3. Amazon EFS *(elastic file system)*

- Descripción: Amazon EFS es un sistema de archivos escalable y completamente administrado que permite compartir almacenamiento entre múltiples instancias EC2.
- Características principales:
  - Escalabilidad automática: crece y se reduce según el uso.
  - Acceso simultáneo: múltiples instancias pueden acceder al mismo sistema de archivos al mismo tiempo.
  - Compatibilidad: basado en el protocolo NFS *(network file system)*.
- Casos de uso:
  - Sistemas de gestión de contenido.
  - Procesamiento de datos compartidos.
  - Sistemas de desarrollo y prueba.
- Ejemplo práctico: un equipo de análisis utiliza EFS para almacenar datos en un entorno compartido, donde múltiples instancias EC2 procesan los archivos simultáneamente.

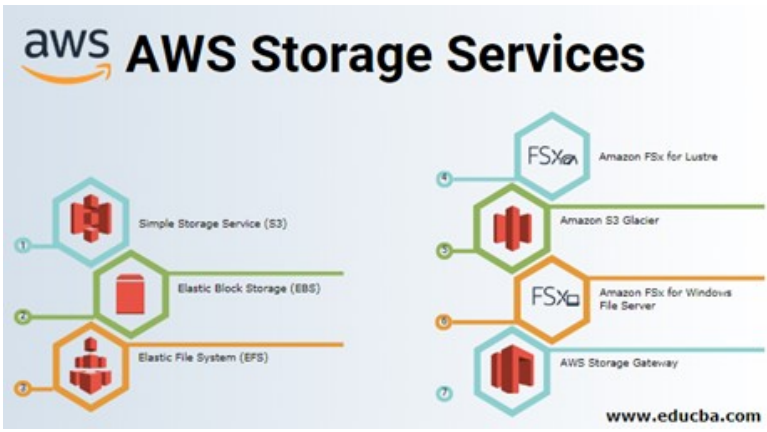

Figura 4. Tipos de almacenamiento datos AWS. Fuente: Pedamkar, 2023.

Tema 2. Almacenamiento escalable

Servicios de almacenamiento en AWS de bases de datos

AWS proporciona una amplia gama de servicios para gestionar bases de datos, cubriendo desde bases de datos relacionales hasta NoSQL. Estos servicios están diseñados para ser escalables, de alto rendimiento y fáciles de administrar, lo que permite a las organizaciones gestionar datos en la nube de manera eficiente. Vamos a profundizar en estos servicios:

1. Amazon RDS *(relational database service)*

- Descripción: Amazon RDS es un servicio administrado que facilita la configuración, operación y escalado de bases de datos relacionales en la nube. Compatible con motores populares como MySQL, PostgreSQL, MariaDB, Oracle y Microsoft SQL Server.
- Características principales:
  - Administración automatizada: actualizaciones de *software, backups* automáticos y recuperación ante fallos.
  - Escalabilidad: ajuste de recursos con pocos clics.
  - Alta disponibilidad: configuración Multi-AZ para tolerancia a fallos.
  - Seguridad: integración con AWS IAM, cifrado en reposo y en tránsito.

- Casos de uso:
  - Aplicaciones web transaccionales.
  - Gestión de ERP o CRM.
  - Almacenamiento de datos estructurados en tiempo real.
- Ejemplo práctico: un comercio electrónico utiliza Amazon RDS para gestionar su base de datos de productos y clientes, con escalabilidad automática durante eventos de alto tráfico como el Black Friday.

2. Amazon Aurora

- Descripción: Amazon Aurora es una base de datos relacional compatible con MySQL y PostgreSQL, diseñada para ofrecer un rendimiento cinco veces mayor que MySQL y tres veces mayor que PostgreSQL estándar.
- Características principales:
  - Rendimiento optimizado: baja latencia y alta velocidad de procesamiento.
  - Alta disponibilidad: replica automáticamente hasta seis copias de datos en tres zonas de disponibilidad.
  - Compatibilidad: totalmente compatible con herramientas y bibliotecas de MySQL y PostgreSQL.
  - Integración con servicios de análisis: integración directa con Amazon Redshift y Athena.

- Casos de uso:
  - Aplicaciones SaaS de alta carga.
  - Bases de datos para aplicaciones empresariales críticas.
  - Migraciones desde bases de datos locales a la nube.
- Ejemplo práctico: una plataforma de *streaming* utiliza Aurora para almacenar datos de usuarios, preferencias y sesiones de reproducción, asegurando una experiencia fluida, incluso con millones de usuarios simultáneos.

3. Amazon DynamoDB

- Descripción: Amazon DynamoDB es un servicio de base de datos NoSQL completamente gestionado diseñado para aplicaciones de alto rendimiento y baja latencia.
- Características principales:
  - Escalabilidad automática: adapta capacidad según la carga.
  - Latencia en milisegundos: ideal para aplicaciones en tiempo real.
  - Modelo flexible: soporta esquemas dinámicos con almacenamiento de clave-valor y documentos.
  - Integración con AWS Lambda: perfecto para arquitecturas *serverless.*

- Casos de uso:
  - Aplicaciones de juegos en línea.
  - Sistemas de recomendación.
  - Bases de datos para IoT y aplicaciones móviles.
- Ejemplo práctico:
  - Una red social utiliza DynamoDB para almacenar perfiles de usuarios, actualizaciones en tiempo real y relaciones entre usuarios.

4. Amazon ElastiCache

- Descripción: Amazon ElastiCache proporciona almacenamiento en memoria para mejorar el rendimiento de aplicaciones mediante caché de datos. Compatible con Redis y Memcached.
- Características principales:
  - Alto rendimiento: ideal para reducir la carga en bases de datos primarias.
  - Persistencia opcional: datos en memoria pueden ser respaldados.
  - Soporte para *clustering:* escalabilidad horizontal para grandes cantidades de datos.
- Casos de uso:
  - Caché de consultas frecuentes en bases de datos.
  - Tablas de clasificación en tiempo real para aplicaciones de juegos.
  - Almacenamiento temporal de sesiones de usuario en aplicaciones web.
- Ejemplo práctico: un portal de noticias utiliza ElastiCache para almacenar consultas populares y minimizar el tiempo de respuesta durante picos de tráfico.

5. Amazon Redshift

- Descripción: Amazon Redshift es un servicio de almacenamiento de datos en columnas diseñado para análisis masivos y consultas rápidas.
- Características principales:
  - Optimizado para análisis: procesa grandes volúmenes de datos de manera eficiente.
  - Compatibilidad con SQL: soporte para herramientas BI estándar, como Tableau y Power BI.
  - Integración con AWS: conexión directa con S3, DynamoDB y Kinesis, para ingestión de datos.
- Casos de uso:
  - *Data warehouses* empresariales.
  - Análisis de grandes conjuntos de datos históricos.
  - Visualización de datos en tiempo real.
- Ejemplo práctico: un *retailer* utiliza Redshift para analizar patrones de compra y planificar inventarios, basándose en predicciones de demanda.

6. Amazon Timestream

- Descripción: Amazon Timestream es una base de datos optimizada para series temporales, diseñada para almacenar y consultar datos con marcas de tiempo, como métricas, *logs* y eventos.
- Características principales:
  - Gestión automática de datos: optimiza datos recientes y antiguos para reducir costos.
  - Consultas rápidas: optimizadas para datos temporales y agregados.
  - Integración: compatible con Amazon QuickSight para visualización.
- Casos de uso:
  - Monitoreo de sistemas y aplicaciones.
  - Análisis de IoT.
  - Gestión de eventos en tiempo real.
- Ejemplo práctico: un sistema de monitoreo de sensores IoT utiliza Timestream para analizar datos de temperatura, humedad y vibración en tiempo real.

7. Amazon Neptune

- Descripción: Amazon Neptune es un servicio de base de datos de grafos completamente administrado, diseñado para manejar relaciones complejas entre datos.
- Características principales:
  - Compatibilidad: compatible con los modelos de grafos Property Graph y RDF.
  - Alta disponibilidad: replica datos en múltiples zonas de disponibilidad.
  - Consultas eficientes: optimizado para consultas con relaciones complejas.
- Casos de uso:
  - Redes sociales para modelar relaciones entre usuarios.
  - Motores de recomendación basados en grafos.
  - Sistemas de detección de fraudes.
- Ejemplo práctico: una plataforma de *e-learning* utiliza Neptune para analizar las conexiones entre cursos, habilidades y usuarios, lo que le permite ofrecer recomendaciones personalizadas.

8. Amazon DocumentDB (con compatibilidad con MongoDB)

- Descripción: Amazon DocumentDB es un servicio de base de datos NoSQL completamente administrado diseñado para almacenar, consultar y procesar datos semiestructurados en formato de documentos JSON. Es altamente compatible con MongoDB, lo que facilita la migración de aplicaciones existentes.
- Características principales:
  - Compatibilidad con MongoDB: diseñado para ser compatible con aplicaciones y

herramientas que utilizan MongoDB. Permite usar las mismas consultas y operaciones sin modificaciones significativas.

- Escalabilidad automática: escalado de almacenamiento independiente del cómputo, hasta 128 TB de datos, sin interrupciones. Los nodos de lectura pueden escalar horizontalmente para gestionar aumentos de tráfico.
- Alta disponibilidad: replica automáticamente seis copias de datos en tres zonas de disponibilidad (AZ). Recuperación ante fallos integrada y rápida.
- Seguridad robusta: cifrado de datos en reposo y en tránsito. Integración con Amazon Virtual Private Cloud (VPC), para aislamiento de red. Compatibilidad con AWS Identity and Access Management (IAM), para un control granular de acceso.
- Gestión sin servidor: actualizaciones automáticas de *software. Backups* continuos y recuperación puntual de datos.

- Casos de uso:
  - Aplicaciones de catálogos: ideal para almacenar productos en aplicaciones de comercio electrónico. Manejo eficiente de datos JSON con esquemas flexibles.
  - Gestión de contenido: para sistemas que requieren almacenamiento y búsqueda de documentos, como blogs o gestores de contenido empresarial.
  - Aplicaciones móviles: almacena datos de usuarios y sesiones con rápida disponibilidad y flexibilidad.
  - Redes sociales: útil para modelar datos no estructurados, tales como publicaciones, comentarios y relaciones de usuarios.
- Ejemplo práctico: gestión de catálogo de productos en una tienda en línea con Amazon DocumentDB. Imagina que estamos desarrollando una aplicación de comercio electrónico donde se necesita almacenar un catálogo de productos. Cada producto tiene información semiestructurada, como nombre, precio, descripción, imágenes y especificaciones técnicas. Este tipo de información puede variar en cada producto, por lo que se necesita una base de datos flexible que permita realizar consultas eficientes y escalables a medida que el catálogo crece.

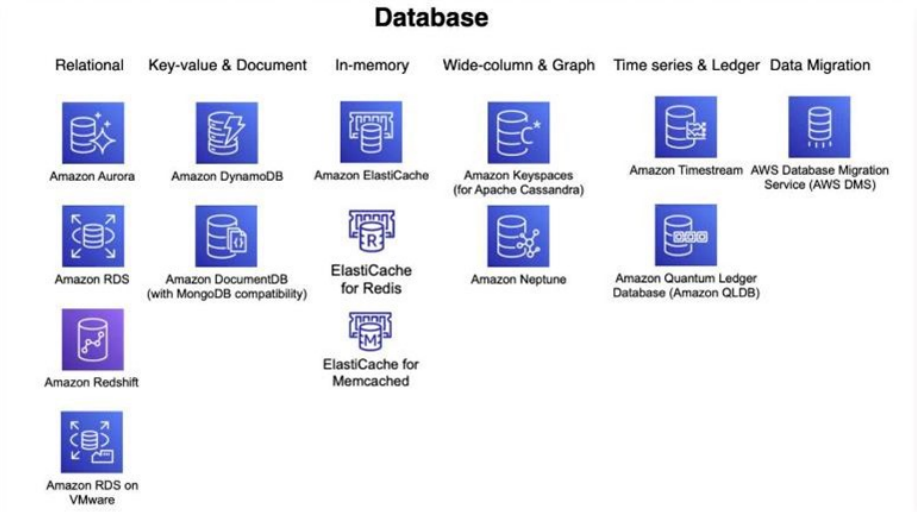

Figura 5. Tipos de bases de datos. Fuente: Patil, 2025.

2.2. BBDD NoSQL

En la actualidad, la cantidad y variedad de datos generados por las aplicaciones modernas han cambiado las reglas del juego en el mundo del almacenamiento y procesamiento de datos. Las bases de datos NoSQL han surgido como una respuesta eficiente y flexible a las limitaciones de las bases de datos relacionales tradicionales, especialmente para manejar datos no estructurados y en grandes volúmenes.

Este apartado te guiará desde los conceptos básicos hasta el entendimiento de sus principales usos, tipos y beneficios.

¿Qué son las bases de datos NoSQL?

El término NoSQL se refiere a bases de datos que no utilizan el modelo relacional tradicional basado en tablas y SQL *(structured query language)*. En cambio, están diseñadas para manejar datos con una estructura más flexible y escalable.

- Definición: una base de datos NoSQL es un sistema de almacenamiento de datos que permite almacenar, recuperar y gestionar grandes volúmenes de datos no estructurados o semiestructurados sin depender de esquemas predefinidos.

- Características principales:
  - Esquema flexible: no requieren una estructura fija de tablas o columnas.
  - Escalabilidad horizontal: se pueden expandir añadiendo más servidores en lugar de aumentar la capacidad de uno solo.
  - Alta disponibilidad y rendimiento: diseñadas para manejar grandes cantidades de transacciones y consultas simultáneamente.
  - Compatibilidad con datos variados: soportan datos en formatos JSON, XML, BSON, entre otros.

Diferencias entre bases de datos relacionales y NoSQL

Se presenta a continuación, una tabla resumen de las diferencias entre ambos tipos:

Aspecto	Bases de datos relacionales	Bases de datos NoSQL
Modelo de datos	Tablas con filas y columnas	Documentos, clave-valor, grafos, columnas
Lenguaje	SQL	Varía según el tipo (consulta no SQL)
Esquema	Esquema fijo y definido	Flexible y dinámico
Escalabilidad	Vertical (más recursos en un servidor)	Horizontal (más servidores)
Consistencia	ACID (atomicidad, consistencia, aislamiento, durabilidad)	Eventual o consistente, según configuración
Casos de uso	Transacciones financieras, ERP	IoT, big data, aplicaciones móviles

Tabla 3. Diferencias entre bases de datos relacionales y NoSQL. Fuente: elaboración propia.

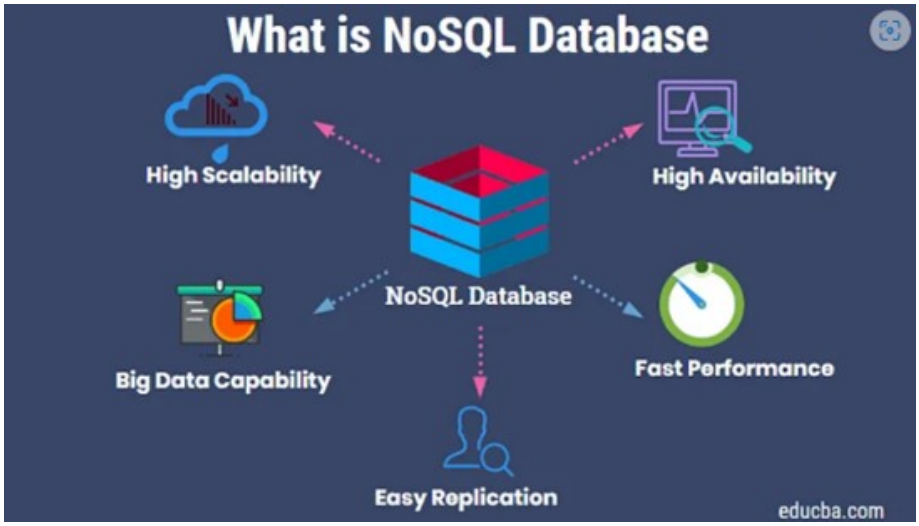

Figura 6. Características BBDD NoSQL. Fuente: Parmar, 2024.

Tipos de bases de datos NoSQL

Las bases de datos NoSQL se dividen en varios tipos, cada uno diseñado para resolver necesidades específicas. A continuación, exploramos los más comunes como son clave-valor, documentales, columnares y de grafos:

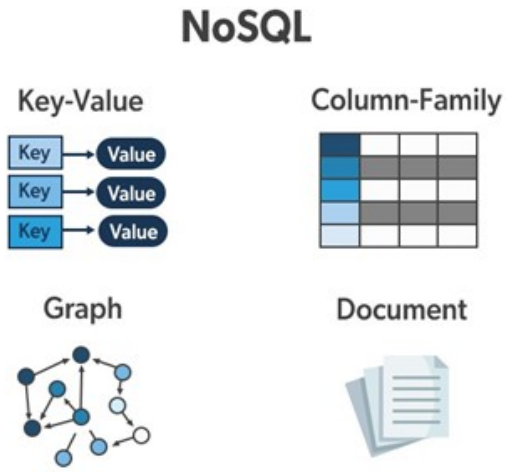

Figura 7. Familias de BBDD NoSQL. Fuente: Castillo, 2023.

Bases de datos clave-valor

En un mundo digital donde la velocidad, la simplicidad y la flexibilidad son esenciales, las bases de datos clave-valor han ganado relevancia, como una de las soluciones más **simples** y **potentes** para manejar **grandes volúmenes** de **datos.**

Este apartado te llevará a través de un recorrido detallado por el modelo de datos clave-valor, su arquitectura, usos, ventajas, limitaciones y ejemplos prácticos.

- ¿Qué es una base de datos clave-valor?:

Una base de datos clave-valor es un sistema de almacenamiento de datos donde cada dato se guarda como un par clave y valor. Este modelo es similar a un diccionario en Python o a un mapa *hash* en otros lenguajes de programación.

- Definición: una base de datos clave-valor almacena datos en pares clave-valor, donde:
  - Clave: es un identificador único para el dato, como un ID o un nombre.
  - Valor: es la información asociada a esa clave, que puede ser simple (un número o una cadena) o compleja (un objeto JSON, una lista, etc.).
- Ejemplo sencillo:

```
plaintext

Clave: "usuario_123"

Valor: {"nombre": "Juan", "edad": 30, "email": "juan@mail.com"}
```

- Arquitectura de una base de datos clave-valor:

La arquitectura de una base de datos clave-valor es sencilla pero eficaz, diseñada para ofrecer rapidez y escalabilidad.

1. Pares clave-valor: los datos se almacenan en una estructura básica que asocia cada clave única con un valor.

2. Almacenamiento en memoria o disco:

- Algunas bases clave-valor, como Redis, almacenan los datos en memoria para obtener una mayor velocidad.

- Otras, como Amazon DynamoDB, los guardan en disco para mayor persistencia.

3. *Hashing* y distribución de claves: las claves se distribuyen entre nodos (en sistemas distribuidos), utilizando algoritmos de *hashing,* lo que asegura una distribución equilibrada y un acceso rápido.

4. Acceso directo por clave: dado que las claves son únicas, el acceso a los valores es directo, lo que elimina la necesidad de búsquedas complejas.

5. Replicación y alta disponibilidad: muchas bases clave-valor replican datos en múltiples nodos para garantizar la disponibilidad y la tolerancia a fallos.

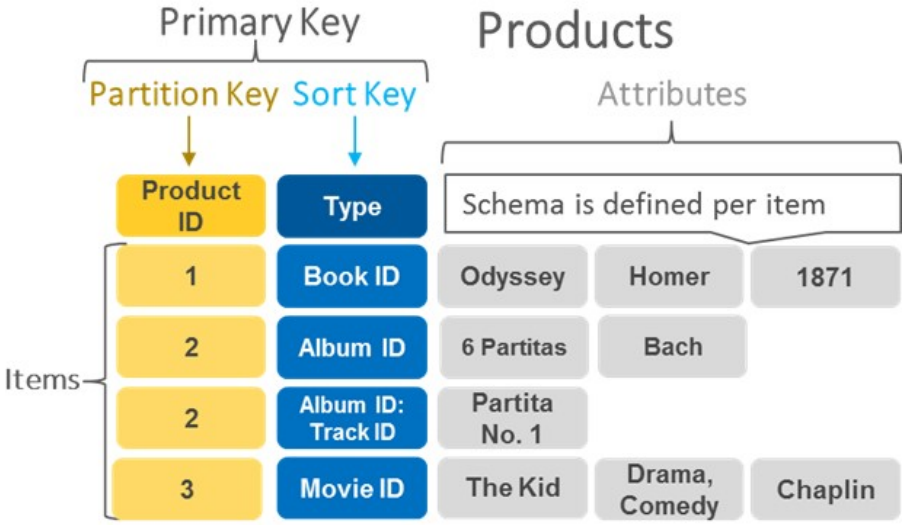

Figura 8. Funcionamiento BBDD Clave-Valor. Fuente: AWS, s. f.

- Casos de uso de las bases de datos clave-valor:

Podemos enumerar los siguientes ejemplos:

1. Caché de datos: las bases clave-valor son ideales para almacenar datos que necesitan ser accedidos rápidamente, como resultados de consultas frecuentes o sesiones de usuario.

- Ejemplo: guardar *tokens* de autenticación de usuarios en Redis.

2. Sistemas de sesiones: almacenar información de las sesiones de usuarios en aplicaciones web.

- Ejemplo: datos de sesión en un sitio de comercio electrónico.

3. Gestión de configuraciones: almacenar configuraciones de aplicaciones, que pueden ser leídas y actualizadas fácilmente.

- Ejemplo: configuración de un sistema de microservicios.

4. Procesamiento de mensajes: utilización en sistemas de mensajería o colas de tareas distribuidas.

- Ejemplo: uso de Redis para manejar tareas asíncronas en una aplicación.

5. IoT *(Internet of things)*: almacenamiento de datos de sensores en tiempo real.

- Ejemplo: recopilación de datos de temperatura en una red de sensores.

- Ventajas de las bases de datos clave-valor:

Destacamos las siguientes:

1. Simplicidad:

- Su modelo es fácil de entender e implementar.
- Ideal para desarrolladores principiantes.

2. Velocidad:

- Permite acceder a los datos en tiempo constante O(1)O(1)O(1), lo que garantiza un rendimiento extremadamente rápido.

3. Escalabilidad horizontal:

- Se pueden añadir más nodos para distribuir los datos y las cargas de trabajo.

4. Flexibilidad en el valor:

- El valor puede ser de cualquier tipo, desde datos simples hasta objetos JSON complejos.

5. Alta disponibilidad:

- La replicación y la redundancia aseguran que los datos estén siempre disponibles.

- Limitaciones de las bases de datos clave-valor:

Como limitaciones de este tipo de bases de datos, destacamos las siguientes:

1. Búsquedas limitadas:

- No soportan consultas complejas, como búsquedas por múltiples atributos (a diferencia de las bases relacionales).
- Ejemplo: no puedes buscar todos los usuarios con edad mayor a 30 directamente.

2. Relaciones entre datos:

- No son adecuadas para manejar relaciones complejas entre entidades.

3. Tamaño de los valores:

- En algunos sistemas, los valores grandes pueden afectar el rendimiento.

4. Mantenimiento del *hashing:*

- En sistemas distribuidos, el rebalanceo de claves entre nodos puede ser complicado.
- Ejemplo práctico en Python. Redis como base de datos clave-valor:

Redis es una de las bases de datos clave-valor más populares, diseñada para trabajar en memoria con alta velocidad.

- Instalación de Redis. Primero, instala la librería de Redis en Python:

```
bash

pip install redis
```

- Conexión a Redis:

```
python

import redis



# Conectar a Redis

redis_client = redis.StrictRedis(host='localhost', port=6379,
decode_responses=True)
```

- Insertar y leer datos:

```
python

# Insertar un dato

redis_client.set("usuario_123", '{"nombre": "Juan", "edad": 30, "email":
"juan@mail.com"}')

# Leer un dato

valor = redis_client.get("usuario_123")

print(valor)
```

- Eliminar datos:

```
python

# Eliminar un dato

redis_client.delete("usuario_123")
```

- Ejemplo avanzado: guardar una lista:

```
python

# Guardar una lista

redis_client.rpush("mi_lista", "item1", "item2", "item3")

# Leer la lista

mi_lista = redis_client.lrange("mi_lista", 0, -1)

print(mi_lista)
```

- Comparación con otros modelos NoSQL:

A continuación, se muestra una tabla comparativa de modelos principales de bases de datos (BBDD) NoSQL:

Aspecto	Clave-Valor	Documental	Columnas	Grafos
Modelo	Clave-Valor	Documentos JSON	Columnas	Nodos y relaciones
Velocidad	Muy alta	Alta	Alta	Moderada
Escenarios	Cachés, sesiones	Aplicaciones web	*Big data,* analítica	Redes sociales
Relaciones complejas	No	Limitadas	Limitadas	Sí

Tabla 4. Comparativa de los modelos principales de BBDD NoSQL. Fuente: elaboración propia.

Bases de datos documentales

Las bases de datos documentales representan un paso innovador en el diseño de sistemas de almacenamiento de datos. Ideales para manejar información semiestructurada y datos jerárquicos, este modelo ha ganado relevancia en aplicaciones web modernas, análisis de datos y entornos dinámicos.

En este apartado, aprenderás qué son las bases de datos documentales, cómo funcionan, sus ventajas y limitaciones, así como ejemplos prácticos para que puedas aplicarlas desde cero.

- ¿Qué es una base de datos documental?:

Una base de datos documental es un tipo de base de datos NoSQL que almacena datos en forma de documentos, usualmente en formatos como JSON, BSON o XML. Cada documento es una unidad independiente que contiene datos y su estructura, lo que proporciona flexibilidad para manejar información diversa.

- Definición: una base de datos documental almacena datos como documentos autocontenidos, donde cada documento puede contener campos y valores (simples o complejos) estructurados de forma jerárquica.

- Ejemplo sencillo de documento JSON:

```
json
{
  "id": "usuario_123",
  "nombre": "Juan",
  "edad": 30,
  "email": "juan@mail.com",
  "pedidos": [
    { "id": "pedido_1", "total": 100 },
    { "id": "pedido_2", "total": 200 }
  ]
}
```

- Características principales de las bases de datos documentales:

1. Almacenamiento flexible:

- Los documentos no necesitan adherirse a un esquema fijo.
- Cada documento puede tener una estructura distinta.

2. Jerarquía de datos:

- Permite almacenar información compleja, como listas o subdocumentos, dentro de un solo documento.

3. Optimización para consultas de documentos:

- Las consultas están diseñadas para trabajar directamente con los datos del documento.

4. Escalabilidad horizontal:

- Estas bases de datos están diseñadas para ser distribuidas y escalarse horizontalmente.

5. Indexación inteligente:

- Los índices pueden crearse en campos individuales.

- Arquitectura de una base de datos documental:

1. Documentos como unidad principal:

- Los datos se almacenan en documentos autocontenidos que son independientes entre sí.

2. Colecciones:

- Los documentos se agrupan en colecciones. Una colección es similar a una tabla en bases de datos relacionales, pero sin restricciones de esquema.

3. Distribución y replicación:

- En sistemas distribuidos, las colecciones se dividen en fragmentos o particiones para distribuir la carga de trabajo.
- La replicación asegura la alta disponibilidad de los datos.

4. API para operaciones CRUD:

- Ofrecen API simples para crear, leer, actualizar y eliminar documentos.

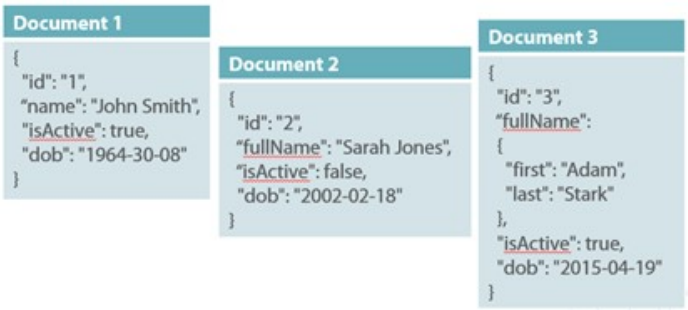

Figura 9. Representación de almacenamiento de datos en bases de datos documentales. Fuente: Koech, 2023.

- Casos de uso de las bases de datos documentales:

1. Aplicaciones web dinámicas:

- Gestión de datos semiestructurados, como perfiles de usuario y contenido generado por los usuarios.

2. Gestión de contenidos:

- Sistemas de blogs, portales de noticias o plataformas de *e-learning,* que manejan contenido variado.

3. Catálogos de productos:

- Comercio electrónico y *retail,* donde cada producto tiene atributos únicos.

4. Sistemas IoT:

- Almacenamiento de datos jerárquicos y actualizaciones en tiempo real de sensores.

5. Procesamiento de datos en tiempo real:

- Aplicaciones que requieren análisis rápido y procesamiento continuo.

- Ventajas de las bases de datos documentales:

1. Flexibilidad estructural:

- Los documentos pueden adaptarse a cambios en los datos sin necesidad de modificar un esquema global.

2. Desarrollo rápido:

- La ausencia de esquemas rígidos permite iteraciones rápidas durante el desarrollo.

3. Alto rendimiento:

- Optimizadas para operaciones de lectura y escritura rápidas en grandes volúmenes de datos.

4. Escalabilidad horizontal:

- Permiten añadir nodos al sistema para manejar el aumento de datos.

5. Relaciones embebidas:

- Los datos relacionados pueden almacenarse dentro del mismo documento, simplificando las consultas.

- Limitaciones de las bases de datos documentales:

1. Tamaño de los documentos:

- Documentos muy grandes pueden ralentizar las operaciones.

2. Relaciones complejas:

- Aunque admiten datos jerárquicos, no son ideales para relaciones muy complejas entre entidades.

3. Falta de estándares:

- La ausencia de un esquema puede complicar el mantenimiento en proyectos grandes.

4. Consistencia eventual:

- En sistemas distribuidos, puede haber un retraso en la sincronización de datos entre nodos.

- Ejemplo práctico con MongoDB:

MongoDB es una de las bases de datos documentales más populares. Veamos cómo trabajar con ella usando Python.