BASES DE DATOS

Trabajo práctico ‘Bases de datos NoSQL’

Integrantes: *Ahumada, Ivan Adolfo ; Arce, Ezequiel ; Borgnia, Erik; Cármenes, Santiago; Civiero, Tomas; Magliotti, Gian Franco ; Rasso, Micaela*

Profesores: *Dra. Leticia M. Seijas, Lic. Claudio Gea, Sr. Francisco Stimmler*

Paradigma de bases de datos NoSQL

Definición del Paradigma NoSQL

El paradigma NoSQL se define como un enfoque de diseño de bases de datos que almacena y consulta datos fuera de las estructuras tabulares tradicionales de las bases de datos relacionales (SQL). Este paradigma se caracteriza por:

- **Estructura flexible**: Operan sin un esquema predefinido (*schema-less*), permitiendo albergar datos en estructuras que agrupan información relacionada, y facilitando la adición libre de campos sin modificar la estructura previa.
- **Naturaleza distribuida**: Generalmente son proyectos de código abierto diseñados para ejecutarse en clústeres. La información se copia y almacena en varios servidores, locales o remotos.
- **Orientación a agregados**: La mayoría están orientadas a agregados, lo que ayuda al funcionamiento en clúster al indicar qué datos se manipulan juntos en un mismo nodo.

Problemas que intenta resolver

NoSQL surgió para solucionar limitaciones específicas de las bases de datos relacionales frente a las demandas tecnológicas modernas:

- **Problemas de escalabilidad**: Las bases de datos SQL solo escalan verticalmente (más hardware en un solo servidor), lo cual tiene un límite físico. NoSQL resuelve esto mediante el escalado horizontal (fragmentación o *sharding*), dividiendo la carga de trabajo entre dos o más servidores cuando un solo servidor no es suficiente.
- **Rigidez de los datos**: Resuelve la dificultad de manejar datos no uniformes o no estructurados. Mientras que SQL requiere esquemas estrictos, NoSQL permite manejar grandes conjuntos de datos dispares y realizar cambios rápidos en el código sin definir cambios estructurales primero.
- **El Teorema CAP**: A diferencia del modelo relacional que prioriza estrictamente la consistencia (ACID), NoSQL aborda los desafíos de los sistemas distribuidos descritos en el teorema CAP, permitiendo sacrificar la consistencia inmediata a favor de la disponibilidad y la tolerancia a particiones de red (fallos de conexión), es decir, si ocurre un fallo de red, el sistema sigue respondiendo peticiones en los nodos disponibles, aunque esto lleve a la muestra de datos desactualizados por el momento en que la red no esté activa.

Relevancia en la actualidad

Su relevancia actual radica en su capacidad para satisfacer las demandas de aplicaciones web modernas, complejas, sitios de comercio electrónico y aplicaciones móviles. Es fundamental por las siguientes razones:

- Velocidad y agilidad: Permite un almacenamiento y procesamiento más rápido para desarrolladores.
- Alta disponibilidad y fiabilidad: Gracias a la replicación en varios servidores, garantiza el acceso a los datos incluso si surgen fallos en algún servidor.
- Manejo de grandes volúmenes: Es la alternativa idónea para administrar conjuntos de datos grandes que cambian rápidamente.

Bases de datos Clave-Valor

En qué consiste el modelo

Este modelo pertenece a la categoría de bases de datos orientadas a agregados. Su funcionamiento se basa en los siguientes principios:

- **Estructura básica**: Consisten en una gran cantidad de agregados, donde cada uno tiene una clave o ID única que se utiliza para acceder a los datos.
- **Opacidad del agregado**: A diferencia de las bases de datos documentales, en el modelo Clave-Valor el agregado es opaco para la base de datos. Para el sistema, el valor almacenado es simplemente un conjunto de bits mayormente sin significado.
- **Acceso**: Solo se puede acceder al agregado mediante una búsqueda basada en su clave. No se pueden realizar consultas basadas en la estructura interna o campos del agregado (a menos que se integren herramientas de búsqueda externas o el sistema específico permitiera ciertas estructuras, como listas o conjuntos).
- **Ventaja**: La opacidad otorga libertad para almacenar lo que se desee dentro del agregado, con la única restricción habitual siendo el límite de tamaño que imponga la base de datos.

Redundancia y consistencia de datos

Al ser bases de datos NoSQL distribuidas, utilizan mecanismos específicos para garantizar la resiliencia y la integridad en un entorno de clúster:

- **Redundancia mediante Hashing Consistente:** Para distribuir las claves y sus réplicas entre los servidores (nodos), se utiliza el algoritmo de **Hashing Consistente**. Esto organiza los nodos en un anillo lógico; si un servidor cae, las claves que le pertenecían se redistribuyen automáticamente a los nodos vecinos sin necesidad de detener el sistema, garantizando así la disponibilidad y la redundancia de datos.

- **Consistencia Ajustable (Quórums N, R, W):** El manejo de la consistencia difiere del modelo relacional estricto. Estas bases de datos permiten configurar el nivel de consistencia por operación utilizando un sistema de votación o **Quórums**:
    - **N:** Número total de réplicas.
    - **W:** Cuántos nodos deben confirmar una escritura.
    - **R:** Cuántos nodos deben responder a una lectura.
    - Si $R + W > N$, se obtiene **Consistencia Fuerte** (siempre se lee el dato más reciente). Si no, se obtiene **Consistencia Eventual**, priorizando la velocidad sobre la precisión inmediata.
- **Resolución de Conflictos con Relojes Vectoriales:** Dado que no suelen tener transacciones ACID distribuidas, pueden ocurrir escrituras simultáneas en diferentes nodos. Para manejar esto, se utilizan **Relojes Vectoriales** (*Vector Clocks*), que permiten al sistema detectar si una versión de un dato es descendiente de otra o si existe un conflicto de versiones que la aplicación debe resolver.

Bases de datos de Documentos

En qué consiste el modelo

Este modelo se basa en la gestión de datos a través de agregados con una estructura interna visible y definida. Sus características principales son:

- **Almacenamiento estructurado**: Los datos se albergan dentro de una estructura de datos compleja, típicamente un documento (JSON, BSON, XML). Operan sin un esquema rígido previo (*schema-less*), lo que permite añadir campos libremente. Además, la base de datos reconoce y valida las estructuras y tipos de datos permitidos dentro del documento.
- **Visibilidad y Acceso**: La base de datos conoce la estructura dentro del agregado. Esto permite funcionalidades como:
    - Enviar consultas basadas en los campos internos del documento.
    - Recuperar sólo partes específicas del agregado en lugar de la totalidad del mismo.
    - Crear índices basados en el contenido interno para acelerar las búsquedas.
- **Identificación**: Cada documento o agregado posee una clave o ID que permite acceder a él, funcionando como una unidad de datos que agrupa información relacionada.

Redundancia y consistencia de datos

En este modelo, la redundancia se aborda tanto desde la infraestructura como desde el diseño de datos:

- **Redundancia por Diseño (Desnormalización):** Un concepto clave para la redundancia en documentos es la **Incrustación (Embedding)**. En lugar de normalizar datos en múltiples tablas (como en SQL), se opta por duplicar datos dentro de los documentos (ej. guardar la dirección del usuario dentro de cada documento de "Pedido"). Esto mejora drásticamente el rendimiento de lectura al evitar JOINs, pero introduce el desafío de mantener la consistencia: si el dato original cambia, se deben actualizar todas sus copias redundantes.
- **Infraestructura de Réplicas (Replica Sets):** Para la redundancia física, bases de datos como MongoDB utilizan **Conjuntos de Réplicas**. Existe un nodo **Primario** que recibe las escrituras y nodos **Secundarios** que replican los datos para garantizar alta disponibilidad si el primario falla.
- **Manejo de la Consistencia:**
    - **Atomicidad por documento:** Soportan transacciones ACID a nivel de un solo documento.
    - **Consistencia Eventual en Lecturas:** Se puede configurar la preferencia de lectura (*Read Preference*) para leer de los nodos secundarios. Esto escala el rendimiento, pero implica aceptar consistencia eventual (posibilidad de leer datos ligeramente obsoletos mientras se replican desde el primario).

SGBD MongoDB

Bases de datos de Documentos

Colección con documentos con diferente estructura

Se utilizan dos dispositivos distintos de computación para comprobar que sucedería al insertar ambos en la misma colección, siendo que tienen distintos campos. Al insertar un "Mouse" en la misma colección que una "Notebook", MongoDB aceptó la operación sin errores, a pesar de tener campos distintos (dpi vs procesador), lo que demuestra la propiedad de Esquema Dinámico. Puede afirmarse que la base de datos no impone una estructura uniforme, trasladando la responsabilidad de manejar la consistencia estructural a la aplicación.

Efecto de UPDATE sobre documentos

Para actualizar un campo se utiliza el operador $set. La operación es atómica a nivel de un solo documento, lo que garantiza la consistencia en esa unidad de datos.

- Insertar un nuevo campo: Al intentar actualizar un campo inexistente, como el campo “marca”, MongoDB lo crea dinámicamente. Esto permite la evolución del esquema sin tiempo de inactividad (downtime).
- Upsert (Update + Insert): Al utilizar la opción { upsert: *true* }, el sistema actualiza el documento si lo encuentra o crea uno nuevo si no existe.

Importancia para la Consistencia: El upsert es crítico en sistemas distribuidos para garantizar la idempotencia. Permite reintentar operaciones de escritura ante fallos de red sin riesgo de duplicar datos o generar inconsistencias.

Utilización de UPDATE: En MongoDB no existe un comando update, se utiliza el operador $set. La operación es atómica a nivel de un solo documento, lo que garantiza la consistencia en esa unidad de datos.

```
db.productos.updateOne(
  { sku: "LAP-001" },
  { $set: { precio: 1400 } }
)
```

Efecto de INSERT sobre documentos

Si MongoDB detecta que el campo especificado en el $set no existe en el documento, lo crea automáticamente. Esto permite la evolución del esquema sin tiempo de inactividad.

Ejemplo: Agregar el campo "proveedor" a un documento que no lo tenía:

```
db.productos.updateOne(
  { sku: "LAP-001" },
```

```
  { $set: { proveedor: "Asus" } } // El campo 'proveedor' se crea ya que no existía
)
```

Para insertar una nueva entrada si no existe (o actualizarla si ya existe), se utiliza la opción upsert (una combinación de Update e Insert).

Concepto: Es una operación fundamental para la consistencia en sistemas distribuidos, ya que garantiza la idempotencia (puedes ejecutar la misma operación varias veces sin crear duplicados ni errores).

```
db.productos.updateOne(
  { sku: "TEC-005" }, // Busca este producto
  { $set: { nombre: "Teclado Mecánico", precio: 80 } },
  { upsert: true } // Si no existe, se crea
)
```

Efecto del DELETE sobre documentos.

Para eliminar registros se utilizan los métodos deleteOne (elimina la primera coincidencia) o deleteMany (elimina todas las coincidencias). Al igual que la escritura, la operación es atómica a nivel de un solo documento. Si el filtro proporcionado no coincide con ningún documento, la operación retorna 0.

```
db.productos.deleteOne(
  { sku: "TEC-005" },
 )
```

También pueden eliminarse campos específicos de un documento sin borrar el documento entero utilizando el operador $unset. Permite refactorizar la estructura de los datos cuando hay atributos que no son necesarios sin afectar al resto de la colección.

```
db.productos.updateOne(
  { sku: "TEC-005" },
  { $unset: { dpi: "" } }
 )
```

Ejemplo de bases de datos de Clave-Valor

Se crea un sistema de "Sesiones de Usuario". En el modelo Clave-Valor, no es importante conocer la estructura de los datos, sólo deben guardarse y recuperarse rápido por su ID. A continuación, se crea una colección llamada “sesiones” que funciona estrictamente como Clave-Valor, con los siguientes parámetros:

- Clave (_id): Identificador único de la sesión.
- Valor (datos): Un "blob" o texto donde guardamos todo junto sin separar por campos.

```
db.sesiones.insertOne({
  _id: "session_user_5599",            // CLAVE
  datos: "items=3;total=1500;active=true" // VALOR
})
```

MongoDB replica este dato automáticamente a los nodos secundarios (Redundancia física). Si el servidor principal explota, el dato session_user_5599 existe en las copias.
En relación a la consistencia, en un modelo Clave-Valor, a menudo se prioriza la velocidad. Esta característica puede expresarse explícitamente al escribir: “Write Concern w:1”, que indica a la base de datos que guarde de inmediato el dato sin esperar a la confirmación de réplicas.

```
db.sesiones.insertOne(
  { _id: "session_user_9999", datos: "..." },
   { writeConcern: { w: 1 } } // Solo confirma el primario. Rápido pero
riesgo de perder datos si cae inmediatamente.
)
```

Ejemplos de bases de datos de Documentos

Se crea un sistema de "Carrito de compras". En el modelo Documentos, sí es importante conocer la estructura de los datos. Para insertar un elemento “Laptop Pro” deben indicarse en completitud sus campos y valores, como puede verse a continuación:

```
// Insertar un producto (Patrón Documentos)
db.productos.insertOne({
   sku: "LAP-X1",
   nombre: "Laptop Pro",
   marca: {                // Sub-documento
      nombre: "Dell",
      pais: "USA"
   },
   tags: ["oferta", "gamer"] // Array indexable
})
```

En las bases de datos de Documentos, puede operarse sobre los campos de los valores, ya que sí se conoce su estructura interna.

Con relación a la consistencia, se puede afirmar que es mucho más compleja que en una Base de Datos relacional. En las Bases de Datos de Documentos, y NoSQL en general, se elige generar redundancia para operar más rápido sobre los datos. Por lo tanto, si se desea actualizar un campo presente en muchos valores, debe hacerse uno a uno. En caso de generarse un error mientras se realiza esta actualización, la base de datos podría quedar con inconsistencias.

```
db.productos.updateMany(
      { "marca.nombre": "Dell" },  // Filtro: Busca todos los Dell
      { $set: { "marca.pais": "China" } }) // Acción: Actualiza el país
```

Comparación

BD de documentos vs. clave-valor

Tanto las bases de datos orientadas a documentos como las de clave-valor tienen ventajas y desventajas según el caso de uso y los requisitos de la aplicación. Las bases de datos orientadas a documentos ofrecen más flexibilidad y expresividad, ya que permiten consultas basadas en contenido, indexación por campos y agregaciones. Sin embargo, esto también conlleva la posibilidad de que haya más gastos generales, incoherencias y complejidad.

Por otro lado, las bases de datos clave-valor ofrecen mayor simplicidad y rendimiento que las bases de datos orientadas a documentos, ya que puede almacenar y recuperar datos con operaciones mínimas, baja latencia y alta disponibilidad, gracias a su modelo simple, donde cada valor se accede únicamente por una clave. Sin embargo, su simplicidad implica que no ofrecen consultas complejas, búsquedas por campos internos ni procesamiento avanzado de datos.

En términos generales, se debe elegir una base de datos orientada a documentos si los datos son complejos, dinámicos o diversos, y si son necesarias funcionalidades como búsquedas por contenido o agregaciones. Por otro lado, una base de datos clave-valor es más adecuada para datos simples, coherentes o binarios que requieren operaciones básicas como lectura, escritura o eliminación, y cuando se prioriza el rendimiento.
Por ejemplo, para una plataforma de publicaciones con distintos formatos, categorías, etiquetas, comentarios y calificaciones se puede beneficiar de una base de datos orientada a documentos, ya que puede manejar la diversidad y complejidad de los datos y permitir consultas y análisis complejos. Otro caso de uso clásico para este tipo de modelo es para catálogos de productos.

En contraste, una aplicación que requiera mantener sesiones de usuario, se beneficiaría del uso de una base de datos clave-valor ya que garantiza el acceso rápido a la información crítica como las preferencias del usuario. Otro caso de uso real sería para manejar el estado en tiempo real de un envío (clave), asumiendo que cada paquete tiene un número de rastreo que corresponde a su estado de envío y ubicación actual (valor), esto permite obtener y actualizar información rápidamente, garantizando una entrega precisa.

NoSQL vs. SQL

A nivel conceptual, el paradigma relacional y el paradigma NoSQL representan dos enfoques diferentes para modelar, almacenar y gestionar la información, cada uno optimizado para contextos distintos.

En primer lugar, el paradigma relacional utiliza un modelo basado en tablas (modelo relacional), donde los datos se representan mediante tuplas (filas) y atributos (columnas). cada entidad del mundo real se modela como una tabla y los vínculos entre entidades se expresan mediante relaciones explícitas , usualmente a través de claves primarias y foráneas. Este modelo requiere un esquema rígido y definido antes de almacenar

información lo que garantiza coherencia estructural y facilita el uso de, por ejemplo, el alterna relacional para realizar consultas declarativas mediante SQL. El énfasis está en la integridad de los datos, la normalización y la consistencia transaccional, apoyándose en las propiedades ACID como base del diseño.

Por su parte, el paradigma NoSQL agrupa una familia de modelos alternativos a relacional, entre los que podemos encontrar: clave-valor, documentos, basado en columnas y grafos. En estos sistemas, los datos se organizan en agregados: unidades de información que se almacenan y manipulan juntas. A diferencia de un esquema rígido, los modelos NoSQL son flexibles o schema-less, lo cual permite que los datos adopten estructuras variadas dentro de una misma colección o conjunto. Las relaciones entre datos suelen ser implícitas o minimizadas puesto que se prioriza la coherencia dentro del agregado antes que entre múltiples entidades. Conceptualmente, este paradigma prioriza la escalabilidad horizontal, el procesamiento distribuido y la adaptación a datos heterogéneos. En muchos casos, esto implica aceptar una consistencia eventual en lugar de una consistencia estricta.

En resumen, el paradigma relacional parte de un modelo matemático formal orientado a estructuras fijas, relaciones explícitas y consistencia fuerte, mientras que el paradigma NoSQL adopta modelos más flexibles, orientadas a agregados y optimizados para escalar en entornos distribuidos, permitiendo representar datos complejos o cambiantes sin al necesidad de un esquema estricto.

MongoDB vs. SGBD relacional

MongoDB almacena los datos en documentos BSON, que son representaciones binarias de documentos JSON, mientras que los SGBD relacionales almacenan la información en tablas. En MongoDB cada documento debe tener un campo _id, cuyo valor debe ser único dentro de la colección, inmutable y puede ser de cualquier tipo que no sea un arreglo.

En la equivalencia de conceptos, las filas se convierten en documentos, las columnas en campos, las tablas en colecciones, y la base de datos pasa a ser un conjunto de colecciones en lugar de un conjunto de tablas. Las vistas se mantienen como concepto en ambos modelos.

Gracias a que MongoDB permite el uso de arreglos, es posible almacenar múltiples valores relacionados dentro de un mismo campo. En cambio, en un SGBD relacional cada valor ocuparía una columna distinta. Por ejemplo, una ubicación geográfica en MongoDB puede guardarse como [latitud, longitud], mientras que en un modelo relacional se necesitan dos columnas separadas: *latitud* y *longitud*.

Además, debido al soporte de arreglos y documentos anidados, en MongoDB ciertas relaciones muchos-a-uno pueden representarse dentro del mismo documento. Esto permite evitar realizar *joins* entre distintas tablas, ya que la información relacionada se almacena junta desde el inicio.

Otra diferencia importante es el manejo de valores nulos. En un modelo relacional, las columnas pueden tomar el valor NULL, lo cual a veces se desaconseja porque no

resulta claro si el dato no existe o simplemente es desconocido. En MongoDB, si un campo no tiene valor, directamente puede omitirse, o colocarse null si se prefiere. También, si se agrega un nuevo campo a los documentos, solo aparece en los documentos que lo necesiten, mientras que en un SGBD relacional agregar una nueva columna genera que muchas filas queden con valores NULL.

El enfoque de MongoDB prioriza almacenar datos muy relacionados dentro del mismo documento para evitar los *joins*, que suelen ser operaciones costosas. Esta práctica se conoce como *embedding*. Sin embargo, para evitar la duplicación excesiva de datos, MongoDB también permite el uso de referencias a otros documentos.

Referencias

- Milvus. (s. f.). *How Does Data Redundancy Work in Document Databases?*. Milvus AI Quick Reference. https://milvus.io/ai-quick-reference/how-does-data-redundancy-work-in-document-databases
- Bariya, J. (2024). *Understanding Quorum-Based Approaches in Distributed Systems*. DEV Community. https://dev.to/jaiminbariya/understanding-quorum-based-approaches-in-distributed-systems-jaimin-bariya-5h1b
- IBM. (s. f.). *CAP theorem*. IBM Topics. https://www.ibm.com/think/topics/cap-theorem
- IBM. (s. f.). *NoSQL databases*. IBM Think. https://www.ibm.com/es-es/think/topics/nosql-databases APA Educación+2Scribbr+2

- BigData.IR. (2016). *Documento sobre bases NoSQL* [PDF]. https://www.bigdata.ir/wp-content/uploads/2016/08/6C594096C6E33AE41A6D365F2C4588C6.pdf Normas APA+1

- MongoDB, Inc. (s. f.). *Mapear los términos y conceptos entre SQL y MongoDB*. MongoDB Developer. https://www.mongodb.com/developer/products/mongodb/map-terms-concepts-sql-mongodb/

- MongoDB, Inc. (s. f.). *¿Qué es NoSQL? Explicado*. MongoDB Resources. https://www.mongodb.com/es/resources/basics/databases/nosql-explained