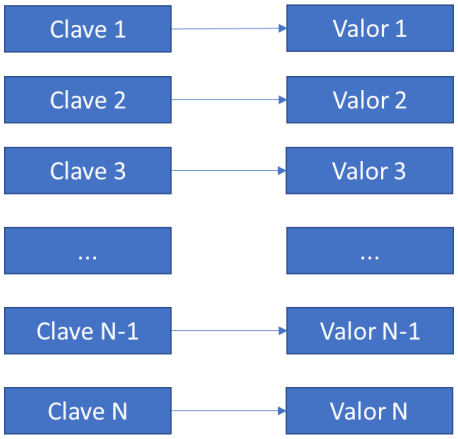

Figura 17. Almacenamiento Clave-Valor.

Las Bases de Datos que implementan almacenamiento clave-valor priorizan el alto rendimiento y la escalabilidad horizontal. Como desventaja, generalmente no se tiene consistencia inmediata, no hay control sobre la redundancia y no poseen referencias externas (integridad referencial). Es tarea de la aplicación realizar las actualizaciones correspondientes para mantener la correspondencia de referencias externas.

La operación de encontrar un valor asociado a una clave se denomina “lookup (indexación)”, y la relación entre una clave y su valor se denomina “correlación (o vinculante)”. La naturaleza poco estructurada de este tipo de almacenamiento hace que estas Bases de Datos sean propicias para lograr una buena escalabilidad horizontal en un ambiente distribuido, sin embargo, es complejo crear una llave única y representativa para cada registro [6, 56, 58].

Una Base de Datos clave-valor posee dos restricciones importantes:

- **Claves distintas:** nunca se van a poder almacenar dos elementos con la misma clave, en otras palabras, las claves deben ser únicas.
- **No hay operaciones de JOINs:** no se pueden realizar consultas sobre un rango de valores o alguna característica particular para elementos almacenados bajo distintas claves.

Algunas implementaciones de almacenamiento clave-valor realizan su operatoria principalmente en la memoria principal (RAM, por su sigla en inglés, Random Access Memory). De este modo, el tiempo de lectura y escritura es en el orden de los microsegundos. Además, en

algunos casos, suelen tener la opción de persistir los datos a disco con el objetivo de resguardar copias de seguridad [6].

Entre los motores de Bases de Datos más populares que implementan en este tipo de almacenamiento se encuentran Redis [56], Amazon DynamoDB [57] y Memcached [58], entre otros:

- **Redis** [56]**:** es el acrónimo de **RE**mote **DI**ctionary **S**erver. Opera de forma similar a como lo hacen el resto de los SGBD clave-valor, pero presenta una serie de ventajas sobre éstas. A diferencia de otros motores de Bases de Datos clave-valor (por ejemplo, Memcached), permite persistir los datos que aloja en memoria principal a disco, de tal manera que, si un nodo se apaga, siempre podría recuperar la última información que haya escrito previamente a disco [6, 56].

  Este motor de Base de Datos es uno de los seleccionados para la experimentación que se presenta en este trabajo y será detallado en el siguiente capítulo.

- **Amazon DynamoDB** [57]**:** es un SGBD NoSQL clave valor distribuido y multi maestro. Amazon ofrece el servicio de Base de Datos de manera totalmente gestionada (Amazon 2012). DynamoDB es un SGBD propietario desarrollado internamente por Amazon y lanzado en 2007, este producto se incluye dentro de la oferta de servicios que se ofrecen como parte de los AWS (Amazon Web Services) [4, 57].

  El servicio de Amazon permite crear nuevas tablas en las que alojar los datos, e interactuar con ellas. No se tiene acceso al código fuente, ni al hardware ni a los servicios que hacen funcionar a la Base de Datos. Al ser una Base de Datos totalmente gestionada, significa que Amazon se encarga de toda la configuración y el mantenimiento de la infraestructura física. DynamoDB es capaz de repartir los datos y el tráfico a través de múltiples servidores de forma que se puedan cubrir las necesidades del usuario [4, 20, 57].

  Empresas como Airbnb [94], Samsung [95] y Toyota [88] aprovechan el rendimiento de DynamoDB para ofrecer soporte a sus cargas de trabajo.

- **Memcached** [58]**:** motor de Base de Datos NoSQL desarrollado por la empresa Danga Interactive, que mantiene el proyecto bajo licencia BSD y que fue pensado para incrementar la velocidad de aplicaciones web dinámicas, reduciendo la carga de la Base de Datos. Memcached hace uso de una tabla hash donde va almacenando los valores asociados a una clave única para cada elemento. Por medio del uso de instrucciones sencillas, se puede almacenar y recuperar información almacenada en la memoria principal de forma muy rápida, haciendo uso de la clave [58].

  Este sistema reduce el número de consultas en la Base de Datos. Lo primero que se hace es consultar al servidor de Memcached si tiene almacenada la información solicitada. Si

es así, la información se devuelve y si no se debe hacer la petición a la Base de Datos ni introducir los datos en Memcached, para que puedan ser accedidos directamente ante un próximo requerimiento. Cada dato introducido tiene un tiempo de expiración; pasado ese tiempo, se elimina, dejando ese espacio para cualquier otro nuevo dato [24, 58].

A continuación, se detallan algunas de las ventajas de Memcached por operar sus estructuras de datos en memoria principal:

- Reduce los niveles de carga de los servidores.
- Permite ajustar el espacio de memoria dedicada, permitiendo indicar la cantidad de memoria destinada para el almacenamiento de información.
- Fácil comunicación entre cliente y servidor, basada en sencillas instrucciones.
- Posibilidad de controlar el tiempo de vida de la información almacenada.

Algunos sitios web como Youtube [77], Facebook [75] o Twitter [76] utilizan Memcached para optimizar el rendimiento y conseguir cachear los datos que tienen que enviar a los usuarios y visitantes.

5.1.5.2 Almacenamiento Documental

Las Bases de Datos documentales se basan en el concepto de "documento". El documento es la unidad principal de su almacenamiento. Estos documentos pueden ser de múltiples formatos, XML, JSON, BJSON, YAML, entre otros. En la figura 18 se presenta la estructura de un documento JSON.

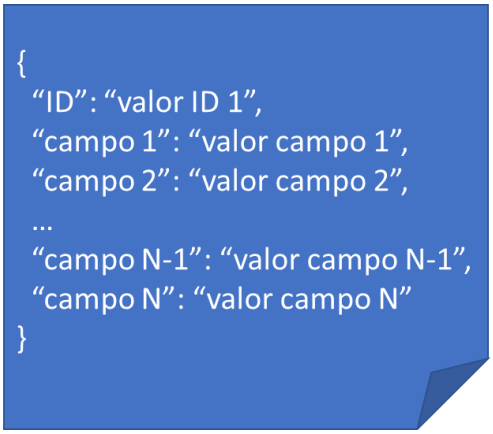

Figura 18. Representación de un documento (JSON)

Algunos de los motores de Bases de Datos que poseen almacenamiento documental, gestionan y almacenan estos documentos en colecciones, análogamente a lo que es una tabla en el modelo relacional, con la diferencia que cada documento dentro de una colección puede tener su propia estructura. Esta categoría de almacenamiento proporciona la capacidad de manejar millones de

lecturas simultáneas, esto se debe a que un documento contiene toda la información que se necesita, es decir, se mantiene toda la información relacionada en solo un documento [5, 6, 13].

Las Bases de Datos que utilizan este tipo de almacenamiento, definen un identificador para cada documento almacenado, esto permite la indexación primaria y su rápida recuperación. Además, se permite la indexación por otros campos del documento lo que mejora el rendimiento, pero hay que evaluar el costo de su mantenimiento. Estas Bases de Datos son apropiadas para el particionado horizontal y en contextos donde la estructura de la información es variable [5, 13, 15].

Entre los motores de Bases de Datos más populares que implementan este tipo de almacenamiento encontramos a: MongoDB [59], Amazon DynamoDB [57], Couchbase [60], Firebase Realtime Database [61], entre otros.

- **MongoDB** [59]**:** este motor de Base de Datos es uno de los seleccionados para la experimentación que se presenta en este trabajo. Será detallado en el siguiente capítulo. Algunas de las empresas que utilizan MongoDB son: LinkedIn [96], CISCO [78], eBay [87], Expedia [97], IBM [70], entre otras. Incluso el CERN (Organización Europea para la Investigación Nuclear) utiliza MongoDB para los grandes volúmenes de datos que genera el acelerador de partículas [29, 59, 98].

- **Couchbase** [60]**:** es una Base de Datos distribuida basada en documentos. Aprovecha una capa de almacenamiento en memoria caché, lo que le permite admitir operaciones muy rápidas de creación, almacenamiento, actualización y recuperación. Se basa en tres principios fundamentales, simple, rápido y elástico.
  Su núcleo es muy simple y directo, y desde la perspectiva del cliente, fácil de usar. También es muy rápido y fácil de instalar y configurar. Es compatible con Memcached, es decir, que, si una aplicación utiliza Memcached, puede almacenar datos en Couchbase [3, 60].

  Couchbase se basa en la estructura básica de almacenamiento de documentos o clave-valor de Memcached. Esto hace que sea muy sencillo almacenar y recuperar datos. No es necesario definir una estructura de datos antes de comenzar a almacenar, y no se requieren consultas complicadas ni lenguajes de consulta para recuperar los datos.

  Couchbase admite tiempos de respuesta inferiores al milisegundo y ha sido optimizado para el almacenamiento de datos de muy alta concurrencia. El sistema es linealmente escalable debido a una arquitectura de "shared nothing (nada compartido)" y puede mejorar el rendimiento general agregando más nodos. El clúster de Couchbase está diseñado para expandirse fácilmente. Para crear un clúster de múltiples nodos, se instala el software en otra máquina y se añade al clúster existente. No es necesario desactivar el clúster actual (ni los clientes ni las aplicaciones que lo utilizan) para realizar esta operación [3, 12, 60].

Algunas de las empresas que utilizan Couchbase son: PayPal [99], Linkedin [96], CISCO [78], entre otros.

- **Firebase Realtime Database** [61]**:** es una Base de Datos gestionada por Google [89] y alojada en la nube. Los datos se almacenan en formato JSON y se sincronizan en tiempo real con cada cliente conectado. Es el servicio de backend que proporciona Google para aplicaciones de Android, aplicaciones de IOS y sitios web [61, 89]. Firebase Realtime Database usa la sincronización de datos para que cada vez que hay un cambio, los dispositivos conectados reciben la actualización en milisegundos. Puede trabajar sin conexión logrando que los datos persistan en el disco y cuando se restablece la conexión, el dispositivo cliente recibe los cambios que faltaban y los sincroniza con el estado actual del servidor.

  Esta Base de Datos permite acceder directamente desde un dispositivo móvil o un navegador web; no es necesario un servidor de aplicaciones. La seguridad y la validación de datos están disponibles a través de las reglas de seguridad de Firebase Realtime Database, estas reglas se basan en expresiones que se ejecutan cuando se leen o se escriben datos.

  Posee una versión o plan de pago para satisfacer las necesidades a gran escala. Se puede dividir la información en diversas instancias de Bases de Datos dentro del mismo proyecto de Firebase. También se puede utilizar "Firebase Authentication" para optimizar el proceso de autenticación de usuarios a la Base de Datos, de esta forma, se controla el acceso a la información. Para ello, se aplican reglas personalizadas de Firebase Realtime Database en cada una de las instancias de la Base de Datos. [13, 26, 33].

5.1.5.3 Almacenamiento Orientado a Columnas

En este tipo de almacenamiento la información se estructura por columnas, en lugar de por filas. La comparación entre el acceso tradicional por filas del modelo relacional y este tipo de almacenamiento se basa principalmente en los accesos a disco. En la figura 19 se presenta gráficamente este tipo de almacenamiento.

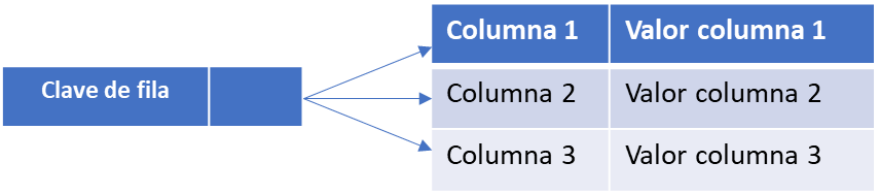

Figura 19. Almacenamiento de Familia de Columnas.

Los accesos a posiciones que se encuentran consecutivas o cercanas requieren de menor tiempo que los accesos a posiciones aleatorias. La mejora en el rendimiento es notoria cuando una consulta involucra gran parte de las filas, pero solamente una pequeña parte de las columnas. Las Bases de Datos que se basan en este tipo de almacenamiento poseen buen rendimiento para las consultas que involucran grandes cantidades de datos. La agrupación lógica en familia de columnas hace que se pueda acceder de una sola vez a toda la información que se encuentra interrelacionada para una determinada entidad. Además, permite que cada entidad posea distinta familia de columnas y solo almacenar las que realmente sean relevantes [6, 7, 8, 62].

Entre los motores de Bases de Datos más populares que implementan este tipo de almacenamiento se encuentran Apache Cassandra [62], Hbase [63], Datastax Enterprise [64], entre otros.

- **Apache Cassandra** [62]: este motor de Base de Datos es uno de los seleccionados para la experimentación que se presenta en este trabajo. Será detallado en el siguiente capítulo.

- **Apache Hbase** [63]: es un Sistema de Gestión de Bases de Datos NoSQL orientado a columnas que se ejecuta sobre Hadoop Distributed File System (HDFS). Esta Base de Datos es desarrollada en Java y fue creada inicialmente a partir de los artículos que liberó Google sobre BigTable en el año 2004. Es la implementación Open Source del proyecto Bigtable de Google [5, 7, 89].
  Apache Hbase es adecuada para el procesamiento de datos en tiempo real o el acceso aleatorio de lectura / escritura a grandes volúmenes de datos. Es una Base de Datos distribuida, persistente, dispersa y ordenada en un mapa multidimensional. Una celda de esta Base de Datos viene definida por una clave primaria de fila (rowkey), por una clave de columna (columnkey) y una marca de tiempo (timestamp) [2, 5, 7].

  HBase utiliza para almacenar la información el HDFS de Hadoop, aunque también puede apoyarse en otros, como Amazon S3 (el sistema de almacenamiento propio de Amazon). Estos sistemas de ficheros tienen la característica de ser distribuidos y tolerantes a fallos [5, 7, 63]. El proyecto de HBase, al ser de código abierto, ha tenido muchos contribuyentes importantes. Por ejemplo, Facebook [75] lo ha utilizado en 2010 para optimizar su sistema de mensajería.

- **DataStax Enterprise (DSE)** [64]: es una plataforma de datos escalable, siempre activa, que se basa en Apache Cassandra y está diseñada principalmente para la nube. DSE integra diversas herramientas de administración y monitoreo en una plataforma unificada. Ofrece disponibilidad continua, rendimiento de escala lineal, simplicidad operativa y fácil distribución de datos en la nube [64]. DSE proporciona una serie de características y beneficios clave para facilitar el desarrollo y la gestión de aplicaciones en la nube, entre estos beneficios se pueden encontrar:

- **Arquitectura altamente escalable**: posee un diseño sin maestro en el que todos los nodos son iguales, lo que brinda simplicidad operativa y capacidades de escalamiento fácil.
- **Replicación**: posee centros de datos múltiples en varias zonas geográficas.
- **Diseño activo en todas partes**: todos los nodos se pueden escribir y leer sin importar dónde se encuentren.
- **Rendimiento de escala lineal**: adicionar nodos produce un aumento lineal en su rendimiento. Por ejemplo, si dos nodos producen 200.000 transacciones por segundo, cuatro nodos entregarán 400.000 transacciones por segundo.
- **Disponibilidad continu**a: ofrece redundancia de datos y de funciones, lo que proporciona un punto único de falla y un tiempo de actividad constante.
- **Detección y recuperación de fallas transparentes**: los nodos que fallan se pueden restaurar o reemplazar fácilmente.
- **Modelos de datos flexibles y dinámicos**: puede implementar almacenamiento tabular, documental (JSON) y de grafos.
- **Protección de datos**: posee una bitácora o registro que garantiza que no se pierdan datos para las transacciones entrantes. Además, posee políticas de seguridad con copias de restauración sencillas para mantener los datos protegidos.
- **Compatibilidad con transacciones**: admite atomicidad, aislamiento y durabilidad de las transacciones con consistencia de datos fuerte o eventual suministrada en un clúster ampliamente distribuido.
- **Compresión de datos**: la posibilidad de tener datos comprimidos hasta en un 80 % sin sobrecarga de rendimiento, colabora en el ahorro en los costos de almacenamiento.
- **CQL (Lenguaje de consulta de Cassandra)**: similar a SQL, lo que facilita la implementación de consultas y el posible traslado desde un RDBMS.

5.1.5.4 Almacenamiento Orientado a Grafos

En este tipo de almacenamiento la información se representa mediante un grafo. Las entidades se modelan como nodos y las relaciones entre los datos se representan mediante aristas, es decir, que cada nodo almacena los punteros a sus nodos adyacentes. En la figura 20 se presenta gráficamente un ejemplo de almacenamiento de grafos.

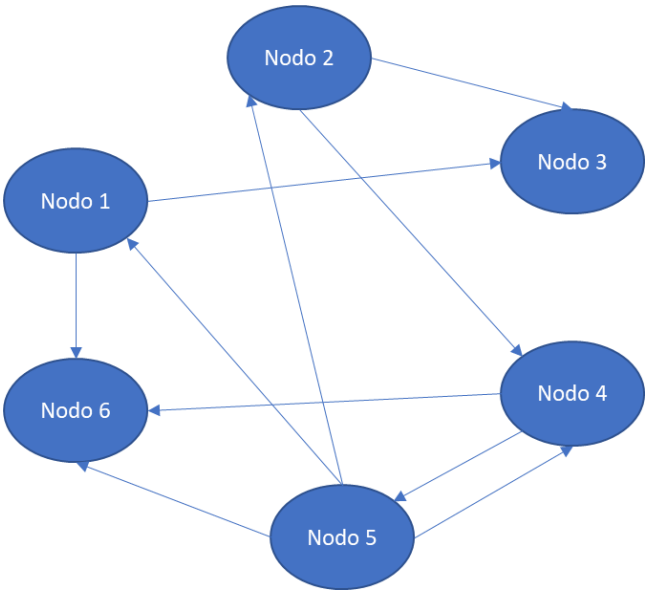

Figura 20. Almacenamiento de Grafo.

En este tipo de almacenamiento, se utiliza la teoría de grafos para efectuar recorridos sobre la Base de Datos. Generalmente, el rendimiento de las Bases de Datos que utilizan este tipo de almacenamiento no está relacionado con el tamaño total del grafo, ni con aquellas consultas que requieran un nivel alto de procesamiento, sino que su rendimiento depende de la parte del grafo que se vea afectada por la consulta a resolver. Además, brinda flexibilidad y agilidad en su mantenimiento, esto se debe a la conexión existente entre sus datos [14, 17, 65].

La utilización de este tipo de almacenamiento depende del contexto de la información y la lógica de negocio. Utilizar una Base de Datos orientada a grafos en un escenario no apropiado hará que no se aproveche todo su potencial. No cualquier problemática es adaptable a este tipo de almacenamiento y tampoco se debe buscar su adaptación forzada porque posiblemente no se podrán explotar las fortalezas en la ejecución de consultas con alto volumen de datos. El motor de Bases de Datos orientado a grafos más popular es Neo4j [65]. No obstante, existen otros motores de Bases de Datos, pero algo menos populares, como JanusGraph [100] o DGraph [101].

- **Neo4J** [65]: este motor de Base de Datos es uno de los seleccionados para la experimentación que se presenta en este trabajo. Será detallado en el siguiente capítulo.

5.1.6 Otras categorías de almacenamiento

5.1.6.1 Almacenamiento en Series Temporales

En este tipo de almacenamiento se considera a la serie temporal como secuencia de datos que generalmente provienen de mediciones tomadas por algún sensor externo. En términos más generales, una Base de Datos de Serie Temporal administra una secuencia de puntos de datos

(no necesariamente números), que típicamente, consisten en mediciones sucesivas realizadas durante un intervalo de tiempo.

El desafío en las Bases de Datos que se basan en este tipo de almacenamiento es proporcionar algoritmos eficientes y métodos de acceso para el procesamiento de consultas, teniendo en cuenta el hecho de que la Base de Datos cambia continuamente a medida que hay nuevos datos disponibles.

Entre los motores de Bases de Datos más populares de almacenamiento de Series Temporales podemos encontrar a InfluxDB [68], Prometheus [102] y Graphite [103], entre otros [43].

5.1.6.2 Almacenamiento Multivalor

Las Bases de Datos que utilizan este tipo de almacenamiento se basan en la propiedad de los atributos multivaluados. Para comprender este tipo de almacenamiento, es necesario conocer el concepto de "Normalización" existente en el proceso de diseño del modelo lógico relacional. Los atributos multivaluados, en un modelo relacional, requieren ser transformados debido a la primera forma normal. Las Bases de Datos que utilizan un almacenamiento multivalor no aplican la primera forma normal, es decir, que admiten la presencia de atributos multivaluados.

En este tipo de almacenamiento se acepta el concepto de redundancia de información y la misma no debe ser separada en estructuras diferentes. Estas Bases de Datos admiten un nivel de desnormalización con el objetivo de mejorar el rendimiento y reducir el número de estructuras destinadas a almacenar la información. Dos de los motores de Bases de Datos más populares que implementan este tipo de almacenamiento son: Adabas [43] y Unidata-Universe [43, 104].

5.1.6.3 Almacenamiento RDF (Resource Description Framework)

Las Bases de Datos que utilizan este tipo de almacenamiento, son un tipo de Base de Datos de grafo que guardan factores semánticos interconectados entre sí. Son capaces de manejar potentes consultas semánticas y utilizar patrones para deducir información de las relaciones existentes.

En este tipo de Bases de Datos los datos almacenados se componen de tres partes vinculadas: objeto, sujeto y predicado. Este formato puede conectar un objeto (denominado objeto) a cualquier otro objeto (denominado sujeto) a través de una relación (denominada predicado). De esta manera se puede inferir situaciones o hechos implícitos sobre la información que se encuentra almacenada.

Permite descubrir relaciones ocultas entre los datos guardados y facilita las técnicas de análisis de texto, como la extracción de datos no estructurados, con el objetivo de clasificarlos y determinar el contexto en el que se aplica, eliminando la ambigüedad.

Uno de los motores de Bases de Datos más populares que poseen únicamente este tipo de almacenamiento es Apache Jena - TDB [43, 105].

5.1.7 Servicios en la Nube (Cloud Database Service)

Los servicios en la nube son modelos que permiten el acceso bajo demanda con el objetivo de compartir un conjunto configurable de recursos de computación (por ejemplo, servidores para el almacenamiento de datos). La computación o servicios en la nube lleva varios años y representa una evolución en la tecnología informática. El avance en la tecnología móvil ha generado la incorporación de una gran variedad de dispositivos electrónicos en la vida cotidiana de los usuarios (Smartphones, Notebooks, Tablets, etc.). Además, existen numerosas aplicaciones y plataformas que hacen uso de servicios en la nube. [4, 13, 10].

Una Base de Datos en la nube está diseñada para un entorno informático virtualizado y se implementa mediante computación en la nube (Cloud Computing), lo que significa utilizar recursos de software y hardware pertenecientes al proveedor de servicios de computación en la nube. Se puede hacer referencia a la computación en la nube como una nueva dimensión en el mundo de la Tecnología de la Información (IT, por sus siglas en inglés, Information Technology) en términos de ahorro de costos y rendimiento más rápido de las aplicaciones. Una Base de Datos en la nube se utiliza principalmente como un servicio y también se la denomina: Base de Datos como servicio [4, 13].
En la actualidad, los proveedores de Bases de Datos relacionales y NoSQL ofrecen implementaciones alternativas de almacenamiento de datos en la nube, similares a las que se pueden descargar e instalar en algún servidor local. Esto permite poder realizar una migración, casi transparente, desde un entorno local a un entorno en la nube. Las Bases de Datos en la nube, cada vez están ganando más popularidad al momento de gestionar y almacenar grandes cantidades de datos. Los motores de Bases de Datos en la nube permiten agregar nodos adicionales cuando sea necesario y aumentar el rendimiento y su performance [13, 59, 64].

Una Base de Datos en la nube administra sus datos en centros ubicados en distintos lugares geográficos. Esto hace que la estructura sea compleja. Existen diferentes métodos para acceder a los servicios en la nube, se puede acceder a través de Internet, o utilizando tecnología móvil, servicios 3G o 4G.

No obstante, es importante conocer las ventajas y desventajas de ambas alternativas (local y en la nube) para evaluar cuál será la alternativa que mejor se adapte a las necesidades del negocio. Existen diferencias a considerar entre una implementación de Base de Datos local y un servicio de Base de Datos en la nube. A continuación, se detallan algunas de estas diferencias:

- **Limitaciones:** generalmente, las Bases de Datos en la nube, en sus versiones libres o gratuitas, establecen limitaciones en las características que ofrecen al usuario, entre estas limitaciones se incluyen el espacio de almacenamiento brindado, la cantidad de lecturas y escrituras en un periodo de tiempo, la cantidad de conexiones simultáneas, entre otras. Alternativamente, ofrecen diversos planes de pago para ampliar estas limitaciones, según las necesidades del cliente.

- **Ubicación de sus servidores:** Al configurar una Base de Datos en la nube, es común, poder seleccionar la ubicación geográfica del clúster. Esto permite ubicar el servidor y sus réplicas más cerca del cliente.
- **Herramientas de monitoreo:** algunas implementaciones incorporan, desde su plan libre o gratuito, herramientas que permiten monitorear la cantidad de conexiones en un momento determinado, el espacio físico ocupado y disponible, la cantidad de operaciones en un rango de tiempo establecido, entre otras.
- **Menor control sobre la infraestructura:** El hardware y su infraestructura estarán en manos del proveedor del servicio, restando trabajo a los administradores de Bases de Datos. Pero hay que tener en cuenta que se pierde control sobre la administración de recursos.

Las tecnologías de Bases de Datos en la nube han cambiado y reestructurado las tareas que deben realizar sus administradores. Hay ciertas características de rutina, que existen en las Bases de Datos locales, por las que ahora, no es necesario preocuparse. Algunas de las tareas que se llevan a cabo cuando se instala un motor de Base de Datos local, de alguna manera, cambian al utilizar un motor de Base de Datos en la nube, dando lugar a otras tareas más cercanas al dominio de la aplicación y a la incorporación y utilización de herramientas específicas que cada proveedor pone a disposición de sus usuarios.

A continuación, se detallan algunos desafíos generales que deben afrontar las implementaciones no relacionales para continuar madurando y establecerse como una alternativa a los motores de Bases de Datos tradicionales.

5.1.8 Algunos desafíos

Actualmente, las Bases de Datos NoSQL se han consolidado como una alternativa real a las tradicionales Bases de Datos relacionales. No obstante, aún deben afrontar ciertos desafíos y/o críticas en general [13, 19, 21]. A continuación, se presentan algunos de los puntos más cuestionados en las Bases de Datos NoSQL:

- **Madurez:** las Bases de Datos relacionales han perdurado en el tiempo con gran éxito y aceptación, debido a esto poseen un gran nivel de maduración en el mercado empresarial brindando estabilidad y seguridad en grandes proyectos. Si bien, el concepto de Bases de Datos NoSQL ya se encuentra instalado de forma exitosa aún se puede considerar que necesitan continuar avanzando y madurando en ciertos aspectos.
- **Soporte**: las tecnologías NoSQL en su mayoría son proyectos “open source”, lo que genera, en ciertas ocasiones, reticencias por temor a no contar con el soporte suficiente ante un problema. Las grandes organizaciones necesitan la tranquilidad de que la información estará segura por más que el sistema falle y, además, requieren soporte oportuno ante cualquier inconveniente.

- **Análisis e inteligencia de negocios:** el gran volumen de información que se genera en algunas aplicaciones posee un valor agregado. Toda esa información sirve a futuro para la toma de decisiones. Por lo tanto, la Inteligencia de Negocio (BI, por sus siglas en inglés, Business Intelligence) es un tema para considerar. Esto genera un gran desafío para las Bases de Datos NoSQL, debido a que deben tener en cuenta la forma de poder integrarse apropiadamente con herramientas de BI.
- **Necesidad de administradores de sistemas:** normalmente, cuando se habla de NoSQL, se hace referencia a la funcionalidad, sus capacidades, rendimiento y de sus ventajas y desventajas con respecto a motores de Bases de Datos relacionales. Pero existe un concepto que también es importante en el manejo de un motor de Base de Datos NoSQL, este es el concepto de administración u operador del sistema. Estos usuarios, serán los encargados de solucionar aquellos problemas que puedan surgir desde el punto de vista de la administración del sistema. Este rol, en algunos casos, no es muy detallado en la documentación de algunos motores de Bases de Datos NoSQL.

Capítulo 6

6.1 Experimentaciones en motores de Bases de Datos Relacionales y Bases de Datos NoSQL

6.1.1 Introducción

Las Bases de Datos relacionales han demostrado cubrir las necesidades de almacenamiento de información en una gran variedad de soluciones informáticas. No obstante, cuando estas Bases de Datos comenzaron a ser utilizadas, los esquemas y el crecimiento del volumen de datos era previsible en el tiempo. Actualmente, la realidad es muy distinta, existen aplicaciones que generan enormes volúmenes de información y de forma no estructurada. Ante esta situación, se utilizan nuevas formas de almacenamiento denominadas Bases de Datos NoSQL.

A partir de aquí, el objetivo será comparar y analizar el uso de Bases de Datos relacionales y Bases de Datos NoSQL en diferentes casos de estudio con diferentes esquemas y volúmenes de información. Se realizará una comparación de resultados y se desarrollarán las conclusiones resultantes.

6.1.2 Motores de Bases de Datos utilizados

Para llevar a cabo estas experimentaciones, se optó por MySQL [47] (motor de Base de Datos relacional), MongoDB [59] (motor de Base de Datos NoSQL con almacenamiento documental), Apache Cassandra [62] (motor de Base de Datos NoSQL que combina un tipo de almacenamiento clave-valor con familia de columnas), Neo4j [65] (motor de Base de Datos con almacenamiento orientado a grafos) y Redis [56] (motor de Base de Datos con almacenamiento clave-valor). Estos motores de Bases de Datos fueron elegidos en base a su popularidad de acuerdo con sus categorías [43]. Además, se seleccionaron y configuraron 2 motores de Bases de Datos Documentales que poseen servicios en la nube, Cloud Firestore [61] y MongoDB Atlas [59]. A continuación, se presenta una breve descripción de cada motor de Base Datos utilizado:

- **MySQL** [47]: la elección de MySQL se debe a que es uno de los sistemas de Base de Datos relacional más reconocidos y populares del mercado con licencia dual (licencia pública general y licencia comercial) [43]. MySQL es Open Source, esto significa que es posible realizar utilizar y modificar el software sin costo económico alguno. Además, se puede estudiar el código fuente y cambiarlo para adaptarlo a las necesidades del usuario.

  MySQL utiliza licencia GPL (GNU General Public License) que define qué es lo que se puede hacer y qué es lo que no se puede hacer con el software en diferentes

circunstancias. En el caso de necesitar algo muy específico, por fuera de la licencia GPL, se puede adquirir una licencia comercial.

MySQL es un Sistema Gestor de Bases de Datos ampliamente usado por su simplicidad y notable rendimiento. Además, es simple en su instalación, puesta en marcha y utilización. Algunas de sus características importantes son:

- Se encuentra desarrollado en C/C++.
- Su velocidad de respuesta es destacable.
- Posee un variado conjunto de tipos de datos.
- Posee múltiples métodos para el almacenamiento de sus tablas.
- Ofrece diferentes configuraciones para mejorar sus prestaciones en casos concretos.
- Su administración se basa en usuarios y privilegios.
- Es confiable en cuanto a estabilidad.
- Funciona en diferentes plataformas.
- Posee Joins muy rápidos usando un multi-join de un paso optimizado.
- Utiliza tablas hash en memoria, que son usadas como tablas temporales.

Además, en MySQL el soporte de transacciones y la integridad referencial (la gestión de claves foráneas) está condicionada a un esquema de almacenamiento de tabla concreto, de forma que, si un usuario no necesita transacciones, puede utilizar un esquema de almacenamiento tradicional, denominado “MyISAM” y obtendrá mayor rendimiento, mientras que, si su aplicación requiere transacciones, podrá utilizar otro esquema denominado “InnoDB” sin ninguna otra restricción o implicación [3, 9, 13, 47].

MySQL es utilizado por numerosas empresas a nivel mundial, entre ellas se destacan, Facebook [75], Twitter [76], YouTube [77], GitHub [79], Uber [83], Booking [81], Spotify [80], entre otras.

- **MongoDB** [59]: su nombre deriva de la palabra inglesa “homongous”. MongoDB es una Base de Datos NoSQL documental, es Open Source, multiplataforma, posee alta disponibilidad, es apta para escalar de forma horizontal y es orientada a documentos. Ha sido diseñada para desarrolladores de aplicaciones modernas y almacena sus datos en colecciones de documentos flexibles (JSON Binario), similar a JSON. Cada documento en MongoDB puede almacenar otros documentos, arreglos o incluso arreglos de documentos. Actualmente, la empresa 10gen es la que se encarga de su mantenimiento y desarrollo [6, 13, 59].

  La utilización de documentos para el almacenamiento de datos posee las siguientes ventajas:

- Los documentos son tipos de datos nativos en muchos lenguajes de programación.
- Los documentos embebidos y los arreglos internos minimizan la necesidad de realizar JOINS muy pesados.
- Tener un esquema dinámico es la base del polimorfismo.

La estructura de los documentos pertenecientes a una misma colección puede ser variable. Esto es una ventaja en contextos en donde la estructura de la información puede cambiar con el tiempo. El uso de documentos facilita la integración con datos nativos existentes en distintos lenguajes de programación, además, la posibilidad de tener documentos embebidos y/o arreglos de documentos, evita la necesidad de realizar operaciones de JOINs.

MongoDB soporta índices secundarios, esto le permite tener un buen desempeño en las consultas, debido a que se puede indexar cualquiera de los campos existentes en un documento.

La replicación en MongoDB adquiere un papel fundamental. Pueden existir varios motivos por los cuales se podría querer tener un entorno replicado de Base de Datos, por ejemplo, para incrementar la capacidad de lectura, para proteger la Base de Datos de diferentes posibles caídas de servidores, como políticas de backup, para incrementar la localización y la disponibilidad de la información, entre otros motivos [8, 15, 27].

MongoDB implementa la replicación entre sus nodos a través de lo que se conoce como “replica set”. Esto es un conjunto de servidores que siguen una arquitectura de maestro – esclavo. Existe un servidor maestro y uno o varios esclavos. El servidor maestro puede realizar operaciones de escritura y lectura, pero los esclavos únicamente permiten la lectura. Además, existen varios tipos de esclavos, cada uno de ellos con una serie de funciones y competencias.

Únicamente puede existir un nodo primario por cada “replica set”. Será el nodo principal o maestro y únicamente este nodo el que acepte las operaciones de escritura por parte de los clientes. El nodo maestro comunica las actualizaciones a los nodos secundarios mediante un log de operaciones llamado “oplog” en donde se escriben todas las operaciones que modifican la Base de Datos. Los nodos secundarios aplican las operaciones del “oplog” de forma asíncrona. Esto significa que entre que el maestro recibe una operación de escritura y el nodo esclavo escribe esos datos, existe un tiempo indeterminado. Por tanto, los nodos secundarios en algunos momentos podrían no devolver a los clientes la información más actualizada [6, 27, 28].

Si el nodo principal queda inaccesible, los nodos secundarios efectuarán lo que se conoce como elecciones con el objetivo de encontrar un nuevo nodo principal para el clúster. Además de los nodos secundarios, se pueden dar dos configuraciones específicas que derivan en dos nodos secundarios especiales:

- **Nodos secundarios ocultos:** son invisibles para las aplicaciones, y además no pueden ser elegidos nodos primarios en las elecciones (tienen la prioridad a cero). Se emplean para labores de reporte y backups.
- **Nodos secundarios retardados:** son similares a los ocultos, son invisibles para las aplicaciones y no pueden ser seleccionados como un nodo maestro en ningún caso. La principal característica de este tipo de nodos es que mantienen una versión histórica de la Base de Datos. Es decir, llevan una cierta demora en la replicación, lo cual los hace perfectos para mantenerlos como nodos backups de corto plazo (actualizaciones erróneas, errores en operaciones, Bases de Datos borradas por error, etc.).

MongoDB distribuye los conjuntos de datos partiendo las colecciones de la Base de Datos. Existen diferentes mecanismos para decidir qué criterio se va a seguir para dividir y repartir los datos. Para realizar un particionado MongoDB se basa en un sistema de claves. Se escoge un campo que sea común a todos los elementos de la colección que se desea particionar y se distribuyen en todos los servidores de una forma más o menos homogénea.

En MongoDB existen dos formas de dividir las colecciones:

- **Particionamiento basado en rangos**: divide la colección en función de los valores del campo que se haya elegido. Los lotes generados son conjuntos de documentos no solapados, y vienen determinados por un valor máximo y otro mínimo de la clave elegida como "range key". Este particionado, posee una ganancia de rendimiento en búsquedas de documentos secuenciales. No obstante, esto mismo puede resultar contraproducente. Si un proceso siempre trata de acceder al mismo rango de datos, el particionado carecería de sentido, puesto que siempre estaría haciendo las consultas sobre el mismo lote.

- **Particionamiento basado en hashes**: genera las claves hash de los valores de un campo concreto común a todos los documentos, y utiliza estos hashes para generar los lotes. Este particionamiento asegura un reparto más homogéneo de los datos. Este tipo de asignación distribuye los datos aleatoriamente y asegura un balance más igualitario al momento de procesar las operaciones en la Base de Datos.

MongoDB es un tipo de Base de Datos de esquema flexible. Esto significa que el usuario determina y declara el esquema de cada uno de los documentos al momento de realizar las inserciones en la Base de Datos. Se cuenta con la flexibilidad de añadir campos cuando son necesarios sin que sean definidos de antemano, u obviar aquellos que no sean necesarios. A su vez, esto requiere que el programador sea metódico en su trabajo, puesto que el sistema no avisará de un posible error u omisión en la estructura de la información [6, 8, 13, 59].

Diferentes empresas como Toyota [88], SEGA [82], AstraZeneca [86], eBay [87], CISCO [78], Google [89], Verizon [85], entre otras, hacen uso de MongoDB.

- **Apache Cassandra** [62]: fue inicialmente desarrollado por Facebook [75] y su código fue liberado en el año 2008. Desde el año 2010 forma parte de los proyectos de Apache. Es de código abierto y se encuentra implementada en Java. Apache Cassandra es una Base de Datos NoSQL que combina dos formas de almacenamiento no estructurado, clave-valor y familia de columnas. Es una Base de Datos altamente escalable, distribuida y eventualmente consistente. Se basa en la tecnología de DynamoDB para la consistencia eventual y de BigTable de Google para el almacenamiento de familia de columnas. Por ser una Base de Datos altamente escalable y distribuida, es posible crear el número de nodos que sean necesarios para su correcto funcionamiento. El usuario o aplicación externa siempre tendrá la sensación de estar trabajando bajo un único sistema. Además, es un sistema preparado para recibir operaciones de escrituras, por lo que cualquier operación que escriba en cualquiera de sus ubicaciones, será eficientemente propagada a todo el clúster [7, 16, 62, 89].

  Apache Cassandra es un motor de Base de Datos descentralizado en donde no existe un único punto de fallo, todos los nodos poseen la misma funcionalidad e importancia. No existe un nodo maestro o un nodo que coordine las actividades del resto de los nodos. Esta descentralización proporciona dos beneficios; es más simple que los sistemas maestro-esclavo y aumenta la disponibilidad. Es poco factible que el servicio se detenga, debido a que la administración no se encuentra centralizada en un único nodo [6, 8, 11].

  El modelo que propone Apache Cassandra para la comunicación entre sus nodos, es un modelo basado en la comunicación P2P (por sus siglas en inglés, peer-to-peer). De esta manera se consigue que todos los nodos sean funcionalmente idénticos entre ellos. De este modo, el diseño P2P hace que el sistema escale de forma sencilla, además de mejorar la disponibilidad de los datos, ya que la pérdida de un nodo no supone una degradación importante del servicio.

  La escalabilidad horizontal, es una de sus características más importantes. Posee la capacidad de continuar ofreciendo un buen desempeño, aun cuando exista un incremento de la demanda de operaciones. Utilizar varios nodos con un costo de hardware moderado, es una buena opción para ampliar un sistema sin entrar en grandes inversiones económicas. Añadir nuevos nodos al clúster se convierte en una tarea trivial, puesto que todos los nodos desempeñan el mismo rol [2, 13].

  Apache Cassandra en un motor de Base de Datos eventualmente consistente, según el teorema de CAP. Es decir, que se prioriza la disponibilidad y posee tolerancia a particiones. Antepone estas características, ante la consistencia. Aunque este comportamiento es configurable. La consistencia eventual es tan solo una posibilidad. Otras alternativas pueden ser:

- **Consistencia estricta**: cada lectura que se realice en el sistema devolverá la escritura más reciente.
- **Consistencia casual**: relaja la carga de sincronización de la consistencia estricta, aplica una visión más semántica de las operaciones. El sistema "descubre" qué operaciones de escritura están relacionadas y hace consistente las llamadas de lectura relacionadas con esa información.

La interacción con Apache Cassandra se realiza con una versión reducida del lenguaje SQL, este lenguaje es conocido como Cassandra Query Language (CQL). Los datos se encuentran desnormalizados de manera que el concepto de JOINS y subconsulta no exista. CQL es la interfaz primaria para la comunicación con Apache Cassandra [11, 16, 62].

Un clúster en Apache Cassandra está conformado por uno o más "keyspaces", un keyspace es análogo al concepto de Base de Datos en el modelo relacional. Un keyspace posee asociado un conjunto de atributos que definen su comportamiento:

- **Factor de replicación**: número de copias que debe guardarse de cada dato de la Base de Datos.
- **Estrategia para las réplicas:** establece qué política se seguirá para alojar las réplicas en el cluster.
- **Colum Families**: un keyspace actúa como contenedor de "column families". El concepto de "column families" es análogo al concepto de tabla en el modelo relacional.

Apache Cassandra está diseñada para funcionar formando clústers de máquinas. Cuanto más distribuido mejor. El protocolo P2P hace que los nodos se comuniquen entre ellos para conocer su estado y repliquen los datos de forma transparente al usuario [13, 62].

Empresas mundialmente conocidas como Facebook [75], Instagram [84], Netflix [73], GitHub [79], entre otras, hacen uso de Apache Cassandra.

- **Redis (RE**mote **DI**ctionary **S**erver**)** [56]: Redis es un gestor de Bases de Datos NoSQL basado en el almacenamiento clave-valor. Opera sus estructuras de datos en memoria principal y, además, permite opcionalmente persistencia. A diferencia de otros motores de Bases de Datos con el mismo tipo de almacenamiento, no trabaja exclusivamente en memoria principal, sino que también brinda la posibilidad de persistir datos, de modo que, si un nodo se cae o falla, puede recuperar la última versión persistida de los datos. Su funcionalidad básica es el de servir como motor de Base de Datos en memoria principal. Esto le confiere una velocidad y tiempos de respuesta mejores que los sistemas de Bases de Datos tradicionales u otros tipos de sistemas NoSQL [4, 12, 20].

Redis reconoce y maneja, en memoria principal, 5 tipos de datos diferentes, cadenas (String), conjuntos (Set), hashes, conjuntos ordenados (Sorted Set) y listas (List). Cuando un dato es agregado, actualizado o consultado, la operación se lleva a cabo directamente en la memoria principal. La persistencia de la información es un proceso configurable y automático que toma los datos de memoria principal y los graba a memoria secundaria [1, 12].

En caso de un fallo o corte de energía se puede recuperar la última versión que se ha persistido. Existen dos alternativas de configuración para la persistencia de los datos:

- **Persistencia RDB (Redis DataBase):** mediante este método de persistencia se realizan copias instantáneas de la Base de Datos cada cierto tiempo, estas copias se denominan "Snapshots". La frecuencia con la que se realizan estas copias es configurable en tiempo y número de cambios. Por ejemplo, se puede configurar que se realice una copia si se modifican cierta cantidad de claves en cierto tiempo.
- **Persistencia AOF (Append Only File):** su comportamiento se basa en dejar constancia de todas las operaciones de escritura que ocurren en la Base de Datos. Redis lleva un archivo de log con las solicitudes de escrituras, de manera que, en caso de ser necesario se puede volver a reconstruir la Base de Datos ejecutando en forma ordenada las operaciones registradas.

En general, se recomienda la implementación de ambos tipos de persistencia. RDB es especialmente recomendable cuando se quiere recuperar la Base de Datos en un momento concreto del tiempo, y AOF para recuperar los últimos datos en caso de que el servidor se apague de forma no controlada. Hay que tener en cuenta que llevar adelante ambos protocolos, implica costos de ocupación en disco y tiempo de procesamiento en sincronización de logs. Hay que evaluar si la pérdida de información de algunos minutos es algo crítico para el sistema [4, 13, 56].

También es posible utilizar solamente RDB para evitar la recarga de los logs que genera AOF. Por otro lado, utilizar solamente el protocolo de AOF, no es algo recomendable. Recuperar la Base de Datos completa, utilizando solamente los logs generados por AOF puede ser algo que requiera un alto costo de procesamiento. Otra opción, es no utilizar persistencia, en ese caso Redis se comportará como una Base de Datos que operará sus estructuras exclusivamente en memoria principal [4, 13, 20].

Redis utiliza un modelo clásico de replicación tipo maestro-esclavo. De este modo, se consigue tener un servidor maestro que será el que mantiene y actualiza la Base de Datos o una parte de ella, y una serie de servidores esclavos que se encargan de tener copias del maestro para su consulta. Algunos puntos destacables de la replicación en Redis son:

- Utiliza replicación asíncrona.
- Los nodos esclavos aceptan conexiones de otros esclavos, formando así una estructura en forma de grafo.
- La replicación es no bloqueante tanto para el maestro como para los esclavos. Es decir, se puede estar en medio de un proceso de replicación y atender al mismo tiempo operaciones de lectura (o escritura en caso de que sea el maestro). En el caso concreto de los nodos esclavos, esto es configurable.

Redis brinda la posibilidad de poder particionar una Base de Datos en múltiples instancias, de manera que cada una de las instancias sea responsable de un subconjunto de los datos. Utiliza básicamente dos tipos de particionado:

- **Particionado por rango**: es un tipo de particionado muy sencillo. Consiste en mapear rangos de objetos a instancias concretas. Un ejemplo sería dividir el ID de los elementos entre el número de instancias disponibles, y asignar un rango a cada nodo.
- **Particionado por Hash**: este particionado consiste en aplicar una función resumen a cada uno de los elementos de la Base de Datos. De esta manera, cuando se va a escribir un nuevo objeto, se calculará su función hash a la cual, posteriormente, se le aplicará la función módulo N donde N será el número de nodos disponibles en el clúster, obteniendo así el servidor concreto al que deberá ir este dato.

Además, posee diferentes variantes para realizar el particionado de la Base de Datos, estas variantes son:

- **Particionado por parte del cliente:** es el cliente el que realiza las operaciones necesarias para determinar en qué nodo debe ser almacenado el dato que está intentando escribir.
- **Particionado mediante proxy:** el cliente contacta con un nodo que hace las funciones de proxy y envía el dato al nodo que corresponda.
- **Particionado por enrutamiento:** los clientes pueden escribir en cualquier nodo del clúster, y será ese mismo nodo el encargado de determinar a la instancia a la que enviará la información.

Redis es utilizado por algunas de las empresas que son conocidas y utilizadas a nivel mundial, como Twitter [76], GitHub [79], StackOverflow [72] e Instagram [84], entre otras.

- **Neo4j** [65]: es una Base de Datos orientada a grafos (BDOG). Está desarrollada en Java por la empresa Neo Technology (Suecia) y es probablemente la Base de Datos basada en grafos más popular y utilizada del mercado. Las BDOG difieren del resto de Bases de

Datos NoSQL principalmente por su diseño y su forma de realizar y aplicar las operaciones [13, 65].

Un grafo es un conjunto de nodos que mantienen una serie de relaciones entre sí. Las relaciones tienen nombre y sentido, y siempre tienen un nodo de inicio y uno de fin. Tanto los nodos como las relaciones poseen propiedades. Los recorridos son consultas que se realizan sobre una parte o la totalidad del grafo. Los nodos o vértices representan entidades u objetos del problema real. Las relaciones se conocen como aristas o arcos y son elementos que representan el vínculo que existe entre los nodos que componen el grafo [14, 17].

En algunas ocasiones representar la información de un problema mediante una estructura de grafo es algo inherente a la propia naturaleza de los datos. Esto es una ventaja con respecto al resto de las Bases de Datos, no obstante, hay que evaluar si el enfoque que el problema necesita realmente es para este tipo de Base de Datos. La principal ventaja en una BDOG se da cuando los datos que se van a almacenar poseen relaciones entre sí o están conectados de alguna manera.

Este tipo de Base de Datos es de diseño gradual, a medida que se va adquiriendo experiencia en el dominio que se está tratando, crece la comprensión sobre cómo tendría que estar diseñada la Base de Datos. Esta es una característica importante debido a que actualmente las metodologías ágiles predominan para dirigir los desarrollos de software.

Es necesario que la Base de Datos también forme parte de esa agilidad y flexibilidad que se demanda en los proyectos de software. La naturaleza sin esquema de las BDOG hace que se perfilen como una opción mucho más viable que las Bases de Datos con esquemas estructurados. Una BDOG permite alterar y modificar partes del modelo de datos de forma rápida y natural, cambios que en otras Bases de Datos pueden llegar a implicar un esfuerzo considerable tanto en tiempo como en rendimiento [13, 18, 25].

Neo4j posee soporte ACID (Atomicity, Consistency, Isolation, Durability) para sus transacciones, es una Base de Datos transaccional, motivo por el cual se acerca a las Bases de Datos relacionales. Todas las operaciones se realizan como transacciones, sin embargo, los datos que se obtienen en algún recorrido de la estructura no se encuentran bloqueados, lo cual podrían ser modificados por otras transacciones. Los bloqueos son administrados por el sistema y se efectúan a nivel de nodo y de relación. También, se permite realizar una configuración manual de bloqueos para lograr un mayor nivel de aislamiento en las operaciones [14, 18, 65].

Neo4j tiene asociado un potente lenguaje denominado Cypher (Neo Technology 2014), que permite realizar consultas y actualizaciones sobre los datos almacenados en la Base de Datos. Cypher es un lenguaje de consulta declarativo, sencillo y potente, que permite que el usuario se centre en el dominio al momento de resolver la operatoria, es decir,

se centra en qué se quiere obtener de un grafo en lugar de pensar cómo obtenerlo [13, 25, 65].

Empresas como Walmart [74], eBay [87] y Cisco [78] hacen uso de Neo4j.

- **Cloud Firestore (en su versión libre)** [61]: Firebase es una plataforma en la nube que brinda Google [89] para el desarrollo de aplicaciones web y móviles. Contiene un conjunto de herramientas para la creación y sincronización de proyectos. Proporciona una serie de ventajas: agilidad en la sincronización de los datos de un proyecto, utilización de un conjunto de herramientas multiplataforma, infraestructura de Google [89], escalabilidad automática, creación de proyectos sin necesidad de un servidor, entre otras. Dentro de los servicios y herramientas que brinda se destaca Cloud Firestore, un motor de Bases de Datos en la nube NoSQL con almacenamiento documental. Cloud Firestore es una Base de Datos flexible y escalable para la programación en servidores, dispositivos móviles y la web. Mantiene los datos sincronizados entre aplicaciones del cliente en tiempo real y ofrece asistencia sin conexión para dispositivos móviles y la Web. Los documentos en Cloud Firestore admiten varios tipos de datos diferentes, desde strings y números simples, hasta objetos complejos. También dentro de los documentos se pueden crear subcolecciones y crear estructuras de datos jerárquicas [5, 13, 61].

  Las consultas de Cloud Firestore son expresivas, eficientes y flexibles. Es posible crear consultas para recuperar datos de un documento sin la necesidad de recuperar la colección completa ni las subcolecciones anidadas. Dentro de los servicios en la nube, es uno de los que posee mayor popularidad [13, 33].

  Cloud Firestore y Firebase Realtime Database (presentado en el capítulo 5) pertenecen a la plataforma de Firebase [61] que brinda Google [89].

- **MongoDB Atlas (en su versión libre)** [59]: esta es una Base de Datos NoSQL como servicio en la nube proporcionada por MongoDB. Es decir, que el usuario utiliza una Base de Datos que se encuentra operada y mantenida por MongoDB. El servicio de MongoDB Atlas incluye de forma automática, la configuración de los servidores y de todo el entorno para la Base de Datos. Además, es compatible con diferentes proveedores de servicios en la nube, como son: Amazon Web Services (AWS), Google Cloud Platform (GPC) y Microsoft Azure. MongoDB Atlas, al igual que MongoDB, es un motor de Base de Datos NoSQL con almacenamiento documental [8, 6, 27, 59].

Existen numerosos ejemplos y experiencias de empresas mundialmente conocidas que hacen uso de la tecnología NoSQL con el objetivo de mejorar la performance de sus servicios. En la figura 21 se presentan los logos de algunas de las empresas que hacen uso de la tecnología NoSQL.

Figura 21. Algunas de las empresas que utilizan tecnología NoSQL [72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89].

6.1.3 Casos de estudio, pruebas de comparación y rendimiento

Se presentan 3 experimentos con el objetivo de obtener métricas de performance para los siguientes motores de Bases de Datos: MySQL [47], MongoDB [59], Apache Cassandra [62], Neo4j [65], Redis [56], Cloud Firestore [61] y MongoDB Atlas [59].

El primer experimento tiene como objetivo analizar el rendimiento de los motores de Bases de Datos MySQL, Cassandra y MongoDB (1 motor de Base de Datos relacional y 2 motores de Bases de Datos NoSQL). Para ello, se exponen 4 casos de estudio en donde se generó aleatoriamente un volumen de datos distinto para cada uno de ellos. Además, se plantean 3 consultas diferentes, luego se expresan los resultados obtenidos y una conclusión al respecto. Las 3 consultas planteadas se basan en el mismo esquema físico de datos para los 4 casos de estudio, la diferencia es el volumen de datos generado aleatoriamente para cada uno de ellos.

El segundo experimento tiene como objetivo analizar el rendimiento de los motores de Bases de Datos MySQL, MongoDB, Apache Cassandra, Neo4j y Redis (1 motor de Bases de Datos relacional y 4 motores de Bases de Datos NoSQL). En este experimento también se exponen 4 casos de estudio, pero cada caso de estudio posee su propio esquema físico de datos para cada motor evaluado, distinto volumen de datos generados aleatoriamente y sus propias consultas.

Además, en este segundo experimento, los volúmenes de datos generados aleatoriamente son más elevados que en el primer experimento, motivo por el cual se incluyen los motores de Bases de Datos utilizados en el primer experimento. Luego, se expresan los resultados obtenidos y una conclusión al respecto.

El tercer y último experimento tiene como objetivo evaluar dos motores de Bases de Datos NoSQL como servicios en la nube que poseen almacenamiento documental, estos motores son Cloud Firestore y MongoDB Atlas. En este experimento se expone un solo caso de estudio y se plantean 3 consultas diferentes. Luego, se expresan los resultados obtenidos y una conclusión al respecto.

En todas las experimentaciones, inicialmente, para cada caso de estudio se realizó el Modelo Conceptual de Datos utilizando un Diagrama Entidad Relación (DER) y posteriormente se realizó la derivación a cada uno de los esquemas físicos correspondientes a cada motor de Base de Datos.

En MySQL (motor de Base de Datos Relacional) se generó el modelo relacional.

En MongoDB (motor de Base de Datos NoSQL) las entidades se representan mediante colecciones de documentos BJSON (JSON binario) y las relaciones se expresan, en caso de que sea necesario, mediante documentos o arreglos de documentos embebidos.

En Apache Cassandra (motor de Base de Datos NoSQL con almacenamiento clave-valor y familia de columnas), el esquema físico para el almacenamiento de sus datos, no se piensa en términos de entidades y relaciones, sino que se representa en términos de consultas, por lo tanto, es importante definir previamente cuáles serán las consultas que se realizarán.

En Neo4j (motor de Base de Datos NoSQL con almacenamiento de grafos), las entidades se representan mediante nodos que pertenecen al grafo y las relaciones son las aristas entre estos nodos. No todos los casos de estudio o escenarios presentan características adecuadas para ser derivados a una representación de grafos. Debido a esto, no se ha incluido Neo4j en el primer experimento.

En Redis (motor de Base de Datos NoSQL con almacenamiento clave-valor), al ser una Base de Datos que opera principalmente en memoria primaria, se generó el conjunto de claves que son necesarias tener vigentes de acuerdo con las consultas a realizar. Redis, solamente fue evaluado en un solo caso de estudio que pertenece al segundo experimento presentado. El gran volumen de datos utilizado, en el resto de los casos de estudio, no es posible de gestionarlo de forma adecuada en la memoria principal. Por este motivo, Redis, fue evaluado en un caso de estudio exclusivo y comparado con el motor de Base de Datos relacional MySQL.

En el caso de los motores de Bases de Datos NoSQL como servicios en la nube, Cloud Firestore y MongoDB Atlas, las entidades fueron representadas como colecciones de documentos y se utilizó unos de los conjuntos de datos que fue generado aleatoriamente para MongoDB. Esto se

pudo llevar a cabo, debido a que los tres motores de Bases de Datos poseen almacenamiento documental.

La ejecución de todas las pruebas se realizó utilizando la misma memoria y sistema operativo (4GB de RAM, Ubuntu versión 18.04). En MySQL se utilizó la versión 5.7, en MongoDB la versión 4.0, en Cassandra la versión 3.11, en Neo4j la versión 4.1 y en Redis la versión 4.0.

6.1.3.1 EXPERIMENTO I

En este primer experimento, los 4 casos de estudio presentados se basan en el mismo esquema conceptual de datos, se aplican las mismas consultas, pero se han generado aleatoriamente diferentes volúmenes de datos para cada caso.

En este experimento se decidió no incluir al motor de Base de Datos Neo4j (con almacenamiento Orientado a Grafos) y no incluir a Redis (con almacenamiento Clave-Valor); esto se debe a que las características de los casos de estudios presentados no presentan características compatibles y/o adecuadas para aplicar en estos motores de Bases de Datos NoSQL.

El esquema conceptual planteado a continuación, representa una empresa que nuclea reservas de viajes online para un conjunto de agencias de turismo. Además, se necesita llevar un registro de sus clientes. Para cada reserva es importante conocer en qué agencia fue realizada, desde qué ciudad de origen será el viaje y cuál es la ciudad de destino de dicha reserva. La figura 22 presenta el esquema conceptual utilizando un Diagrama de Entidad - Relación (DER) para el contexto planteado.

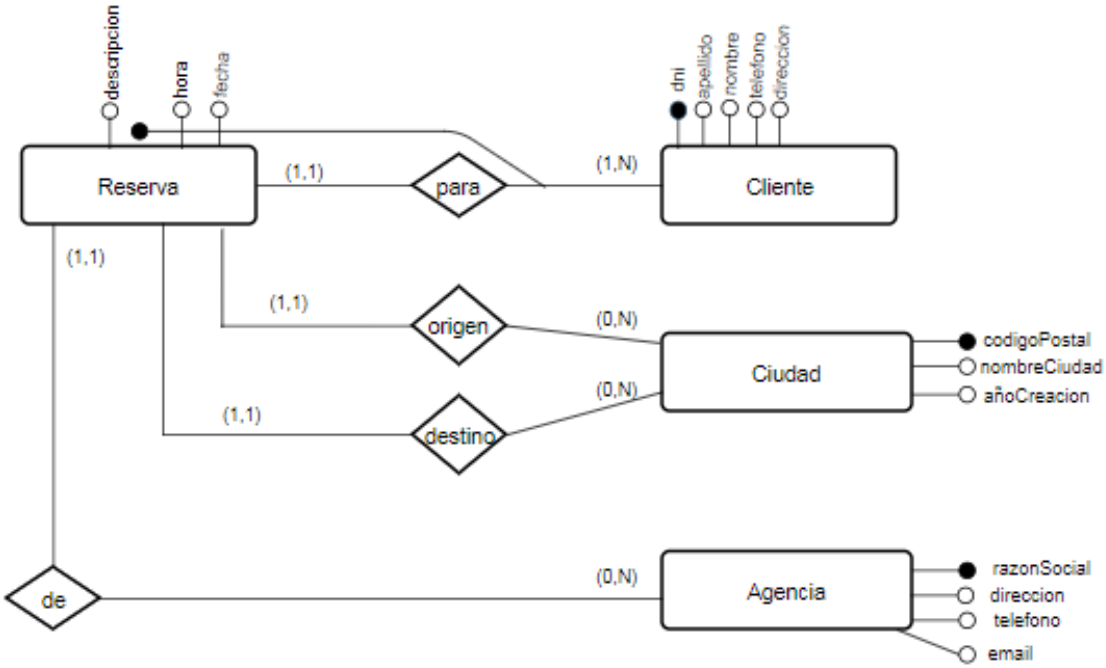

Figura 22. Esquema conceptual expresado mediante un Diagrama Entidad - Relación.

A continuación, se presentan los esquemas físicos resultantes para los motores de Bases de Datos incluidos en este primer experimento (MySQL, MongoDB y Apache Cassandra) y que son el resultado de un proceso de derivación del Diagrama Entidad-Relación expresado en la figura 22.

La figura 23 presenta el esquema relacional para el motor de Base de Datos MySQL. Previamente, se realizó el esquema lógico relacional y después se derivó al esquema relacional. No obstante, el esquema lógico relacional no presentó cambios respecto al esquema conceptual planteado en la figura 22.

```
AGENCIA = (RAZONSOCIAL(PK), DIRECCION, TELEFONO, E-MAIL)
CIUDAD  = (CODIGOPOSTAL(PK), NOMBRE, ANIOCREACION)
CLIENTE = (DNI(PK), NOMBRE, APELLIDO, TELEFONO, DIRECCION)
RESERVA = ((FECHA, HORA, DNI(FK))(PK), CPORIGEN(FK), CPDESTINO(FK),
           RAZONSOCIAL(FK), DESCRIPCION)
```

Figura 23. Esquema Relacional generado para MySQL.

En el esquema relacional resultante de la figura 23, la clave primaria de cada relación se expresa mediante la sigla en inglés PK (Primary Key) y las claves foráneas mediante la sigla en inglés FK (Foreing Key).

El esquema físico en MongoDB quedó determinado por una colección que conforma a las reservas en el esquema conceptual de datos. En el caso de las relaciones con cardinalidad de

uno a muchos, se representan utilizando documentos embebidos. La figura 24 presenta la estructura documental generada para MongoDB.

```
RESERVA : {
  _ID,
  HORA,
  FECHA,
  AGENCIA: {E-MAIL, TELEFONO, DIRECCION, RAZONSOCIAL},
  CLIENTE: {DNI, NOMBRE, APELLIDO, TELEFONO, DIRECCION},
  DESCRIPCION,
  ORIGEN: {NOMBRE, CODIGOPOSTAL, ANIOCREACION},
  DESTINO: {NOMBRE, CODIGOPOSTAL, ANIOCREACION}
}
```

Figura 24. Estructura del documento en MongoDB para la colección de reservas.

Cada documento que se genera en la colección posee un campo "_ID" que se utiliza para identificar de forma unívoca a dicho documento dentro de la colección correspondiente. El "_ID" es la clave principal para los documentos dentro de la colección. Los campos "Agencia", "Cliente", "Origen" y "Destino", son documentos embebidos que poseen su propia estructura de campos.

Para el motor de Base de Datos Apache Cassandra, el esquema físico de información se expresa de acuerdo con las consultas a realizar. La figura 25 plantea el esquema físico resultante en Apache Cassandra de acuerdo con el esquema de datos y a las consultas que impactarán contra dicho esquema de datos.

```
RESERVAS  = ((DNI, FECHA, HORA)(PK), APELLIDO, NOMBRE, ORIGEN,
             DESTINO, RAZONSOCIAL)
PROBLEMAS = ((DNI, FECHA, HORA)(PK), ORIGEN, DESTINO, DESCRIPCION,
             RAZONSOCIAL)
```

Figura 25. Esquema físico propuesto en Apache Cassandra para el sistema de reservas.

En este esquema físico para Apache Cassandra, se conforman dos tablas a través de las cuales se pueden responder las consultas que se plantean para este caso de estudio. La clave primaria de cada tabla se expresa mediante la sigla en inglés PK (Primary Key).

A continuación, se plantean 3 consultas para las pruebas de rendimiento en cada caso de estudio perteneciente a este primer experimento:

- **Consulta I.1 (I.1)**: Reportar los datos del cliente y la cantidad de reservas realizadas para cada cliente.
- **Consulta I.2 (I.2)**: Reportar los datos de las reservas en las que se describe algún problema.
- **Consulta I.3 (I.3)**: Reportar los datos de las reservas realizadas por un cliente en particular.

A continuación, se presenta la implementación de cada una de estas 3 consultas en los 3 motores de Bases de Datos utilizados en este experimento I.

6.1.3.1.1 Implementaciones de la consulta I.1

En la figura 26, se muestra la implementación de la consulta I.1 en SQL (Structured Query Language) para el motor de Base de Datos Relacional MySQL.

```
MYSQL:

SELECT C.DNI, C.NOMBRE, C.APELLIDO, COUNT(*) AS CANTIDAD
FROM CLIENTE C INNER JOIN RESERVA R ON (C.DNI = R.DNI)
GROUP BY C.DNI, C.NOMBRE, C.APELLIDO
```

Figura 26. Implementación de la consulta I.1 en SQL para MySQL.

En este caso, para implementar la consulta I.1 en SQL, es necesario realizar una operación de JOIN entre la relación de clientes y la relación de reservas. Además, requiere de aplicar un agrupamiento por DNI de cliente para contar y proyectar la cantidad de reservas realizadas por cada cliente.

En la figura 27, se presenta la implementación de la consulta I.1 utilizando CQL (Cassandra Query Language) para el motor de Base de Datos Orientado a Columnas Apache Cassandra.

```
CASSANDRA:

SELECT DNI, NOMBRE, APELLIDO, COUNT(DNI) AS CANTIDAD
FROM RESERVAS
GROUP BY C.DNI
```

Figura 27. Implementación de la consulta I.1 en CQL para Apache Cassandra.

La implementación de la I.1 en CQL es similar a la implementación presentada para MySQL en SQL. En este caso, no es necesario realizar una operación de JOIN, esto se debe a que en Apache Cassandra la tabla que contiene las reservas fue generada con el objetivo de responder a las consultas correspondientes sin necesidad de operaciones de JOIN.

En la figura 28, se presenta la implementación de consulta I.1 para el motor de Base de Datos Documental MongoDB.

Para la resolución de las consultas en MongoDB, se utiliza el MongoDB Shell (mongosh). Este es un entorno que se basa en JavaScript y Node.js completamente funcional para interactuar con implementaciones de MongoDB. Se puede utilizar MongoDB Shell para realizar consultas y operaciones directamente con la Base de Datos.

```
MONGODB:

DB.RESERVAS.AGGREGATE([ {$GROUP : { _ID: { DNI:
"$CLIENTE.DNI", NOMBRE: "$CLIENTE.NOMBRE", APELLIDO:
"$CLIENTE.APELLIDO" }, "CANTIDAD": { $SUM: 1 } } } ])
```

Figura 28. Implementación de la consulta I.1 en MongoDB Shell.

Para la resolución de C.I.1 en MongoDB se utiliza la operación de agregación (aggregate). Las operaciones de agregación permiten procesar varios documentos y retornar resultados calculados.

6.1.3.1.2 Implementaciones de la consulta I.2

En la figura 29, se presenta la implementación de la consulta I.2 en SQL (Structured Query Language) para el motor de Base de Datos Relacional MySQL.

```
MYSQL:

SELECT R.FECHA, R.HORA, CO.NOMBRE, CD.NOMBRE, R.RAZONSOCIAL,
       R.DESCRIPCION
FROM RESERVA R INNER JOIN CIUDAD CO ON (CO.CODIGOPOSTAL =
     R.CPORIGEN) INNER JOIN CIUDAD CD ON (CD.CODIGOPOSTAL =
     R.CPDESTINO)
WHERE (DESCRIPCION LIKE "%PROBLEMA%")
```

Figura 29. Implementación de la consulta I.2 en SQL para MySQL.

En este caso, para implementar I.2 en SQL, es necesario realizar dos operaciones de JOIN entre la relación reserva y la relación ciudad. Esto se debe a que es necesario obtener la ciudad de origen y la ciudad de destino de cada reserva. Además, en la resolución propuesta, se aplica un filtro utilizando el operador "LIKE" de SQL, el cual permite realizar la búsqueda de un patrón específico dentro de una cadena de texto. El operador "LIKE" se puede combinar con caracteres que se denominan comodines, como, por ejemplo, el signo "%", lo que significa que la cadena puede contener cualquier otra cadena en donde se encuentre dicho signo. Para la consulta C.I.2 en MySQL, el filtro aplicado se corresponde con aquellas reservas que contienen la palabra "PROBLEMA" en cualquier parte de su descripción.

En la figura 30, se presenta la implementación de la consulta I.2 utilizando CQL (Cassandra Query Language) para el motor de Base de Datos Orientado a Columnas Apache Cassandra.

```
CASSANDRA:

SELECT *
FROM PROBLEMAS
```

Figura 30. Implementación de la consulta I.2 en CQL para Apache Cassandra.

La implementación de I.2 en CQL difiere a la implementación presentada para MySQL en SQL. En este caso, no es necesario realizar operaciones de JOIN entre tablas, esto se debe a que en Apache Cassandra se generó una tabla específica de problemas con el objetivo de responder a esta consulta sin necesidad de operaciones de JOIN.

En la figura 31, se presenta la implementación de consulta I.2 para el motor de Base de Datos Documental MongoDB utilizando MongoDB Shell.

```
MONGODB:

DB.RESERVAS.FIND( { DESCRIPCION: { $REGEX:
/PROBLEMA/, $OPTIONS: "I" } }, { FECHA: 1, HORA: 1,
"ORIGEN.NOMBRE": 1, "DESTINO.NOMBRE": 1,
"AGENCIA.RAZONSOCIAL": 1, DESCRIPCION: 1, _ID: 0 })
```

Figura 31. Implementación de la consulta I.2 en MongoDB Shell.

Para la resolución de I.2 en MongoDB se utiliza la operación de búsqueda "FIND" que brinda MongoDB Shell y permite filtrar documentos de una colección. Se utiliza el operador "$REGEX", que es análogo al operador "LIKE" en SQL.

6.1.3.1.3 Implementaciones de la consulta I.3

En la figura 32, se muestra la implementación de la consulta I.3 en SQL (Structured Query Language) para el motor de Base de Datos Relacional MySQL

```
MYSQL:

SELECT R.FECHA, R.HORA, CO.NOMBRE, CD.NOMBRE, R.RAZONSOCIAL
FROM RESERVA R INNER JOIN CIUDAD CO ON (CO.CODIGOPOSTAL =
     R.CPORIGEN) INNER JOIN CIUDAD CD ON (CD.CODIGOPOSTAL =
     R.CPDESTINO)
WHERE (R.DNI = 28764545)
```

Figura 32. Implementación de la consulta I.3 en SQL para MySQL.

En este caso, para implementar I.3 en SQL, se necesitan las mismas operaciones de JOIN que en I.2. Esto se debe a que hay que obtener la proyección de la ciudad de origen y la ciudad de destino de cada reserva realizada por un cliente en particular. En esta consulta I.3, se aplica una condición de filtro para obtener solamente las reservas realizadas por un cliente en particular.

En la figura 33, se presenta la implementación de la consulta I.3 utilizando CQL (Cassandra Query Language) para el motor de Base de Datos Orientado a Columnas Apache Cassandra.

```
CASSANDRA:

SELECT FECHA, HORA, ORIGEN, DESTINO, RAZONSOCIAL
FROM RESERVAS
WHERE DNI = 28764545
```

Figura 33. Implementación de la consulta I.3 en CQL para Apache Cassandra.

La implementación de I.3 en CQL es similar a la implementación presentada para MySQL en SQL. En este caso, no es necesario realizar una operación de JOIN, esto se debe a que en Apache Cassandra la tabla que contiene las reservas fue generada con el objetivo de responder a las consultas correspondientes sin necesidad de operaciones de JOIN.

En la figura 34, se presenta la implementación de la consulta I.3 para el motor de Base de Datos Documental MongoDB utilizando MongoDB Shell.

```
MONGODB:

DB.RESERVAS.FIND( { "CLIENTE.DNI": 28764545 }, {
FECHA: 1, HORA: 1, "ORIGEN.NOMBRE": 1,
"DESTINO.NOMBRE": 1, "AGENCIA.RAZONSOCIAL": 1, _ID: 0
} )
```

Figura 34. Implementación de la consulta I.3 en MongoDB Shell.

Para la resolución de esta consulta en MongoDB se utiliza la operación de búsqueda "FIND" que brinda MongoDB Shell y permite filtrar documentos de una colección.

A continuación, se presentan los resultados obtenidos para cada prueba de performance en cada uno de los 4 casos de estudio y para los 3 motores de Bases de Datos.

6.1.3.1.4 Caso de estudio I.A

Para este primer caso de estudio se han generado 1.260.000 reservas online de forma aleatoria.

A continuación, en la tabla 1 se expresan los tiempos medidos en segundos de la ejecución de cada consulta para los motores de Bases de Datos MySQL, Apache Cassandra y MongoDB. En este caso de estudio se conformó un conjunto de 9 pruebas de performance.

Tabla 1. Resultados del caso de estudio I.A expresados en segundos.

Consultas	*MySQL*	*MongoDB*	*Cassandra*
I.1	6,9	34,31	9,83
I.2	61,62	14,90	4,73
I.3	0,52	0,85	0,06

6.1.3.1.5 Caso de estudio I.B

En este segundo caso de estudio el volumen de reservas online generadas aleatoriamente es de 4.233.386 reservas.

A continuación, en la tabla 2 se expresan los tiempos medidos en segundos de la ejecución de cada consulta para los motores de Bases de Datos MySQL, Apache Cassandra y MongoDB. En este caso de estudio se conformó un conjunto de 9 pruebas de performance.

Tabla 2. Resultados del caso de estudio I.B expresados en segundos.

Consultas	*MySQL*	*MongoDB*	*Cassandra*
I.1	22,5	77,28	53,41
I.2	224,29	51,52	11,6
I.3	0,56	2,1	0,12

6.1.3.1.6 Caso de estudio I.C

En este tercer caso de estudio se ha aumentado el volumen de reservas online, generadas aleatoriamente, a 10.473.392 reservas.

A continuación, en la tabla 3 se expresan los tiempos medidos en segundos de la ejecución de cada consulta para los motores de Bases de Datos MySQL, Apache Cassandra y MongoDB. En este caso de estudio se conformó un conjunto de 9 pruebas de performance.

Tabla 3. Resultados del caso de estudio I.C expresados en segundos.

Consultas	*MySQL*	*MongoDB*	*Cassandra*
I.1	52,60	145,12	78,6
I.2	768,45	97,96	59,51
I.3	1,11	3,5	0,13

6.1.3.1.7 Caso de estudio I.D

En este cuarto y último caso de estudio se ha aumentado el volumen de reservas online, generadas aleatoriamente, a 20.120.254 reservas.

A continuación, en la tabla 4 se expresan los tiempos medidos en segundos de la ejecución de cada consulta para los motores de Bases de Datos MySQL, Apache Cassandra y MongoDB. En este caso de estudio se conformó un conjunto de 9 pruebas de performance.

Tabla 4. Resultados del caso de estudio I.D expresados en segundos.

Consultas	*MySQL*	*MongoDB*	*Cassandra*
I.1	128,44	198,59	139,23
I.2	1948,12	132,69	75,21
I.3	1,64	3,93	0,18

6.1.3.1.8 Discusión de los resultados obtenidos.

Para la primera consulta I.1 en los 4 casos de estudio, se observa que MySQL obtiene un mejor tiempo de respuesta. No obstante, a medida que aumenta el volumen de datos en los distintos casos presentados, los tiempos de respuesta se acercan a los obtenidos con Apache Cassandra. Estas consultas requieren el uso de la función de agregación COUNT y se debe tener en cuenta que MySQL optimiza el uso de las funciones de agregación en relación con los otros 2 motores de Bases de Datos. Esta podría ser la justificación por la cual su desempeño es mejor.

Para la segunda consulta I.2 en los 4 casos de estudio, Apache Cassandra logra el mejor tiempo de respuesta. Esto podría justificarse debido a que los esquemas físicos en Apache Cassandra se generan en base a las consultas del sistema. En este caso, Apache Cassandra posee una tabla diseñada específicamente en términos de estas consultas. MongoDB se ubica en segundo lugar, dado que tiene que realizar un filtro de documentos, pero sin necesidad de operaciones de JOIN. En este caso MySQL es el menos eficiente, esto se debe a que el filtro aplicado retorna un conjunto de datos considerable, a los cuales se les aplica operaciones de JOIN, afectando de este modo el tiempo de respuesta.

En la tercera consulta I.3, a pesar del incremento en el volumen de información, los tiempos de respuesta poseen una performance similar para todos los casos de estudio, independientemente del volumen de datos. Esto se debe a que en la implementación de la consulta en todos los motores de Bases de Datos se aplica un filtro por un atributo que se encuentra indexado. Finalmente, los tiempos resultantes ubican a Apache Cassandra en el mejor lugar, seguida por MySQL y MongoDB.

6.1.3.2 EXPERIMENTO II

En este segundo experimento se presentan 4 casos de estudio y para cada caso se plantea un esquema conceptual diferente. Para cada uno de ellos se realizó el Diagrama de Entidad Relación (DER) y posteriormente, se realizó la derivación al esquema físico correspondiente a cada motor de Base de Datos utilizado.
En este experimento se utilizó MySQL (motor de Base de Datos Relacional), MongoDB (motor de Base de Datos Documental), Apache Cassandra (motor de Base de Datos de Familia de Columnas), Neo4j (motor de Base de Datos orientado a Grafos) y Redis (motor de Base de Datos Clave-Valor).

Además, se ha generado un gran volumen de datos para todos los casos de estudio y los modelos y esquemas físicos de información sobre el cual se impactan las consultas son diferentes.

En los 3 primeros casos de estudios presentados no fue posible evaluar el motor de Base de Datos Redis debido a que el gran volumen de datos utilizado no se puede gestionar adecuadamente en memoria principal.

A continuación, se presentan los 4 casos de estudio realizados para este experimento.

6.1.3.2.1 Caso de estudio II.A

En este primer caso de estudio, el esquema conceptual representa a una bandeja de entrada para un sistema de mensajería interna. En este caso se tienen los usuarios y los mensajes que se envían entre ellos. Para este caso de estudio se han generado de forma aleatoria 30.000.000 de mensajes en total correspondientes a 6.000.000 de usuarios distintos. La figura 35 presenta el Diagrama Entidad Relación (DER) para este primer caso de estudio.

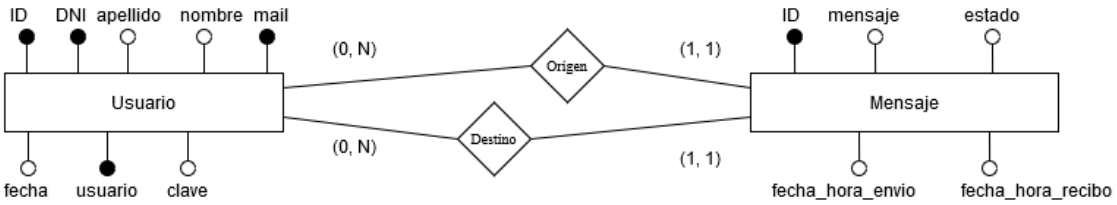

Figura 35. Esquema conceptual expresado mediante un DER para el sistema de mensajería interna.

La figura 36 presenta el modelo relacional para el motor de Base de Datos MySQL resultante del esquema conceptual presentado. Previamente, se realizó el esquema lógico relacional y después se derivó al esquema relacional. No obstante, el esquema lógico relacional no presentó cambios respecto al esquema conceptual de datos descripto.

```
USUARIO = (ID(PK), DNI, APELLIDO, NOMBRE, MAIL, FECHA, USUARIO,
           CLAVE)
MENSAJE = (ID(PK), MENSAJE, ESTADO, FHENVIO, FHRECIBO, UORIGEN(FK),
           UDESTINO(FK)
```

Figura 36. Esquema relacional para MySQL.

En este esquema relacional la clave primaria de cada relación se expresa mediante la sigla en inglés PK (Primary Key) y las claves foráneas mediante la sigla en inglés FK (Foreing Key).

La figura 37 presenta el esquema físico para la estructura documental de MongoDB. En este caso, el esquema físico quedó conformado por una colección que contiene el conjunto de los documentos que representa a los mensajes y las relaciones con cardinalidad de uno a muchos fueron derivadas utilizando documentos embebidos.

```
MENSAJE : {
  _ID,
  MENSAJE,
  ESTADO,
  FHENVIO,
  FHRECIBO,
  UORIGEN: {DNI, APELLIDO, NOMBRE, MAIL, USUARIO},
  UDESTINO: {DNI, APELLIDO, NOMBRE, MAIL, USUARIO}
}
```

Figura 37. Estructura de un documento en MongoDB.

Cada documento que se genera en la colección posee un campo "*_ID*" que se utiliza para identificar de forma unívoca a dicho documento dentro de la colección que corresponda. Es la clave principal para los documentos dentro de una colección. En este caso el campo “UORIGEN” Y “UDESTINO” son documentos embebidos que representan al usuario que envía el mensaje (UORIGEN) y el usuario que recibe el mensaje (UDESTINO) respectivamente.

En el caso de Apache Cassandra, el esquema físico de información se expresa de acuerdo con las consultas a realizar. La figura 38 plantea el esquema físico resultante en Apache Cassandra de acuerdo con el esquema conceptual propuesto para este caso de estudio y a las consultas que impactarán contra dicho esquema.

```
MENSAJE = ((UORIGEN, UDESTINO, FHENVIO)(PK), FHRECIBO, OMAIL, DMAIL,
          MENSAJE, ESTADO)
```

Figura 38. Esquema físico para Apache Cassandra.

Para el esquema físico de Apache Cassandra se conformó una tabla a través de la cual se pueden responder a las 3 consultas que impactarán en este caso de estudio.

Para el caso de Neo4j, se crearon nodos para representar a los usuarios y nodos para representar a los mensajes. En la figura 39 se muestra una parte del grafo que se generó para este caso de estudio. Los nodos de color azul representan a los usuarios, mientras que los grises representan a los mensajes. Las aristas representan las relaciones de envío y recepción de los mensajes entre usuarios.

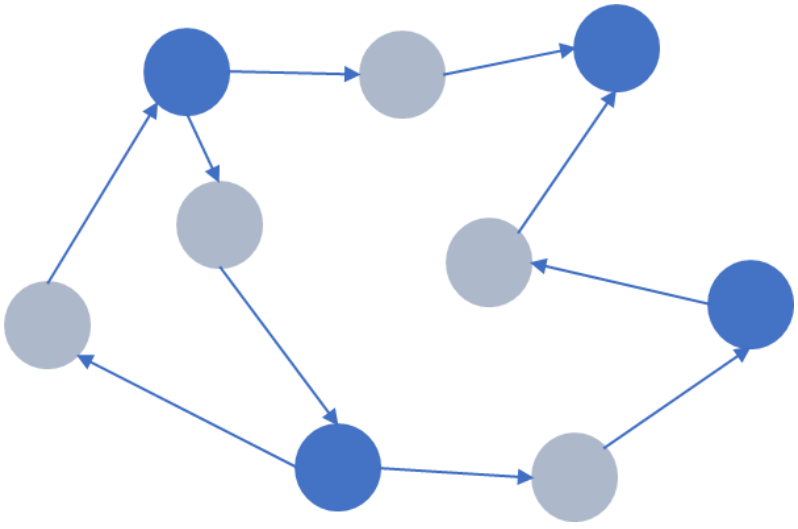

Figura 39. Fragmento del grafo que se generó en Neo4j para este primer caso de estudio.

A continuación, se presentan 3 consultas para realizar las pruebas de rendimiento en este primer caso de estudio (II.A):

- **Consulta II.A.1 (II.A.1)**: Mensajes enviados y/o recibidos entre dos usuarios específicos.
- **Consulta II.A.2 (II.A.2)**: Buscar una cadena o patrón de caracteres entre los mensajes enviados y/o recibidos de un usuario específico.

- **Consulta II.A.3 (II.A.3)**: Mensajes recibidos, pero no leídos por un usuario específico.

A continuación, se presenta la implementación de cada una de las 3 consultas en los 4 motores de Bases de Datos utilizados en este primer caso de estudio del experimento II.

La figura 40 muestra la implementación de la consulta II.A.1 en SQL (Structured Query Language) para el motor de Base de Datos Relacional MySQL.

```
MYSQL:

SELECT MENSAJE, FHENVIO, ESTADO
FROM MENSAJE M INNER JOIN USUARIO O ON (O.ID = M.UORIGEN)
     INNER JOIN USUARIO D ON (D.ID = M.UDESTINO)
WHERE ((O.USUARIO = "USER256") AND (D.USUARIO = "USER1")) OR
      ((O.USUARIO = "USER1") AND (D.USUARIO = "USER256"))
ORDER BY FHENVIO DESC
```

Figura 40. Implementación de la consulta II.A.1 en SQL para MySQL.

En este caso, para implementar la consulta II.A.1 en SQL, se realizan dos operaciones de JOIN entre la relación de mensajes y la relación de usuarios. Esto se debe a que se necesita obtener los mensajes entre dos usuarios específicos, sin discriminar el usuario emisor y el usuario receptor para el mensaje correspondiente. Además, se aplica un ordenamiento descendente del resultado por fecha en que ha sido enviado el mensaje.

La figura 41 presenta la implementación de la consulta II.A.1 utilizando CQL (Cassandra Query Language) para el motor de Base de Datos Apache Cassandra.

```
CASSANDRA:

SELECT MENSAJE, FHENVIO, ESTADO
FROM MENSAJE
WHERE(UORIGEN IN ("USER256", "USER1")) AND (UDESTINO
      IN ("USER1", "USER256"))
ORDER BY FHENVIO DESC
```

Figura 41. Implementación de la consulta II.A.1 en CQL para Apache Cassandra.

La implementación de II.A.1 en CQL es similar a la implementación presentada para MySQL en SQL. En este caso, no es necesario realizar una operación de JOIN, esto se debe a que en Apache Cassandra la tabla que contiene los mensajes fue generada con el objetivo de responder a las consultas correspondientes sin necesidad de operaciones de JOIN.

La figura 42 muestra la implementación de la consulta II.A.1 para el motor de Base de Datos Documental MongoDB utilizando MongoDB Shell.

```
MONGODB:

DB.MENSAJES.FIND({ $OR: [ { $AND: [ {
"UORIGEN:USUARIO": "USER1" }, { "UDESTINO.USUARIO":
"USER256" } ] }, { $AND: [ { "UORIGEN.USUARIO":
"USER256" }, { "UDESTINO.USUARIO": "USER1" } ] } ] },
{ MENSAJE: 1, FHENVIO: 1, ESTADO: 1 }).SORT({
FHENVIO: -1 })
```

Figura 42. Implementación de la consulta II.A.1 para MongoDB.

Para la resolución de la consulta II.A.1 en MongoDB se utiliza la operación de búsqueda "FIND" que brinda MongoDB Shell y permite filtrar documentos de una colección. Además, se aplica el operador "SORT" con el parámetro -1 para ordenar la colección de documentos resultantes de forma descendente.

La figura 43 muestra la implementación de la consulta II.A.1 para el motor de Base de Datos Neo4j utilizando el lenguaje Cypher.

Cypher es el lenguaje de consulta para Neo4j que le permite recuperar datos de una Base de Datos orientada a grafos. Este lenguaje está inspirado en SQL y permite concentrarse en los datos que se desean obtener del grafo sin tener que pensar en cómo obtenerlos. Se considera un lenguaje sencillo de aprender e incorporar por su similitud con SQL.

```
NEO4J:

MATCH (U1: USUARIO)-[R1]-(M: MENSAJE)-[R2]-(U2:
       USUARIO)
WHERE (ID(U1) = 34000550) AND (ID(U2) = 34000805)
RETURN U1.USUARIO, R1, M, R2, U2.USUARIO;
```

Figura 43. Implementación de la consulta II.A.1 en Neo4j.

La cláusula "MATCH" de Cypher permite especificar los patrones que Neo4j buscará en la Base de Datos. Esta es la forma principal de obtener datos en el conjunto actual de enlaces dentro del Grafo. A continuación, se presenta la tabla 5 en donde se expresan los tiempos obtenidos en segundos de la ejecución de las 3 consultas en los 4 motores de Bases de Datos (MySQL, MongoDB, Cassandra y Neo4j). En este caso de estudio se conformó un conjunto de 12 pruebas de performance.

Tabla 5. Resultados del caso de estudio II.A expresados en segundos.

Consultas	*MySQL*	*MongoDB*	*Cassandra*	*Neo4j*
II.A.1	0,33	385	0,35	0.24
II.A.2	259	178	9	0.15
II.A.3	20	166	72	0.24

6.1.3.2.2 Caso de estudio II.B

En este segundo caso de estudio la información que se representa mediante un esquema conceptual corresponde a una central de monitoreo que necesita registrar y procesar eventos procedentes de dos fuentes diferentes (tránsito y meteorológica). Para este caso de estudio se han generado aleatoriamente 40.000.000 de eventos en total. En la figura 44 se presenta el Diagrama Entidad Relación (DER) para la situación mencionada.

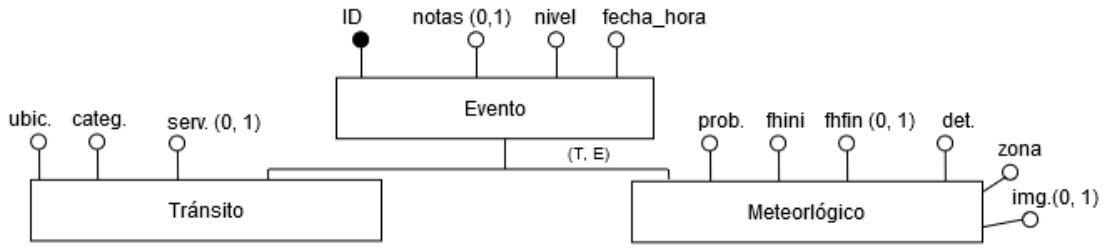

Figura 44. Esquema conceptual, expresado mediante un DER, para el presente caso de estudio.

Para representar conceptualmente a los eventos de tránsito y eventos meteorológicos se optó por utilizar una jerarquía de herencia con cobertura total exclusiva. Esto se debe a que ambos tipos de eventos comparten características similares entre sí.

La figura 45 presenta el modelo relacional (MySQL) resultante del esquema conceptual presentado. Previamente, se realizó el esquema lógico relacional y después se derivó al esquema relacional. Para el esquema lógico relacional se transforma la jerarquía de herencia dejando las tres entidades bajo el concepto de relación “es_un”.

```
EVENTO      = (ID(PK), NOTAS?, NIVEL, FECHAHORA)
TRANSITO    = (ID(PK)(FK), UBICACION, CATEGORIA, SERVICIO?)
METEOROLOGICO = (ID(PK)(FK), PROBABILIDAD, FHINICIO, FHFIN?,
                 DETALLE, ZONA, IMAGEN?)
```

Figura 45. Esquema relacional para MySQL.

En este esquema relacional, la clave primaria de cada relación se expresa mediante la sigla en inglés PK (Primary Key) y las claves foráneas mediante la sigla en inglés FK (Foreing Key). Además, los atributos que son opcionales se destacan por un signo de pregunta (?).

Para el motor de Base de Datos MongoDB, se generó una colección de eventos, en donde cada documento representa un evento, de tránsito o meteorológico, con los campos que correspondan en cada caso. En la figura 46 se presenta un ejemplo para la estructura documental de MongoDB para el caso de un evento de tránsito.

```
EVENTO : {
  _ID,
  ID,
  TIPO,
  NIVEL,
  FECHA,
  HORA,
  UBICACION,
  CATEGORIA
}
```

Figura 46. Estructura de un documento en MongoDB para un evento de tránsito.

Para el modelo físico de Apache Cassandra, se generó, como en los casos de estudio anteriores, teniendo en cuenta las consultas planteadas.

Para generar el grafo en Neo4j se generaron nodos que pertenecen a la categoría de eventos de tránsito y nodos que pertenecen a la categoría de eventos meteorológicos. Este caso de estudio II.B, no resulta adecuado para implementar mediante almacenamiento orientado a grafos, por ese motivo, el modelo físico generado en Neo4j puede no ser el apropiado.

A continuación, se plantean 3 consultas para realizar las pruebas de rendimiento en este caso de estudio II.B:

- **Consulta II.B.1 (II.B.1)**: Eventos meteorológicos que superen una determinada probabilidad de ocurrencia.
- **Consulta II.B.2 (II.B.2)**: Eventos de tránsito con una determinada categoría en un período de tiempo.
- **Consulta II.B.3 (II.B.3)**: Cantidad de eventos con un determinado nivel de prioridad (Alto, Medio o Bajo).

Los esquemas físicos y la resolución de consultas correspondientes a cada uno de los motores de Bases de Datos utilizados en este caso de estudio se generaron siguiendo las mismas reglas aplicadas al primer caso de estudio (II.A).

La tabla 6 expresa los tiempos obtenidos en segundos de la ejecución de cada consulta. En este caso de estudio se conformó un conjunto de 12 pruebas de performance.

Tabla 6. Resultados del caso de estudio II.B expresados en segundos.

Consultas	*MySQL*	*MongoDB*	*Cassandra*	*Neo4j*
II.B.1	200	119	123	360
II.B.2	233	117	120	134
II.B.3	81	106	200	288

6.1.3.2.3 Caso de estudio II.C

Para este tercer caso de estudio, el esquema conceptual que se plantea es para representar la información de vuelos pertenecientes a diferentes aerolíneas y con diversos aeropuertos de origen y destino.

En este caso, se ha generado aleatoriamente la información sobre 40.000.000 de vuelos entre 10.000 aeropuertos distintos. La figura 47 presenta el Diagrama Entidad Relación (DER) para la situación mencionada.

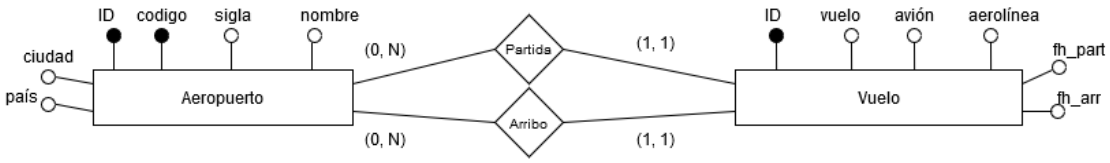

Figura 47. Esquema conceptual expresado mediante un DER.

La figura 48 presenta el esquema relacional (MySQL) resultante para el esquema conceptual planteado. Previamente, se realizó el esquema lógico relacional y después se derivó al esquema relacional. No obstante, el esquema lógico relacional no presentó cambios respecto al esquema conceptual de datos generado.

```
AEROPUERTO = (ID(PK), CODIGO(CK), SIGLA, NOMNBRE, CIUDAD, PAIS)
VUELO      = (ID(PK), VUELO, AVION, AEROLINEA, FHPARTIDA, FHARRIBO,
              AORIGEN(FK), ADESTINO(FK))
```

Figura 48. Esquema relacional para MySQL.

Para este esquema relacional, la clave primaria de cada relación se expresa mediante la sigla en inglés PK (Primary Key), las claves foráneas mediante la sigla en inglés FK (Foreing Key) y las claves candidatas mediante la sigla en inglés CK (Candidate Key).

En la figura 49 se presenta el esquema físico para la estructura documental de MongoDB. En este caso, el esquema físico quedó conformado por una colección que contiene el conjunto de los documentos que representa a los vuelos y las relaciones con cardinalidad de uno a muchos fueron derivadas utilizando documentos embebidos. Los campos “AORIGEN” Y “ADESTINO” son documentos embebidos que representan los datos del aeropuerto de origen y el aeropuerto de destino respectivamente.

```
VUELO : {
  _ID,
  ID,
  VUELO,
  AVION,
  AEROLINEA,
  FHARRIBO,
  FHPARTIDA,
  AORIGEN: {CODIGO, SIGLA, NOMBRE, CIUDAD, PAIS},
  ADESTINO: {CODIGO, SIGLA, NOMBRE, CIUDAD, PAIS}
}
```

Figura 49. Estructura de un documento en MongoDB.

El modelo físico de Cassandra, se generó, como en los casos de estudio anteriores, teniendo en cuenta las consultas planteadas.

Para el caso de Neo4j, se crearon nodos para representar los aeropuertos y nodos para representar los vuelos. En la figura 50 se muestra una parte del grafo que se generó para este caso de estudio. Los nodos de color azul representan a los aeropuertos, mientras que los grises representan vuelos. Las aristas representan las relaciones de partidas y arribos de los vuelos entre un aeropuerto origen y un aeropuerto de destino.

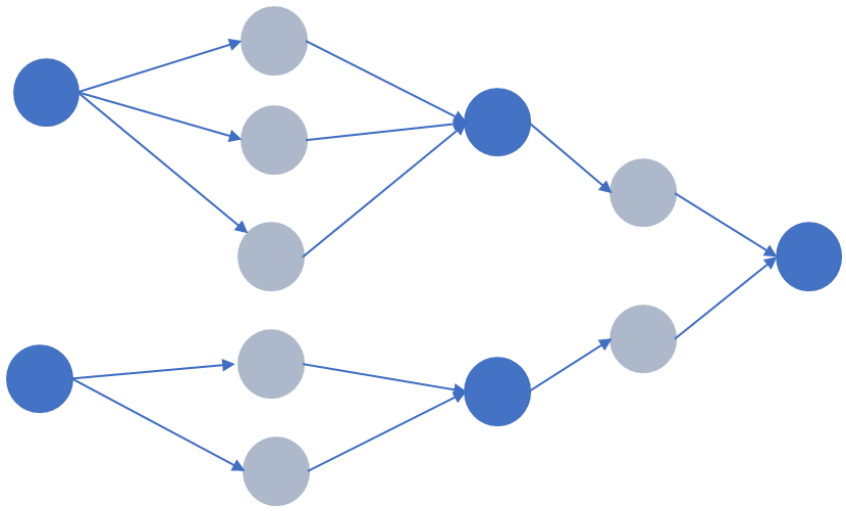

Figura 50. Fragmento del Grafo que se ha generado en Neo4j para el sistema de vuelos.

A continuación, para este caso de estudio, se plantean 2 consultas para realizar las pruebas de rendimiento:

- **Consulta II.C.1 (II.C.1)**: Vuelos directos entre un aeropuerto de origen y un aeropuerto de destino.
- **Consulta II.C.2 (II.C.2)**: Vuelos que posean una escala entre un aeropuerto de origen y un aeropuerto de destino.

Los esquemas físicos y la resolución de consultas correspondientes a cada uno de los motores de Bases de Datos utilizados en este caso de estudio se generaron siguiendo las mismas reglas aplicadas al primer caso de estudio (II.A).

La tabla 7 expresa los tiempos obtenidos en segundos de la ejecución de cada consulta. En este caso de estudio se conformó un conjunto de 9 pruebas de performance.

Tabla 7. Resultados del caso de estudio II.C expresados en segundos.

Consultas	***MySQL***	***MongoDB***	***Cassandra***	***Neo4j***
II.C.1	18	166	1	0,1
II.C.2	1720	-	1,5	0,4

En el caso de la consulta II.C.2 en MongoDB, no se encontró una implementación que logre finalizar exitosamente para poder determinar una métrica de performance.

6.1.3.2.4 Caso de estudio II.D

En este cuarto y último caso de estudio se presenta un esquema para evaluar específicamente el motor de Bases de Datos Redis en comparación con el motor de Base de Datos MySQL. El esquema conceptual planteado representa un historial de búsquedas recientes para un sitio WEB.

El motor de Base de Datos Redis (Remote Dictionary Server) es un motor de Base de Datos clave-valor que se destaca por disponer sus estructuras de datos en memoria principal, pero, además, posee la característica de que permite aplicar persistencia de datos. Redis, para lograr el máximo rendimiento, trabaja con un conjunto de datos en memoria principal y opcionalmente permite de forma configurable grabar estos datos en disco. Estas particularidades hacen que Redis sea específico y adecuado para ciertos casos de uso, como, por ejemplo, aplicaciones de almacenamiento de caché, administración de sesiones de usuario, análisis en tiempo real, entre otros.

Para este caso de estudio se han generado 100.000 usuarios y 10.000.000 búsquedas en total. La figura 51 presenta el Diagrama Entidad Relación (DER) del problema planteado.

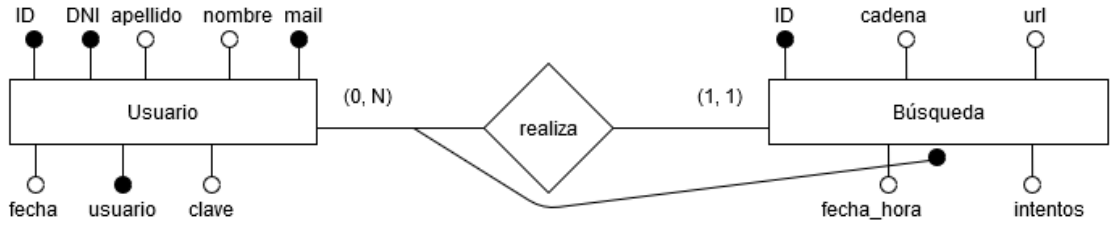

Figura 51. Esquema conceptual, expresado mediante un DER, para el presente caso de estudio.

En el esquema conceptual planteado, la entidad "Búsqueda" posee un identificador mixto, el cual se compone del identificador del usuario más la fecha y hora de la búsqueda.

La figura 52 presenta el esquema relacional (MySQL) resultante del esquema conceptual presentado. Previamente, se realizó el esquema lógico relacional y después se derivó al esquema relacional. No obstante, el esquema lógico relacional no presentó cambios respecto al modelo conceptual de datos descripto.

```
USUARIO  = (ID(PK), DNI(CK), APELLIDO, NOMBRE, MAIL(CK), FECHA,
            USUARIO(CK), CLAVE)
BUSQUEDA = (ID(PK), CADENA,URL, INTENTOS, (FHBUSQUEDA, ID(FK))(CK))
```

Figura 52. Esquema relacional para MySQL.

En este esquema relacional, la clave primaria de cada relación se expresa mediante la sigla en inglés PK (Primary Key), las claves foráneas mediante la sigla en inglés FK (Foreing Key) y las claves candidatas mediante la sigla en inglés CK (Candidate Key).

A continuación, para este caso de estudio, se plantean 2 consultas para realizar las pruebas de rendimiento:

- **Consulta II.D.1**: Obtener las 5 búsquedas más recientes para un usuario determinado.
- **Consulta II.D.2**: Obtener las 5 búsquedas más frecuentes para un usuario determinado.

El esquema físico y la resolución de consultas para MySQL se generó siguiendo las mismas reglas aplicadas que en los casos de estudios anteriores. En el caso de Redis, se generaron en memoria principal, las 10.000.000 de claves correspondientes y bajo cada clave los datos básicos obtenidos de la entidad usuario.
La tabla 8 expresa los resultados obtenidos en microsegundos. En este caso de estudio se conformó un conjunto de 4 pruebas de performance.

Tabla 8. Resultados del caso de estudio II.D expresados en **microsegundos**.

Consultas	*MySQL*	*Redis*
C.II.D.1	580000 (0.58 seg.)	0,48
C.II.D.2	550000 (0.55 seg.)	0,53

6.1.3.2.5 Discusión de los resultados obtenidos

A continuación, se presenta un resumen sobre los resultados obtenidos para cada caso de estudio:

- **Caso de estudio II.A:** en este primer caso de estudio, Neo4j obtiene el mejor rendimiento. No obstante, debido a que la cantidad de nodos "mensajes" tiene un crecimiento exponencial, el rendimiento puede llegar a degradarse. En el caso de la consulta II.A.1, los motores de Bases de Datos Apache Cassandra y MySQL obtienen tiempos próximos a Neo4j. Esto se debe a la característica de los campos utilizados en el filtro de la consulta. En el caso de MongoDB, presenta tiempos superiores con respecto a los otros 3 motores de Bases de Datos. Las consultas que involucran campos que pertenecen a documentos embebidos pueden afectar su performance.

  Para la consulta II.A.2, Apache Cassandra obtiene mejor tiempo que los motores de Bases de Datos MySQL y MongoDB, esto se debe a que se utilizó la implementación de un índice para realizar búsquedas relativas. MongoDB obtiene mejor rendimiento que MySQL; las búsquedas relativas y el uso de JOINs afectan el rendimiento en MySQL.

  Para la consulta II.A.3, MySQL obtiene mejor rendimiento que los motores de Bases de Datos MongoDB y Apache Cassandra. Realizar consultas que involucren columnas que no son parte de la Primary Key o Clustering Key en Apache Cassandra, afecta su rendimiento.

- **Caso de estudio II.B:** en este segundo caso de estudio para este segundo experimento, ninguno de los 4 motores de Bases de Datos (MySQL, MongoDB, Apache Cassandra y Neo4j) obtiene el mejor rendimiento para todas las consultas planteadas.

  En las consultas II.B.1 y II.B.2, MongoDB obtiene el mejor rendimiento. En estas consultas, el esquema planteado para MongoDB no utiliza documentos embebidos y no

requiere almacenar campos opcionales, evitando comparaciones nulas. En MySQL las operaciones de JOINs afectan su performance. En Apache Cassandra, las consultas planteadas involucran campos que no están presentes en la Primary Key o Clustering Key, lo que afecta su tiempo de respuesta. En Neo4j el esquema del caso de estudio no es apropiado para el almacenamiento de grafos, dado que no existen relaciones entre los nodos del grafo.

En el caso de la consulta II.B.3, MySQL optimiza la función de agregación COUNT, mejorando así, su tiempo de respuesta en comparación a los otros 3 motores de Bases de Datos (MongoDB, Apache Cassandra y Neo4j).

- **Caso de estudio II.C:** en este tercer caso de estudio Neo4j obtiene el mejor rendimiento para las consultas planteadas. Esto es debido a que dichas consultas son resueltas eficientemente en el almacenamiento de grafos. Apache Cassandra, obtiene mejores tiempos que los motores de Bases de Datos MySQL y MongoDB, esto se debe a que el esquema físico en Apache Cassandra se plantea en términos de consultas. En MySQL la baja de performance entre la consulta II.C.1 y la consulta II.C.2 se debe a los JOINs necesarios para resolver cada consulta.

- **Caso de estudio II.D:** En este último caso de estudio se plantearon dos consultas y se evalúan dos motores de Bases de Datos, Redis y MySQL. Redis obtiene el mejor rendimiento. Esto se debe a que Redis define íntegramente sus estructuras en memoria RAM donde opera los datos, mientras que MySQL por su parte distribuye su procesamiento entre memoria RAM y memoria secundaria. No obstante, gestionar un volumen de datos muy alto en memoria primaria trae algunos inconvenientes y desventajas, entre ellas, la pérdida de llaves en el caso de no configurar una estrategia de persistencia adecuada.

6.1.3.3 EXPERIMENTO III (Motores de Bases de Datos NoSQL como servicios en la Nube).

Este último experimento se realiza con el objetivo de evaluar dos motores de Bases de Datos como servicios en la nube, Cloud Firestore y MongoDB Atlas. Para llevar a cabo este experimento se utilizó el segundo caso de estudio presentado en el experimento II (caso de estudio II.B).

6.1.3.3.1 Caso de estudio III.A

En este caso de estudio la información que se representa mediante un esquema conceptual corresponde a una central de monitoreo que necesita registrar y procesar eventos procedentes de dos fuentes diferentes (tránsito y meteorológica). En la figura 53 se presenta el Diagrama Entidad Relación (DER) para la situación mencionada.

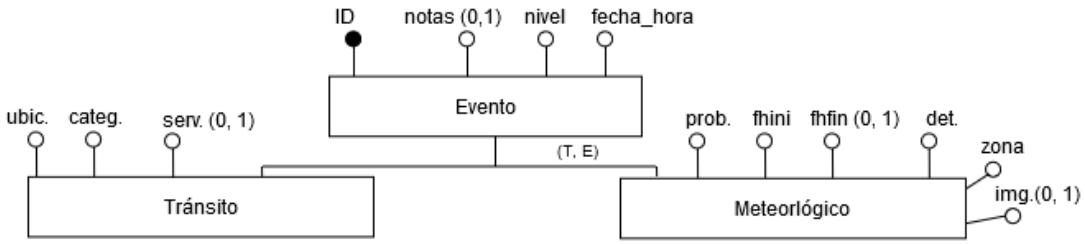

Figura 53. Esquema conceptual, expresado mediante un DER, para el presente caso de estudio.

Para representar conceptualmente a los eventos de tránsito y eventos meteorológicos se optó por utilizar una jerarquía de herencia con cobertura total exclusiva. Esto se debe a que ambos tipos de eventos comparten características similares entre sí.

A continuación, se plantean 3 consultas para realizar las pruebas de rendimiento en este caso de estudio III.A:

- **Consulta III.A.1 (III.A.1)**: Eventos meteorológicos que superen una determinada probabilidad de ocurrencia.
- **Consulta III.A.2 (III.A.2)**: Eventos de tránsito con una determinada categoría en un período de tiempo.
- **Consulta III.A.3 (III.A.3)**: Cantidad de eventos de un determinado nivel de prioridad (Alto, Medio o Bajo).

En este caso, en ambos motores de Bases de Datos (Cloud Firestore y MongoDB Atlas), cada entidad del esquema conceptual fue representada mediante colecciones de documentos y las relaciones correspondientes son derivadas utilizando documentos embebidos. Para ambos motores de Bases de Datos (Cloud Firestore y MongoDB Atlas) se evalúan los planes libres. El volumen de datos generado es menor comparado con el utilizado en los motores de Bases de Datos que fueron instalados localmente (MySQL, MongoDB, Apache Cassandra, Neo4j y Redis), esto se debe a la cantidad de lecturas y escrituras permitidas en un período de tiempo en plan gratuito. Además, se desarrolló una aplicación local y se estableció la conexión remota con ambos motores de Bases de Datos logrando insertar unos 150.000 documentos aproximadamente. A continuación, en la tabla 9 se expresan los tiempos obtenidos en segundos de la ejecución de cada consulta para este caso de estudio. En este caso de estudio se conformó un conjunto de 6 pruebas de performance.

Tabla x. Resultados del caso de estudio III.A expresados en segundos.

Consultas	***Cloud Firestore***	***MongoDB Atlas***
III.A.1	39	19
III.A.2	3	3
III.A.3	2	4

6.1.3.3.2 Discusión de los resultados obtenidos.

Se realizó un comparativo entre dos motores de Bases Datos NoSQL como servicio en la nube (Cloud Firestore y MongoDB Atlas), ambos con almacenamiento documental y en su plan ofrecido gratuitamente. Se utilizó uno de los 4 casos de estudios aplicados en el experimento II (caso de estudio II.B), pero con un menor volumen de datos, esto se debe a las restricciones del plan libre que brindan ambos motores de Bases de Datos (Cloud Firestore y MongoDB Atlas).

Se realizó un conjunto de 6 pruebas de comparación y rendimiento mediante la implementación de 3 consultas en ambos motores de Base de Datos. En resumen, exceptuando la consulta III.A.1, no hay grandes diferencias en los tiempos de respuesta obtenidos. La transferencia de un gran número de documentos y el estado de la conectividad en el instante de la consulta pueden afectar la performance de su tiempo de respuesta.

Capítulo 7

7.1 Conclusiones

En los capítulos que componen este trabajo se ha realizado un resumen de conceptos generales de Base de Datos, cómo ha evolucionado el almacenamiento de información a través de los años, y los distintos tipos de Bases de Datos (Jerárquicas, de Red, Relacionales y Orientadas a Objetos). Así mismo, se ha presentado el paradigma Relacional de Bases de Datos, el cual es el más popular y el más utilizado hasta la actualidad [43]. Se han presentado algunas de sus características más importantes, propiedades ACID, transacciones, reglas de integridad, entre otros conceptos.

Luego, el trabajo se ha concentrado en las Bases de Datos No Relacionales (NoSQL). Estas Bases de Datos utilizan una variedad de implementaciones para acceder y administrar datos (Clave-Valor, Documental, Familia de Columnas y Grafos, entre otras). Estos tipos de Bases de Datos están optimizados específicamente para aplicaciones que requieren grandes volúmenes de datos, baja latencia y modelos de datos flexibles, lo que se logra mediante la flexibilización de algunas restricciones. Además, se han explicado conceptos importantes que poseen estas Bases de Datos, como el Teorema de CAP y las propiedades BASE (Base Availability, Soft State y Eventual Consistency).

Las Bases de Datos NoSQL son flexibles y versátiles, proponen soluciones a problemas que en otros paradigmas han sido difíciles de abordar. No obstante, estas Bases de Datos no han reemplazado a las tradicionales Bases de Datos Relacionales, las cuales poseen una serie de características que resultan importantes para el almacenamiento de información y que se han detallado en el capítulo 4.

El propósito fundamental de este trabajo ha sido realizar experimentos que permitan evaluar comparativamente los motores de Bases de Datos MySQL, MongoDB, Apache Cassandra, Neo4j, Redis y dos motores de Bases de Datos como servicio en la nube, Cloud Firestore y MongoDB Atlas. Exceptuando a MySQL, motor de Base de Datos relacional, los restantes son todos motores de Bases de Datos NoSQL.

El primer experimento realizado se concentró en un estudio comparativo de 3 motores diferentes de Bases de Datos (MySQL, MongoDB y Apache Cassandra). Se plantearon 4 casos de estudio que se basaron en el mismo esquema conceptual para generar el esquema físico correspondiente a cada motor de Base de Datos, y en cada uno de ellos con distintos volúmenes de datos generados aleatoriamente. En cada caso de estudio se realizaron 3 consultas para cada uno de los 3 motores de Bases de Datos MySQL (relacional), MongoDB (NoSQL documental) y

Apache Cassandra (NoSQL familia de columnas). Se conformaron así, un conjunto de 36 pruebas, con el objetivo de obtener métricas de performance.

Como análisis de los resultados obtenidos en este primer experimento se pudo concluir que en línea general no hay un motor de Base de Datos que logre el mejor tiempo de respuesta para todas las consultas evaluadas. Para este primer experimento, los resultados obtenidos dependen del tipo de consulta planteada, el modelo físico generado y el volumen de datos asociado a cada caso de estudio.

El segundo experimento de este trabajo se concentró en un estudio comparativo entre 5 motores de Bases Datos, 4 NoSQL (MongoDB, Apache Cassandra, Redis y Neo4j) y 1 Relacional (MySQL), para 4 casos de estudios diferentes. En cada uno de estos 4 casos de estudios se ha planteado un esquema conceptual diferente con su correspondiente esquema físico. Además, se definieron un conjunto de consultas específico para cada caso, resultando 36 consultas en total.

Para el primer y segundo caso de estudio se definieron 3 consultas en 4 motores de Bases de Datos diferentes (MySQL, MongoDB, Cassandra y Neo4j), formando un conjunto de 24 pruebas.

Para el tercer caso de estudio se definieron 2 consultas para 4 motores de Bases de Datos diferentes (MySQL, MongoDB, Cassandra y Neo4j), formando así un conjunto de 8 pruebas.
En el cuarto y último caso de estudio, de este segundo experimento, se definieron 2 consultas para 2 motores de Bases de Datos diferentes (MySQL y Redis), formando así un conjunto de 4 pruebas.

Los resultados obtenidos en cada caso de estudio fueron analizados con el objetivo de determinar si alguno de los motores de Bases de Datos utilizados se comportaba mejor que otro. Al disponer de un gran volumen de información no hay un motor de Bases de Datos que obtenga el mejor rendimiento en todas las consultas planteadas. A medida que aumenta el volumen de datos, es necesario evaluar cuales son las alternativas de almacenamiento, es decir, que tipo de motor de Bases de Datos es conveniente utilizar con el objetivo de brindar mejores prestaciones.

En el tercer experimento, se utilizaron 2 motores de Bases de Datos como servicios en la nube, estos motores fueron Cloud Firestore (Google) y MongoDB Atlas. Para las pruebas se utilizó un caso estudio perteneciente al segundo experimento, en el cual se plantearon 3 consultas, formando así, un conjunto de 6 pruebas de performance. En las pruebas realizadas ninguno de los dos motores de Bases de Datos (Cloud Firestore y MongoDB Atlas) logró obtener el mejor rendimiento para las 3 consultas planteadas y, además, al ser motores de Bases de Datos como servicios en la nube, el recurso de conexión a la red y la transferencia de los documentos afecta la performance de cada ejecución.

En el caso de la primera consulta (C.III.A.1), se obtienen tiempos elevados en ambos motores de Bases de Datos. Esto se debe a que el resultado obtenido para la consulta evaluada requiere de la transferencia de una gran cantidad de documentos.

En el caso de la segunda consulta (C.III.A.2), no se perciben diferencias en el rendimiento obtenido por ambos motores de Bases de Datos. Es importante aclarar, que en el caso de Cloud Firestore, se realizó la creación de un índice para aplicar el filtrado requerido en la consulta. En este aspecto en Firebase, se puede apreciar un punto en común con Apache Cassandra, en donde, por cuestiones de eficiencia, no es posible ejecutar cualquier tipo de consulta.

En el caso de la tercera consulta (C.III.A.3), los resultados obtenidos son similares en ambos motores de Bases de Datos. En este caso no se ha encontrado observación alguna en lo que respecta a la consulta planteada y el resultado obtenido.

Las experimentaciones realizadas en este trabajo han brindado un aprendizaje y experiencia importante. La utilización de motores de Bases de Datos NoSQL ha permitido explorar nuevas características y variantes para el almacenamiento de datos, así mismo, poder evaluar y comparar distintos comportamientos en un contexto controlado de datos.
Este trabajo puede ser utilizado como base o referencia en distintos ámbitos (educativo, profesional, científico, etc.).

A partir de las investigaciones realizadas para llevar a cabo este trabajo, se han realizado 2 publicaciones en el Congreso Argentino de Ciencias de la Computación (CACIC), en el año 2019 [21] y en el año 2020 [18] y, 1 publicación como capítulo de libro para un libro de Comunicaciones en Ciencias de la Información y Computación, editado por Springer en el año 2021 [13].

Además, este trabajo sirve de base para futuras publicaciones y/o trabajos sobre Bases de Datos NoSQL.

7.2 Trabajo Futuro

Ante la variedad de resultados obtenidos y la falta de prevalencia de un motor de Bases de Datos con el mejor rendimiento, como trabajo futuro se prevé plantear escenarios para realizar un escalamiento horizontal de los esquemas planteados y realizar una nueva evaluación en ese nuevo contexto.
Además, se buscará incorporar nuevos experimentos y reforzar las existentes con alternativas de motores de Bases de Datos relacionales y NoSQL en diversos ambientes.

También, se pretende experimentar con otras categorías de Bases de Datos, como son NewSQL, Bases de Datos de Series Temporales (por ejemplo, InfluxDB [68] o Kdb+ [69]), Bases de Datos de motor de búsqueda (por ejemplo, Elasticsearch [66] o Splunk [67]), entre otros.

Finalmente, se prevé estudiar y experimentar la utilización de motores de Bases de Datos NoSQL específicos para dispositivos móviles.

Bibliografía

1. T. T. Le and X. Lam Pham, "Towards NoSQL databases: Experiences from actual projects," 2022 3rd International Conference on Big Data Analytics and Practices (IBDAP), 2022, pp. 15-20, doi: 10.1109/IBDAP55587.2022.9907664. https://ieeexplore.ieee.org/document/9907664
2. Khan, W., Kumar, T., Cheng, Z., Raj, K., Roy, A. M., & Luo, B. (2022). SQL and NoSQL Databases Software architectures performance analysis and assessments--A Systematic Literature review. arXiv preprint arXiv:2209.06977. https://doi.org/10.48550/arXiv.2209.06977
3. A. D. Díaz Erazo, M. Raúl Morales Morales, V. K. Pineda Chávez and S. Leonardo Morales Cardoso, "Comparative Analysis of performance for SQL and NoSQL Databases," 2022 17th Iberian Conference on Information Systems and Technologies (CISTI), 2022, pp. 1-14, doi: 10.23919/CISTI54924.2022.9820292. https://ieeexplore.ieee.org/document/9820292
4. Perianayagam, S., Vig, A., Terry, D., Sivasubramanian, S., Sorenson III, J. C., Mritunjai, A., ... & Sosothikul, S. (2022). Amazon {DynamoDB}: A Scalable, Predictably Performant, and Fully Managed {NoSQL} Database Service. In 2022 USENIX Annual Technical Conference (USENIX ATC 22) (pp. 1037-1048). https://www.sciencedirect.com/science/article/pii/S0743731521002100
5. Bao, L., Yang, J., Wu, C. Q., Qi, H., Zhang, X., & Cai, S. (2022). XML2HBase: Storing and querying large collections of XML documents using a NoSQL database system. Journal of Parallel and Distributed Computing, 161, 83-99.
6. N. B. Seghier and O. Kazar, "Performance Benchmarking and Comparison of NoSQL Databases: Redis vs MongoDB vs Cassandra Using YCSB Tool," 2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI), 2021, pp. 1-6, doi: 10.1109/ICRAMI52622.2021.9585956. https://ieeexplore.ieee.org/document/9585956
7. X. Cui and W. Chen, "Performance Comparison Test of HBase and Cassandra Based on YCSB," 2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS), 2021, pp. 70-77, doi: 10.1109/ICIS51600.2021.9516864. https://ieeexplore.ieee.org/document/9516864
8. J. M. A. Araujo, A. C. E. de Moura, S. L. B. da Silva, M. Holanda, E. d. O. Ribeiro and G. L. da Silva, "Comparative Performance Analysis of NoSQL Cassandra and MongoDB Databases," 2021 16th Iberian Conference on Information Systems and Technologies (CISTI), 2021, pp. 1-6, doi: 10.23919/CISTI52073.2021.9476319. https://ieeexplore.ieee.org/document/9476319
9. M. Ha and Y. Shichkina, "The Query Translation from MySQL to MongoDB Taking into Account the Structure of the Database," 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), 2021, pp. 383-386, doi: 10.1109/ElConRus51938.2021.9396591. https://ieeexplore.ieee.org/document/9396591
10. N. I. Abo Dabowsa, A. M. Maatuk, S. M. Elakeili and M. Akhtar Ali, "Converting Relational Database to Document-Oriented NoSQL Cloud Database," 2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer

Engineering MI-STA, 2021, pp. 381-386, doi: 10.1109/MI-STA52233.2021.9464488. https://ieeexplore.ieee.org/document/9464488

11. M. Ha and Y. A. Shichkina, "An Approach to Translating a Database from MySQL to Cassandra," 2021 II International Conference on Neural Networks and Neurotechnologies (NeuroNT), 2021, pp. 1-4, doi: 10.1109/NeuroNT53022.2021.9472806. https://ieeexplore.ieee.org/document/9472806

12. T. S. Ivanova and E. A. Ivanov, "Research and Development of the Method of Investigating the Possibility of Transformation Relational Databases to NoSQL," 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), 2021, pp. 2090-2093, doi: 10.1109/ElConRus51938.2021.9396363. https://ieeexplore.ieee.org/document/9396363

13. Marrero, L., Olsowy, V., Tesone, F., Thomas, P., Delia, L., Pesado, P. (2021). Performance Analysis in NoSQL Databases, Relational Databases and NoSQL Databases as a Service in the Cloud. In: Pesado, P., Eterovic, J. (eds) Computer Science – CACIC 2020. CACIC 2020. Communications in Computer and Information Science, vol 1409. Springer, Cham. https://doi.org/10.1007/978-3-030-75836-3_11

14. R. J. Sholichah, M. Imrona and A. Alamsyah, "Performance Analysis of Neo4j and MySQL Databases using Public Policies Decision Making Data," 2020 7th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), 2020, pp. 152-157, doi: 10.1109/ICITACEE50144.2020.9239206. https://ieeexplore.ieee.org/document/9239206

15. C. Huang, M. Cahill, A. Fekete and U. Rohm, "Deciding When to Trade Data Freshness for Performance in MongoDB-as-a-Service," 2020 IEEE 36th International Conference on Data Engineering (ICDE), 2020, pp. 1934-1937, doi: 10.1109/ICDE48307.2020.00207. https://ieeexplore.ieee.org/document/9101730

16. L. Tseng, H. Pan and Y. Wu, "Tutorial: Deep Dive into Apache Cassandra: Theory, Design, and Application," 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2020, pp. 1-2, doi: 10.1109/PerComWorkshops48775.2020.9156261.
https://ieeexplore.ieee.org/document/9156261

17. L. G. Wiseso, M. Imrona and A. Alamsyah, "Performance Analysis of Neo4j, MongoDB, and PostgreSQL on 2019 National Election Big Data Management Database," 2020 6th International Conference on Science in Information Technology (ICSITech), 2020, pp. 91-96, doi: 10.1109/ICSITech49800.2020.9392041. https://ieeexplore.ieee.org/document/9392041

18. Marrero, L., Olsowy, Tesone, F., V., Thomas, P., Delia, L., Pesado, P. (2020). Análisis de performance en Bases de Datos NoSQL y Bases de Datos Relacionales. XXVI Congreso Argentino de Ciencias de la Computación 2020 (CACIC 2020). Universidad Nacional de la Matanza, San Justo, Buenos Aires, 5 al 9 de octubre de 2020. ISBN 978-987-4417-90-9. http://sedici.unlp.edu.ar/handle/10915/114202

19. Marrero, L., Thomas, P., Pasini, A., Bertone, R. A., Ibáñez, E. J., Aguirre, V., Olsowy, Tesone, F., Pesado, P. (2020). Aspectos de la Ingeniería de Software, Bases de Datos Relacionales y Bases de Datos No Relacionales para el desarrollo de Sistemas de Software en Escenarios

Híbridos. XXII Workshop de Investigadores en Ciencias de la Computación 2020 (WICC 2020). Universidad Nacional de la Patagonia Austral, El Calafate, Santa Cruz, Argentina. Mayo 2020 http://sedici.unlp.edu.ar/handle/10915/104026

20. Meier, A., Kaufmann, M. (2019). SQL & NoSQL Databases. Springer Fachmedien Wiesbaden GmbH, part of Springer Nature 2019. Springer Vieweg Wiesbaden ISBN: 978-3-658-24548-1. https://link.springer.com/book/10.1007/978-3-658-24549-8

21. Marrero, L., Olsowy, V., Thomas, P., Delia, L., Tesone, F., Fernández Sosa, J., Pesado, P. (2019). Un estudio comparativo de bases de datos relacionales y bases de datos NoSQL. XXV Congreso Argentino de Ciencias de la Computación 2019 (CACIC 2019). Universidad Nacional de Río Cuarto, Córdoba, 14 al 18 de octubre de 2019. ISBN 978-987-688-377-1. http://sedici.unlp.edu.ar/handle/10915/91403

22. Marrero, L., Thomas, P., Pasini, A., Bertone, R. A., Ibáñez, E. J., Ripodas, A., Aguirre, V., Olsowy, Tesone, F., Capecci, E. M., Pesado, P. (2019). Aspectos de ingeniería de software y bases de datos para el desarrollo de sistemas de software en escenarios híbridos. XXI Workshop de Investigadores en Ciencias de la Computación 2019 (WICC 2019). Universidad Nacional de San Juan, San Juan, Argentina. Abril 2019. ISBN 978-987-3619-27-4. http://sedici.unlp.edu.ar/handle/10915/77088

23. Ajit, S., Sultan, A.. Data Modeling with NoSQL Database (2019). Publicado de forma independiente. ISBN 978-1072978374.

24. Modigliani, S., Vera, C., Lund, M. I. (2018). NoSQL: modelos de datos y sistemas de gestión de bases de datos. XX Workshop de Investigadores en Ciencias de la Computación 2018 (WICC 2018). Universidad Nacional del Nordeste, Chaco - Corrientes, Argentina. Abril 2018. ISBN: 978-987-3619-27-4. http://sedici.unlp.edu.ar/handle/10915/67258

25. Dan, L., Ming Ming, L. (2018). A Performance Optimization Scheme for Migrating Hive Data to Neo4j Database. International Symposium on Computer, Consumer and Control (IS3C) 2018. ISBN: 978-1-5386-7036-1. https://ieeexplore.ieee.org/document/8644938

26. H. Zheng and X. Jiang, "Design and Implementation of College Consumption Analysis System Based on NoSQL Database," 2018 13th International Conference on Computer Science & Education (ICCSE), 2018, pp. 1-5, doi: 10.1109/ICCSE.2018.8468719. https://ieeexplore.ieee.org/document/8468719

27. S. Ghule and R. Vadali, "Transformation of SQL system to NoSQL system and performing data analytics using SVM," 2017 International Conference on Trends in Electronics and Informatics (ICEI), 2017, pp. 883-887, doi: 10.1109/ICOEI.2017.8300833. https://ieeexplore.ieee.org/abstract/document/8300833

28. A. El Alami and M. Bahaj, "Migration of a relational databases to NoSQL: The way forward," 2016 5th International Conference on Multimedia Computing and Systems (ICMCS), 2016, pp. 18-23, doi: 10.1109/ICMCS.2016.7905665. https://ieeexplore.ieee.org/document/7905665

29. S. Chickerur, A. Goudar and A. Kinnerkar, "Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications," 2015 8th International

Conference on Advanced Software Engineering & Its Applications (ASEA), 2015, pp. 41-47, doi: 10.1109/ASEA.2015.19. https://ieeexplore.ieee.org/document/7433067

30. A. U. Sepulveda, R. C. Loyola and M. W. Hernandez, "Performance comparison slowly changing dimensions using model relational and object-relational," 2015 34th International Conference of the Chilean Computer Science Society (SCCC), 2015, pp. 1-6, doi: 10.1109/SCCC.2015.7416582. https://ieeexplore.ieee.org/document/7416582.

31. Bajpayee, R., Sinha, S. P., & Kumar, V. (2015). Big data: a brief investigation on NoSQL databases. International journal of innovations & advancement in computer science, 4(1), 28-35.

32. Abraham Silberschatz, Henry F. Korth, S. Sudarshan (2014). Fundamentos de Bases de Datos 6ta. Edición. McGraw-Hill Interamericana de España S.L., 2014. ISBN: 8448190335, 9788448190330.

33. R. Burtica, E. M. Mocanu, M. I. Andreica and N. Ţăpuş, "Practical application and evaluation of no-SQL databases in Cloud Computing," 2012 IEEE International Systems Conference SysCon 2012, 2012, pp. 1-6, doi: 10.1109/SysCon.2012.6189510. https://ieeexplore.ieee.org/document/6189510

34. Bertone, R., Thomas, P. (2011). Introducción a las bases de datos: fundamentos y diseño. Prentice Hall / Pearson Educación, 2011. ISBN: 9876151363, 9789876151368.

35. N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," in Computer, vol. 43, no. 2, pp. 12-14, Feb. 2010, doi: 10.1109/MC.2010.58. https://ieeexplore.ieee.org/document/5410700

36. Ramez A. E., Navathe, S. B. (2007). Fundamentos de Sistemas de Bases de Datos 5ta. Edición. Pearson 2007. ISBN: 9788478290857, 9788478291151.

37. C. J., Date (2001). Introducción a las Bases de Datos. Pearson Educación, 2001. ISBN: 9684444192, 9789684444195.

38. Johnson, J. L. (2000). Bases de Datos: modelos, lenguajes, diseños. OXFORD University Press, 2000. ISBN: 9706134611, 9789706134615.

39. Batini, C., Ceri, S., Navathe, S. B (1994). Diseño Conceptual de Bases de Datos, un enfoque de entidades-interrelaciones. Addison-Wesley Iberoamericana. ISBN 0-201-60120-6 (1994).

40. American National Standards Institute. Standards Planning and Requirements Committee. Study Group on Database Management Systems. Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems. ACM, 1975.

41. Representación de una Base de Datos Orientada Objetos mediante un Diagrama de Clases UML. https://www.editorialpencil.es/%E2%96%B7-dos-ejemplos-de-diagramas-de-clases-uml-dos-mil-veintiuno/. Accedido en Octubre 2022

42. Marketing 4 Ecommerce. https://marketing4ecommerce.net/. Accedido en Octubre 2022.

43. DB-Engines Ranking (https://db-engines.com/en/ranking). Accedido en Octubre 2022.

44. Biografías y Vidas. https://www.biografiasyvidas.com/biografia/h/hollerith.htm. Accedido en Octubre 2022.

45. Edgar Frank Codd. https://es.wikipedia.org/wiki/Edgar_Frank_Codd. Accedido en Octubre 2022.
46. Oracle (https://www.oracle.com/). Accedido en Octubre 2022.
47. MySQL (https://www.mysql.com/). Accedido en Octubre 2022.
48. SQL Server (https://www.microsoft.com/es-es/sql-server/sql-server-downloads). Accedido en Julio 2022.
49. PostgreSQL (https://www.postgresql.org/). Accedido en Octubre 2022.
50. IBM DB2 (https://www.ibm.com/ar-es/products/db2). Accedido en Octubre 2022.
51. Microsoft Access (https://www.microsoft.com/es-es/microsoft-365/access). Accedido en Julio 2022.
52. SQLite (https://www.sqlite.org/index.html). Accedido en Octubre 2022.
53. MariaDB Server (https://mariadb.org/). Accedido en Octubre 2022.
54. Snowflake (https://www.snowflake.com/?lang=es). Accedido en Octubre 2022.
55. DOMO (https://www.domo.com/). Accedido en Octubre 2022.
56. Redis (https://redis.io/). Accedido en Octubre 2022.
57. Amazon DynamoDB (https://aws.amazon.com/es/dynamodb/). Accedido en Octubre 2022.
58. Memcached (https://memcached.org/). Accedido en Octubre 2022.
59. MongoDB (https://www.mongodb.com/es). Accedido en Octubre 2022.
60. Couchbase (https://www.couchbase.com/). Accedido en Octubre 2022.
61. Firebase (https://firebase.google.com/?hl=es). Accedido en Octubre 2022.
62. Apache Cassandra (https://cassandra.apache.org/_/index.html). Accedido en Octubre 2022.
63. Apache HBase (https://hbase.apache.org/). Accedido en Octubre 2022.
64. DataStax Enterprise (https://www.datastax.com/). Accedido en Octubre 2022.
65. Neo4j (https://neo4j.com/). Accedido en Octubre 2022.
66. Elasticsearch. https://www.elastic.co/es/what-is/elasticsearch. Accedido en Octubre 2022.
67. Splunk. https://www.splunk.com/. Accedido en Octubre 2022.
68. InfluxDB. https://www.influxdata.com/. Accedido en Octubre 2022.
69. Kdb+. https://kx.com/. Accedido en Octubre 2022.
70. IBM - IMS. https://www.ibm.com/products/ims. Accedido en Octubre 2022.
71. GNU. https://www.gnu.org/. Accedido en Octubre 2022.
72. Stack Overflow. https://es.stackoverflow.com/. Accedido en Octubre 2022.
73. Netflix. https://www.netflix.com/ar/. Accedido en Octubre 2022.
74. Walmart. https://www.walmart.com/. Accedido en Octubre 2022.
75. Facebook. https://www.facebook.com/. Accedido en Octubre 2022.
76. Twitter. https://twitter.com/?lang=es/. Accedido en Octubre 2022.
77. Youtube. https://www.youtube.com/. Accedido en Octubre 2022.
78. CISCO. https://www.cisco.com/c/es_ar/index.html. Accedido en Octubre 2022.
79. GitHub. https://github.com/.Accedido en Octubre 2022.

80. Spotify. https://www.spotify.com/ar/. Accedido en Octubre 2022.
81. Booking. https://www.booking.com/. Accedido en Octubre 2022.
82. SEGA. https://www.sega.es/.Accedido en Octubre 2022.
83. Uber. https://www.uber.com/ar/es/. Accedido en Octubre 2022.
84. Instagram. https://www.instagram.com/. Accedido en Octubre 2022.
85. Verizon. https://espanol.verizon.com/. Accedido en Octubre 2022.
86. AstraZeneca. https://www.astrazeneca.com/. Accedido en Octubre 2022.
87. eBay. https://ar.ebay.com/. Accedido en Octubre 2022.
88. Toyota. https://www.toyota.com.ar/. Accedido en Octubre 2022.
89. Google. https://www.google.com/. Accedido en Octubre 2022.
90. FedEx. https://www.fedex.com/es-ar/home.html. Accedido en Octubre 2022.
91. Grupo BIMBO. https://www.grupobimbo.com/. Accedido en Octubre 2022.
92. GOL Transportes Aéreos. https://www.voegol.com/. Accedido en Octubre 2022.
93. CouchDB. https://couchdb.apache.org/. Accedido en Octubre 2022.
94. Airbnb. https://www.airbnb.com.ar/. Accedido en Octubre 2022.
95. Samsung. https://www.samsung.com/ar/dedwdewd/. Accedido en Octubre 2022.
96. LinkedIn. https://ar.linkedin.com/. Accedido en Octubre 2022.
97. Expedia. https://www.expedia.com.ar/. Accedido en Octubre 2022.
98. CERN. https://home.cern/. Accedido en Octubre 2022.
99. PayPal. https://www.paypal.com/ar/home?locale.x=es_AR. Accedido en Octubre 2022.
100. JanusGraph. https://janusgraph.org/. Accedido en Octubre 2022.
101. DGraph.https://dgraph.io/. Accedido en Octubre 2022.
102. Prometheus. https://prometheus.io/. Accedido en Octubre 2022.
103. Graphite. https://graphiteapp.org/. Accedido en Octubre 2022.
104. Unidata-Universe.https://www.ibm.com/docs/es/iis/11.5?topic=databases-universe-unidata. Accedido en Octubre 2022.
105. Apache Jena - TDB. https://jena.apache.org/documentation/tdb/. Accedido en Octubre 2022.