PUCE Virtual
EDUCACIÓN FLEXIBLE

Adquisición, gestión y gobernanza de datos

No SQL Databases

Clase 5

MAESTRÍA EN
SISTEMAS DE INFORMACIÓN
Mención Data Science

La excelencia no se improvisa

INTRODUCCIÓN

Las bases de datos NoSQL han surgido como una solución crucial para enfrentar las limitaciones de los sistemas de gestión de bases de datos relacionales tradicionales, especialmente en el contexto de la explosión de datos y la creciente necesidad de escalabilidad. A diferencia de las bases de datos SQL, que organizan la información en tablas estructuradas y utilizan el lenguaje SQL para las consultas, las bases de datos NoSQL ofrecen una variedad de modelos de datos, incluidos documentos, grafos, clave-valor y columnas anchas. Esta flexibilidad permite a las organizaciones gestionar datos no estructurados y semiestructurados de manera más eficiente, facilitando el análisis en tiempo real y la integración de datos provenientes de diversas fuentes.

El auge de las aplicaciones web y móviles, junto con el crecimiento del Big Data, ha impulsado la adopción de bases de datos NoSQL en diversos sectores. Estas bases de datos están diseñadas para escalar horizontalmente, lo que permite gestionar grandes volúmenes de información sin sacrificar el rendimiento. Además, su capacidad para adaptarse a esquemas dinámicos y su alta disponibilidad las convierte en una opción atractiva para empresas que buscan innovar y responder rápidamente a las demandas del mercado. Este análisis profundizará en las características, ventajas y desventajas de las bases de datos NoSQL, así como en casos prácticos que ilustran su aplicación en entornos reales.

RDA2

Esquematizar metodologías apropiadas para la obtención de modelos inductivos (predictivos) y deductivos (descriptivos) que faciliten los procesos de toma de decisiones basadas en datos en las instituciones.

Clase 5: No SQL Databases

Teorema ACID en Bases de Datos

El teorema ACID es un conjunto de propiedades fundamentales que garantizan la fiabilidad de las transacciones en bases de datos. ACID es un acrónimo que representa Atomicidad, Consistencia, Aislamiento y Durabilidad. Estas propiedades son cruciales en sistemas de gestión de bases de datos relacionales, donde la integridad de los datos debe mantenerse a lo largo de múltiples operaciones concurrentes (Elmasri & Navathe, 2016).

La **Atomicidad** asegura que una transacción se ejecute completamente o no se ejecute en absoluto. Por ejemplo, en el caso de una transferencia de dinero entre dos cuentas bancarias, si se deduce el dinero de una cuenta, pero la adición a la otra falla, la operación debe revertirse. Esto significa que, ante cualquier fallo, el sistema debe regresar al estado inicial antes de la transacción (Date, 2004). De esta manera, se garantiza que el saldo total no se vea afectado de manera incorrecta.

La **Consistencia** garantiza que una transacción lleve la base de datos de un estado válido a otro estado válido. Siguiendo con el ejemplo bancario, si la suma total de los saldos de todas las cuentas debe ser siempre igual a un valor determinado (por ejemplo, el capital inicial), cualquier operación que altere esta suma debe hacerlo de tal manera que el nuevo estado también respete esta regla. Si una transacción deja el sistema en un estado inconsistente, se debe abortar (Korth & Silberschatz, 2010).

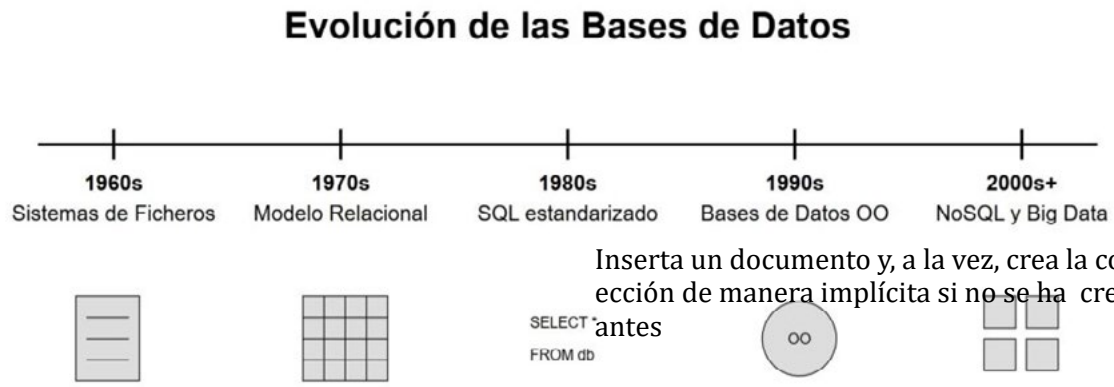
El **Aislamiento** se refiere a la capacidad de ejecutar transacciones concurrentes de manera que no interfieran entre sí. Por ejemplo, si dos usuarios intentan retirar dinero de la misma cuenta al mismo tiempo, el sistema debe asegurarse de que ambas transacciones se manejen de forma aislada. Esto se logra mediante mecanismos como bloqueos o control de concurrencia, lo que asegura que cada transacción opere con datos consistentes (Elmasri & Navathe, 2016).

Finalmente, la **Durabilidad** asegura que una vez que una transacción ha sido confirmada, su efecto persiste incluso en caso de fallos del sistema. Por ejemplo, si un cliente completa una compra en línea y el sistema se apaga inmediatamente después, los datos de la transacción deben guardarse de tal manera que se puedan recuperar al reiniciar el sistema. Esto es esencial para mantener la confianza de los usuarios en el sistema (Date, 2004).

A continuación, se muestra una figura que indica la evolución de las bases de datos entre los años 1960 y 2000.

Figura 1

Evolución de las Bases de Datos.



Fuente: https://jhonmosquera.com/bases-de-datos-historia/

Introducción a las bases de datos NoSQL

Las bases de datos NoSQL han surgido como una alternativa vital a las bases de datos relacionales tradicionales, especialmente en contextos donde la escalabilidad y la flexibilidad son cruciales. A diferencia de las bases de datos SQL, que utilizan un modelo de datos estructurado basado en tablas y relaciones, las bases de datos NoSQL ofrecen una variedad de modelos de datos, incluyendo documentos, clave-valor, columna ancha y grafos. Esta diversidad permite a las organizaciones manejar grandes volúmenes de datos no estructurados y semiestructurados de manera más eficiente, facilitando el análisis en tiempo real y la rápida adaptación a cambios en los requisitos de datos (Moniruzzaman & Hossain, 2018).

Un ejemplo prominente de base de datos NoSQL es MongoDB, que utiliza un modelo de datos basado en documentos. En MongoDB, los datos se almacenan en formato BSON (Binary JSON), lo que permite una representación más rica y flexible de la información. Esto resulta especialmente útil en aplicaciones web y móviles, donde los esquemas de datos pueden cambiar con frecuencia. Por ejemplo, en una aplicación de comercio electrónico, cada producto podría tener diferentes atributos, lo que sería difícil de gestionar en un esquema rígido de base de datos relacional (Chen et al., 2017).

Otra categoría de bases de datos NoSQL es la de clave-valor, como Redis. Redis almacena datos en pares de clave-valor, lo que permite un acceso extremadamente rápido a la información. Esto es particularmente útil en aplicaciones que requieren alta velocidad y baja latencia, como en sistemas de caché o gestión de sesiones de usuario. Un caso de uso típico de Redis podría ser la gestión de sesiones en una aplicación web, donde se necesita acceder a datos de usuario rápidamente (Pereira, 2020).

Las bases de datos de grafos, como Neo4j, se centran en la representación de relaciones entre datos. Estas bases son ideales para aplicaciones que requieren una comprensión profunda de las conexiones, como redes sociales, sistemas de recomendación o análisis de fraude. Por ejemplo, en una red social, Neo4j puede utilizarse para modelar las conexiones entre usuarios y sus interacciones, permitiendo consultas complejas sobre las relaciones de los datos (Robinson et al., 2015).

En resumen, las bases de datos NoSQL ofrecen soluciones versátiles para gestionar datos en entornos donde la flexibilidad, la escalabilidad y el rendimiento son esenciales. Su diversidad de modelos y capacidades permite a las organizaciones adaptarse a las necesidades cambiantes del mercado y a las complejidades del Big Data.

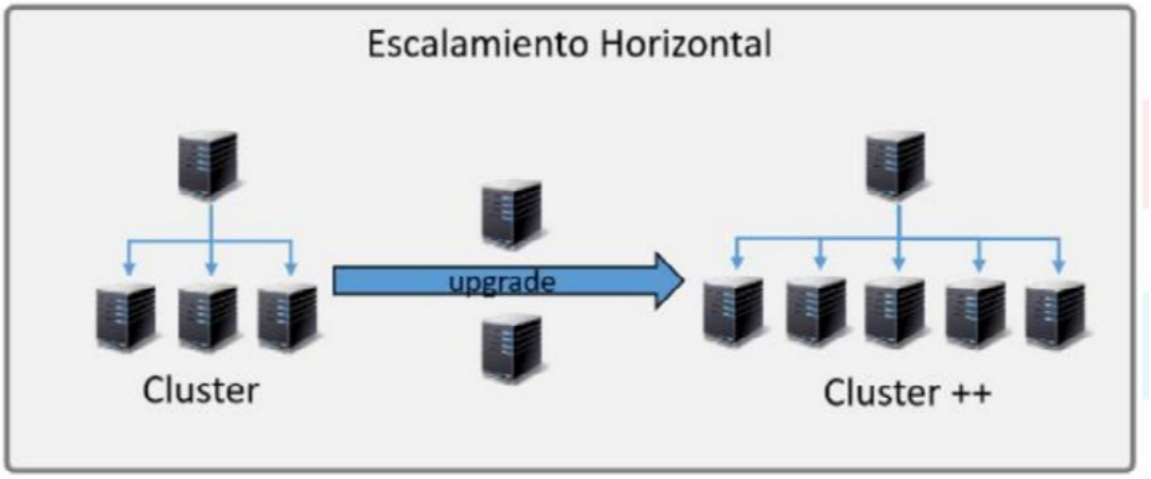
Características Esenciales de las Bases de Datos NoSQL

Las bases de datos NoSQL se han convertido en una solución popular para gestionar grandes volúmenes de datos en un entorno dinámico. A diferencia de las bases de datos relacionales, las bases de datos NoSQL ofrecen características que permiten una mayor flexibilidad, escalabilidad y rendimiento. A continuación, se describen algunas de sus características esenciales.

- **Escalabilidad Horizontal**
  Una de las características más destacadas de las bases de datos NoSQL es su capacidad para escalar horizontalmente. Esto significa que pueden expandirse agregando más servidores en lugar de aumentar la capacidad de un solo servidor. Por ejemplo, en un sistema de gestión de contenido que experimenta un aumento repentino en el tráfico, una base de datos NoSQL como Cassandra puede distribuir los datos entre múltiples nodos, manteniendo el rendimiento incluso bajo cargas pesadas (Sharma & Sharma, 2020) (ver figura 2).
- **Flexibilidad del Esquema**
  Las bases de datos NoSQL permiten un enfoque flexible para la gestión de datos, lo que significa que no requieren un esquema fijo como las bases de datos relacionales. Esto es particularmente útil en aplicaciones donde los datos pueden cambiar con frecuencia. Por ejemplo, en una base de datos de documentos como MongoDB, se pueden almacenar diferentes estructuras de datos en la misma colección, lo que permite una mayor adaptabilidad a las necesidades del negocio (Chen et al., 2017).
- **Alto Rendimiento y Baja Latencia**
  Las bases de datos NoSQL están diseñadas para ofrecer un alto rendimiento y baja latencia en operaciones de lectura y escritura. Esto las hace ideales para aplicaciones en tiempo real, como análisis de datos o aplicaciones web interactivas. Por ejemplo, Redis, una base de datos en memoria, permite acceder a los datos casi instantáneamente, lo que resulta útil en casos de uso como gestión de sesiones de usuario y almacenamiento en caché (Pereira, 2020).
- **Capacidad para Manejar Datos No Estructurados**
  A diferencia de las bases de datos relacionales que se centran en datos estructurados, las bases de datos NoSQL son aptas para manejar datos no estructurados y semiestructurados. Esto incluye imágenes, vídeos, documentos de texto y otros formatos de datos que no se ajustan a un esquema predefinido. Por ejemplo, una base de datos de grafos como Neo4j es excelente para modelar relaciones complejas entre datos no estructurados, como las interacciones en una red social (Robinson et al., 2015).
- **Consistencia Eventual**
  Mientras que las bases de datos relacionales suelen seguir el modelo ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), muchas bases de datos NoSQL adoptan un enfoque de consistencia eventual. Esto significa que, en lugar de garantizar que todas las transacciones sean consistentes en todo momento, permiten que los datos se sincronicen con el tiempo. Este enfoque es útil en aplicaciones distribuidas donde la disponibilidad y la partición son prioritarias, como en sistemas de comercio electrónico (Moniruzzaman & Hossain, 2018).

Figura 2

Escalamiento Horizontal



Fuente: Elaboración Propia

Teorema CAP en Bases de Datos

El teorema CAP, formulado por Eric Brewer en 2000, establece que es imposible para un sistema de bases de datos distribuido garantizar simultáneamente las tres propiedades: Consistencia, Disponibilidad y Tolerancia a Particiones. Estas características son fundamentales para comprender el comportamiento y las decisiones de diseño en sistemas de bases de datos distribuidas. El teorema CAP se ha convertido en un marco crucial para los arquitectos de sistemas, especialmente en el contexto de bases de datos NoSQL (Brewer, 2000).

Consistencia
La consistencia implica que todas las lecturas de un dato deben reflejar el mismo valor en un sistema. Esto significa que, después de completar una transacción, todas las partes del sistema deben estar en un estado coherente. Por ejemplo, en un sistema bancario distribuido, si un usuario realiza una transferencia de dinero, el saldo debe ser actualizado en todos los nodos inmediatamente para evitar que otros usuarios vean saldos obsoletos. Sin embargo, lograr esta consistencia en sistemas distribuidos puede ser complicado, especialmente en situaciones de alta latencia (Lynch, 2006).

Disponibilidad
La disponibilidad asegura que cada solicitud de un usuario recibe una respuesta, ya sea de éxito o de error. Un sistema altamente disponible está diseñado para funcionar incluso si algunos de sus componentes fallan. Por ejemplo, en servicios de streaming como Netflix, es crítico que el sistema esté siempre disponible para los usuarios, incluso si algunos servidores se caen. Este enfoque puede llevar a decisiones que priorizan la disponibilidad sobre la consistencia, lo que significa que los usuarios pueden recibir datos que no están completamente actualizados en el momento de la solicitud (Cachin, 2016).

Tolerancia a Particiones

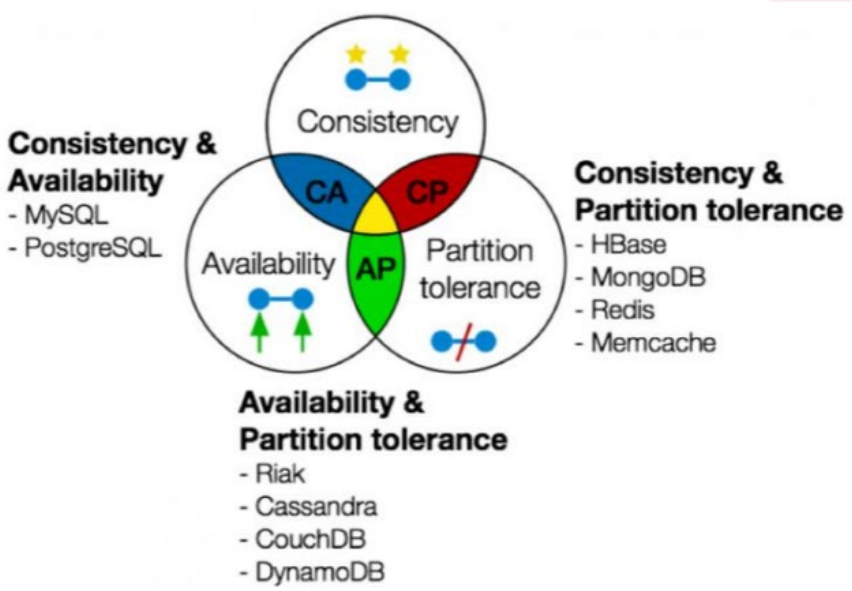
La tolerancia a particiones se refiere a la capacidad del sistema para seguir funcionando a pesar de fallos en la comunicación entre nodos. En un entorno distribuido, la red puede dividirse en segmentos que no pueden comunicarse entre sí. Por ejemplo, si un grupo de servidores pierde la conexión con el resto del sistema, un sistema que prioriza la tolerancia a particiones puede continuar operando en la parte de la red que todavía está activa. Sin embargo, esto puede implicar una falta de consistencia, ya que los nodos

desconectados pueden tener datos diferentes (Gilbert & Lynch, 2002).

El teorema CAP sugiere que un sistema de bases de datos distribuido solo puede proporcionar dos de las tres propiedades simultáneamente. Esto ha llevado a la clasificación de bases de datos en diferentes tipos según cómo manejan estas propiedades. Por ejemplo, los sistemas que priorizan la consistencia y la disponibilidad, como las bases de datos relacionales tradicionales, pueden fallar en situaciones de partición. Por otro lado, las bases de datos NoSQL, como Cassandra, están diseñadas para ser altamente disponibles y tolerantes a particiones, a menudo sacrificando la consistencia temporal (Kappa et al., 2015).

Figura 3

Teorema CAP en bases de datos



Fuente: https://debeando.com/teorema-cap.html

Teorema BASE en Bases de Datos NoSQL

El teorema BASE es un enfoque alternativo al teorema ACID, diseñado para sistemas de bases de datos NoSQL que priorizan la escalabilidad y la disponibilidad sobre la consistencia estricta. BASE es un acrónimo que representa **Básicamente Disponible**, **Estado Suave** y **Eventual Consistency**. Este marco es particularmente relevante en aplicaciones que requieren alta disponibilidad y una respuesta rápida, como en entornos distribuidos y en la nube (Brewer, 2000).

- **Básicamente Disponible**

  La propiedad "básicamente disponible" significa que el sistema garantiza que cada solicitud recibe una respuesta, aunque esta respuesta pueda no ser la más actual. Esto contrasta con la disponibilidad estricta, donde se espera que los datos devueltos sean consistentes en todo momento. Por ejemplo, en aplicaciones de redes sociales, si un usuario publica una actualización, es posible que otros usuarios vean la publicación de inmediato, incluso si el sistema no ha terminado de propagarla a todos los nodos (Liu et al., 2017).

- **Estado Suave**

El concepto de “estado suave” se refiere a que los datos en un sistema pueden estar en un estado intermedio y no necesariamente reflejan un estado consistente en todos los nodos en todo momento. Esto significa que los datos pueden cambiar y evolucionar a medida que se producen las actualizaciones. Por ejemplo, en sistemas de recomendación en línea, un usuario puede ver recomendaciones que se basan en información que no está completamente actualizada, pero que aún resulta útil. Esta flexibilidad permite a los sistemas manejar cambios de datos de manera más dinámica (Sadalage & Fowler, 2012).

- **Eventual Consistency**

  La consistencia eventual es una característica clave del teorema BASE, donde el sistema garantiza que, si no se realizan nuevas actualizaciones, eventualmente todos los nodos del sistema convergerán a un estado consistente. Por ejemplo, en un sistema de comercio electrónico, si un producto se actualiza en un nodo, es posible que otros nodos no reflejen inmediatamente esta actualización. Sin embargo, con el tiempo, todos los nodos alcanzarán la misma información, lo que permite que el sistema siga siendo útil incluso sin consistencia inmediata (Vogels, 2009).

El enfoque BASE es especialmente adecuado para aplicaciones que priorizan la velocidad y la disponibilidad, como las plataformas de redes sociales y los servicios de streaming. Al aceptar una consistencia eventual en lugar de una estricta, los desarrolladores pueden diseñar sistemas que responden más rápidamente a las demandas de los usuarios y escalan con mayor facilidad (Lowe, 2016).

A continuación, se muestra una figura con la comparación entre los teoremas ACID y BASE.

Figura 4

Comparación entre Teorema ACID y BASE.

ACID	BASE
Strong consistency	Weak consistency – stale data OK
Isolation	Availability first
Focus on "commit"	Best effort
Nested transactions	Approximate answers OK
Availability?	Aggressive (optimistic)
Conservative (pessimistic)	Simpler!
Difficult evolution (e.g. schema)	Faster
	Easier evolution

*Lo que pierde el enfoque **BASE vs. ACID** en cuanto a **consistencia**, lo compensa con la **disponibilidad**, **simpleza**, **velocidad** y la facilidad de **evolución**.*

Fuente: Elaboración Propia

El siguiente video muestra un ejemplo en el que se contrasta el teorema ACID con el teorema BASE desde el minuto 43 hasta el 46: ACID vs. BASE.

Clasificación de las Bases de Datos NoSQL

Las bases de datos NoSQL se clasifican en varias categorías según su modelo de datos y la forma en que gestionan la información. Cada tipo ofrece características y ventajas específicas que se adaptan a diferentes necesidades y escenarios de uso. A continuación, se describen las principales categorías de bases de datos NoSQL: clave-valor, documentos, columnares y de grafos.

Bases de Datos Clave-Valor

Las bases de datos clave-valor son el tipo más simple de NoSQL. En este modelo, los datos se almacenan como pares de clave y valor, donde cada clave es única y se utiliza para recuperar su valor asociado. Este enfoque es extremadamente eficiente para operaciones de lectura y escritura. Un ejemplo notable es Redis, que es ampliamente utilizado para almacenar datos temporales y en memoria, como sesiones de usuario y cachés (Pereira, 2020). Debido a su simplicidad y velocidad, estas bases de datos son ideales para aplicaciones que requieren un alto rendimiento.

Figura 5

Pares Clave - Valor

Fuente: Elaboración Propia

Bases de Datos Columnares

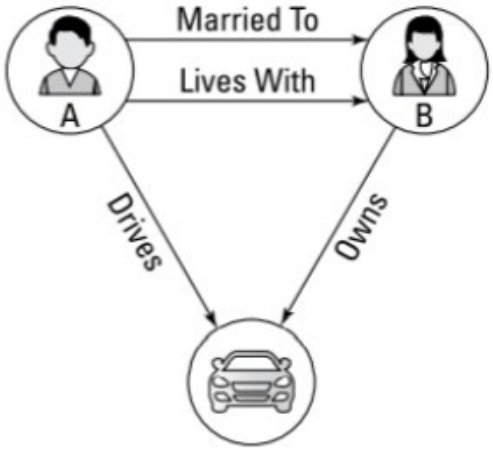
Las bases de datos columnares almacenan datos en columnas en lugar de filas, lo que permite una mayor compresión y optimización de consultas analíticas. Este tipo es especialmente útil para aplicaciones que requieren análisis en tiempo real y consultas complejas. Un ejemplo es Apache Cassandra, que se utiliza a menudo en aplicaciones de Big Data que necesitan gestionar grandes volúmenes de datos distribuidos (Sharma & Sharma, 2020). La arquitectura de Cassandra permite una escalabilidad horizontal, lo que la hace ideal para empresas que experimentan un rápido crecimiento en sus datos.

Bases de Datos Basadas en Grafos

Las bases de datos de grafos están diseñadas para gestionar datos que se representan como nodos y relaciones. Este modelo es extremadamente útil para aplicaciones que requieren una comprensión profunda de las relaciones entre diferentes entidades, como redes sociales o sistemas de recomendación. Neo4j es una de las bases de datos de grafos más conocidas y se utiliza para modelar relaciones complejas, permitiendo realizar consultas avanzadas sobre las conexiones entre datos (Robinson et al., 2015). Este enfoque permite a los desarrolladores explorar y analizar la estructura de los datos de una manera que no es posible con otros tipos de bases de datos.

Figura 6

Ilustración de relaciones entre elementos de una base de datos relacional



Fuente: Elaboración Propia

Bases de Datos Documentales

Las bases de datos documentales almacenan datos en formatos de documentos, como JSON o BSON. Este modelo permite que cada documento tenga una estructura diferente, lo que proporciona flexibilidad en el almacenamiento de datos. MongoDB es uno de los ejemplos más conocidos de este tipo de base de datos. Su capacidad para manejar datos semiestructurados la convierte en una opción popular para aplicaciones web, donde los esquemas de datos pueden cambiar rápidamente (Chen et al., 2017). Esto permite a los desarrolladores iterar rápidamente sobre el diseño de la base de datos sin necesidad de migraciones complejas.

MongoDB

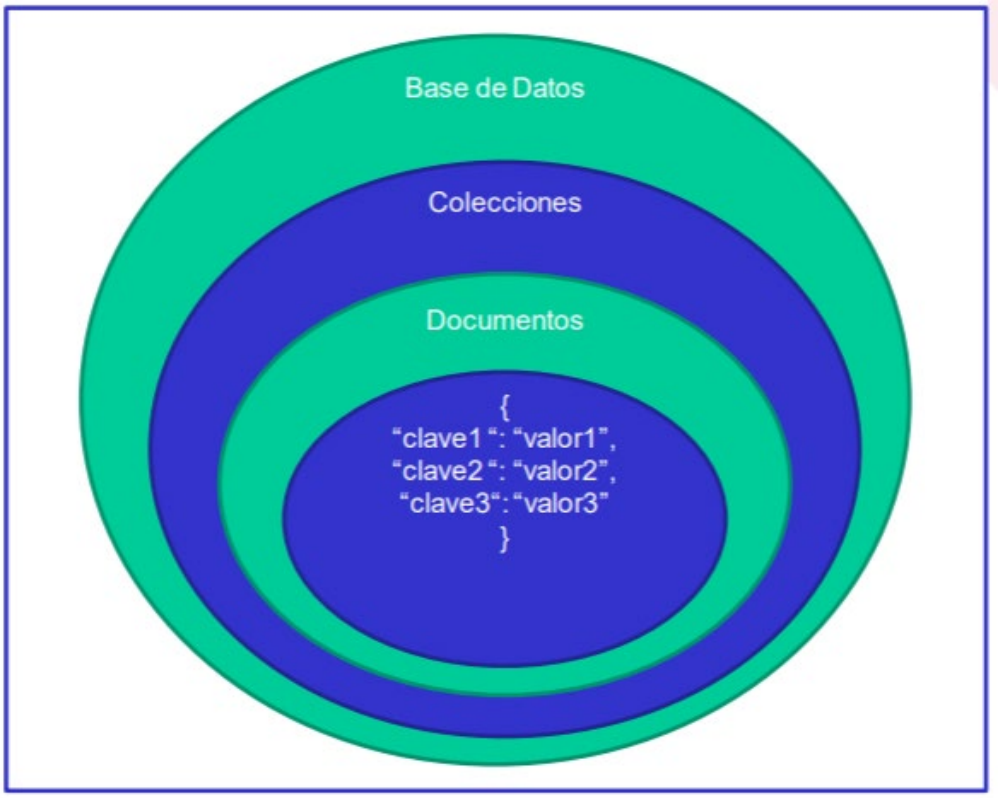
MongoDB es una base de datos documental que se puede usar en diferentes escenarios. A continuación, se mencionan algunos casos en los que es especialmente útil:

- Cuando la aplicación va a tener un crecimiento acelerado.
- Cuando la aplicación tendrá servidores en la nube.
- Cuando es necesario montar una base de datos lo más rápido posible.
- Cuando es posible que la estructura de los datos cambie.

A continuación, se presenta una figura que muestra la arquitectura de MongoDB Server.

Figura 7

Ilustración de la Arquitectura de MongoDB Server



Fuente: Elaboración Propia

El siguiente video muestra un ejemplo de instalación de MongoDB en Windows desde el minuto 1:20:30 hasta el 1:32:00: Instalación de Mongo DB.

Comandos CRUD (Create, Read, Update, Delete) de Mongo DB

Creación de Base de Datos:

use almacen: Comando que sirve para crear una base de datos

db: Muestra la base de datos en la que estamos ubicados

show dbs: Muestra todas las bases de datos creadas. Si la base de datos no tiene información no va a aparecer.

Insertar, Crear:

```
db.usuarios.insert({

"cedula": "1715237051",
"nombre": "Eduardo Montero",

"clave": "xxxxx",

"país": "Ecuador"

})
db.createCollection ("productos")
show collections:

Eliminar
db.productos.drop():

db.dropDatabase()

Actualizar

db.productos.insert({

id : "1",

nombre: "arroz",

valor: 1.5,

stock: 100

})

db.productos.update(
{"id": "a"},

{$set:{"valor": 50}}

)
```

Crea una colección de manera explícita
Muestra las colecciones creadas en la base de datos
Elimina una colección

Elimina una base de datos

Modifica un documento

db.productos.find().pretty()

Eliminar

Elimina un documento

db.productos.deleteOne({
“id”:”a”

})

Referencias

Brewer, E. A. (2000). Towards robust distributed systems. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing* (pp. 7-8).

Cachin, C. (2016). Architecture of a Cloud Storage System. In *Cloud Storage Security* (pp. 1-24). Springer. https://doi.org/10.1007/978-3-319-26961-5_1

Chen, L., Zhou, W., & Zhao, J. (2017). The performance of NoSQL databases for big data applications. *Journal of Computer and System Sciences, 94*, 65-75. https://doi.org/10.1016/j.jcss.2017.02.003

Date, C. J. (2004). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson.

Gilbert, S., & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News, 33*(2), 51-59. https://doi.org/10.1145/58027.58062

Kappa, S., Collell, J., & Tormo, R. (2015). Analyzing the CAP Theorem in Distributed Systems: A Survey. *ACM Computing Surveys, 48*(3), 1-36. https://doi.org/10.1145/2747690

Korth, H. F., & Silberschatz, A. (2010). *Database System Concepts* (6th ed.). McGraw-Hill.

Liu, H., Li, X., & Zeng, X. (2017). The future of distributed databases: A review of NoSQL and NewSQL databases. *Journal of Computer Science and Technology, 32*(5), 915-925. https://doi.org/10.1007/s11390-017-1751-7

Lowe, R. (2016). *NoSQL for mere mortals*. Addison-Wesley.

Lynch, N. A. (2006). *Distributed Algorithms*. Morgan Kaufmann.

Moniruzzaman, A. B. M., & Hossain, S. (2018). NoSQL databases: A brief summary of their design, architecture, and applications. *IEEE Access, 7*, 104603-104615. https://doi.org/10.1109/ACCESS.2018.2870872

Pereira, L. (2020). *Redis in Action: A hands-on guide to using Redis*. Manning Publications.

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph Databases* (2nd ed.). O'Reilly Media.

Sadalage, P. J., & Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley.

Sharma, S., & Sharma, A. (2020). NoSQL databases: Overview, features, and future trends. *International Journal of Computer Applications, 975*, 1-7. https://doi.org/10.5120/ijca2020920144

Vogels, W. (2009). Eventually consistent. *Communications of the ACM, 52*(1), 40-44. https://doi.org/10.1145/1435417.1435432

Consistencia eventual

La “consistencia eventual” es un concepto en bases de datos NoSQL que se refiere a un modelo de consistencia en el cual, después de realizarse actualizaciones en el sistema, todos los nodos convergerán eventualmente hacia un estado consistente, aunque no necesariamente de manera inmediata.

Neo4j
Es una de las bases de datos de grafos más populares. Ofrece un modelo de datos flexible y un lenguaje de consulta llamado Cypher, que facilita la navegación y manipulación de grafos.

PUCE Virtual
EDUCACIÓN FLEXIBLE

La excelencia no se improvisa

síguenos

MDASC - 12 - 241