Un estudio comparativo de Bases de Datos Relacionales y Bases de Datos NoSQL

Luciano Marrero[1], Verena Olsowy [1], Pablo Thomas[1], Lisandro Delia[1], Fernando Tesone[1], Juan Fernandez Sosa[1], Patricia Pesado[1]

[1] Instituto de Investigación en Informática LIDI
Facultad de Informática - Universidad Nacional de La Plata – Argentina
Centro Asociado Comisión de Investigaciones Científicas de la Provincia de Buenos Aires

{lmarrero, volsowy, pthomas, ldelia, ftesone, jfernandez, ppesado}@lidi.info.unlp.edu.ar

Resumen: 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 realidad, se utilizan nuevas formas de almacenamiento denominadas bases de datos no relacionales o NoSQL. En este trabajo se compara y analiza el uso de bases de datos relacionales y NoSQL en diferentes casos de estudio con diferentes esquemas y volúmenes de información.

Keywords: Bases de Datos Relacionales, Bases de Datos NoSQL, Almacenamiento Documental, Almacenamiento de Familia de Columnas.

1 Introducción

El concepto de almacenamiento y procesamiento de datos relacionado con las ciencias informáticas se remonta al siglo XIX, al año 1884, donde Herman Hollerith [1], creó una máquina automática que perforaba una cinta de papel dividida en espacios, logrando así, persistir datos binarios [1]. Para fines del siglo XIX aparece el almacenamiento magnético, el cual dió origen en el siglo XX, a las cintas magnéticas (1950) y a los discos magnéticos (1960) [16]. No obstante, hasta entonces, los datos eran almacenados en sistemas de archivos secuenciales, complejos de administrar a medida que crecían en tamaño [15, 16]. El concepto de base de datos se utiliza por primera vez en el año 1963 (en un simposio, California, EEUU). En esa misma década, con el auge de los discos magnéticos, se da origen a la primera generación de bases de datos, las bases de datos de red y las bases de datos jerárquicas [11, 16]. En la década del 70, Edgar Frank Codd, publica un artículo (A Relational Model of Data for Large Shared Databanks) en el que aplicaba conceptos matemáticos de álgebra relacional para el tratamiento de grandes volúmenes de datos [16]. Este trabajo, dio

origen a la segunda generación de bases de datos, las bases de datos relacionales. En sus inicios, los sistemas relacionales fueron fáciles de utilizar pero poco prácticos por su lentitud. Esto cambió en la década del 80 con el desarrollo de hardware más potente, logrando que las bases de datos relacionales se consoliden en el mercado empresarial [13, 14]. Asociado al modelo relacional surge el lenguaje SQL (Structured Query Language), basado en álgebra y cálculo relacional, este lenguaje permite administrar y recuperar información de estos sistemas de base de datos. Posteriormente, para mediados de la década del 80, SQL se convierte en estándar.

Para fines de la década del 80 y principios de los 90, se comienza a utilizar un nuevo estilo de programación, llamado programación orientada a objetos (OOP, por sus siglas en inglés). Las estructuras de datos generadas por estos lenguajes son complejas y difíciles de almacenar en los motores de bases de datos relacionales. En consecuencia, surge una nueva categoría de sistemas de bases de datos, los sistemas de bases de datos orientados a objetos (ODBMS, por su sigla en inglés) [16].

Desde la década de los 90 hasta la actualidad, los cambios tecnológicos tienen un ritmo vertiginoso. Con el auge de la comunicación a través de internet, el volumen de datos a administrar ha crecido de manera exponencial. Actualmente, considerar las bases de datos solamente como un repositorio de información, puede llevar a una administración incorrecta de los datos. La dinámica en la generación de información ha cambiado, entidades y personas utilizan aplicaciones de software para compartir y difundir información. En respuesta a este avance, hace algunos años surgió un nuevo grupo de bases de datos, las bases de datos no relacionales o bases de datos NoSQL (no solo SQL).

En [29] se realizó una estudio comparativo de rendimiento entre 3 motores de Bases de Datos NoSQL (MongoDB, Cassandra y HBase), mientras que en [30] se realiza un estudio similar entre el motor de bases de datos relacional Oracle y el motor MongoDB.

Este trabajo se centra en 3 pruebas de rendimiento para 4 casos de estudio, en 3 motores de bases de datos distintos: MySQL como ejemplo de motor de base de datos relacional; MongoDB y Apache Cassandra como ejemplos de bases de datos no relacionales, totalizando así, un conjunto de 36 escenarios distintos.

A partir de la sección 2 el trabajo se organiza del siguiente modo, se detalla brevemente el panorama actual de las bases de datos relacionales, en la sección 3 se brinda una introducción al concepto de bases de datos NoSQL, en la sección 4 se explican características de los motores de bases de datos seleccionados; en la sección 5 se resumen los resultados obtenidos y finalmente se presentan las conclusiones y trabajo futuro.

2 Panorama actual de las bases de datos relacionales.

Los Sistemas de Gestión de Bases de Datos Relacionales (RDBMS, por sus siglas en inglés) hacen uso de transacciones con el fin de preservar la integridad de las bases de datos. El uso de transacciones afecta el rendimiento en un entorno concurrente debido al bloqueo de datos para preservar la propiedades ACID (Atomicidad,

Consistencia, Aislamiento, Durabilidad). [3, 10, 12]. Por otra parte, dado que se necesita administrar grandes volúmenes de información, los sistemas necesitan ser escalables. La escalabilidad en un sistema de base de datos, se refiere a la posibilidad de mejorar sus prestaciones agregando nuevos recursos. La escalabilidad vertical es económicamente costosa y tiene un límite. La escalabilidad horizontal es difícil de aplicar debido al modelo rígido y estructurado que presentan las bases de datos relacionales. Estas bases de datos se constituyen de un conjunto de tablas, en las cuales la consulta de información puede implicar relacionar varias tablas, y a medida que el volúmen de datos crece, realizar operaciones de JOIN entre tablas afecta notablemente el rendimiento. [13, 14, 15].

3 Introducción a las Bases de Datos NoSQL.

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. NoSQL se distingue de los tradicionales sistemas de gestión de bases de datos relacionales en diversos aspectos, siendo el más notorio, el de no poseer un lenguaje de consulta estructurado (SQL) como lenguaje principal. Además, no requieren de una estructura fija y tabular, no soportan operaciones JOIN, no garantizan por completo las propiedades de ACID, y en general son adecuadas para la escalabilidad horizontal [2, 3]. Para NoSQL es necesario sacrificar algunas características de las bases de datos relacionales para obtener nuevas prestaciones o mejoras en algunos aspectos sin afectar el rendimiento del sistema. Para lograr esto, se propone una visión más liviana y optimista, aceptando que la consistencia en la base de datos esté en un estado de flujo o que la disponibilidad se logre mediante el apoyo a fallas. [2, 4].

NoSQL propone un sistema llamado "BASE (Básicamente Disponible, Estado Suave, Consistencia Eventual)" qué es diferente a las propiedades de ACID de las bases datos relacionales. En resumen, a través de las propiedades BASE se logra disponibilidad básica (Base Availability), esto significa que el sistema se encontrará disponible la mayoría del tiempo. Con el estado débil (Soft State) el sistema se vuelve más flexible en cuanto a consistencia y con la consistencia eventual (Eventual Consistency) se garantiza que el sistema eventualmente se volverá consistente.[3, 7]

Existen cuatro categorías de almacenamiento para bases de datos NoSQL:

Almacenamiento Clave/Valor: simples en cuanto a su implementación, almacenan datos como un conjunto de pares "clave/valor" (key-value). La clave representa un identificador único que puede retornar un objeto complejo y arbitrario de información, denominado valor (value). Por ejemplo, Redis y Amazon DynamoDB, entre otros, implementan este tipo de almacenamiento [23, 24].

Almacenamiento Documental: el concepto central de este tipo de almacenamiento es el documento. Una base de datos NoSQL documental, almacena, recupera y gestiona datos de documentos. Estos documentos encapsulan y codifican datos o información bajo algún formato estándar (XML, YAML, JSON, BSON). Por

ejemplo, MongoDB y Apache CouchDB, entre otros, son implementaciones de bases de datos documentales [20, 25].

Almacenamiento de Familia de Columnas: en este tipo de almacenamiento los datos se encuentran organizados por columnas, en lugar de por filas. Las bases de datos que utilizan esta forma de almacenamiento tienden a ser un híbrido entre las clásicas bases de datos relacionales y la tecnología orientada a columna. Por ejemplo, Cassandra y Apache HBase, entre otros, utilizan este tipo de almacenamiento [22, 26].

Almacenamiento de Grafos: en este tipo de almacenamiento se representa la base de datos bajo el concepto de un grafo. Permite almacenar la información como nodos de un grafo y sus respectivas relaciones con otros nodos, y se aplica la teoría de grafos para recorrer la base de datos; son muy útiles para almacenar información en modelos con muchas relaciones entre distintas entidades o nodos. Por ejemplo, Neo4j y OrientDB, entre otros, permiten este tipo de almacenamiento [27, 28].

4 Motores de Bases de Datos utilizados en los casos de estudio

Para realizar la experimentación presentada en este trabajo, se optó por MySQL (un motor de base de dato relacional), MongoDB (motor con almacenamiento documental) y Apache Cassandra (motor que combina un tipo de almacenamiento clave-valor con familia de columnas), estos últimos son los 2 motores de bases de datos NoSQL más populares [17, 19, 21, 22].

La elección de MySQL se debe a que es uno de los sistemas de base de datos relacional más reconocido y popular del mercado con licencia dual (licencia pública general y licencia comercial). MySQL es utilizado por numerosas empresas a nivel mundial, entre ellas se destacan, Facebook, Twitter, YouTube, GitHub, Uber, Latam Airlines, Booking.com, Spotify, entre otras. [17, 18, 19]

La elección de MongoDB se debe a que es la base de datos NoSQL documental con mayor popularidad. MongoDB es de código abierto, multiplataforma, escrita en C++, posee almacenamiento documental y prioriza la disponibilidad. Es una base de datos adecuada para contextos donde los datos no siempre poseen una estructura fija. En la actualidad, diferentes empresas (Coinbase, SEGA, Royal Bank of Scotland, Telefónica, Facebook, Nokia, entre otras) hacen uso de MongoDB en alguna parte de su infraestructura de datos. [17, 20, 21]

La elección de Apache Cassandra se basa en que es la base de datos NoSQL con almacenamiento en familia de columnas con mayor popularidad. Apache Cassandra es multiplataforma, diseñada para trabajar de forma distribuida con grandes volúmenes de datos. Su escalabilidad lineal, disponibilidad y su tolerancia a fallas la convierten en una plataforma adecuada para datos de misión crítica. Cassandra se origina en el año 2008, en Facebook, por una necesidad de atender búsquedas de su mensajería en tiempo real ante un volumen muy elevado de datos [17, 22].

5 Casos de estudio, pruebas de comparación y rendimiento.

Para representar la información de cada caso de estudio, se realizó previamente el modelo conceptual de datos utilizando un diagrama de entidad relación. Posteriormente, se realizó la derivación al esquema físico correspondiente a cada motor de base de datos.

En MySQL se generó el modelo relacional.

En MongoDB las entidades se representan mediante colecciones de documentos BJSON y las relaciones se expresan mediante documentos embebidos, esto último se debe a que en MongoDB no existe el concepto de JOIN [4, 5, 20].

El caso de Apache Cassandra, los datos no se almacenan en términos de entidades, sino que se representan en términos de consultas, por lo tanto, es importante definir las consultas que se realizarán sobre el sistema.[6, 8, 22].

La ejecución de todas las pruebas se realizaron utilizando la misma infraestructura de hardware y sistema operativo. Se utilizó la versión 5.7 del motor de base de datos MySQL, la versión 4.0 del motor de base de datos MongoDB y la versión 3.11 del motor de bases de datos Apache Cassandra. Para cada caso de estudio el volumen de datos utilizado se generó de forma aleatoria.

A continuación, se detallan los resultados obtenidos de las pruebas realizadas en los 4 casos de estudio. Sólo en el primero de ellos se presenta el modelo conceptual, los esquemas físicos, el código de las consultas y los tiempos obtenidos; y en todos los casos solo se expresan los tiempos resultantes.

5.1 Caso de estudio 1

Se representa una empresa que nuclea reservas de viajes online para un conjunto de agencias de turismo. Actualmente, se llevan registradas 1.260.000 reservas online.

La figura 1 presenta el modelo entidad-relación del problema en cuestión.

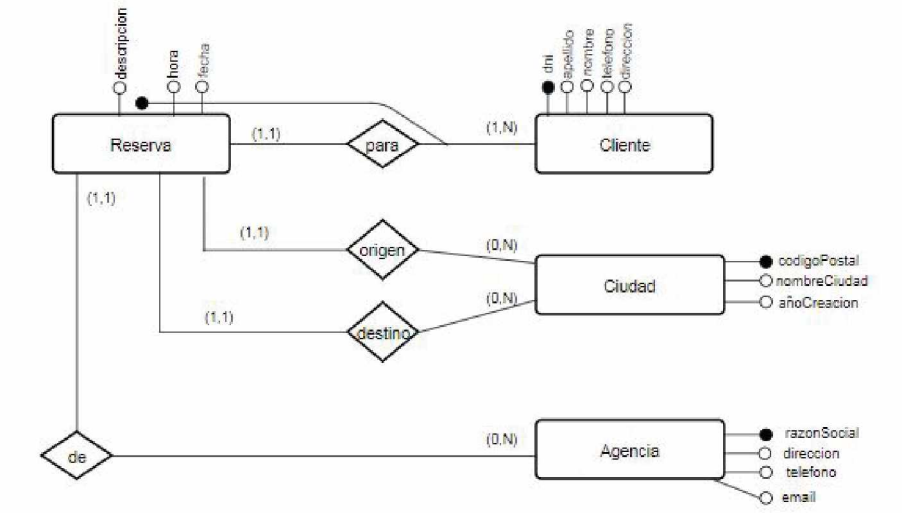

Figura 1. Esquema conceptual expresado mediante un Diagrama Entidad - Relación.

La figura 2 presenta el esquema relacional derivado del modelo conceptual expresado en la figura 1.

AGENCIA = {RAZONSOCIAL(PK), dirección, teléfono, e-mail}
CIUDAD = {CODIGOPOSTAL(PK), nombreCiudad, anioCreación}
CLIENTE = {DNI(PK), nombre, apellido, teléfono, dirección}
RESERVA = {(FECHA, HORA, DNI(*FK*))(*PK*), cpOrigen(*FK*), cpDestino(*FK*), razonSocial(*FK*) , descripción}

Figura 2. Esquema Relacional.

La figura 3 muestra el formato de los datos en un documento para MongoDB.

```
Reservas:{
    _id,
    hora ,
    fecha,
    agencia: { email , telefono,  direccion , razonSocial },
    cliente : {  dni,  nombre ,  apellido,  telefono, direccion},
    descripcion ,
    ciudadOrigen : {anioCreacion, codigoPostal ,  nombreCiudad},
    ciudadDestino : {anioCreacion,codigoPostal , nombreCiudad}
}
```

Figura 3. Esquema para MongoDB.

En Cassandra, el esquema físico de información se expresa de acuerdo a las consultas a realizar. La figura 4 plantea dicho esquema físico.

RESERVAS = {(**dni**, *fecha*, *hora*)(PK), apellido, nombre, ciudaddestino, ciudadorigen, razonsocial }
PROBLEMAS = {(**dni**, *fecha*, *hora*)(PK), ciudaddestino, ciudadorigen, descripción, razonsocial}

Figura 4. Esquema físico de Apache Cassandra.

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

"C1: Reportar los datos y la cantidad de reservas realizadas para cada cliente."
"C2: Reportar los datos de reservas en los que se describe algún problema."
"C3: Reportar las reservas realizadas por un cliente en particular."

En la figura 5, 6 y 7 se muestran las soluciones de las 3 consultas para cada motor de base de datos.

MySQL:

```
select c.dni, c.nombre, c.apellido, count(r.dni) as cantidad
from cliente c inner join reserva r on (c.dni = r.dni)
group by c.dni, c.nombre, c.apellido, c.telefono, c.direccion
```

CASSANDRA:

```
select dni, nombre, apellido, count(dni) as cantidad
from reservas
group by dni
```

MONGO:

```
db.reservas.aggregate([ {$group : {_id:{ dni : "$cliente.dni",
nombre:"$cliente.nombre", apellido: "$cliente.apellido"},
"cantidad": { $sum: 1 } } } ])
```

Figura 5. Implementación de C1.

MySQL:

```
select r.fecha, r.hora, co.nombreCiudad, cd.nombreCiudad,
r.razonSocial, r.descripcion
from reservas r
inner join ciudad co on co.codigoPostal=r.cpOrigen
inner join ciudad cd on cd.codigoPostal=r.cpDestino
where descripcion like "%problema%"
```

CASSANDRA:

```
select * from problemas
```

MONGO:

```
db.reservas.find({descripcion:{$regex: /?roblema/,
$options:"i"}}, {fecha:1, hora:1,
"ciudadOrigen.nombreCiudad":1, "ciudadDestino.nombreCiudad":1,
"agencia.razonSocial":1, descripcion:1, _id:0 })
```

Figura 6. Implementación de C2.

```
MySQL
select r.fecha, r.hora, co.nombreCiudad, cd.nombreCiudad, r.razonSocial
from reservas r
inner join ciudad co on co.codigoPostal=r.cpOrigen
inner join ciudad cd on cd.codigoPostal=r.cpDestino
where r.dni = 28764545

CASSANDRA
select fecha, hora, ciudadOrigen, ciudadDestino, razonSocial
from reservas
where dni = 28764545

MONGO
db.reservas.find({"cliente.dni":28764545}, {fecha:1, hora:1,
"ciudadOrigen.nombreCiudad":1, "ciudadDestino.nombreCiudad":1,
"agencia.razonSocial":1, _id:0 })
```

Figura 7. Implementación de C3.

En la tabla 1 se expresan los tiempos obtenidos en segundos de la ejecución de cada consulta.

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

Consultas	MySQL	MongoDB	Cassandra
C1	6,9	34,31	9,83
C2	61,62	14,90	4,73
C3	0,52	0,85	0,06

5.2 Caso de estudio 2

En este caso de estudio se presentan los movimientos de cuentas en una entidad bancaria que ha alcanzado las 4.233.386 movimientos de cuenta.

En la tabla 2 se expresan los tiempos obtenidos en segundos de la ejecución de cada consulta.

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

Consultas planteadas	MySQL	MongoDB	Cassandra
C1: Reportar cantidad de movimientos por cuenta.	22,5	77,28	53,41
C2: Reportar los movimientos donde el concepto contenga la cadena "alquiler de cochera".	224,29	51,52	11,6
C3: Reportar los movimientos realizados desde una cuenta en particular.	0,56	2,1	0,12

5.3 Caso de estudio 3

El tercer caso de estudio plantea un sistema de compras on-line a nivel nacional que ha alcanzado unas 10.473.392 compras.

La tabla 3 expresa los tiempos obtenidos en segundos de la ejecución de cada consulta.

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

Consultas planteadas	MySQL	MongoDB	Cassandra
C1: Reportar la cantidad de compras por cliente.	52,60	145,12	78,6
C2: Reportar las compras que contienen la cadena "Excelente" en su opinión.	768,45	97,96	59,51
C3: Reportar las compras realizadas por un usuario en particular.	1,11	3,5	0,13

5.4 Caso de estudio 4

El cuarto y último caso presenta un registro de las líneas de teléfonos celulares activas para una empresa nacional que posee 20.120.254 líneas telefónicas.

La tabla 4 expresa los tiempos obtenidos en segundos de la ejecución de cada consulta.

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

Consultas planteadas	MySQL	MongoDB	Cassandra
C1: Reportar la cantidad de líneas telefónicas para cada plan.	128,44	198,59	139,23
C2: Reportar las líneas telefónicas que contienen la cadena "Prepago con deuda" en la descripción de su estado.	19484,12	132,69	75,21
C3: Reportar las líneas telefónicas para una localidad en particular.	1,64	3,93	0,18

6 Discusión de los resultados

En la primera consulta para 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 experimentos, los tiempos de respuesta se acercan a los obtenidos con Cassandra. Estas consultas requieren el uso de la función de agregación COUNT. MySQL optimiza el uso de las funciones de agregación en relación a los

otros 2 motores [4, 9, 20, 22]. Esta podría ser la justificación por la cual su desempeño es mejor.

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

En la tercera consulta, 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 datos se aplica un filtro por un atributo que se encuentra indexado. Finalmente, los tiempo resultantes ubican a Cassandra en el mejor lugar, seguida por MySQL y MongoDB.

7 Conclusiones

Este trabajo se concentra en un estudio comparativo entre diferentes motores bases datos.

En primer lugar, se resume brevemente la evolución de las diferentes formas de almacenamiento de datos. Posteriormente, se plantean las restricciones existentes en las bases de datos relacionales para almacenar grandes volúmenes de información, entre ellos, el manejo de transacciones, las operaciones JOINS de SQL y la dificultad en la escalabilidad. Esta situación origina el análisis y estudio de una alternativa diferente para almacenar información , las bases de datos NoSQL.

Las bases de datos NoSQL no poseen un lenguaje de consulta estructurado, no requieren de una estructura fija y son adecuadas para la escalabilidad horizontal; por lo que mitigan algunos de los problemas que presentan las bases de datos relacionales a través de las propiedades BASE (básicamente disponible, estado flexible, consistencia eventual). Existen 4 tipos de almacenamientos para bases de datos NoSQL: Clave/Valor, Documental, Familia de Columnas y Grafos.

Se plantean 4 casos de estudios cada uno de ellos para diferentes dominios y con distintos volúmenes de datos. Para el primero se ha presentado el desarrollo en forma completa. En cada uno de ellos se plantean 3 modelos consultas para 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.

Finalmente, se realiza una discusión de los resultados obtenidos. En línea general se puede concluir que no hay un motor de base de datos que logre el mejor tiempo de respuesta para todas las consultas evaluadas. Los resultados obtenidos para los casos de estudio planteados, dependen del tipo de consulta planteada y el volumen de datos asociado.

Estas conclusiones tienen concordancia con las conclusiones obtenidas en [29] y [30].

8 Trabajo Futuro

Como trabajo futuro, se prevé incorporar nuevas pruebas y reforzar las existentes con nuevos motores de bases de datos relacionales, y las categorías de bases de datos no relacionales no contempladas en este trabajo (clave-valor, orientadas a grafos). Además, se buscará explorar la capacidad de escalamiento horizontal de las bases de datos NoSQL y de las bases de datos relacionales.

Finalmente se experimentará con bases de datos relacionales basados en NewSQL.

9 Bibliografía

1. Herman Hollerith. https://es.wikipedia.org/wiki/Herman_Hollerith. Accedido en agosto 2019.
2. Ajit Singh, Sultan Ahmad. Data Modeling with NoSQL Database. ISBN 978-1072978374 (2019).
3. 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 (WICC 2019, Universidad Nacional de San Juan). ISBN 978-987-3619-27-4. http://sedici.unlp.edu.ar/handle/10915/77088.
4. Jitender Kumar, Varsha Garg. Security analysis of unstructured data in NOSQL MongoDB database. International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN) 2017. https://ieeexplore.ieee.org/document/8284495.
5. Benymol Jose, Sajimon Abraham. Exploring the merits of nosql: A study based on MongoDB. International Conference on Networks & Advances in Computational Technologies (NetACT) 2017. https://ieeexplore.ieee.org/document/8076778.
6. Shivendra Kumar Pandey, Sudhakar. Context based Cassandra query language. 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT) 2017.https://ieeexplore.ieee.org/document/8204142.
7. Roshni Bajpayee, Sonali Priya Sinha y Vinod Kumar. Big Data: A Brief investigation on NoSQL Databases. International Journal of Innovations & Advancement in Computer Science (IJIACS). ISSN 2347. Volume 4, Issue 1. January 2015.
8. Deepak Vohra. NoSQL Web Development with Apache Cassandra. Cengage Learning, 2015. ISBN 9781305576773.
9. Guoxi Wang, Jianfeng Tang. The NoSQL Principles and Basic Application of Cassandra Model. International Conference on Computer Science and Service System 2012. https://ieeexplore.ieee.org/document/6394574.
10. Enrique J. Reinosa, Calixto A. Maldonado, Roberto Muñoz, Luis E. Damiano, Maximiliano A. Abrutsky. Base de Datos. Alfaomega. ISBN 978-987-1609-31-4 (2012).
11. Abraham Silberschatz, Henry F. Korth, S. Sudarshan. Fundamentos de Bases de Datos, Quinta Edición. McGraw Hill. ISBN 84-481-2021-3 (2006).

12. Ramez A. Elmasri, Shamkant B. Navathe. Fundamentos de Sistemas de Bases de Datos. Tercera Edición. Pearson (Addison Wesley). ISBN 84-782-9051-6 (2002).
13. C. J. Date. Introducción a los Sistemas de Bases de Datos. Séptima Edición. Pearson Educación. Prentice Hall. ISBN 968-444-419-2 (2001).
14. Carlo Batini, Stefano Ceri, Shamkant B. Navathe. Diseño Conceptual de Bases de Datos, un enfoque de entidades-interrelaciones. Addison-Wesley / Díaz de Santos. ISBN 0-201-60120-6 (1994).
15. Michael J. Folk y Bill Zoellick. Estructuras de Archivos, Un Conjunto de Herramientas Conceptuales. ISBN 0-201-62923-2
16. Burton Grad y Thomas J. Bergin. History of Database Management Systems. https://ieeexplore.ieee.org/document/5370775.
17. Ranking de bases de datos según su popularidad. https://db-engines.com/en/ranking. Accedido en Agosto 2019.
18. MySQL. https://www.mysql.com/. Accedido en agosto 2019.
19. Clientes de MySQL. https://www.mysql.com/customers/. Accedido en agosto 2019.
20. MongoDB. https://www.mongodb.com/. Accedido en agosto 2019.
21. Empresas que utilizan MongoDB. https://www.mongodb.com/who-uses-mongodb.. Accedido en agosto 2019.
22. Apache Cassandra. http://cassandra.apache.org/. Accedido en agosto 2019.
23. Redis. https://redis.io/. Accedido en agosto 2019.
24. Amazon DynamoDB. https://aws.amazon.com/es/dynamodb/. Accedido en agosto 2019.
25. Apache CouchDB. http://couchdb.apache.org/. Accedido en agosto 2019.
26. Apache HBase. http://hbase.apache.org/. Accedido en agosto 2019.
27. Neo4j. https://neo4j.com/. Accedido en agosto 2019.
28. OrientDB. https://orientdb.com/. Accedido en agosto 2019.
29. Evaluating NoSQL performance: Which database is right for your data?. https://jaxenter.com/evaluating-nosql-performance-which-database-is-right-for-your-data-107481.html. Accedido en Septiembre 2019.
30. Una comparación de rendimiento entre Oracle y Mongodb. http://www.scielo.org.co/pdf/cein/v26n1/v26n1a07.pdf. Accedido Septiembre 2019.