1. Document Stores (Bases de datos orientadas a documentos)

Son bases de datos diseñadas para almacenar y recuperar documentos.

Los documentos pueden estar en formatos como JSON, XML, BSON, entre otros.

Poseen una estructura jerárquica (en forma de árbol).

Los documentos son auto-descritos, es decir, incluyen la descripción de los datos.

Cada documento tiene una clave de partición.

El motor de la base de datos puede examinar el contenido interno del documento.

Se diferencian de los key-value stores porque estos no interpretan la estructura interna del valor.

2. JSON (JavaScript Object Notation)

Características

Formato de texto usado para almacenar e intercambiar datos.

Todo objeto JavaScript puede convertirse a JSON.

La sintaxis se basa en:

Pares nombre–valor.

Valores permitidos: cadenas, números, objetos JSON, arreglos, booleanos y null.

Separación por comas.

Uso de llaves {} para objetos.

Uso de corchetes [] para arreglos.

Soporta JSON Schema para validación de estructura.

Ejemplo

Incluye atributos simples, objetos anidados y arreglos, mostrando su flexibilidad estructural.

3. XML (eXtensible Markup Language)

Características

Formato de texto para guardar e intercambiar datos.

Incluye:

Prólogo XML (versión y codificación).

Elementos con etiqueta de inicio y cierre.

Atributos (usados para metadatos como identificadores).

Contenido textual u otros elementos.

Elementos vacíos.

Soporta XML Schema para definición estructural.

Ejemplo

Presenta una estructura jerárquica con elementos anidados y atributos.

4. Ejemplos de Document Stores

El documento menciona y muestra visualmente algunos de los principales motores:

MongoDB

Apache CouchDB

RavenDB

OrientDB

Terrastore

Microsoft Azure DocumentDB

También se referencia el ranking de DB-Engines para este tipo de bases de datos.

5. MongoDB – Información general

Sitio oficial: https://www.mongodb.com/

Lanzado en 2009.

Implementado en C++.

Licenciamiento: Open Source (GNU Affero GPL) y comercial.

Acceso mediante:

Línea de comandos (JavaScript).

APIs para múltiples lenguajes: C, C#, C++, Java, Python, JavaScript, Ruby, Scala, entre otros.

Lenguaje de consulta con expresividad similar a SQL.

6. MongoDB – Características principales

Alto desempeño en lectura y escritura.

Escalabilidad horizontal mediante sharding.

Alta disponibilidad y tolerancia a fallos.

Arquitectura Always On.

Soporte de índices secundarios.

Lenguaje de consulta expresivo.

Consistencia variable.

Herramientas de administración, seguridad, monitoreo e integración.

Objetivo: “Tomar lo mejor de los mundos relacional y NoSQL”.

7. Bases de datos, colecciones y documentos

Una instancia MongoDB puede contener múltiples bases de datos.

Cada base de datos puede contener múltiples colecciones.

Las colecciones almacenan documentos JSON (internamente BSON).

Tamaño máximo de documento: 16 MB.

Cada documento tiene un identificador único _id.

Los nombres son sensibles a mayúsculas y minúsculas.

Soporte para índices secundarios.

8. Colecciones en MongoDB

Agrupan documentos relacionados.

No imponen un esquema rígido.

Los documentos suelen tener propósitos similares.

La partición se define mediante un shard key:

Puede ser uno o varios atributos.

Es inmutable.

Soporta valores faltantes.

Permite validación de esquema opcional:

Atributos obligatorios.

Validaciones de tipo y rangos.

9. Arquitectura: MongoDB Sharded Cluster

Componentes

Shards: contienen subconjuntos de los datos.

Replica Set: conjunto de nodos que implementan replicación y recuperación ante fallos.

Mongos: servicio de enrutamiento que dirige las solicitudes.

Config Server:

Almacena metadatos del clúster.

Mantiene la configuración y autenticación.

Se despliega como replica set.

10. Replica Sets

Arquitectura maestro–esclavo.

Escrituras en el nodo primario.

Lecturas por defecto desde el primario (configurable).

Replicación asíncrona.

Elección automática de nuevo primario ante fallos.

Recomendaciones:

Al menos 3 nodos.

Número impar de nodos.

Si la mayoría cae, el sistema queda en modo read-only.

11. Sharding

Se realiza a nivel de colección.

Uso de shard key para partición.

Los datos se dividen en chunks.

Los chunks se distribuyen entre shards.

Balanceo automático por defecto.

Estrategias:

Hashing.

Rangos de valores.

Hashed Sharding

Distribución uniforme usando hash MD5.

Ideal para cargas intensivas de escritura.

Requiere alta cardinalidad.

Adecuado para llaves monotónicas.

Sharding por Rangos

Llaves cercanas permanecen juntas.

Optimiza consultas por rangos.

12. Chunk Splits

Tamaño por defecto del chunk: 128 MB.

División automática durante inserciones o actualizaciones.

No se revierte una vez ejecutado.

Trade-offs entre chunks pequeños y grandes en balanceo y eficiencia.

13. Selección del Shard Key

Alta cardinalidad es esencial.

Evita cuellos de botella y falta de escalabilidad.

Las consultas que no usan shard key provocan broadcast a todos los shards.

Impacta directamente el rendimiento y escalabilidad.

14. Zonas (Zones)

Agrupan shards.

Permiten:

Aislamiento de datos.

Proximidad geográfica.

Asignación según desempeño.

15. Transacciones y consistencia

Atomicidad por defecto a nivel de documento.

Soporta transacciones multi-documento.

Configuración mediante:

Read Concern Levels: available, local, majority, linearizable, snapshot.

Write Concern Levels: majority, w=1…n.

Las escrituras pueden ser visibles antes de ser durables.

16. Storage Backend

Permite combinar diferentes motores.

WiredTiger (por defecto):

Compresión.

Control de concurrencia a nivel de documento.

In-memory:

Máximo desempeño.

Latencia predecible.

Analítica en tiempo real.

17. Índices secundarios

Mejoran la eficiencia de consultas.

Impactan negativamente las escrituras.

Se definen por colección.

Se actualizan sincrónicamente.

Buenas prácticas:

Indexar campos de alta selectividad.

Preferir intersección de índices sobre índices compuestos.

Uso de índices parciales.

Casos donde no se usan índices:

Wildcards iniciales.

Negaciones o desigualdades.

18. Modelado de datos en MongoDB

Debe basarse en:

Patrones de acceso.

Consultas frecuentes.

Actualizaciones.

Evolución del esquema.

Decisión clave:

Datos desnormalizados vs datos normalizados.

Datos desnormalizados

Mejor desempeño en lectura.

Menos consultas.

Se acepta duplicación.

La aplicación maneja la integridad.

Límite: documentos muy grandes y crecimiento constante.

Datos normalizados

Mayor flexibilidad y consistencia.

Más consultas para resolver referencias.

Adecuado para relaciones N-M y grandes jerarquías.

19. Recomendaciones de diseño

Diseñar el esquema según los requerimientos de la aplicación.

Optimizar para los casos de uso más frecuentes.

Agrupar datos usados juntos.

Evitar joins en lectura.

Realizar joins en escritura.

Duplicar datos cuando sea necesario.

20. Operaciones en MongoDB

CRUD

Crear colección: db.createCollection()

Insertar: insert

Consultar: find, aggregate

Actualizar: updateOne, updateMany

Eliminar: deleteOne, deleteMany

Crear índices: createIndex

Consultas

Filtros con operadores: $eq, $gt, $lt, $in, $and, $or, etc.

Proyecciones con inclusión/exclusión de campos.

Soporte para:

Documentos anidados.

Arreglos.

Arreglos de documentos.

Agregaciones

Pipeline con etapas como $match, $group, $sort, $project, $limit, $unwind.

21. APIs y Lenguajes soportados

MongoDB ofrece drivers oficiales para: Java, Python, JavaScript, C#, C, C++, Ruby, PHP, Go, Scala, Erlang, Node.js, entre otros.

22. Referencias

W3Schools – JSON y XML.

Manual oficial de MongoDB 7.0.

NoSQL Distilled – Fowler & Sadalage, 2013.