**Portafolio Módulo 4: Bases de Datos Para Ingenieros de Datos**

El siguiente Documento consolida el trabajo realizado durante el módulo 4 en el cual se desarrolla cada uno de los ejercicios indicados en clases y los que se deben desarrollar como término de cada clase.

**Clase M4 AE1: Tecnologías de Bases de Datos. Comparativa entre RDBMS y NOSQL:**

Las bases de datos relacionales utilizan un sistema rígido organizado en filas y columnas y que generan alta integridad en los datos. las bases de datos no relacionales utilizan diversos modelos como documentos clave valor grafos y tiene un esquema flexible que permite almacenar datos sin una estructura fija. A continuación, una tabla con las principales diferencias:

|  |  |  |
| --- | --- | --- |
| Característica | RDBMS (Relacional) | NoSQL (No Relacional) |
| Modelo de datos | Tablas, filas y columnas. Los datos están conectados por relaciones (claves primarias y foráneas). | Varios modelos: documentos, clave-valor, grafos, columnas. |
| Esquema | Esquema rígido y predefinido. Los datos deben ajustarse a la estructura de la tabla. | Esquema flexible y dinámico. Permite almacenar datos de diferentes tipos sin una estructura fija. |
| Escalabilidad | Principalmente vertical (aumentar la capacidad del servidor, como RAM y CPU). | Principalmente horizontal (añadir más servidores al clúster para distribuir la carga). |
| Integridad de datos | Alta, garantizada por las propiedades ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), que aseguran que las transacciones sean confiables. | Baja o "consistencia eventual". Las propiedades BASE (Básicamente Disponible, Consistencia Eventual, Estado Flexible) priorizan la disponibilidad y el rendimiento sobre la consistencia inmediata. |
| Lenguaje de consulta | Utiliza SQL (Structured Query Language), un lenguaje estándar y robusto para consultas complejas. | No tiene un lenguaje estandarizado. Cada sistema tiene su propia API o lenguaje de consulta (ej. MQL en MongoDB). |

**Ejercicio FlashMarket:**

**Comparativa RDBMS vs NoSQL Caso de negocio: Plataforma de entrega rápida tipo “FlashMarket”**

**Una startup chilena llamada FlashMarket ha lanzado una aplicación móvil que permite a los usuarios realizar pedidos de productos de supermercados y tiendas de barrio con entregas en menos de 30 minutos.**

**El sistema permite:**

**· Gestionar perfiles de usuario, direcciones, y métodos de pago.**

Se sugiere el uso de una Base de Datos Relacional, que permite manejarla con tablas para mejor comodidad de uso.

Se pueden usar diversas claves, que nos permiten hacer consultas para reportes.

Nos permite agrupar por diferentes criterios, ejemplo coordinar los despachos.

**· Mostrar inventario actualizado por tienda en tiempo real.**

Se sugiere una Base de Datos NO SQL, debido a la disponibilidad que requiere la información, incluso ante caídas o fallas del sistema.

**· Registrar cientos de pedidos por minuto durante horas punta.**

Se sugiere una Base de Datos NO SQL, debido a la disponibilidad que requiere la información, la velocidad requerida y la posibilidad de aumentar la capacidad del servicio en momentos clave.

Debe ser flexible para soportar para poder crecer en caso de ser necesario, por lo que sugiere que pueda crecer de manera horizontal.

**· Funcionar de forma estable en varias ciudades simultáneamente.**

Se sugiere una Base de Datos NO SQL, que se puede distribuir en diferentes nodos, distribuyendo la carga en el procesamiento de datos y brindando redundancia ante posibles caídas.

Aunque los datos no estén disponibles el sistema debe mantenerse en operación

· **Mantener la experiencia del cliente sin errores, incluso cuando hay desconexiones o latencias.**

La Base de Datos NO SQL, permite entregar una experiencia ininterrumpida y sin fallas, con su tolerancia a fallas y gracias a la distribución en diferentes nodos.

Aunque los datos no estén disponibles el sistema debe mantenerse en operación.

Además, los usuarios pueden:

**· Consultar el historial de compras.**

Base de Datos Relacional, permite realizar consultas fidedignas, con la posibilidad de compartir la información para análisis con diversas aplicaciones.

**· Recibir recomendaciones personalizadas según su ubicación y hábitos.**

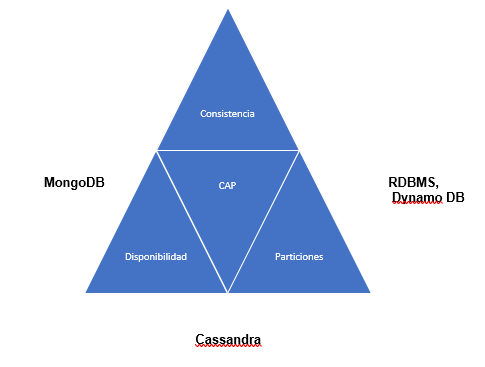
Base de datos NO SQL, debido a las diversas opiniones de los clientes y permite verificar hábitos de compra, ubicación, etc.

**Conclusión:**

Se sugiere un Híbrido: Relacional y No SQL.

NOSQL para la gran cantidad de datos en tiempo real con disponibilidad constante y potenciando con Bases Relacionales para todo lo que se relaciona a los datos de los clientes, compras confirmadas y pagos.

**Mapa Visual del teorema CAP:**



El **teorema CAP** afirma que es imposible que un sistema de datos distribuidos garantice simultáneamente las tres propiedades siguientes:

**Consistencia (Consistency**): Todos los nodos del sistema muestran los mismos datos en un momento dado.

**Disponibilidad (Availability**): Cada solicitud de un cliente recibe una respuesta, incluso si hay fallos en algunos nodos.

**Tolerancia a particiones (Partition Tolerance):** El sistema sigue funcionando incluso cuando hay pérdida de comunicación entre nodos.

Según este teorema, un sistema puede garantizar solo dos de estas tres propiedades a la vez. Esto significa que, dependiendo del caso de uso, se debe priorizar entre consistencia, disponibilidad o tolerancia a particiones.

**Preguntas de la clase:**

1. **¿Qué diferencias observas entre una base de datos relacional y una NoSQL, en cuanto a estructura y flexibilidad?**

Las Bases de Datos son Estructuradas, son más rígidas. Las Bases de datos NoSQL son más flexibles y soportan otro tipo de datos.

1. **¿Por qué crees que empresas como Netflix o Facebook optan por bases de datos NoSQL?**

Las Bases de Datos NoSQL no son estructuradas, permiten crecimiento horizontal, pueden estar divididas en regiones.

1. **¿Qué significa que una base de datos sea "eventualmente consistente"? ¿En qué casos puede ser un problema?**

Quiere decir que los datos se no actualizan hasta un procesamiento específico, lo que es propio de las Bases No SQL, que sacrifican consistencia por disponibilidad y poder estar particionadas.

**Clase M4 AE2: Tecnologías de Bases de Datos. Optimización de Consultas en una RDBMS:**

**Diagnóstico y Consulta de Una Consulta Lenta:**

En el ejercicio planteado por el docente se crean 3 bases de datos mediante CREATE TABLE y posteriormente se le ingresa información mediante INSERT TO. Se muestra una consulta a la Base de datos con SELECT \*, lo que genera una consulta de todas las bases de datos utilizando muchos recursos y cargando datos que podrían ser innecesarios para la consulta. Se sugiere realizar consultas a los campos necesarios mejorando los JOIN.

El uso de EXPLAIN nos permite visualizar el ”plan de ejecución” sin que se realice la consulta, lo que nos permitiría revisar y mejorar la consulta. Por otro lado, EXPLAIN ANALYZE nos permite ejecutar la consulta y podremos visualizar los tiempos de ejecución y poder detectar errores o cuellos de botella, es más lento. Ya que realiza la consulta.

**Código Original:**

SELECT \*

FROM pedidos

JOIN clientes ON pedidos.cliente\_id = clientes.id

JOIN productos ON pedidos.producto\_id = productos.id

WHERE UPPER(clientes.nombre) = 'JUAN PÉREZ'

AND YEAR(pedidos.fecha) = 2023

AND productos.categoria LIKE '%tecnología%'

AND pedidos.estado = 'entregado';

**Mejora en el código:**

SELECT pedidos.id, pedidos.fecha, pedidos.estado, clientes.nombre, productos.nombre AS producto, productos.categoria

FROM pedidos

JOIN clientes ON pedidos.cliente\_id = clientes.id

JOIN productos ON pedidos.producto\_id = productos.id

WHERE UPPER(clientes.nombre) = 'JUAN PÉREZ'

AND pedidos.fecha >= '2023-01-01'

AND pedidos.fecha < '2024-01-01'

AND productos.categoria LIKE '%tecnología%'

AND pedidos.estado = 'entregado';

**Preguntas a la clase:**

**1. ¿Por qué una consulta que funciona correctamente puede no ser eficiente?**

Una consulta que devuelve el resultado correcto puede no ser eficiente porque la forma en que está escrita o el diseño de la base de datos la obligan a realizar operaciones innecesarias o costosas. La eficiencia no se mide por si la consulta funciona, sino por la cantidad de recursos que consume (tiempo de CPU, memoria, operaciones de E/S de disco) para ejecutarse.

**2. ¿Qué impacto tiene el uso de SELECT \* en bases de datos grandes?**

El uso de SELECT \* (seleccionar todas las columnas) en bases de datos grandes tiene un impacto negativo en el rendimiento debido a que requiere de muchos recursos y obliga a leer toda la base de datos, cuando posiblemente sólo se requiere de algunos datos en forma específica. Esto disminuye los tiempos de respuesta y consume muchos recursos.

**3. ¿Cuándo conviene normalizar y cuándo desnormalizar una base de datos?**

La normalización y la desnormalización son técnicas de diseño que buscan equilibrar la integridad de los datos con el rendimiento de las consultas.

Cuándo normalizar: Cuando la consistencia de los datos es crítica, Cuando se requiere hacer muchas consultas a l base de datos

Cuándo desnormalizar: En almacenes de datos (Data Warehouses) y sistemas de análisis (OLAP), donde el rendimiento de las consultas de lectura (SELECT) es la prioridad principal y las operaciones de escritura son menos frecuentes. Cuando el volumen de datos es muy grande y las lecturas se vuelven muy lentas.

**M4 AE3 Bases de Datos No Relacionales**

**Diseña tu Primera Base de datos NoSQL**

**Diseña la estructura de una base de datos NoSQL para una app de mensajería en tiempo real (tipo WhatsApp). Debe incluir el tipo de base de datos elegido, cómo se modelarían los datos y cómo se escalaría:**

Para este Caso se recomienda el uso de una base de datos NoSQL para documentos, que permita almacenar mensajes, usuarios, fechas y horas.

Debe incluir Datos como:

**Usuarios**, donde estará la información de los usuarios y sus contactos

**Mensajes**, donde estarán los mensajes, con fechas y horas.

La escalabilidad sería horizontal, lo que permitiría crecimiento en caso de ser necesario y redundancia y tolerancia a fallos de los nodos.

**Preguntas a la clase:**

**¿Por qué surgieron las bases de datos NoSQL?**

Las Bases de Datos NoSQL Surgen para flexibilizar el tipo de datos y la posibilidad de fácil crecimiento.

**¿Qué ventajas tiene un modelo documental frente a uno relacional?**

Las Bases de datos NoSQL son más flexibles, más rápidas y con alta disponibilidad.

**¿En qué situaciones sería contraproducente usar una base NoSQL?**

En Bases de datos pequeñas, rígidas, muy estructuradas y que requieren de alta consistencia y disponibilidad.

**M4 Clase AE4 Simulación de diseño y operación de un cluster:**

**1: Definir el número de centros de datos y nodos.**

Se sugiere crear 2 sentros de datos: América y Europa, para asegurar disponibilidad en ambos. Cada Centro de datos puede contar con 3 nodos, 6 en total.

**2: Seleccionar la estrategia de replicación**

La estrategia de replicación ideal para un clúster de múltiples centros de datos es NetworkTopologyStrategy. Esta estrategia te permite especificar el factor de replicación para cada centro de datos por separado. Así, el clúster puede ser escalable horizontalmente y puede mantener la alta disponibilidad si una región completa falla.

**3: Proponer un keyspace y una tabla**

Se propone un keyspace llamado streaming\_data y una tabla user\_views para registrar las visualizaciones de contenido:

CREATE KEYSPACE streaming\_data WITH replication = {

'class': 'NetworkTopologyStrategy',

'datacenter1': '3',

'datacenter2': '3'

};

USE streaming\_data;

CREATE TABLE user\_views (

user\_id UUID,

episode\_id UUID,

timestamp TIMESTAMP,

PRIMARY KEY ((user\_id), episode\_id, timestamp)

);

**4: Escribir consultas optimizadas**

A continuación, se presentan dos consultas CQL optimizadas para este modelo de datos.

**Consulta 1**: Buscar todos los episodios vistos por un usuario específico:

SELECT episode\_id, timestamp FROM user\_views WHERE user\_id = 9e2832c3-442c-47d0-80d4-722168923a1a

Esta consulta es muy eficiente porque usa la clave de partición user\_id, lo que permite a Cassandra encontrar los datos de un usuario directamente en una sola partición.

**Consulta 2**: Buscar el último episodio visto por un usuario específico:

SELECT episode\_id FROM user\_views WHERE user\_id = 9e2832c3-442c-47d0-80d4-722168923a1

Esta consulta también usa la clave de partición y aprovecha la clave de agrupación timestamp para obtener el último registro de forma muy rápida.

**5: Exposición del diseño**

El diseño utiliza **NetworkTopologyStrategy** para lograr alta disponibilidad y rtolerancia a fallos en 2 centros de datos. La elección de **user\_id** como clave de partición permite optimizar las consultas en el historial de visualizaciones de un usuario, así se minimiza el número de nodos que deben ser consultados.

**CLASE M4 AE5 Mongodb**

**Modelado de Colecciones para una APP de E-commerce**

Se consideró el caso de una tienda de ropa, donde se embebieron los datos de tallas y colores, pues el tipo de vestido se encuentra precisamente con esas variantes que pertenecen al producto:

{

"nombre": "Vestido Floral Primavera",

"descripcion": "Vestido largo estampado, ideal para eventos.",

"precio\_base": 59.99,

"categoria": "mujer > vestidos",

"marca": "Mango",

"variantes": [

{

"color": "rojo",

"talle": "S",

"sku": "MNG-VFL-RO-S",

"precio": 59.99

“reviews”:10

{“stocksucursalnorte:5

“stocksucursalsur:5

“stocksucursalcentro:5

},

{

"color": "rojo",

"talle": "M",

"sku": "MNG-VFL-RO-M",

"precio": 59.99

“stock”:10

},

{

"color": "azul",

"talle": "S",

"sku": "MNG-VFL-AZ-S",

“stock”:8

**Preguntas a la Clase:**

**1 ¿En qué casos es más conveniente usar documentos embebidos en lugar de documentos relacionados?**

Es más conveniente usar documentos embebidos cuando la relación entre los datos es de "uno a pocos" y los datos secundarios no se consultan sin el documento principal. Esto evita las uniones o joins que son costosas en una base de datos no relacional como MongoDB. Permiten una consulta más rápida.

**2 ¿Cómo afecta el diseño de colecciones al rendimiento de MongoDB?**

Un buen diseño disminuye los datos que el motor de la base de datos necesita leer de disco para satisfacer una consulta. Al usar un esquema adecuado, puedes evitar joins costosos y optimizar las consultas, ganando rendimiento. Por el contrario, un diseño pobre puede llevar a documentos muy grandes o a tener que hacer múltiples consultas para obtener toda la información, lo que disminuye el rendimiento.

**3 ¿Qué ventajas tiene consultar documentos con $and y $regex comparado a una consulta básica?**

Consultar documentos con $and y $regex en MongoDB permite realizar búsquedas más complejas y flexibles que una consulta básica. Con ello se mejora el rendimiento y se optimiza las búsquedas.

**M4 AE6 DynamoDB**

**Cree la tabla Comunas y creen las acciones de un CRUD en Dynamo DB**

En el ejercicio se busca practicar el CRUD (Crear/Create, Leer/Read, Actualizar/Update y Eliminar/Delete) en elementos de una base de datos en DynamoDB, si bien el proceso no es complejo, la nomenclatura es bastante susceptible a errores, pues la falta de un guion, apóstrofo o espacio genera errores en el código. Se recomienda el uso de herramientas que permitan automatizar estos procesos, como **AWS Management Console.**

##crear tabla##

aws dynamodb create-table `

--table-name Comunas `

--attribute-definitions AttributeName=ComunaID,AttributeType=S `

--key-schema AttributeName=ComunaID,KeyType=HASH `

--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 `

--endpoint-url http://localhost:8000 `

--region us-west-2

###put item### Crear###

aws dynamodb put-item --table-name Comunas --item '{\"ComunaID\": {\"S\": \"001\"}, \"Nombre\": {\"S\": \"Talca\"}}' --endpoint-url http://localhost:8000 --region us-west-2

aws dynamodb put-item --table-name Comunas --item '{\"ComunaID\": {\"S\": \"003\"}, \"Nombre\": {\"S\": \"Santiago\"}}' --endpoint-url http://localhost:8000 --region us-west-2

aws dynamodb put-item --table-name Comunas --item '{\"ComunaID\": {\"S\": \"002\"}, \"Nombre\": {\"S\": \"La Florida\"}}' --endpoint-url http://localhost:8000 --region us-west-2

###get item### Leer###

aws dynamodb get-item `

--table-name Comunas `

--key '{\"ComunaID\": {\"S\": \"001\"}}' `

--endpoint-url http://localhost:8000 `

--region us-west-2

###scan###Buscar

aws dynamodb scan `

--table-name Comunas `

--endpoint-url http://localhost:8000 `

--region us-west-2

###update###Modificar###

aws dynamodb update-item `

--table-name Comunas `

--key '{\"ComunaID\": {\"S\": \"001\"}}' `

--update-expression "SET Nombre= :cambiacomuna" `

--expression-attribute-values '{\":cambiacomuna\": {\"S\": \"Curicó\"}}' `

--endpoint-url http://localhost:8000 `

--region us-west-2

###delete###Borrar###

aws dynamodb delete-item `

--table-name Comunas `

--key '{\"ComunaID\": {\"S\": \"001\"}}' `

--endpoint-url http://localhost:8000 `

--region us-west-2

**Diseño de una Tabla y escritura de datos en DynamoDB:**

###Crear Tabla###

aws dynamodb create-table `

--table-name Reservas `

--attribute-definitions AttributeName=ReservasID,AttributeType=S `

--key-schema AttributeName=ReservasID,KeyType=HASH `

--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 `

--endpoint-url http://localhost:8000 `

--region us-west-2

###put item##

aws dynamodb put-item --table-name Reservas --item '{\"ReservasID\": {\"S\": \"001\"}, \"Usuario\": {\"S\": \"Pablo LLanos\"}, \"Destino\": {\"S\": \"Brasil\"}, \"Fecha\": {\"S\": \"20/01/2026\"},\"Precio\": {\"N\": \"500000\"}}' --endpoint-url http://localhost:8000 --region us-west-2

aws dynamodb put-item --table-name Reservas --item '{\"ReservasID\": {\"S\": \"002\"}, \"Usuario\": {\"S\": \"Ignacio Llanos\"}, \"Destino\": {\"S\": \"Brasil\"}, \"Fecha\": {\"S\": \"20/01/2026\"},\"Precio\": {\"N\": \"400000\"}}' --endpoint-url http://localhost:8000 --region us-west-2

aws dynamodb put-item --table-name Reservas --item '{\"ReservasID\": {\"S\": \"003\"}, \"Usuario\": {\"S\": \"Luis Rojas\"}, \"Destino\": {\"S\": \"España\"}, \"Fecha\": {\"S\": \"10/01/2026\"},\"Precio\": {\"N\": \"1000000\"}}' --endpoint-url http://localhost:8000 --region us-west-2

###SCAN###

aws dynamodb scan `

--table-name Reservas `

--endpoint-url http://localhost:8000 `

--region us-west-2

###get item###

aws dynamodb get-item `

--table-name Reservas `

--key '{\"ReservasID\": {\"S\": \"001\"}}' `

--endpoint-url http://localhost:8000 `

--region us-west-2

**Preguntas a la Clase:**

**¿En qué casos conviene usar DynamoDB por sobre una base relacional?**

Cuando se quieren usar bases de datos escalables y rápidas. Además, Dynamo DB es un servicio sin servidor.

**¿Cómo afecta el diseño de la clave primaria al rendimiento?**

La clave primaria indica la forma en como se realizará la búsqueda, por lo que nos permite mejorar el rendimiento en las consultas.

**¿Qué ventaja ofrece la integración de DynamoDB con AWS Lambda?**

Ambas tecnologías están son de AWS, por lo que están integradas, por lo que los triggers o gatilladores se implementan de buena manera. Ambas son tecnologías Serverless, por lo que son altamente escalables y de bajo costo.

**Retomar el ejercicio anterior (tabla Reservas Viajes) y crear una carga de ítems por lote y una recuperación de ítems por lote**

aws dynamodb batch-write-item`

--request-items '{

\"Reservas\": [

{

\"PutRequest\": {

\"Item\": {

\"ReservasID\": { \"S\": \"004\" },

\"Usuario\": { \"S\": \"Carlos León\" },

\"Destino\": { \"S\": \"Brasil\" },

\"Fecha\": { \"S\": \"02/02/2026\" } ,

\"Precio\": { \"N\": \"600000\" }

}

}

},

{

\"PutRequest\": {

\"Item\": {

\"ReservasID\": { \"S\": \"005\" },

\"Usuario\": { \"S\": \"Juan Pérez\" },

\"Destino\": { \"S\": \"México\" },

\"Fecha\": { \"S\": \"02/02/2026\" } ,

\"Precio\": { \"N\": \"800000\" }

}

}

},

{

\"PutRequest\": {

\"Item\": {

\"ReservasID\": { \"S\": \"006\" },

\"Usuario\": { \"S\": \"Raúl Valdivia\" },

\"Destino\": { \"S\": \"España\" },

\"Fecha\": { \"S\": \"03/02/2026\" } ,

\"Precio\": { \"N\": \"1000000\" }

}

}

}

]

}' `

--endpoint-url http://localhost:8000 `

--region us-west-2

aws dynamodb batch-write-item `

--request-items '{

\"Reservas\": [

{“ReservasID”:{“S”: “001”}},

{“ReservasID”:{“S”: “002”}},

]

}’`

**Escritura y Recuperación en lote:**

**##Preparación del Entorno##**

aws dynamodb create-table `

--table-name Usuarios `

--attribute-definitions AttributeName=UsuarioID,AttributeType=S `

--key-schema AttributeName=UsuarioID,KeyType=HASH `

--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 `

--endpoint-url http://localhost:8000 `

--region us-west-2

**## escritura por lotes##**

aws dynamodb batch-write-item `

--request-items '{

\"Usuarios\": [

{

\"PutRequest\": {

\"Item\": {

\"UsuarioID\": { \"S\": \"001\" },

\"Nombre\": { \"S\": \"Carlos León\" },

\"Edad\": { \"N\": \"35\" },

\"Ciudad\": { \"S\": \"Talca\" }

}

}

},

{

\"PutRequest\": {

\"Item\": {

\"UsuarioID\": { \"S\": \"002\" },

\"Nombre\": { \"S\": \"Beatriz Soto\" },

\"Edad\": { \"N\": \"40\" },

\"Ciudad\": { \"S\": \"Santiago\" }

}

}

},

{

\"PutRequest\": {

\"Item\": {

\"UsuarioID\": { \"S\": \"003\" },

\"Nombre\": { \"S\": \"Diego Rojas\" },

\"Edad\": { \"N\": \"31\" },

\"Ciudad\": { \"S\": \"Valparaiso\" }

}

}

}

]

}' `

--endpoint-url http://localhost:8000 `

--region us-west-2

**##Búsqueda por lotes##**

aws dynamodb batch-get-item `

--request-items '{

\"Usuarios\": {

\"Keys\": [

{ \"UsuarioID\": { \"S\": \"001\" } },

{ \"UsuarioID\": { \"S\": \"003\" } }

]

}

}' `

--endpoint-url http://localhost:8000 `

--region us-west-2

**Preguntas a la clase:**

**¿Por qué es útil usar Batch en lugar de operaciones individuales?**

Se realiza la carga de datos en un solo proceso de escritura, ahorra tiempo y dinero. Se disminuye el tráfico de datos mejorando el uso de recursos.

**Calcula cuantas acciones se ahorrarían si se usara batch en vez de Put item.**

Se tendría que haber usado 1 put item por cada usuario (3 en total). Se tendrían que haber realizado 2 consultas individuales (2 en total). En síntesis, el proceso de disminuye de 5 a 2 operaciones.

**¿Qué riesgos si no planificamos bien el Particionamiento de Datos?**

Se pierde velocidad en la búsqueda de datos, se puede incurrir en altos costos al tener que rediseñar o reestructurar las bases.

**Análisis Final**

Las Bases de datos NoSQL son más flexibles que las Bases RDBMS, el permitir crecimiento horizontal entrega ventajas como la distribución en diferentes centros de datos con varios nodos y la posibilidad de crecimiento casi infinito. En las bases de datos No SQL MongoDB y DynamoDB son bases de datos NoSQL muy populares, pero tienen diferencias clave en su arquitectura, administración y casos de uso, en la siguiente tabla se muestra una comparativa de ambas bases de datos:

|  |  |  |
| --- | --- | --- |
| Característica | MongoDB | DynamoDB |
| Modelo de datos | Documentos (BSON). | Clave-valor, documentos. |
| Gestión | Autohospedado o servicio administrado (MongoDB Atlas). | Completamente administrado por AWS (servicio serverless). |
| Modelo de esquemas | Dinámico y flexible. Los documentos en una colección pueden tener estructuras diferentes. | Rígido. El diseño de la clave primaria es obligatorio y crítico para el rendimiento. |
| Lenguaje de consulta | MongoDB Query Language (MQL), muy rico y potente. | Operaciones de API simples (Get, Put, Update, Query, Scan). Las consultas se basan en la clave primaria. |
| Rendimiento | Alto rendimiento, pero la consistencia y el rendimiento pueden variar según la configuración. | Rendimiento predecible y constante con una latencia de un solo dígito en milisegundos. |
| Escalabilidad | El sharding (fragmentación) se puede configurar manualmente o de forma automática con MongoDB Atlas. | Escalabilidad automática y transparente. Diseñado para escalar a millones de peticiones por segundo. |
| Transacciones | Soporte para transacciones ACID en múltiples documentos y colecciones. | Soporte para transacciones ACID solo dentro de una sola tabla. |
| Costo | Basado en el tamaño del clúster y el uso, con opciones de precios variadas. | Basado en unidades de capacidad de lectura/escritura (RCU/WCU) o en el uso bajo demanda. |
| Integración | Amplia integración con diversas herramientas y entornos de desarrollo, agnóstico de la nube. | Integración nativa y profunda con el ecosistema de AWS (Lambda, Kinesis, S3). |
| Caso de uso | Ideal para sistemas de gestión de contenido, catálogos de productos y aplicaciones con esquemas cambiantes y consultas complejas. | Ideal para aplicaciones con grandes volúmenes de datos, rendimiento predecible y baja latencia, como IoT, juegos o plataformas de anuncios. |

DynamoDB: Es la mejor opción para aplicaciones que necesitan un rendimiento predecible y consistente a cualquier escala, como back-ends de juegos, IoT, aplicaciones móviles o plataformas de anuncios. Su integración nativa con el ecosistema de AWS (Lambda, S3, Kinesis) la hace ideal para arquitecturas sin servidor.

MongoDB: Es más adecuada para aplicaciones que requieren un esquema flexible, consultas complejas y la capacidad de realizar análisis en la base de datos (a través de su framework de agregación). Es comúnmente utilizada en sistemas de gestión de contenidos (CMS), catálogos de productos y aplicaciones de comercio electrónico.

En resumen, elige **MongoDB** si buscas flexibilidad en el esquema, un lenguaje de consulta rico y control sobre la infraestructura. Opta por **DynamoDB** si priorizas la escalabilidad automática, la baja latencia garantizada y una solución sin servidor que se integre perfectamente con AWS.