

# Módulo 7: Ingeniería fundamental de Big Data

<b>INTRODUCCIÓN .....</b>	<b>4</b>
INGENIERÍA DE DATOS .....	4
INGENIERÍA DE BIG DATA .....	4
CARACTERÍSTICAS Y DESAFÍOS DE LA INGENIERÍA DE BIG DATA .....	5
INGENIERÍA DE BIG DATA .....	5
PÓSTER DEL MAPA MENTAL .....	7
<b>TERMINOLOGÍA Y CONCEPTOS DE ALMACENAMIENTO DE BIG DATA.....</b>	<b>8</b>
SHARDING .....	9
REPLICACIÓN .....	11
REPLICACIÓN: MAESTRO-ESCLAVO .....	11
REPLICACIÓN: PEER-TO-PEER .....	14
SHARDING Y REPLICACIÓN .....	17
COMBINAR SHARDING Y REPLICACIÓN MAESTRO-ESCLAVO.....	18
COMBINAR SHARDING Y REPLICACIÓN PEER-TO-PEER.....	18
LECTURA .....	18
TEOREMA CAP .....	21
LECTURA .....	23
ACID .....	23
LECTURA .....	27
BASE .....	27
EJERCICIO 7.1: COMPLETE LOS ESPACIOS EN BLANCO.....	31
<b>CARACTERÍSTICAS DEL DISPOSITIVO DE ALMACENAMIENTO DE BIG DATA .....</b>	<b>37</b>
ESCALABILIDAD .....	37
ESCALABILIDAD: ESCALABILIDAD VERTICAL .....	38
ESCALABILIDAD: ESCALABILIDAD HORIZONTAL.....	38
ESCALABILIDAD .....	38
REDUNDANCIA Y DISPONIBILIDAD .....	38
ACCESO RÁPIDO.....	39
ALMACENAMIENTO A LARGO PLAZO .....	39
ALMACENAMIENTO SIN ESQUEMA.....	40
ALMACENAMIENTO DE BAJO COSTO.....	41

<b>DISPOSITIVOS DE ALMACENAMIENTO EN DISCO .....</b>	<b>47</b>
DISPOSITIVO DE ALMACENAMIENTO EN DISCO .....	47
DISPOSITIVO DE ALMACENAMIENTO EN DISCO: SISTEMA DE ARCHIVOS DISTRIBUIDO .....	47
DISPOSITIVO DE ALMACENAMIENTO EN DISCO: BASE DE DATOS .....	50
RDBMS .....	50
DISPOSITIVO DE ALMACENAMIENTO EN DISCO: NoSQL .....	54
DISPOSITIVO DE ALMACENAMIENTO EN DISCO: CARACTERÍSTICAS DE NoSQL .....	54
DISPOSITIVO DE ALMACENAMIENTO EN DISCO: JUSTIFICACIÓN DE NoSQL .....	55
DISPOSITIVO DE ALMACENAMIENTO EN DISCO: TIPOS DE NoSQL .....	56
NoSQL: LLAVE-VALOR (KEY-VALUE).....	58
LECTURA .....	59
NoSQL: DOCUMENTO .....	59
LECTURA .....	61
NoSQL: BASADO EN COLUMNAS.....	61
LECTURA .....	63
NoSQL: GRAFO .....	63
LECTURA .....	66
NewSQL.....	66
EJERCICIO 7.2: SELECCIONE EL DISPOSITIVO DE ALMACENAMIENTO CORRECTO .....	67
<b>CARACTERÍSTICAS DEL MOTOR DE PROCESAMIENTO DE BIG DATA.....</b>	<b>73</b>
PROCESAMIENTO DE DATOS DISTRIBUIDOS/PARALELOS .....	73
PROCESAMIENTO DE DATOS SIN ESQUEMA .....	73
SOPORTE PARA MÚLTIPLES CARGAS DE TRABAJO.....	74
ESCALABILIDAD LINEAL .....	74
REDUNDANCIA Y TOLERANCIA A ERRORES .....	75
BAJO COSTO .....	75
EJERCICIO 7.3: ENCUENTRE EL TÉRMINO CORRESPONDIENTE PARA CADA ENUNCIADO.....	76
<b>PROCESAMIENTO FUNDAMENTAL DE BIG DATA.....</b>	<b>81</b>
PROCESAMIENTO DE BIG DATA: CLUSTER .....	82
PROCESAMIENTO DE BIG DATA: MODO POR LOTES (BATCH PROCESSING) .....	83
PROCESAMIENTO DE BIG DATA: MODO EN TIEMPO REAL .....	83
<b>PRESENTACIÓN DEL MOTOR DE PROCESAMIENTO DE MAPREDUCE.....</b>	<b>89</b>
CONCEPTOS DE MAPREDUCE .....	90
MAPREDUCE: MAPEAR.....	90

MAPREDUCE: COMBINAR .....	91
MAPREDUCE: DIVIDIR .....	92
MAPREDUCE: MEZCLAR Y CLASIFICAR .....	93
MAPREDUCE: REDUCIR .....	94
MAPREDUCE: EJEMPLO .....	95
EJERCICIO 7.4: ORGANICE LAS ETAPAS DE MAPREDUCE .....	98
<b>DISEÑO FUNDAMENTAL DEL ALGORITMO DE MAPREDUCE .....</b>	<b>102</b>
PARALELISMO DE TAREAS .....	103
PARALELISMO DE DATOS .....	104
DISEÑO DEL ALGORITMO DE MAPREDUCE .....	104
DISEÑO DEL ALGORITMO DE MAPREDUCE: CONSIDERACIONES.....	105
EJERCICIO 7.5: COMPLETE LOS ESPACIOS EN BLANCO.....	106
<b>RESPUESTAS A LOS EJERCICIOS .....</b>	<b>112</b>
RESPUESTAS AL EJERCICIO 7.1 .....	112
RESPUESTAS AL EJERCICIO 7.2 .....	113
RESPUESTAS AL EJERCICIO 7.3 .....	113
RESPUESTAS AL EJERCICIO 7.4 .....	113
RESPUESTAS AL EJERCICIO 7.5 .....	114
<b>EXAMEN B90.07 .....</b>	<b>115</b>
<b>KIT DE AUTOAPRENDIZAJE DEL MÓDULO 7.....</b>	<b>115</b>
<b>INFORMACIÓN Y RECURSOS DE CONTACTO .....</b>	<b>116</b>
COMUNIDAD DE AITCP .....	116
INFORMACIÓN GENERAL DEL PROGRAMA .....	116
INFORMACIÓN GENERAL ACERCA DE LOS MÓDULOS DEL CURSO Y LOS KITS DE AUTOAPRENDIZAJE ..	116
INQUIETUDES ACERCA DEL EXAMEN DE PEARSON VUE .....	116
PROGRAMACIÓN DE TALLERES DIRIGIDOS AL PÚBLICO Y GUIADOS POR INSTRUCTORES .....	116
TALLERES PRIVADOS GUIADOS POR INSTRUCTORES .....	117
CONVERTIRSE EN UN ENTRENADOR CERTIFICADO .....	117
INQUIETUDES GENERALES SOBRE BDSCP .....	117
NOTIFICACIONES AUTOMÁTICAS.....	117
RETROALIMENTACIÓN Y COMENTARIOS .....	117

# Introducción

Este es el cuaderno de trabajo oficial del **Módulo 7 Ingeniería fundamental de Big Data** para el curso del BDSCP y su respectivo **Examen B90.07** de Pearson VUE.

El objetivo de este documento es brindar conocimientos sobre la ingeniería fundamental de Big Data, lo cual incluye, entre otros:

- Terminología y conceptos de almacenamiento de Big Data
- Características del dispositivo de almacenamiento de Big Data
- Dispositivos de almacenamiento en disco
- Características del motor de procesamiento de Big Data
- Procesamiento fundamental de Big Data
- Presentación del motor de procesamiento de MapReduce
- Diseño fundamental del algoritmo de MapReduce

## Ingeniería de datos

La ingeniería de datos es el campo que se encarga del desarrollo, prueba, implementación y mantenimiento de las soluciones de procesamiento de datos por medio de la recopilación, análisis, transformación, unión, procesamiento y manejo de datos.

Dos actividades principales que comprenden la ingeniería de datos son el **almacenamiento y procesamiento de datos**, el cual por lo general está estructurado. La tarea de un ingeniero de datos es hacer los datos aptos para varios tipos de análisis de datos (Data Analysis), incluyendo el desarrollo de modelos (minería de datos (Data Mining) y otros algoritmos específicos del proceso de negocios) y presentación de informes.

## Ingeniería de Big Data

Dentro del ámbito de Big Data, la ingeniería de datos comprende el **desarrollo de soluciones de procesamiento de datos con un alto nivel de distribución, escalabilidad y tolerancia a errores para procesar grandes cantidades de datos** con el fin de reunir información. La ingeniería de Big Data comprende el procesamiento de datos para respaldar el ciclo de vida del análisis de Big Data, como se analiza en el Módulo 2.

Los ingenieros de Big Data aseguran la disponibilidad de los datos para que los científicos de datos desarrollen modelos y productos de datos. Es necesario que tengan conocimientos acerca de varias alternativas tecnológicas de procesamiento y almacenamiento de datos para adquirir, almacenar y procesar datos que generalmente son de carácter semiestructurado y sin estructurar.

## **Características y desafíos de la ingeniería de Big Data**

Muchos de los desafíos a los que se enfrenta la ingeniería de Big Data están relacionados con el manejo de las tres V principales de Big Data:

- **volumen**: datasets a escala de internet y lotes asociados, y procesamiento de datos en tiempo real
- **velocidad**: procesamiento de grandes cantidades de datos estructurados, sin estructurar y semiestructurados que son recibidos rápidamente, incluyendo la extracción de datos relevantes de datasets semiestructurados y sin estructurar
- **variedad**: recopilación y agregación de datos provenientes de distintas fuentes con esquemas dispares o sin ningún esquema
- **importación y exportación de grandes cantidades de datos desde y hacia tecnologías de almacenamiento tradicionales**, incluyendo OLTP (sistemas CRM, ERP, SCM) y sistemas OLAP (bodega de datos digital (Data Warehouse))
- **validación y limpieza (Cleansing)** de datos en tiempo real o por lotes y creación de modelos de datos eficientes
- **establecimiento de un entorno óptimo de procesamiento y almacenamiento de datos** con base en el tipo de datos y sus requerimientos de procesamiento
- **desarrollo de algoritmos eficientes de procesamiento de datos** que se ejecuten en clusters de computadoras
- **desarrollo de Big Data Pipelines y aplicaciones de Big Data** que pueden incluir visualizaciones de datos importantes

## **Ingeniería de Big Data**

La certificación como Ingeniero de Big Data proporciona conocimientos profundos acerca de los conceptos y las características del dispositivo de almacenamiento y de los mecanismos del motor de procesamiento que se analizaron en el Módulo 2. Entender estos mecanismos clave es esencial para el Ingeniero de Big Data, ya que son el fundamento de cualquier entorno de solución de Big Data.

El objetivo de todo el conjunto de temas presentados en los Módulos 7 y 8 es **permitirles a los ingenieros, asesores y otros profesionales relacionados con Big Data diseñar y construir soluciones de procesamiento de datos en un entorno Big Data usando tecnologías contemporáneas**.

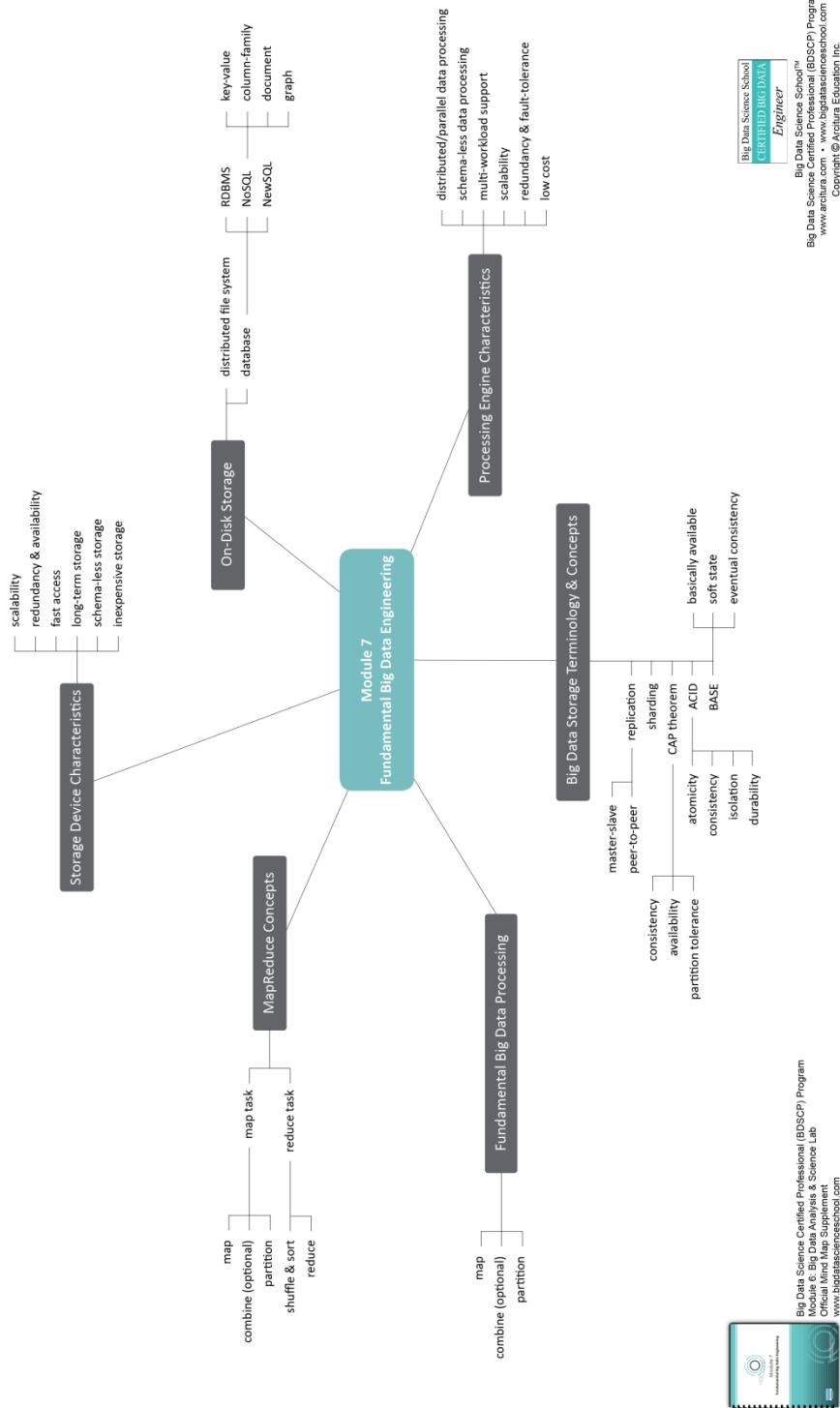
Específicamente, el Módulo 7 explora los temas que abordan el conjunto exclusivo de retos y características de la ingeniería de Big Data. Estos temas incluyen:

- los conceptos, características y tecnologías en disco que soportan el procesamiento por lotes (Batch Processing) de datasets de Big Data

- una introducción a un framework de procesamiento de datos que soporte el procesamiento por lotes (Batch Processing) de datasets de Big Data
- las consideraciones fundamentales para el diseño de algoritmos de procesamiento de datos que funcionan en clusters de computadoras

## Póster del mapa mental

El *Póster del mapa mental del Módulo 7 del BDSCP* adjunto a este cuadernillo ofrece una representación visual alternativa de los principales temas abordados en este curso.



# Terminología y conceptos de almacenamiento de Big Data

La capacidad de almacenar cantidades voluminosas de datos recibidos rápidamente (volumen, velocidad y variedad característicos de Big Data) es esencial para generar cualquier valor a partir de los datasets de Big Data. Los datos pueden almacenarse usando dispositivos basados en disco o en memoria.

Por lo general, los datos deben ser almacenados en un disco antes de que puedan ser procesados. Sin embargo, esto solo se aplica para **el modo de procesamiento por lotes (Batch Processing)**. En el **modo de procesamiento en tiempo real**, los datos son procesados primero en la memoria y después son almacenados en el disco.

Por lo general, los datos adquiridos no se encuentran en un formato o una estructura que pueda ser procesada directamente. Por esto, como resultado de la actividad de manipulación (wrangling) de datos (limpieza (cleansing), filtrado (filtering), preparación de datos), deben ser almacenados nuevamente.

Los datos también deben ser almacenados una vez sean procesados, como resultado de la analítica y con fines de archivo. El almacenamiento es necesario generalmente:

- cuando los datasets son adquiridos o cuando los datos son generados dentro del ámbito empresarial
- cuando los datos son manipulados para que sean aptos para el análisis de datos (Data Analysis)
- cuando los datos son procesados como resultado de una actividad ETL o cuando el producto es el resultado de una operación analítica

## NOTA

Los datasets de Big Data no pueden ser procesados en el modo por lotes ni en tiempo real. En la sección *Presentación del motor de procesamiento de MapReduce* se aborda el modo por lotes, y el modo de procesamiento en tiempo real es analizado en el Módulo 8.

Antes de explorar los tipos de mecanismos del dispositivo de almacenamiento de Big Data, se presentará la siguiente terminología y conceptos básicos.

- **sharding**
- **replicación**
- **teorema CAP**
- **ACID**
- **BASE**

## Sharding

Sharding es el proceso de **particionar horizontalmente un gran dataset en un grupo de datasets más pequeños y manejables llamados shards**. Los shards están distribuidos entre distintos nodos, en donde un nodo es un servidor o una computadora (Figura 7.1). Cada shard es almacenado en un nodo aparte y cada nodo solamente es responsable por los datos que almacena. Cada shard comparte el mismo **esquema**, y todos los shards representan colectivamente la base de datos completa.

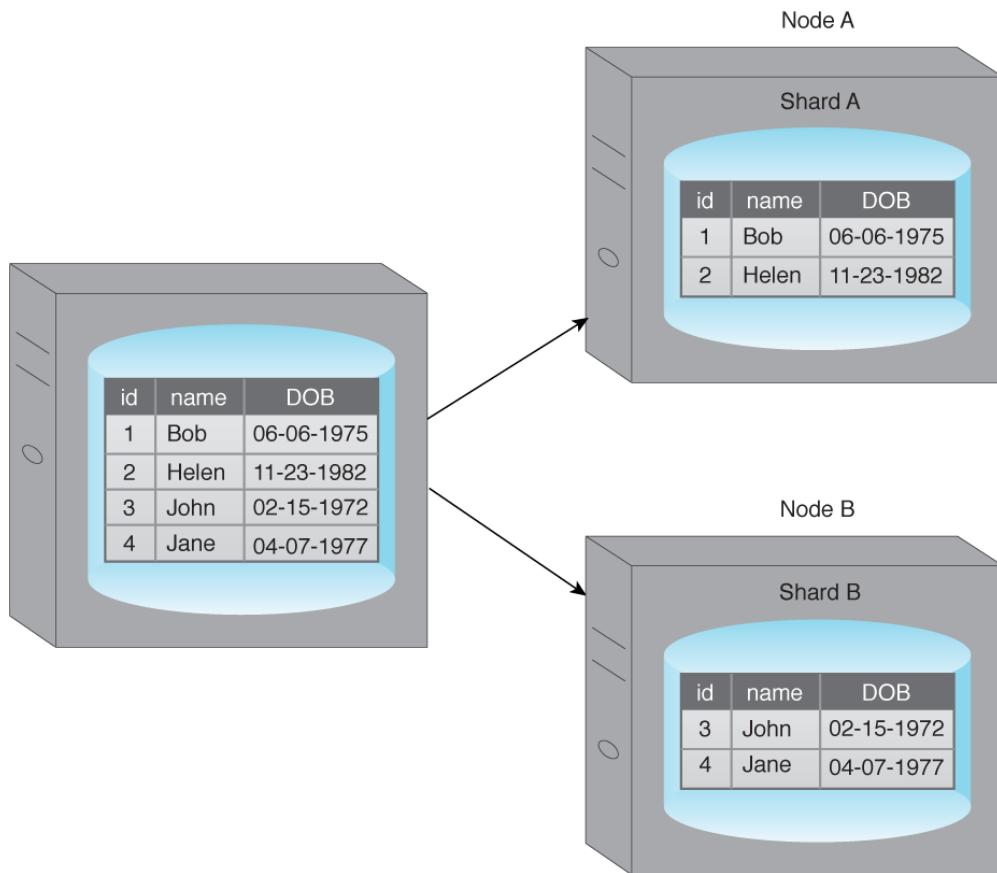


Figura 7.1 – Ejemplo de sharding, en donde un dataset es dividido entre el Nodo A y el Nodo B, lo que da como resultado los Shard A y Shard B, respectivamente.

El sharding puede o no ser transparente para el cliente. En el sharding, **se distribuye una carga de procesamiento entre distintos nodos para lograr la escalabilidad horizontal**, la cual es respaldada como un método para incrementar la capacidad al agregar recursos de capacidad mayor o similar junto a los recursos existentes.

La escalabilidad horizontal es un método para incrementar la capacidad agregando recursos de capacidad mayor o similar a los recursos existentes. Como cada nodo es responsable de una parte de todo el dataset, los tiempos de lectura y escritura mejoran en gran medida.

La Figura 7.2 ilustra los siguientes pasos:

1. a, b. De forma independiente, cada shard permite lecturas y escrituras en los subconjuntos específicos de datos por los que es responsable.
2. Dependiendo de la consulta, puede que los datos deban buscarse en los dos nodos.

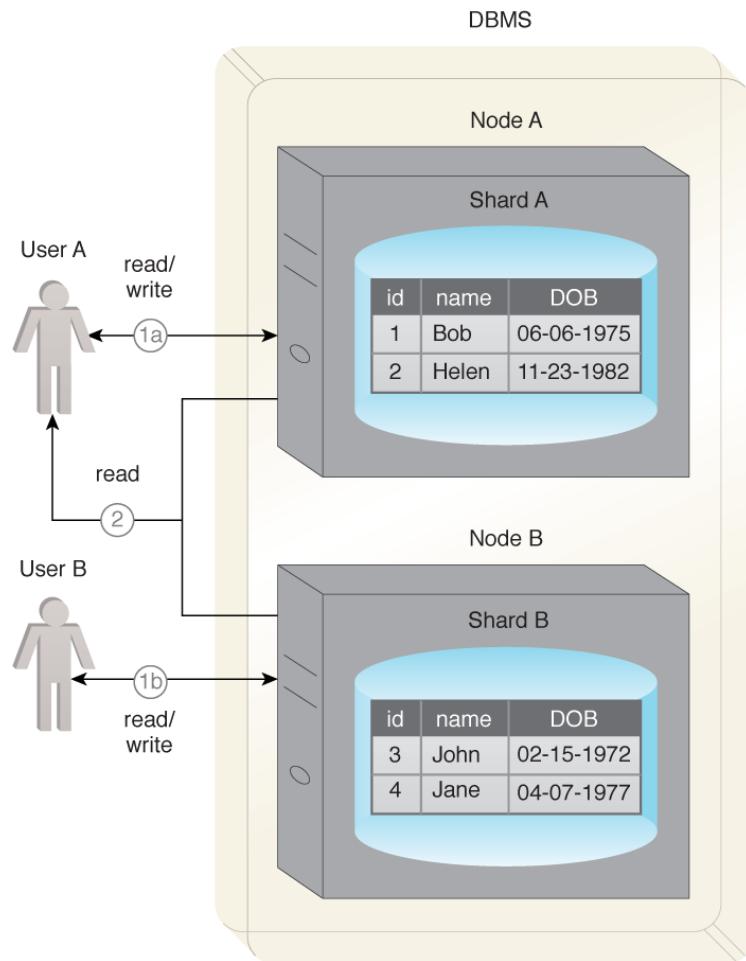


Figura 7.2 – Ejemplo de sharding, en donde los datos son buscados tanto en el Nodo A como en el Nodo B.

Un beneficio del sharding es que **brinda una tolerancia parcial a errores**. En caso de que un nodo falle, solamente se verán afectados los datos guardados en dicho nodo.

Para la partición de los datos, los patrones de la consulta deben ser tomados en cuenta, de tal forma que los shards no se conviertan en cuellos de botella en términos de rendimiento. Por ejemplo, las consultas que requieren datos de varios shards implicarán una disminución en el rendimiento. **La ubicación de los datos, o conservar los datos a los que frecuentemente se tiene acceso en un solo shard, ayuda a contrarrestar tales problemas de rendimiento.**

## Replicación

La replicación almacena múltiples copias de un dataset, conocidas como *réplicas*, en varios nodos (Figura 7.3). Esto proporciona escalabilidad, disponibilidad y tolerancia a los errores. Existen dos métodos diferentes para implementar la replicación:

- maestro-esclavo
- peer-to-peer

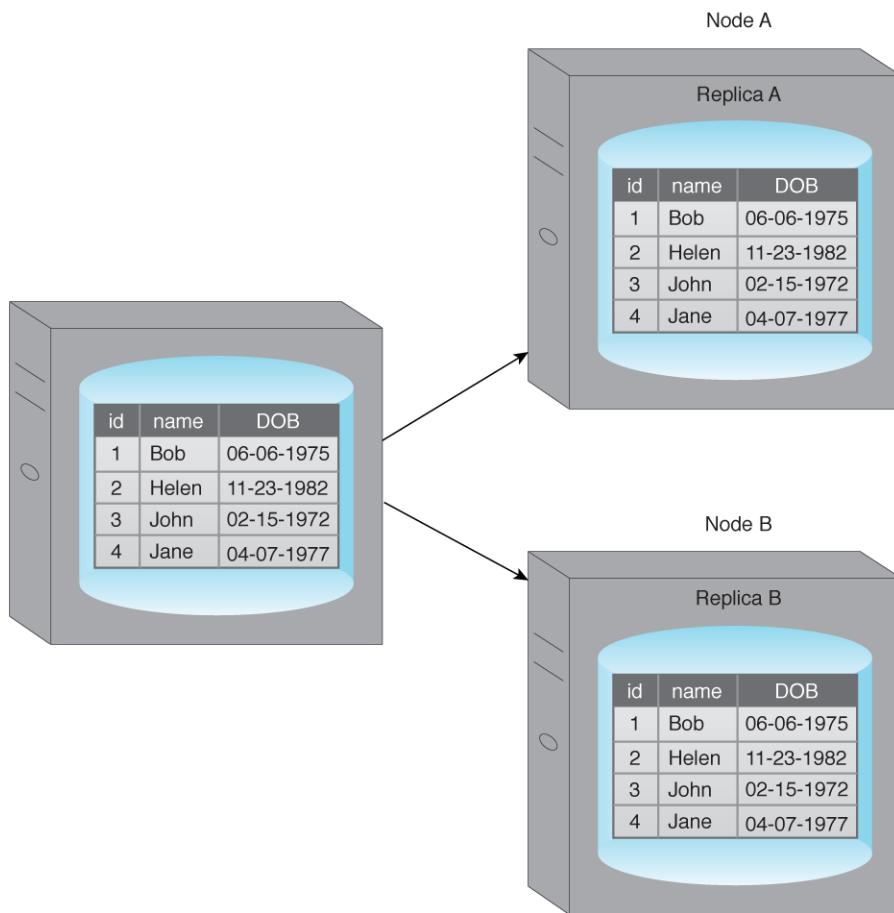


Figura 7.3 – Ejemplo de replicación, en donde un dataset es replicado en los Nodos A y B, lo que da como resultado las Réplicas A y B.

## Replicación: maestro-esclavo

Los nodos son organizados en una configuración maestro-esclavo en la que todos los datos son insertados y actualizados por medio de un nodo maestro. Una vez guardados, los datos son replicados en varios nodos esclavos. Todas las solicitudes externas de operaciones de escritura (insertar, actualizar, eliminar) ocurren en el nodo maestro, mientras que los datos pueden ser leídos desde cualquier nodo esclavo. En la Figura 7.4, el nodo maestro administra las operaciones de escritura. Los datos pueden ser leídos tanto en el nodo Esclavo A como en el nodo Esclavo B.

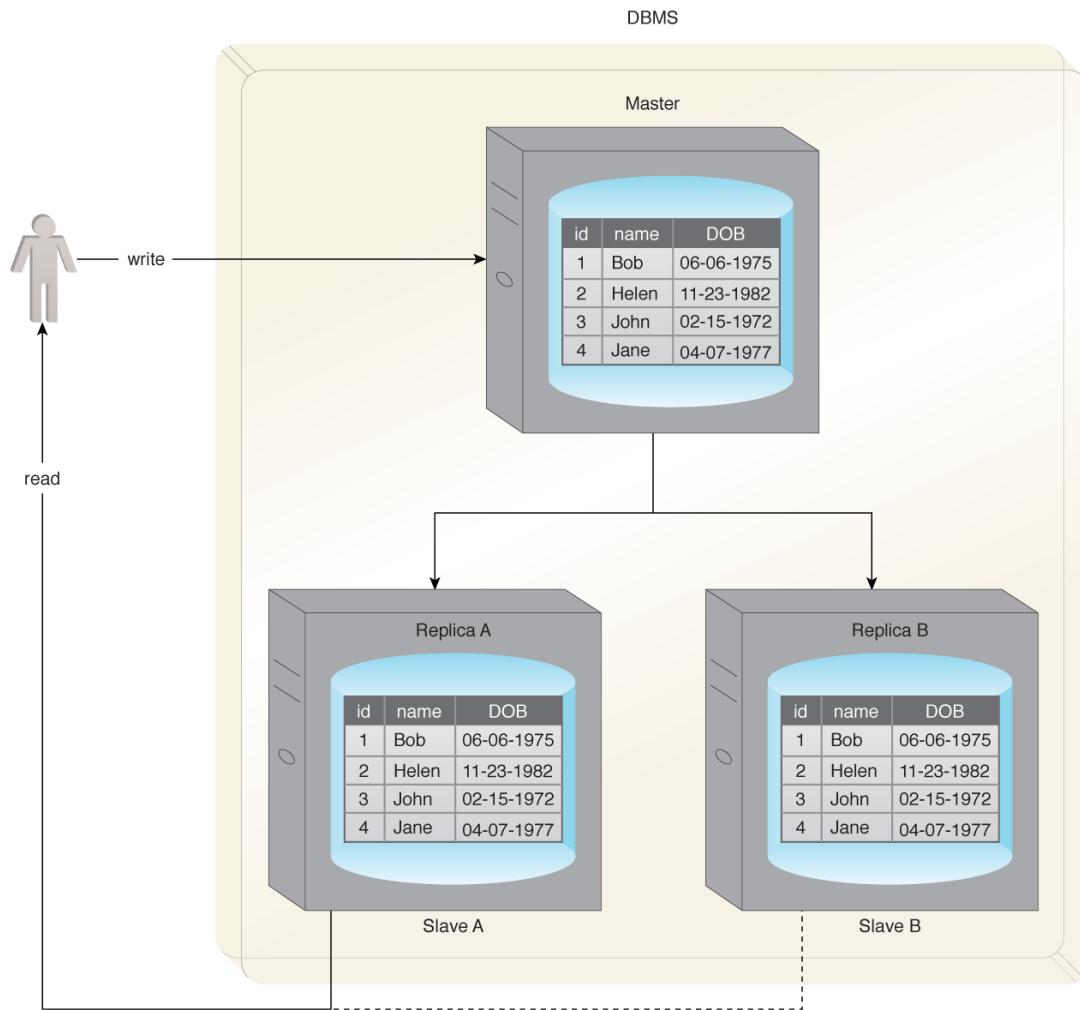


Figura 7.4 – Ejemplo de replicación maestro-esclavo, en donde el nodo Maestro A es el único punto de contacto para todas las operaciones de escritura, mientras que los datos pueden ser leídos desde los nodos Esclavos A y B.

La replicación maestro-esclavo es ideal para cargas con operaciones intensivas de lectura en vez de cargas de operaciones intensivas de escritura, ya que la escalabilidad horizontal puede satisfacer de forma sencilla las crecientes demandas de operaciones de lectura por medio de nodos esclavos adicionales. **Las operaciones de escritura son consistentes, ya que todas son coordinadas por el nodo maestro.** Esto significa que el desempeño de operaciones de escritura se ve afectado a medida que aumenta la cantidad de operaciones de escritura. En caso de que falle el nodo maestro, siguen siendo posibles las operaciones de lectura por medio de cualquiera de los nodos esclavos. Un nodo esclavo puede ser configurado como un nodo de respaldo del nodo maestro.

En caso de una falla del nodo maestro, las operaciones de escritura no serán soportadas hasta que el nodo maestro se restablezca. Este se puede restablecer desde el respaldo del nodo maestro o se puede elegir un nuevo nodo maestro entre los nodos esclavos. La inconsistencia de operaciones de lectura puede ser un problema si se lee un nodo esclavo antes de que se copie la actualización en el mismo desde el nodo maestro.

Para garantizar la consistencia de lectura, se puede implementar un sistema de votación en donde se declare consistente una operación de lectura si la mayoría de los nodos esclavos contienen la misma versión del registro. La implementación de tal sistema de votación requiere un mecanismo de comunicación rápido y confiable entre los nodos esclavos.

En la Figura 7.5:

1. El Usuario A actualiza los datos.
2. El nodo maestro copia los datos en el Nodo esclavo A.
3. Antes de que los datos sean copiados en el Nodo esclavo B, el Usuario B trata de leer los datos actualizados en el Nodo esclavo B, lo que resulta en una inconsistencia de lectura.
4. Finalmente, gracias a actualización hecha por el nodo maestro, los datos se volverán consistentes en el Nodo esclavo B.

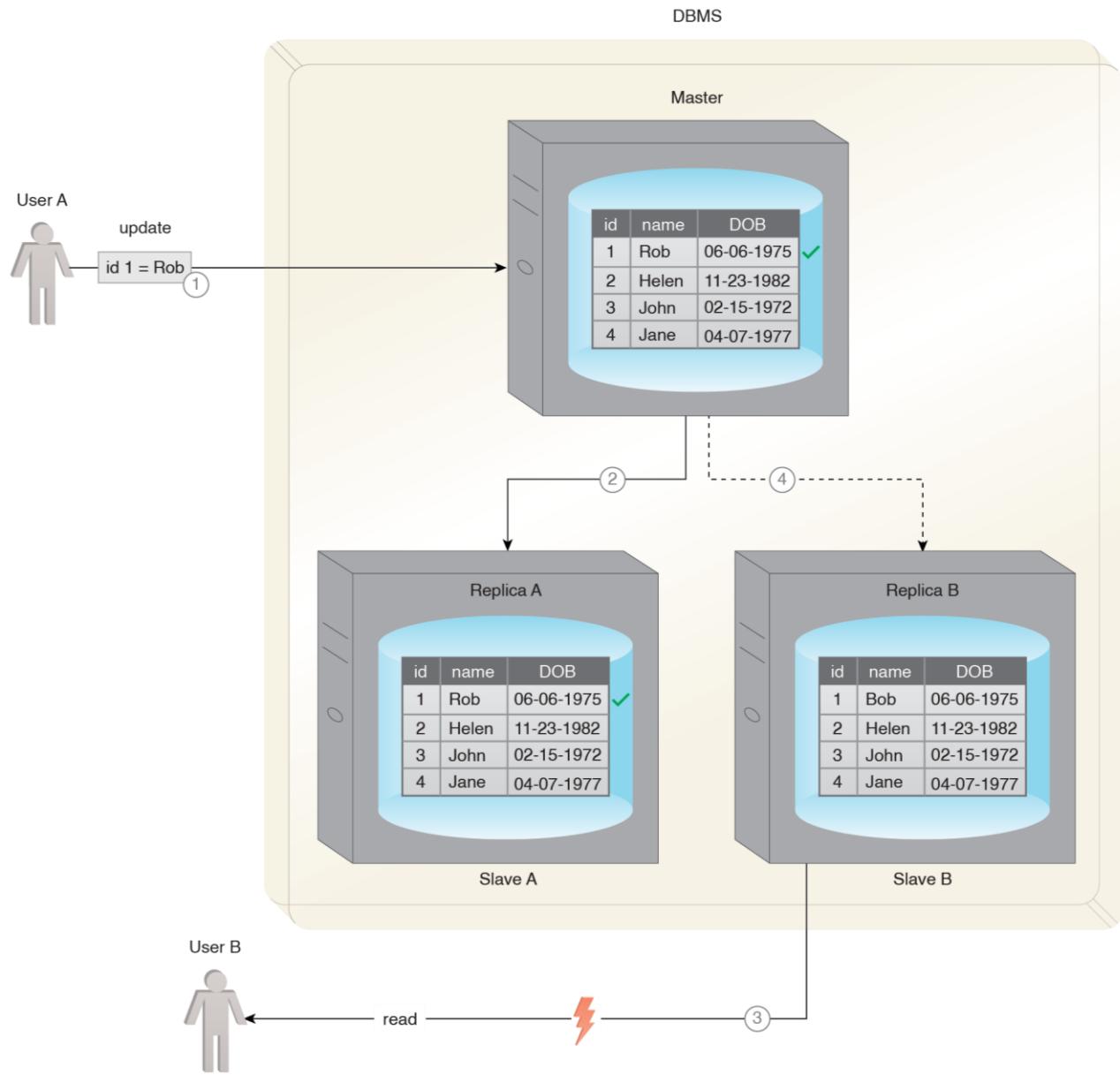


Figura 7.5 – Ejemplo de la replicación maestro-esclavo, en donde ocurre la inconsistencia de lectura.

### Replicación: peer-to-peer

En la replicación peer-to-peer, todos los nodos operan en el mismo nivel. En otras palabras, no existe una configuración maestro-esclavo. Cada nodo, conocido como peer, es capaz de manejar tanto operaciones de lectura como operaciones de escritura. Cada operación de escritura es copiada a todos los peers, como se muestra en la Figura 7.6.

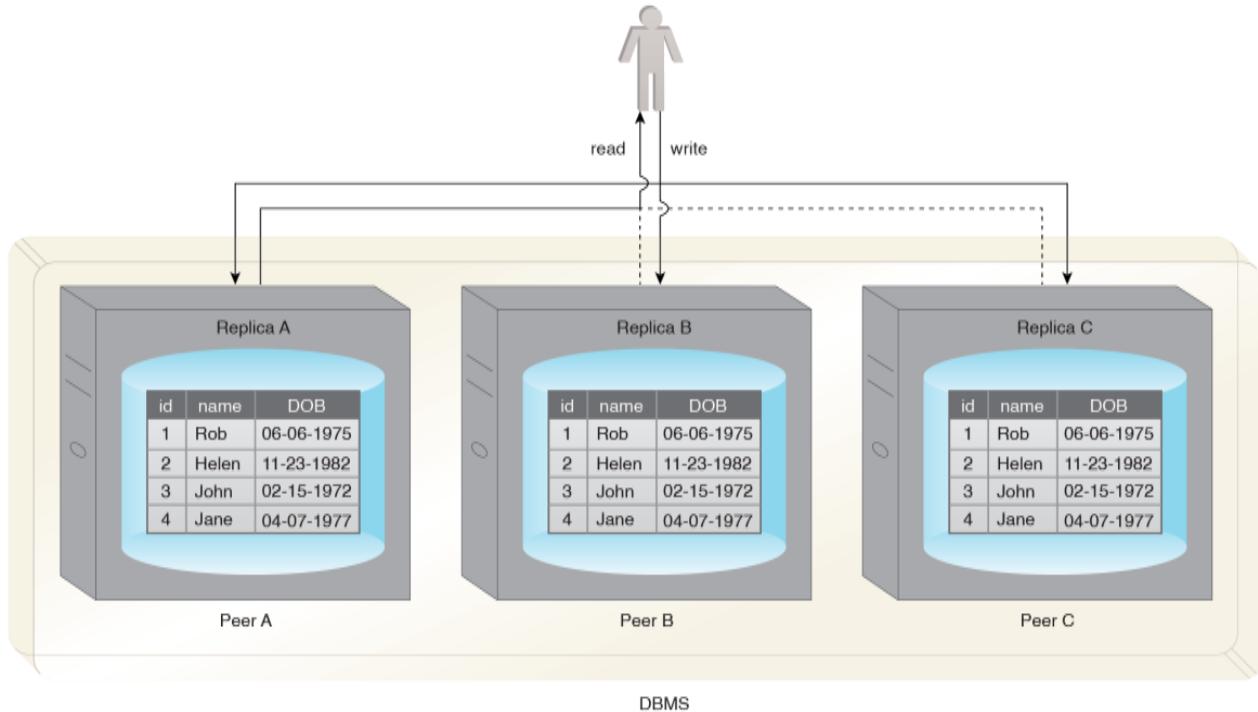


Figura 7.6 – Las operaciones de escritura son copiadas a los Peers A, B y C simultáneamente. Se leen los datos en el Peer A, pero también se pueden leer en los Peers B o C.

La replicación peer-to-peer tiende a escribir inconsistencias que ocurren como resultado de una actualización simultánea de los mismos datos en diferentes peers. Esto se puede manejar implementando la concurrencia pesimista o la concurrencia optimista.

- **La concurrencia pesimista** es un enfoque proactivo que hace uso del bloqueo para asegurar que solo una actualización sea exitosa a la vez. Sin embargo, esto afecta la disponibilidad, ya que la base de datos permanece como no disponible hasta que todo bloqueo sea liberado.
- **La concurrencia optimista** es un enfoque reactivo que no hace uso del bloqueo. En vez de esto, permite que la inconsistencia ocurra primero y restaura la consistencia después del hecho.

Debido a esto, los peers pueden permanecer inconsistentes por un periodo de tiempo antes de alcanzar la consistencia. Sin embargo, la base de datos permanece disponible ya que no hay bloqueos. **Al igual que la configuración maestro-esclavo, las operaciones de lectura pueden ser inconsistentes en el periodo de tiempo transcurrido mientras alguno de los peers está siendo actualizado.** Sin embargo, las operaciones de lectura se vuelven consistentes a largo plazo cuando las actualizaciones han sido copiadas a todos los peers.

Para asegurar la consistencia en la lectura, se puede implementar un sistema de votación en el cual la lectura sea considerada consistente si la mayoría de los peers contienen la misma versión del registro. La implementación de tal sistema de votación requiere un mecanismo de comunicación rápido y confiable entre los peers.

En la Figura 7.7:

1. El Usuario A actualiza los datos.
2. a. Los datos son copiados al Peer A.  
b. Los datos son copiados al Peer B.
3. Antes de que los datos sean copiados al Peer C, el Usuario B trata de leer los datos en el Peer C, lo que resulta en una inconsistencia de lectura.
4. Los datos serán actualizados a largo plazo en el Peer C, y la base de datos será consistente de nuevo.

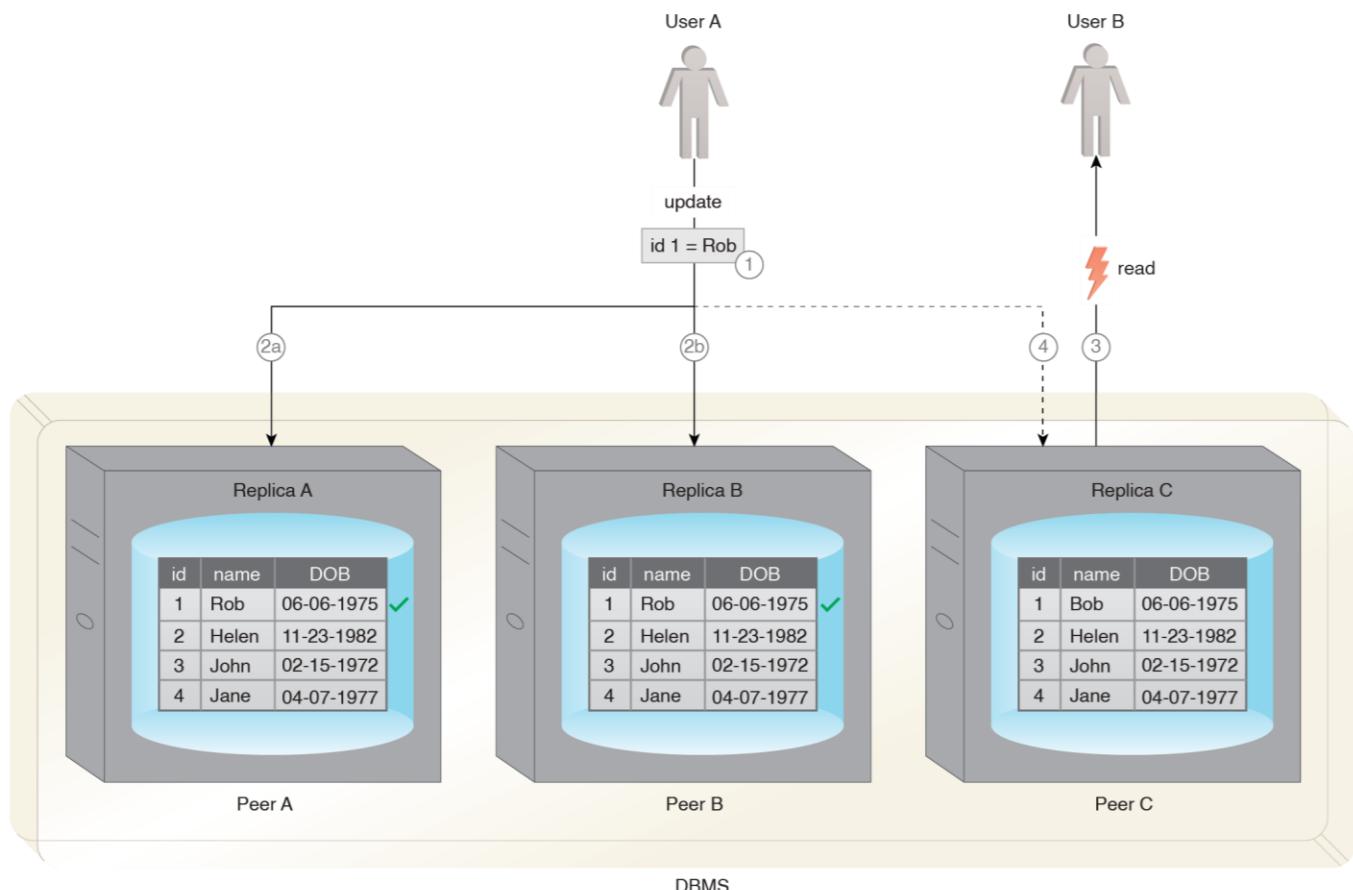


Figura 7.7 – Ejemplo de la replicación peer-to-peer, en la cual ocurre una inconsistencia de lectura.

## Sharding y replicación

El sharding y la replicación se pueden combinar para mejorar la limitada tolerancia a errores que ofrece el sharding, y al mismo tiempo, beneficiarse de la mayor disponibilidad y escalabilidad que ofrece la replicación (Figura 7.8).

Esta sección abarca las siguientes combinaciones:

- sharding y replicación maestro-esclavo
- sharding y replicación peer-to-peer

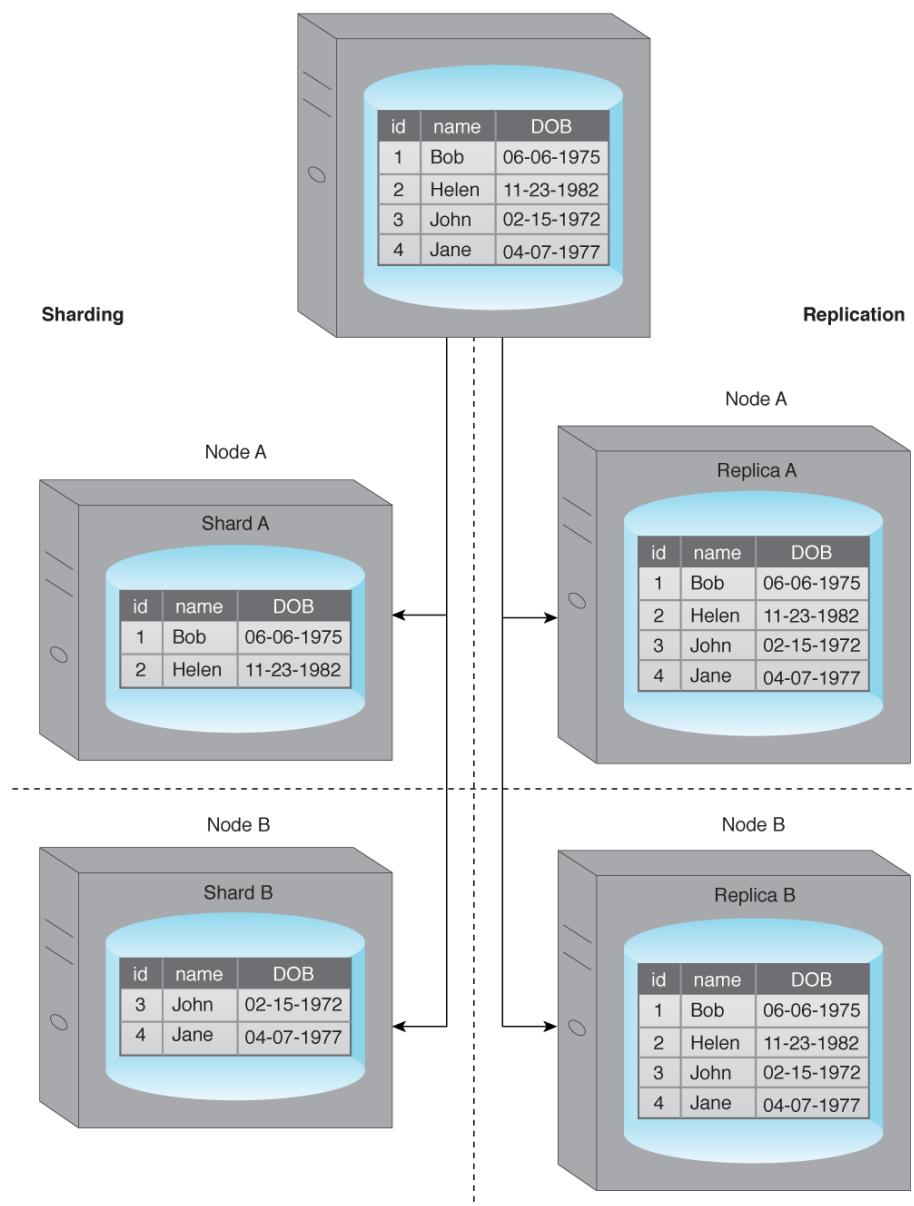


Figura 7.8 – Comparación entre el sharding y la replicación, que muestra cómo un dataset es distribuido entre dos nodos según cada enfoque.

## Combinar sharding y replicación maestro-esclavo

Varios shards se convierten en esclavos de un solo maestro, mientras que el maestro mismo es un shard. Aunque es posible tener más de un maestro, **un shard-esclavo únicamente puede ser manejado por un shard-maestro.**

El shard-maestro se encarga de mantener la consistencia en las operaciones de escritura. Sin embargo, esto significa que la tolerancia a errores (respecto a las operaciones de escritura) se ve afectada si el shard maestro deja de funcionar o si ocurre una suspensión de red. **Las réplicas de shards se conservan en múltiples nodos esclavos** con el fin de proporcionar escalabilidad y tolerancia a errores para las operaciones de lectura.

En la Figura 7.9:

- Cada nodo actúa como maestro y como esclavo para diferentes shards.
- El Nodo A regula las operaciones de escritura (id = 2) en el Shard A, ya que es el maestro para el Shard A.
- El Nodo A replica los datos (id = 2) al Nodo B, el cual es esclavo para el Shard A.
- Tanto el Nodo B como el Nodo C pueden poner a disposición información de forma directa para realizar operaciones de lectura (id = 4), ya que los dos contienen el Shard B.

## Combinar sharding y replicación peer-to-peer

Cada shard es replicado en múltiples peers y cada peer es responsable únicamente de un subconjunto de datos, no del dataset completo. Colectivamente, esto ayuda a lograr mayor escalabilidad y tolerancia a errores. Ya que no hay un maestro involucrado, **no hay un solo punto de falla** respecto a las operaciones de lectura y escritura.

En la Figura 7.10:

- Cada nodo contiene réplicas de dos shards diferentes.
- Las operaciones de escritura (id = 3) se replican tanto en el Nodo A como en el Nodo C (Peers), ya que estos son responsables del Shard C.
- Tanto el Nodo B como el Nodo C pueden poner a disposición información para realizar operaciones de lectura (id = 6), ya que los dos contienen el Shard B.

## Lectura

En la sección *Sharding y Replicación*, en las páginas 38 a 44 del libro *Resumen de NoSQL* incluido en este módulo, encontrará un análisis más detallado sobre este tema.

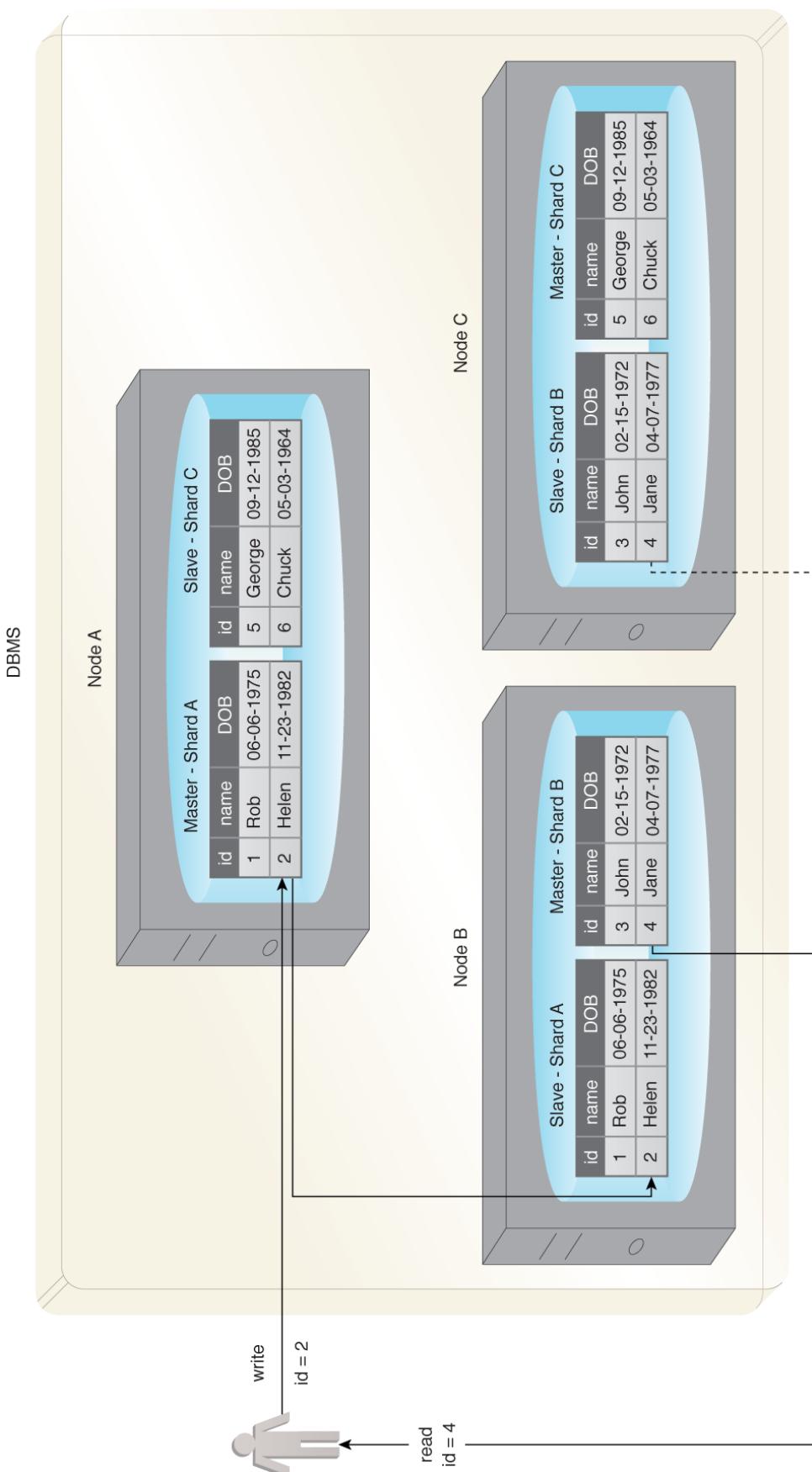


Figura 7.9 – Ejemplo de combinación de sharding y replicación maestro-esclavo.

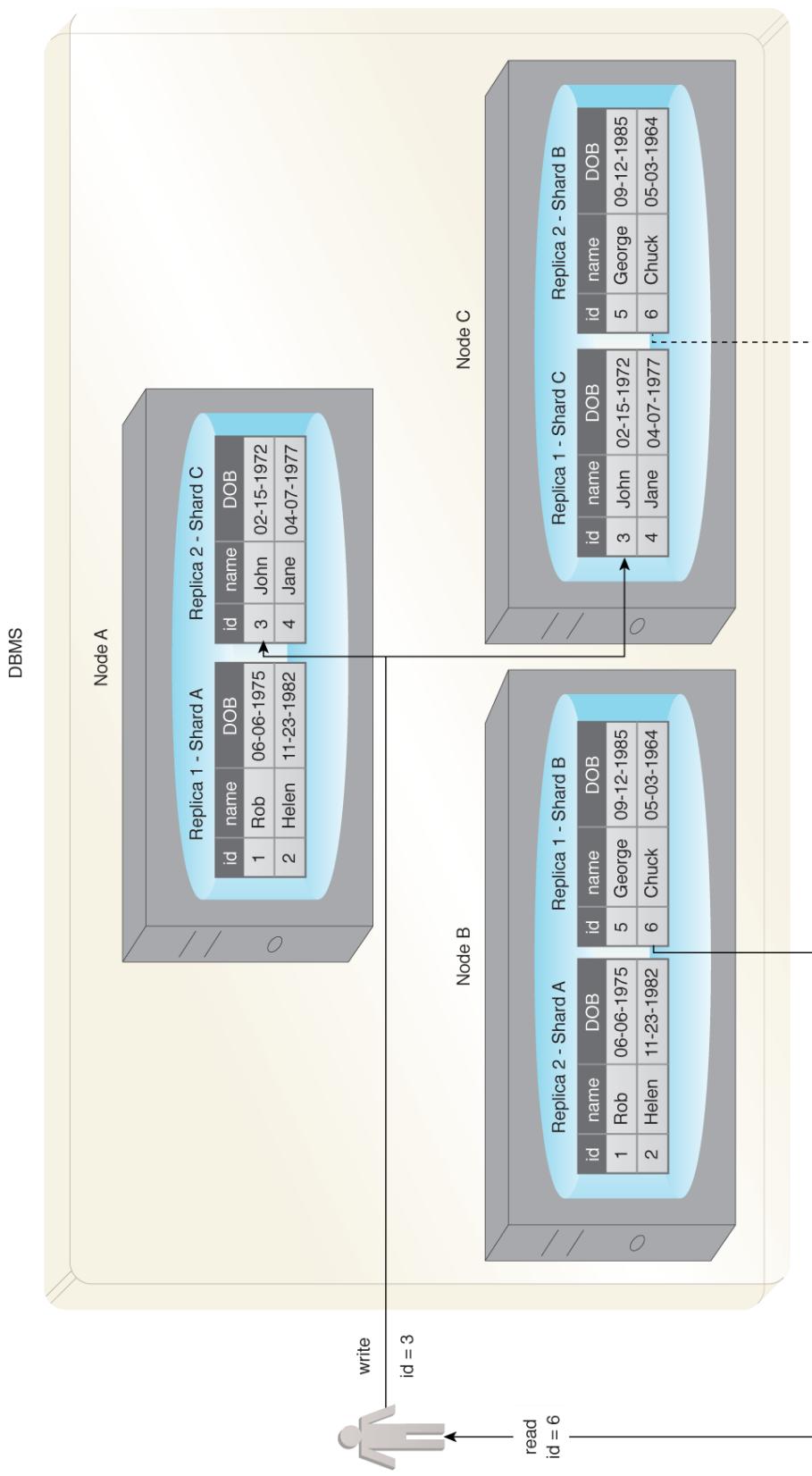


Figura 7.10 – Ejemplo de combinación de sharding y replicación peer-to-peer.

## Teorema CAP

El teorema de consistencia, disponibilidad y tolerancia al particionado (CAP, por sus siglas en inglés) establece que un sistema de archivos distribuido, particularmente una base de datos que funciona en cluster, solo puede proporcionar dos de las tres propiedades a continuación:

- **Consistencia:** una operación de lectura desde cualquier nodo muestra los mismos datos en varios nodos (Figura 7.11).
- **Disponibilidad:** una solicitud de lectura o escritura siempre será reconocida como un éxito o un fracaso (Figura 7.12).
- **Tolerancia a la partición:** el sistema de base de datos puede tolerar la suspensión de la comunicación, la cual separa el cluster en varios silos, y puede seguir atendiendo las solicitudes de lectura y escritura (Figura 7.12).

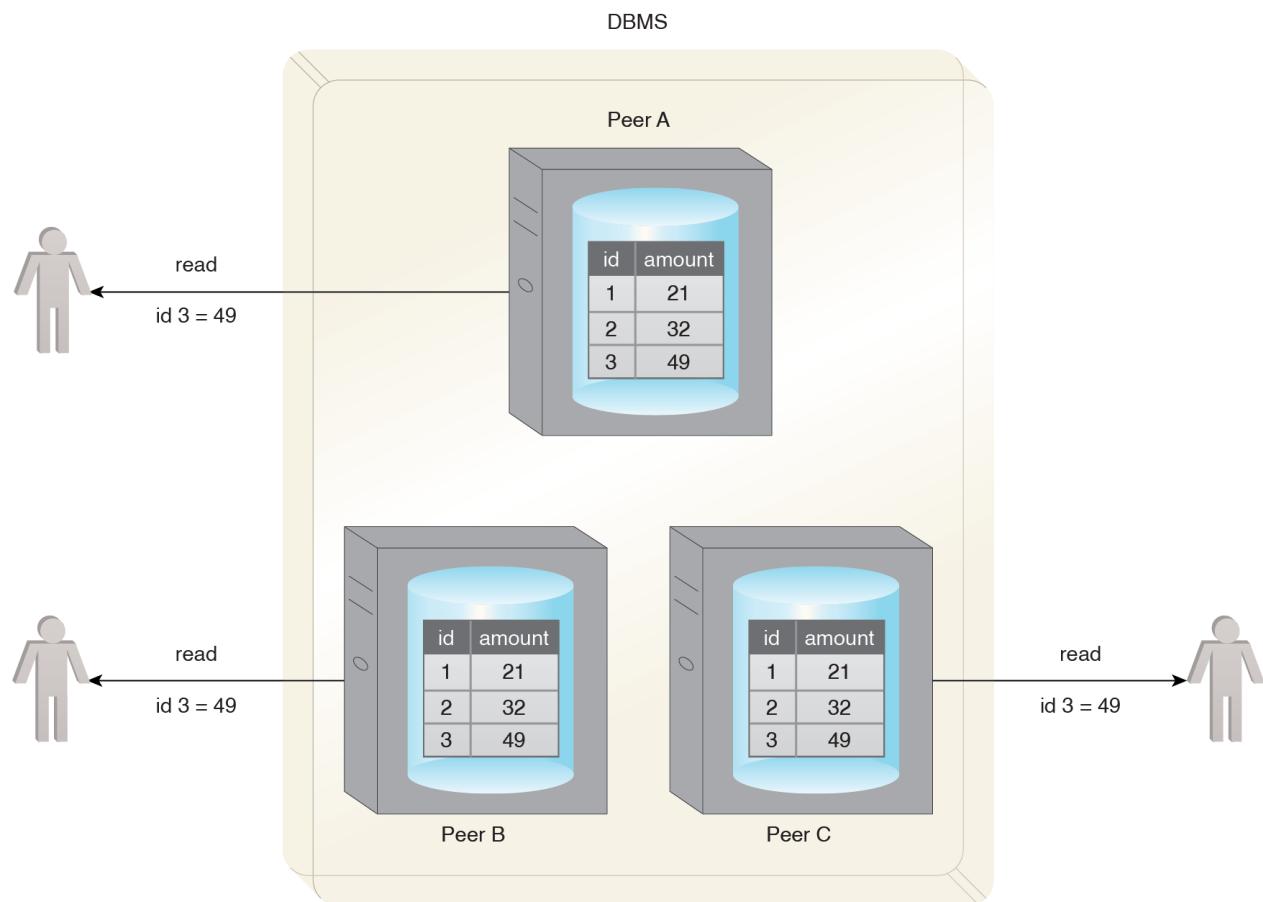


Figura 7.11 – Consistencia: los tres usuarios ven el mismo valor en la columna cantidad, aunque sean tres nodos diferentes los que proporcionan el registro.

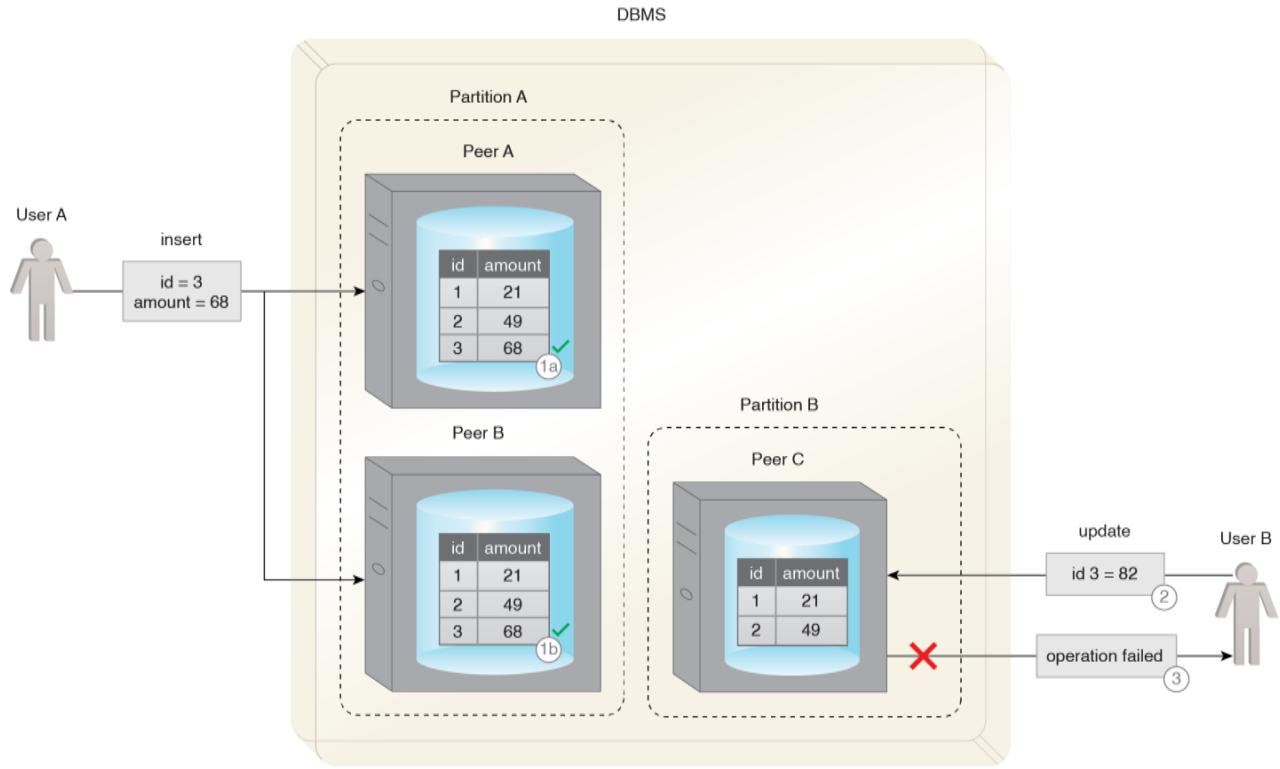


Figura 7.12 – Disponibilidad y tolerancia a la partición: en caso de una falla de comunicación, las solicitudes de ambos usuarios (1, 2) siguen siendo atendidas. Sin embargo, en el caso del Usuario B, la actualización falla, ya que el registro con id = 3 no se ha copiado en el Peer C. Se le notifica debidamente al usuario (3) que falló la actualización.

Si se requiere consistencia (**C**) y disponibilidad (**A**), los nodos disponibles deben comunicarse para asegurar la consistencia (**C**). Por lo tanto, la tolerancia a la partición (**P**) no es posible.

Si se requiere consistencia (**C**) y tolerancia a la partición (**P**), los nodos no pueden permanecer disponibles (**A**), ya que los nodos no estarán disponibles mientras que alcanzan la consistencia (**C**), lo cual no puede pasar mientras se cuente con tolerancia a la partición (**P**).

Si se requiere disponibilidad (**A**) y tolerancia a la partición (**P**), entonces la consistencia (**C**) no es posible, debido al requerimiento de comunicación de los datos entre los nodos. Así, la base de datos puede permanecer disponible (**A**) pero con resultados inconsistentes.

En una base de datos distribuida, la escalabilidad y tolerancia a errores pueden mejorarse por medio de nodos adicionales, aunque esto repercuta en la consistencia (**C**), la cual también puede causar que la disponibilidad (**A**) se vea afectada debido a la latencia causada por una mayor comunicación entre los nodos.

Los sistemas de base de datos distribuidos no pueden ser 100% tolerantes a la partición (**P**). Aunque la suspensión de comunicación sea algo inusual y temporal, siempre se debe contar con tolerancia a la partición (**P**). De este modo, CAP se trata más acerca de ser **CP** o ser **AP**.

Por otro lado, los RDBMS son **CA**, ya que por lo general están disponibles (**A**) y son consistentes (**C**) a la vez. Por lo general, los RDBMS se ejecutan en un solo nodo; así, la

tolerancia a la partición (**P**) no representa una gran preocupación. Cabe señalar que la elección entre **CP** y **AP** depende completamente de los requerimientos del sistema. Este teorema también es conocido como el teorema de Brewer, por el nombre de su proponente.

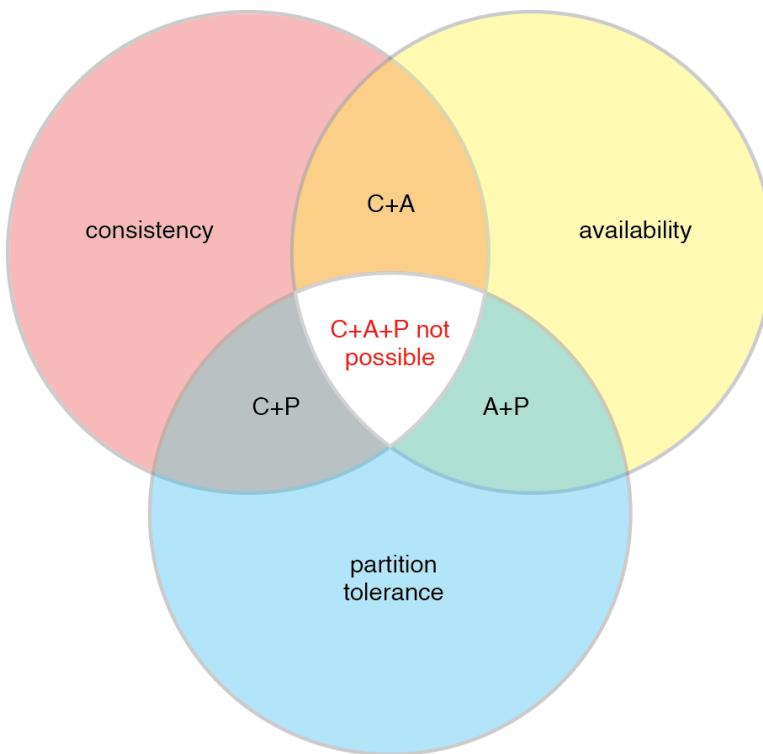


Figura 7.13 – Diagrama de Venn que resume el teorema CAP.

## Lectura

En la sección *Antecedentes adicionales del Teorema CAP*, en las páginas 53 a 56 del libro *Resumen de NoSQL* incluido en este módulo, encontrará un análisis más detallado sobre el teorema CAP.

## ACID

ACID (por su acrónimo en inglés) es un principio de diseño de bases de datos que comprende:

- atomicidad (*Atomicity*)
- consistencia (*Consistency*)
- aislamiento (*Isolation*)
- durabilidad (*Durability*)

La **atomicidad** asegura que todas las operaciones serán siempre exitosas o fallidas por completo. En otras palabras, no hay operaciones parciales.

En la Figura 7.14:

1. Un usuario trata de actualizar tres registros como parte de una operación.
2. Dos registros son actualizados exitosamente antes de la aparición de un error.
3. Como resultado, la base de datos reduce cualquier efecto parcial de la operación y devuelve el sistema a su estado anterior.

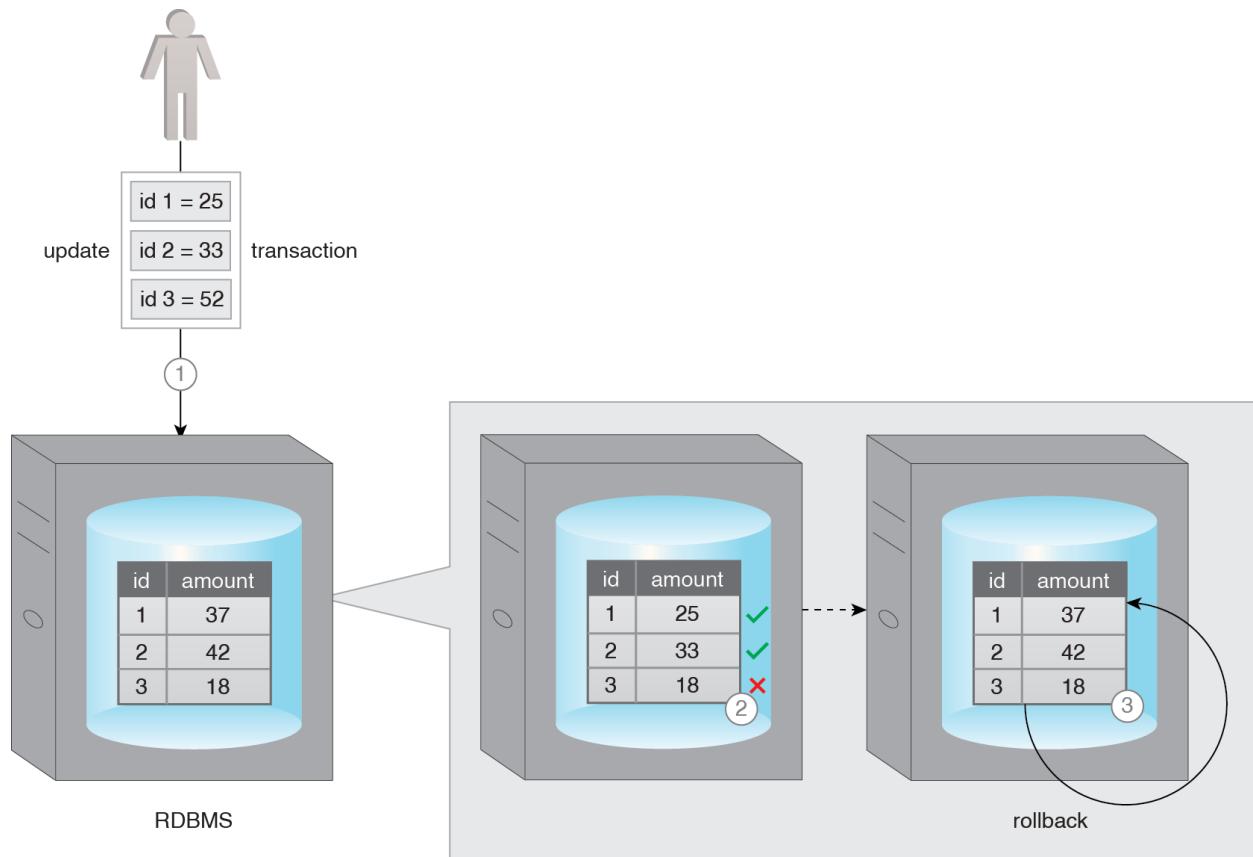


Figura 7.14 – Ejemplo de la propiedad de atomicidad de ACID.

La **consistencia** garantiza que una base de datos solo permitirá datos válidos y que siempre será consistente después de una operación. Se garantiza que cualquier operación de escritura seguida de una lectura inmediata es consistente para múltiples clientes.

En la Figura 7.15:

1. Un usuario intenta actualizar la columna cantidad en la tabla de tipo flotante con un valor varchar.
2. La base de datos aplica su verificación de validación y rechaza esta actualización debido a un valor nulo en la columna cantidad.

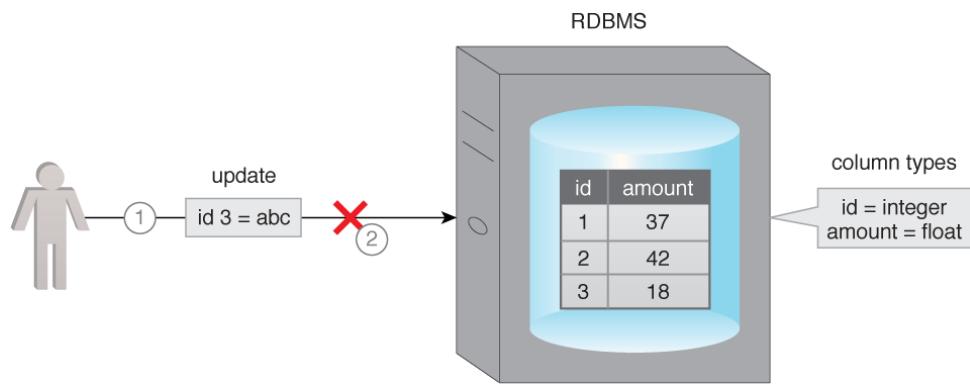


Figura 7.15 – Ejemplo de la propiedad de consistencia de ACID.

El **aislamiento** asegura que los resultados de una operación no son visibles a otras operaciones hasta que finalice dicha operación.

En la Figura 7.16:

1. El Usuario A intenta actualizar dos registros como parte de una operación.
2. La base de datos actualiza exitosamente el primer registro.
3. Sin embargo, antes de que pueda actualizar el segundo, el Usuario B intenta actualizar el mismo registro. La base de datos no permite la actualización del Usuario B hasta que la actualización del Usuario A sea completamente exitosa o fallida.

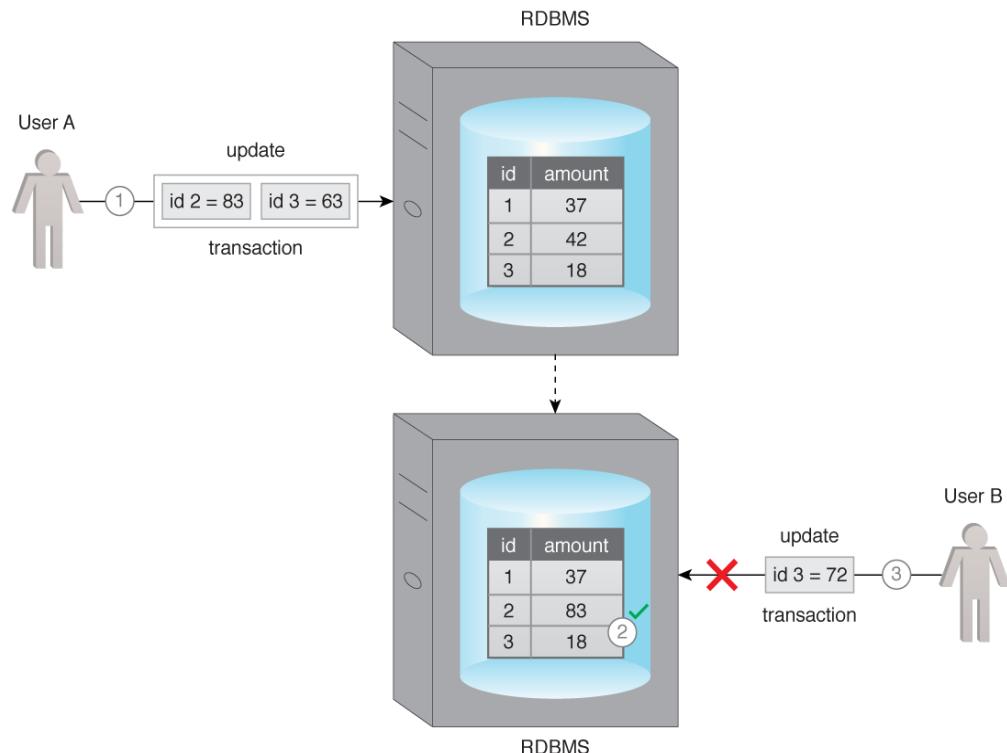


Figura 7.16 – Ejemplo de la propiedad de aislamiento de ACID.

La **durabilidad** garantiza que los resultados de una operación sean permanentes. En otras palabras, una vez se haya realizado la operación, no se puede revertir. Esto es independiente de cualquier fallo del sistema.

En la Figura 7.17:

1. Un usuario actualiza un registro como parte de una operación.
2. La base de datos actualiza el registro exitosamente.
3. Justo después de esta actualización, ocurre un fallo de electricidad. La base de datos mantiene su estado mientras no hay electricidad.
4. La electricidad regresa.
5. El registro se muestra de conformidad con la última actualización existente cuando el usuario lo solicitó.

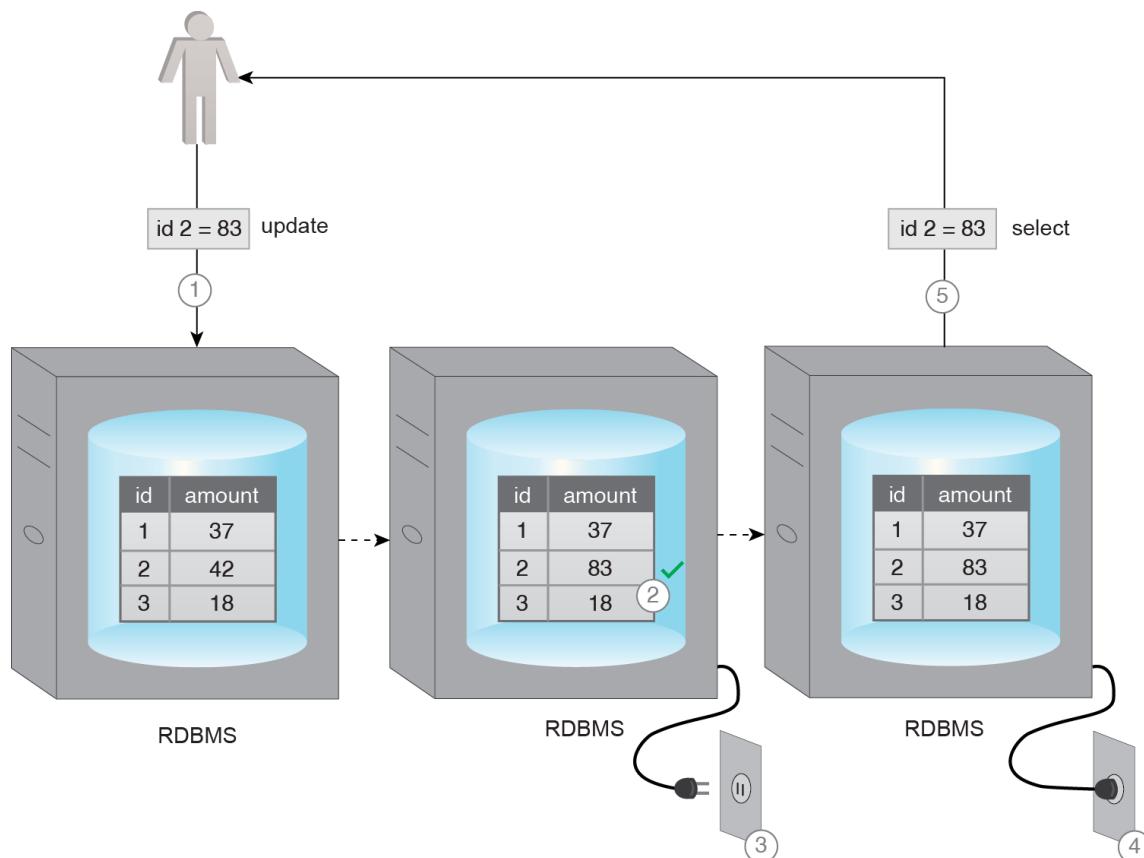


Figura 7.17 – Ejemplo de la propiedad de durabilidad de ACID.

La Figura 7.18 muestra los resultados de la aplicación del principio ACID:

1. El Usuario A intenta actualizar un registro como parte de una operación.
2. La base de datos valida el valor y la actualización se aplica exitosamente.
3. Después de completar con éxito la operación, cuando el Usuario B y el Usuario C soliciten el mismo registro, la base de datos les proporciona a ambos usuarios el valor actualizado.

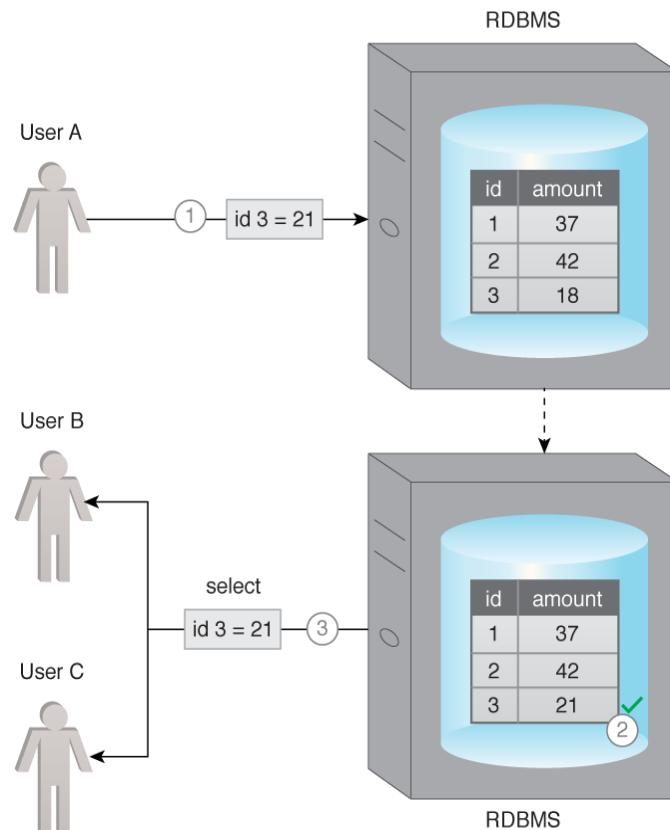


Figure 7.18 – Ejemplo que ilustra los resultados de la aplicación del principio ACID.

## Lectura

En la sección *ACID y Bases de datos NoSQL*, en las páginas 19 y 20 del libro *Resumen de NoSQL* incluido en este módulo, encontrará un análisis más detallado sobre ACID y los dispositivos de almacenamiento NoSQL.

## BASE

BASE (por su acrónimo en inglés) es un principio de diseño de bases de datos basado en el teorema CAP y seguido por sistemas de bases de datos que hacen uso de la tecnología distribuida. El acrónimo BASE representa:

- disponibilidad todo el tiempo (*basically available*)

- **estado flexible (soft state)**
- **consistencia a largo plazo (eventual consistency)**

“Disponibilidad todo el tiempo” se refiere a que la base de datos siempre reconocerá la solicitud del cliente, ya sea como datos solicitados o una notificación de éxito o error.

En la Figura 7.19, ambos usuarios tienen acceso a la información, aunque el sistema de base de datos haya sido particionado como resultado de una falla de red.

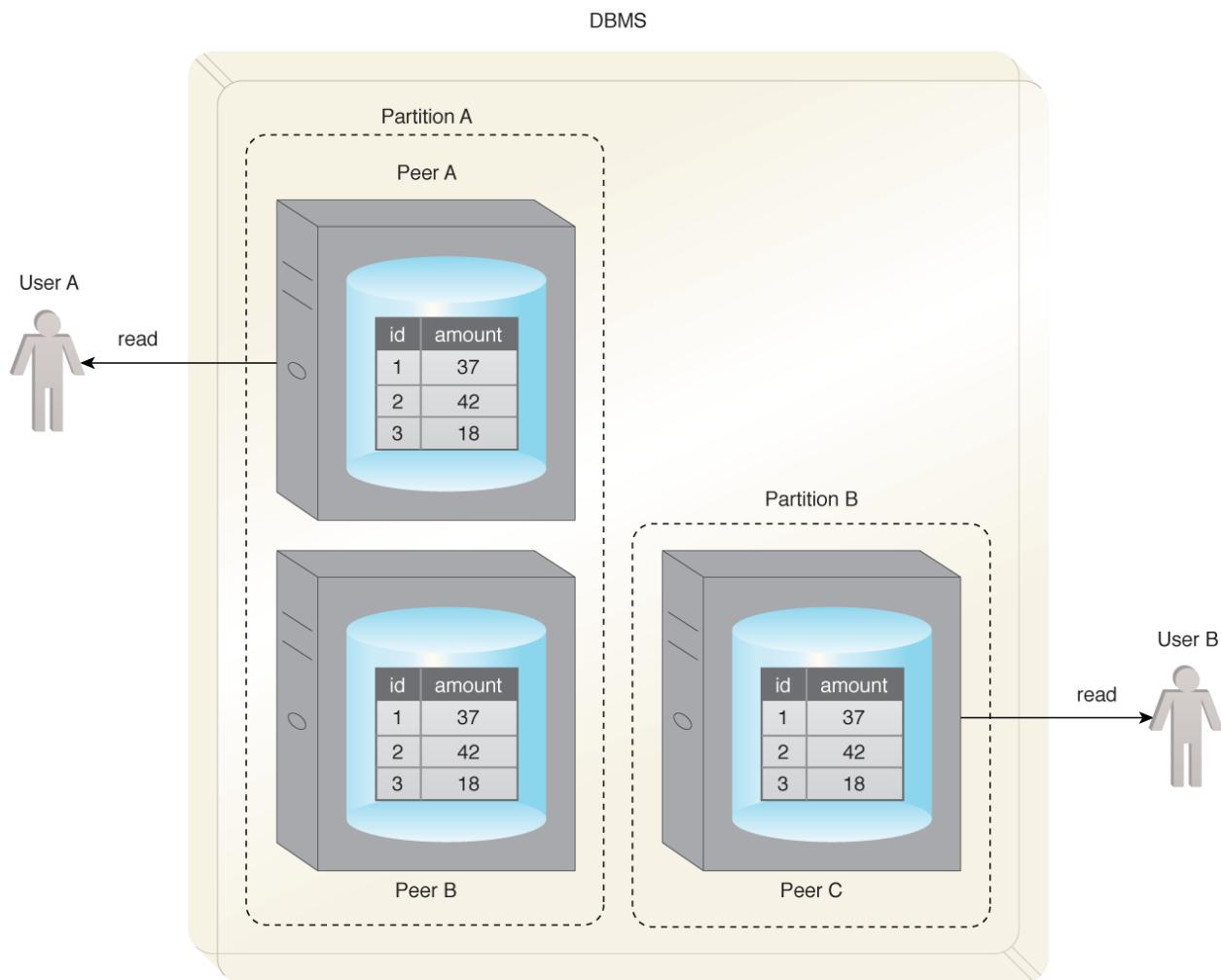


Figura 7.19 – La información está disponible para el Usuario A y el Usuario B, a pesar de la partición del sistema de base de datos causada por una falla de red.

El “**estado flexible**” implica que la base de datos puede no tener un estado consistente cuando se lean los datos, y por lo tanto, los resultados pueden cambiar si se solicitan estos mismos datos de nuevo. Esto se debe a que los datos podrían actualizarse para ser consistentes, aunque ningún usuario haya escrito los datos entre las dos operaciones de lectura. Esta propiedad está estrechamente relacionada con la consistencia a largo plazo.

En la Figura 7.20:

1. El Usuario A actualiza un registro en el Peer A.
2. Antes de que otros peers puedan actualizarse, el Usuario B solicita el mismo registro en el Peer C.
3. Ahora la base de datos tiene un estado flexible, y el Usuario B ve los datos anteriores.

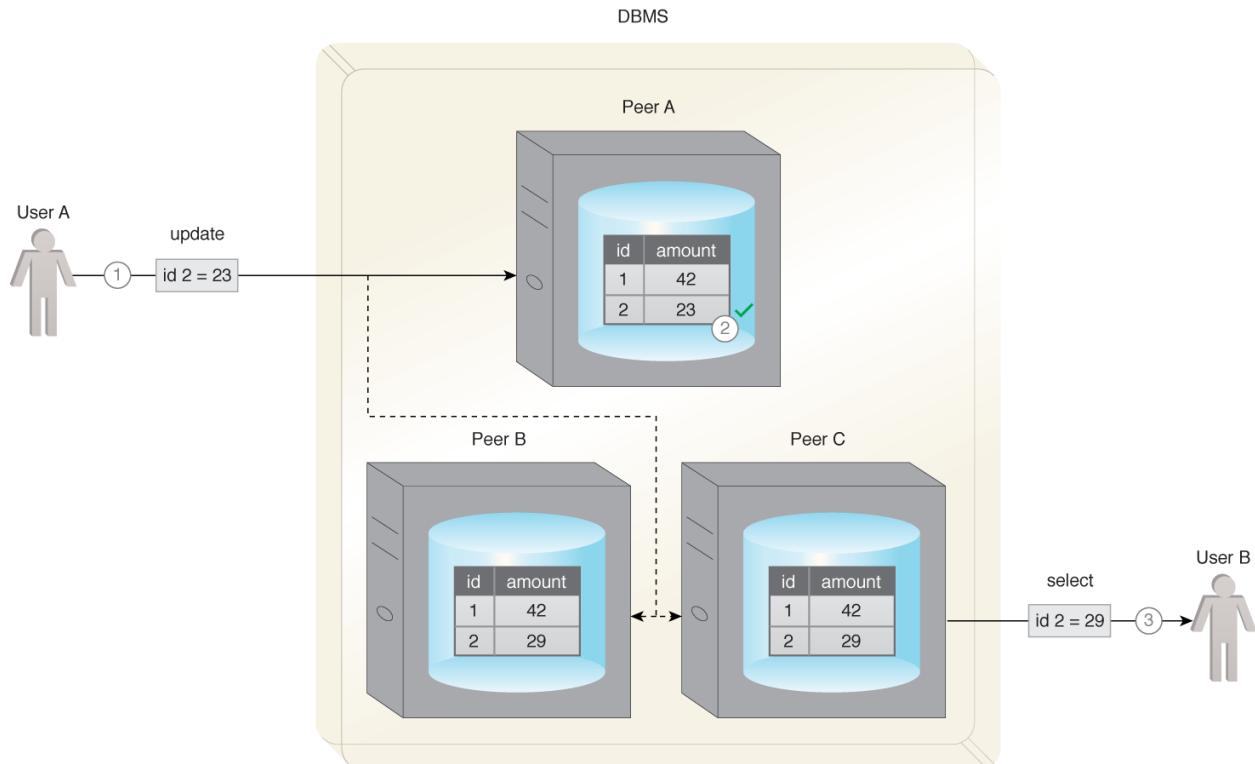


Figura 7.20 – Ejemplo de la propiedad de estado flexible de BASE.

La “**consistencia a largo plazo**” es el estado en el cual las operaciones de lectura llevadas a cabo por diferentes clientes justo después de una operación de escritura podrían no tener resultados consistentes. La base de datos solo alcanza la consistencia una vez los cambios se hayan propagado por todos los nodos. Por lo tanto, mientras la base de datos esté en proceso de alcanzar el estado de consistencia a largo plazo, se encontrará en un estado flexible.

En la Figura 7.21:

1. El Usuario A actualiza un registro.
2. El registro solo se actualiza en Peer A. Antes de que se actualicen otros Peers, el Usuario B solicita el mismo registro.
3. La base de datos ahora se encuentra en un estado flexible. El Peer C devuelve los datos obsoletos al Usuario B.
4. Sin embargo, la consistencia se alcanza a largo plazo y el Usuario C recibe el valor correcto.

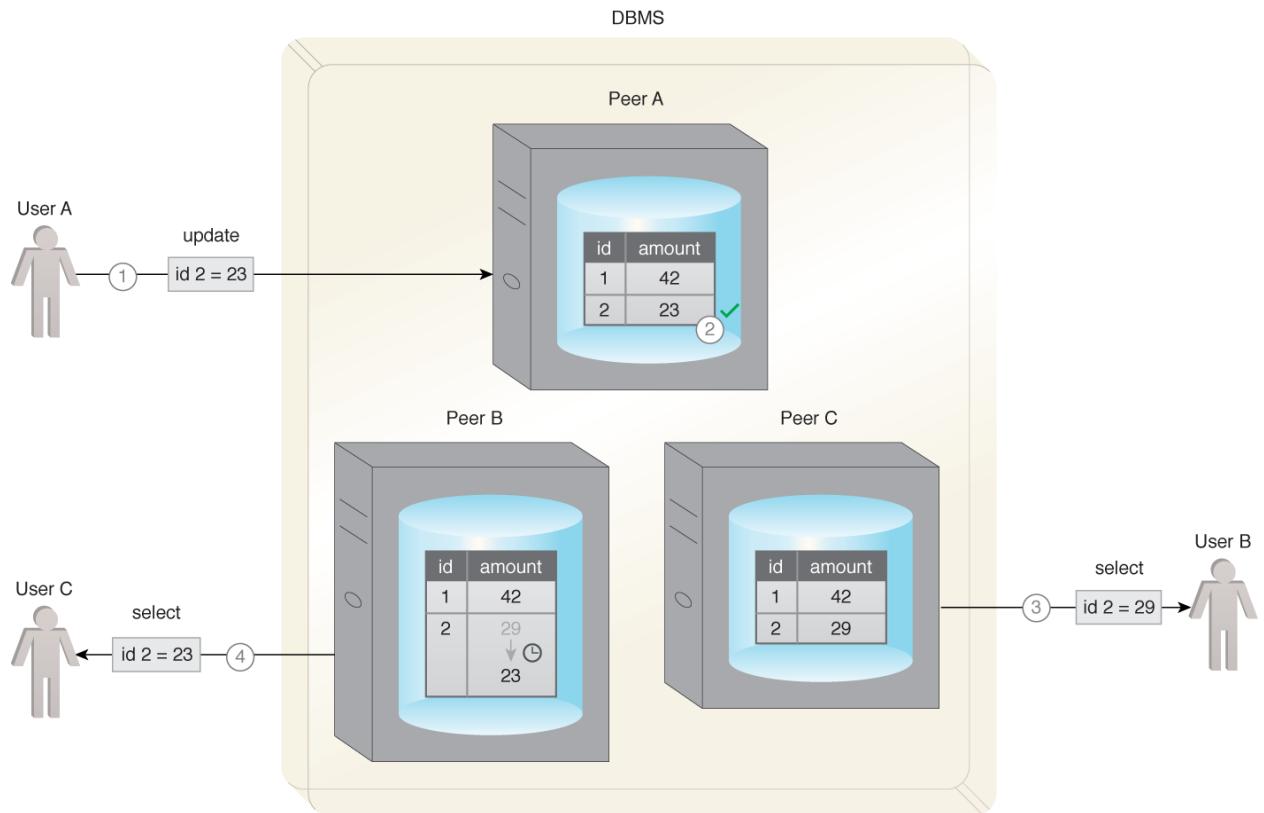


Figura 7.21 – Ejemplo de la propiedad de consistencia a largo plazo de BASE.

BASE enfatiza la **disponibilidad** sobre la consistencia inmediata, a diferencia de ACID, el cual asegura la consistencia inmediata a expensas de la disponibilidad debido al bloqueo de registros. Este enfoque flexible hacia la consistencia permite que las bases de datos compatibles con BASE atiendan las solicitudes de múltiples clientes sin latencia alguna, aunque suministren resultados inconsistentes. Sin embargo, por lo general, las bases de datos compatibles con BASE no son usadas para sistemas transaccionales, en los que la falta de consistencia puede convertirse en un problema.

### Ejercicio 7.1: complete los espacios en blanco

1. \_\_\_\_\_ es el proceso de particionar horizontalmente un gran dataset en un grupo de datasets más pequeños y manejables llamados \_\_\_\_\_, distribuidos entre múltiples nodos.
  
2. En el sharding, los \_\_\_\_\_ deben ser tomados en cuenta, de tal forma que los shards no se conviertan en cuellos de botella en términos del rendimiento.
  
3. \_\_\_\_\_ crea múltiples copias de un dataset, conocidas como \_\_\_\_\_, y las almacena en varios nodos.
  
4. Los dos métodos para implementar la replicación incluyen la replicación \_\_\_\_\_ y la replicación \_\_\_\_\_.
  
5. En la replicación \_\_\_\_\_, el nodo \_\_\_\_\_ es el único punto de contacto para todas las operaciones de escritura, mientras que los datos pueden ser leídos desde cualquier nodo \_\_\_\_\_.
  
6. En la replicación \_\_\_\_\_ no hay nodos \_\_\_\_\_ y \_\_\_\_\_, y todos los nodos, llamados \_\_\_\_\_, operan al mismo nivel.
  
7. El sharding y la replicación se pueden combinar para mejorar la limitada \_\_\_\_\_ que ofrece el sharding, y al mismo tiempo, beneficiarse de la mayor \_\_\_\_\_ y \_\_\_\_\_ que ofrece la replicación.
  
8. El teorema CAP establece que un sistema de archivos distribuido solo puede proporcionar dos de las tres propiedades, que son \_\_\_\_\_, \_\_\_\_\_ y \_\_\_\_\_.

9. En el contexto del teorema CAP, las bases de datos relacionales proporcionan

\_\_\_\_\_ y \_\_\_\_\_.

10. En el contexto del diseño de bases de datos, el acrónimo BASE representa

\_\_\_\_\_, \_\_\_\_\_ y \_\_\_\_\_.

*Las respuestas al ejercicio se encuentran al final de este cuadernillo.*

## Notas

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---





## **Notas / Bocetos**

# **Características del dispositivo de almacenamiento de Big Data**

Big Data está integrado por datasets que no se pueden almacenar usando soluciones de almacenamiento tradicionales, principalmente debido al gran volumen de datos que contienen. La velocidad es un factor que hace que las soluciones tradicionales de bases de datos sean inapropiadas para el almacenamiento de Big Data, principalmente debido a su diseño centralizado que ofrece poca o nula escalabilidad.

La variedad característica de los datasets Big Data requiere atención especial, ya que se estima que el 80% de los datasets Big Data contiene datos sin estructurar, y las soluciones de almacenamiento tradicionales no soportan el almacenamiento de datos semiestructurados y sin estructurar de una forma escalable y eficiente.

Debido a las características únicas de Big Data, los datasets Big Data no pueden conservarse usando soluciones de almacenamiento tradicionales. Sin embargo, antes de elegir un mecanismo de almacenamiento alternativo para un entorno de solución de Big Data, se deben considerar las siguientes características en cuanto a dispositivos de almacenamiento:

- **escalabilidad**
- **redundancia y disponibilidad**
- **acceso rápido**
- **almacenamiento a largo plazo**
- **almacenamiento sin esquema**
- **almacenamiento de bajo costo**

## **Escalabilidad**

Los datasets de Big Data se presentan en volúmenes enormes, a gran velocidad, y generalmente son adquiridos a partir de varias fuentes. El resultado son **grandes cantidades de datos en un periodo corto de tiempo**. Con la posibilidad de encontrar nuevas percepciones acerca de la forma en que funcionan los negocios, las empresas —como parte de sus actividades de adquisición de datos— están guardando más y más datos generados tanto al interior de la empresa como obtenidos de fuentes externas.

Quizá no sea posible **readquirir** datos debido a **los altos costos de adquisición**; por ejemplo, cuando se compra un dataset a un proveedor de datos. Además, puede que no sea posible generar datos de nuevo si los eventos que generaron los datos inicialmente fueron únicos; por ejemplo, un medidor inteligente que funcionó en un momento determinado. En cualquier caso, se hace necesaria una solución de almacenamiento escalable, con suficiente capacidad para los requisitos actuales y futuros de captura de datos.

Además de almacenar los datos sin procesar, se requiere un almacenamiento adicional para conservar los datos creados como resultado de la **actividad de manipulación (wrangling)** de

**datos**. En el caso de la unión de varios datasets, el requerimiento de almacenamiento de datos aumentará en tamaño, debido a la necesidad de conservar tanto los datasets originales como el dataset adicional.

Se necesita mayor almacenamiento con el fin de conservar los resultados analíticos de una actividad de análisis de datos (Data Analysis). Para abordar la creciente demanda de almacenamiento de datos, **un dispositivo de almacenamiento puede estar sometido a escalabilidad vertical o a escalabilidad horizontal**.

### **Escalabilidad: escalabilidad vertical**

La **escalabilidad vertical** (vertical scaling), es una estrategia para aumentar la capacidad de recursos reemplazando un dispositivo existente de baja capacidad por uno de mayor capacidad. Por ejemplo, para duplicar la capacidad de un dispositivo de almacenamiento, los datos en un disco de 500 GB pueden ser copiados a un disco de 1 TB. **La escalabilidad vertical causará interrupción e inactividad del sistema**.

### **Escalabilidad: escalabilidad horizontal**

La **escalabilidad horizontal** (scaling out) es una estrategia para aumentar la capacidad de los recursos agregando dispositivos con capacidad similar o mayor al dispositivo existente. Por ejemplo, para duplicar la capacidad de almacenamiento, se puede añadir un disco de 500 GB a un disco existente de 500 GB. **La escalabilidad horizontal no causa interrupción ni inactividad del sistema**.

### **Escalabilidad**

Teniendo en cuenta las diferencias entre escalabilidad vertical y horizontal, un dispositivo de almacenamiento debería poder hacer una escalabilidad horizontal, ya que esta es una estrategia de menor costo que permite aumentar la capacidad de almacenamiento sin incurrir en inactividad e interrupciones del sistema. **El sharding puede ayudar a crear un almacenamiento escalable, ya que los shards pueden ser almacenados en los nodos que son agregados por medio de escalabilidad horizontal** (Figura 7.22).

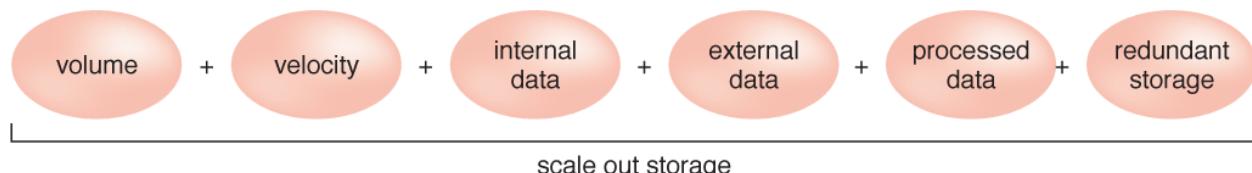


Figura 7.22 – Ejemplo de los componentes del almacenamiento de escalabilidad horizontal.

### **Redundancia y disponibilidad**

Los datasets de Big Data, en forma procesada o sin procesar, son activos de la empresa y requieren atención con respecto a su almacenamiento. Es posible que varias funciones de la

empresa necesiten adquirir valor de estos datasets, algunas veces de forma simultánea. En una empresa, esta dependencia en los datasets de Big Data requiere un dispositivo de almacenamiento confiable que sea **tolerante a errores** y esté **altamente disponible**.

Un entorno de solución de Big Data generalmente está compuesto por clusters que se construyen usando servidores básicos conectados por medio de una red con gran ancho de banda. Con más nodos y conexiones de red, aumentan las posibilidades de que un nodo deje de estar disponible, ya sea por una avería del hardware —tal como un fallo de disco— o por un fallo en la red. Como resultado, es necesaria **redundancia en el almacenamiento** para garantizar el acceso ininterrumpido a los datos en caso de una falla de un dispositivo de almacenamiento, proporcionando de esta forma alta disponibilidad y tolerancia a errores.

Para brindar tal redundancia, un dispositivo de almacenamiento implementa el sharding y la replicación automática con una configuración de **sharding y replicación maestro-esclavo** o **sharding y replicación peer-to-peer**.

## Acceso rápido

El análisis de Big Data generalmente implica el procesamiento en tiempo real y por lotes. **El análisis en tiempo real requiere capacidades de lectura/escritura rápida** que normalmente son implementadas por medio de soluciones de almacenamiento en memoria. Por otro lado, **el procesamiento por lotes (Batch Processing) requiere acceso a secuencia de datos con alta capacidad de procesamiento**, implementados mediante dispositivos tradicionales de almacenamiento basado en discos, o unidades más recientes de estado sólido que ofrecen mejor rendimiento a un costo mayor. A la vez, **los datos que son recibidos a una mayor velocidad requieren un dispositivo de almacenamiento que sea compatible con operaciones más rápidas de escritura con mínima sobrecarga**; por ejemplo, la validación de esquemas durante el tiempo de escritura en lugar del tiempo de lectura.

## Almacenamiento a largo plazo

El principio básico de Big Data es obtener valor de grandes cantidades de datos. Para ello, se requiere que las empresas **conserven los datos durante períodos prolongados de tiempo** para crear conjuntos más grandes de datos, de forma que los análisis futuros sean más profundos gracias a que cuentan con más datos históricos disponibles. Esta característica trae consigo la necesidad de un dispositivo de almacenamiento con **mayor capacidad de almacenamiento, que sea confiable y que pueda ser puesto en línea sin retrasos significativos**.

Tradicionalmente, los datos históricos que no son necesarios son archivados en un almacenamiento offline. Sin embargo, esto implica que dichos datos no están disponibles para análisis instantáneos. En comparación, el análisis de Big Data requiere que los datos históricos estén disponibles online para descubrir patrones ocultos que permitan obtener información valiosa. Como resultado, **los datos históricos se mantienen online al añadir más capacidad de almacenamiento** por medio de la escalabilidad.

En los entornos tradicionales de almacenamiento de datos, una vez que una base de datos alcanza el tope de su capacidad, los datos son descargados en unidades de cinta. En los

entornos Big Data, en lugar de descargarlos en unidades de cinta, los datos históricos se mantienen online y la capacidad de almacenamiento sencillamente aumenta mediante la escalabilidad (Figura 7.23).

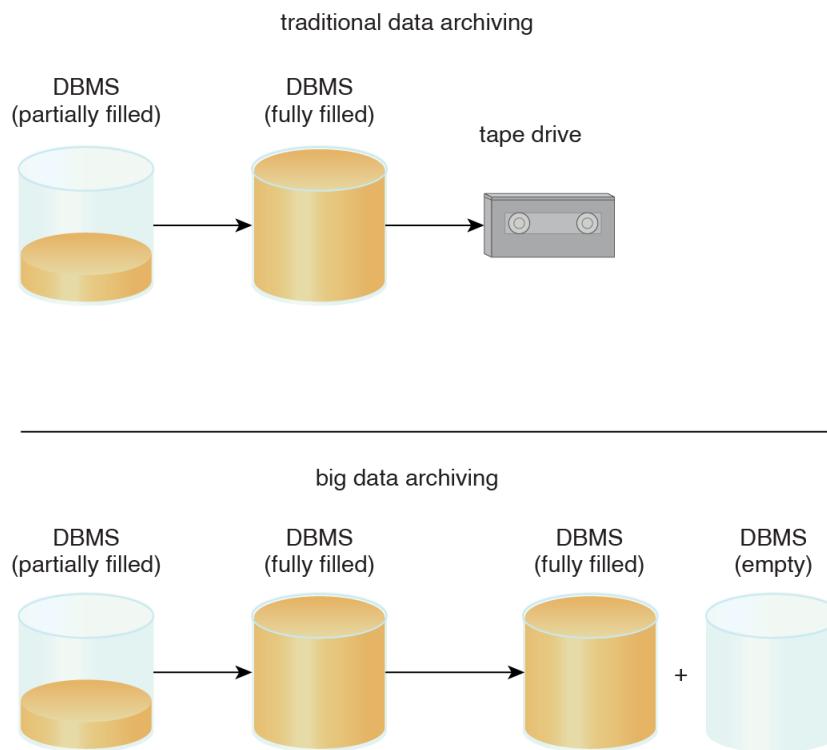


Figura 7.23 – Comparación entre el sistema tradicional de archivado y el archivado de Big Data.

### Almacenamiento sin esquema

Los conjuntos de datos Big Data tienen múltiples formatos con esquemas limitados o sin esquemas, como datos semiestructurados y sin estructurar. **No es posible aplicar un esquema sin antes tener conocimiento de la estructura de los datos.** De igual manera, tener algunos conocimientos sobre la estructura de los datos no garantiza que los mismos se ajusten a la misma estructura en el futuro, como en el caso de los datos sin estructurar. Esto requiere que el dispositivo de almacenamiento **sea compatible con la persistencia de datos sin esquema, así como capacidad adicional para que los esquemas cambien** durante el procesamiento sin interrumpir las aplicaciones existentes ni generar tiempos de inactividad.

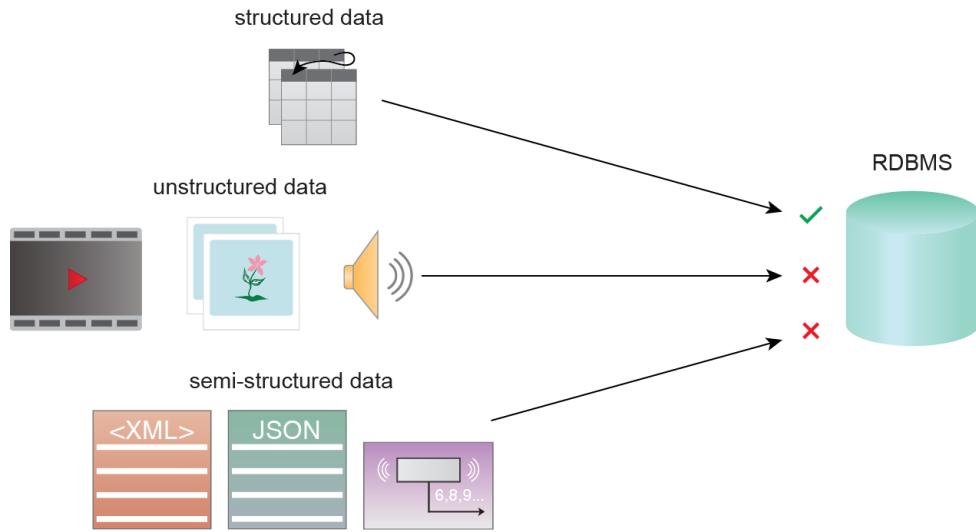


Figura 7.24 – Ejemplo de almacenamiento tradicional.

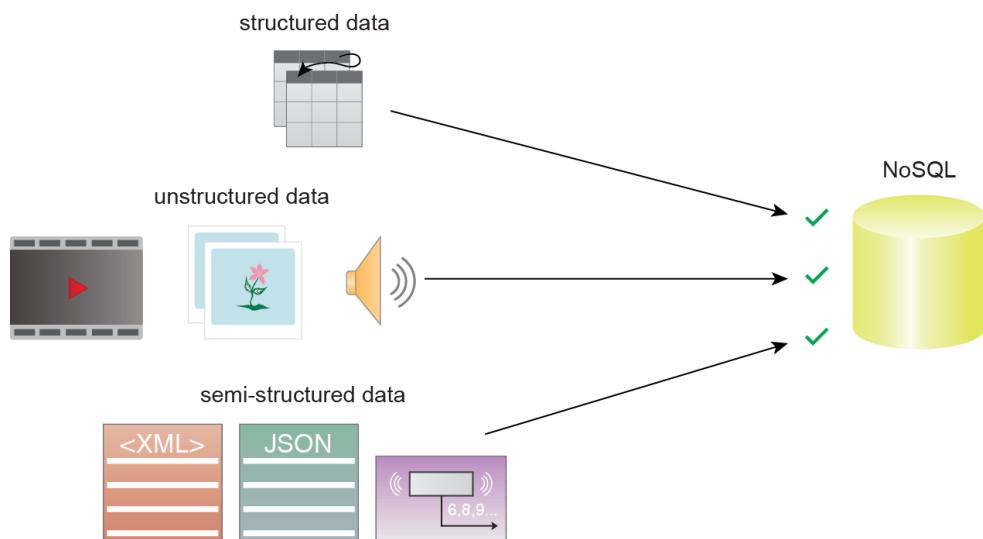


Figura 7.25 – Ejemplo de almacenamiento de Big Data.

## Almacenamiento de bajo costo

El costo de los dispositivos de almacenamiento puede ser preocupante con respecto a todas las características necesarias para almacenar datos voluminosos, proporcionar replicación y ofrecer compatibilidad con el almacenamiento a largo plazo. Un dispositivo de almacenamiento debe utilizar eficientemente los recursos subyacentes de disco para así no desperdiciar el espacio de almacenamiento.

Por lo general, el uso de dispositivos de almacenamiento patentados requiere que los nodos existentes sean reemplazados con nodos más costosos, con el fin de aumentar las capacidades, hasta que lleguen a un límite. **Un dispositivo de almacenamiento debe hacer uso**

de hardware básico que pueda ser mejorado de manera que los costos sigan siendo bajos a medida que las empresas acumulan más y más datos.

## Notas

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---





## **Notas / Bocetos**

# Dispositivos de almacenamiento en disco

Dependiendo del medio de almacenamiento utilizado, los dispositivos de almacenamiento se pueden dividir en dos tipos:

- **almacenamiento en disco**
- **almacenamiento en memoria**

La siguiente sección describe los tipos y características de un dispositivo de almacenamiento en disco, así como su idoneidad, teniendo en cuenta las características de almacenamiento de Big Data ya establecidas. Los dispositivos de almacenamiento en memoria son presentados en el Módulo 8.

## Dispositivo de almacenamiento en disco

Por lo general, el almacenamiento en disco utiliza unidades de disco duro de bajo costo para un almacenamiento a largo plazo. El almacenamiento en disco puede ser implementado por medio de un sistema de archivos distribuido o una base de datos, como se muestra en la Figura 7.26.

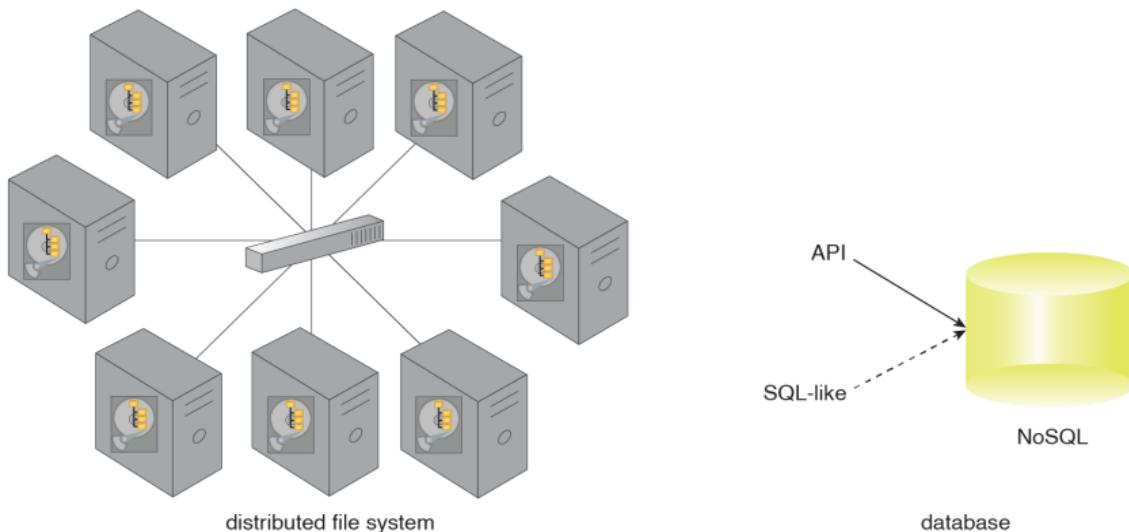


Figura 7.26 – Símbolos utilizados para representar un sistema de archivos distribuido y una base de datos NoSQL.

## Dispositivo de almacenamiento en disco: sistema de archivos distribuido

Un dispositivo de almacenamiento en el que se implementa un sistema de archivos distribuido ofrece almacenamiento de datos sencillo y de acceso rápido, y además es capaz de almacenar grandes datasets no relacionales, como datos semiestructurados y sin estructurar. A pesar de que está basado en mecanismos simples de bloqueo de archivos para controlar la concurrencia, ofrece capacidad de lectura/escritura rápida, lo cual maneja la característica de velocidad de Big Data.

Los sistemas de archivos distribuidos son una buena opción cuando se debe acceder a los datos en modo de streaming sin operaciones aleatorias de lectura ni escritura (Figura 7.27).

Un sistema de archivos distribuido que no sea ideal para los datasets compuestos por una cantidad de archivos pequeños crea una actividad excesiva de búsqueda en el disco, lo que ralentiza el acceso general a los datos. Asimismo, hay una mayor sobrecarga cuando se procesan múltiples archivos muy pequeños, ya que normalmente el motor de procesamiento genera procesos dedicados en tiempo de ejecución para procesar cada uno de los archivos antes de que los resultados sean sincronizados desde el otro lado del cluster.

Debido a estas limitaciones, los sistemas de archivos distribuidos funcionan mejor con menos archivos, pero de mayor tamaño, a los cuales se accede de forma secuencial. Por lo general, múltiples archivos pequeños son combinados en un solo archivo para así contar con un almacenamiento y procesamiento óptimos. Tenga en cuenta que los sistemas de archivos distribuidos no ofrecen ninguna capacidad de búsqueda por defecto.

Al momento de considerar los sistemas de archivos en el ámbito de Big Data, los únicos candidatos verdaderos son los sistemas de archivos distribuidos. Esto es porque los sistemas de archivos no distribuidos no pueden ser empleados en entornos de cluster, lo cual es un requisito para procesar grandes datasets usando clusters. (La arquitectura en cluster es presentada en la sección *Procesamiento fundamental de Big Data*, más adelante.)

Un dispositivo de almacenamiento con un sistema de archivos distribuido es ideal si se deben almacenar datasets sin procesar de gran tamaño, o cuando se requiere el archivado de los datasets. Además, esto ofrece una opción de bajo costo para el almacenamiento de grandes cantidades de datos que permanecen online durante un periodo de tiempo. Esto se debe a que sencillamente se pueden añadir más discos sin necesidad de descargar los datos a dispositivos de almacenamiento offline, como unidades de cinta.

Al igual que cualquier sistema de archivos, los sistemas de archivos distribuidos son independientes de los datos que son almacenados, y por lo tanto son compatibles con el almacenamiento de datos sin esquema. En general, un dispositivo de almacenamiento con sistema de archivos distribuido ofrece redundancia por defecto y una alta disponibilidad, al copiar los datos a múltiples ubicaciones mediante la replicación.

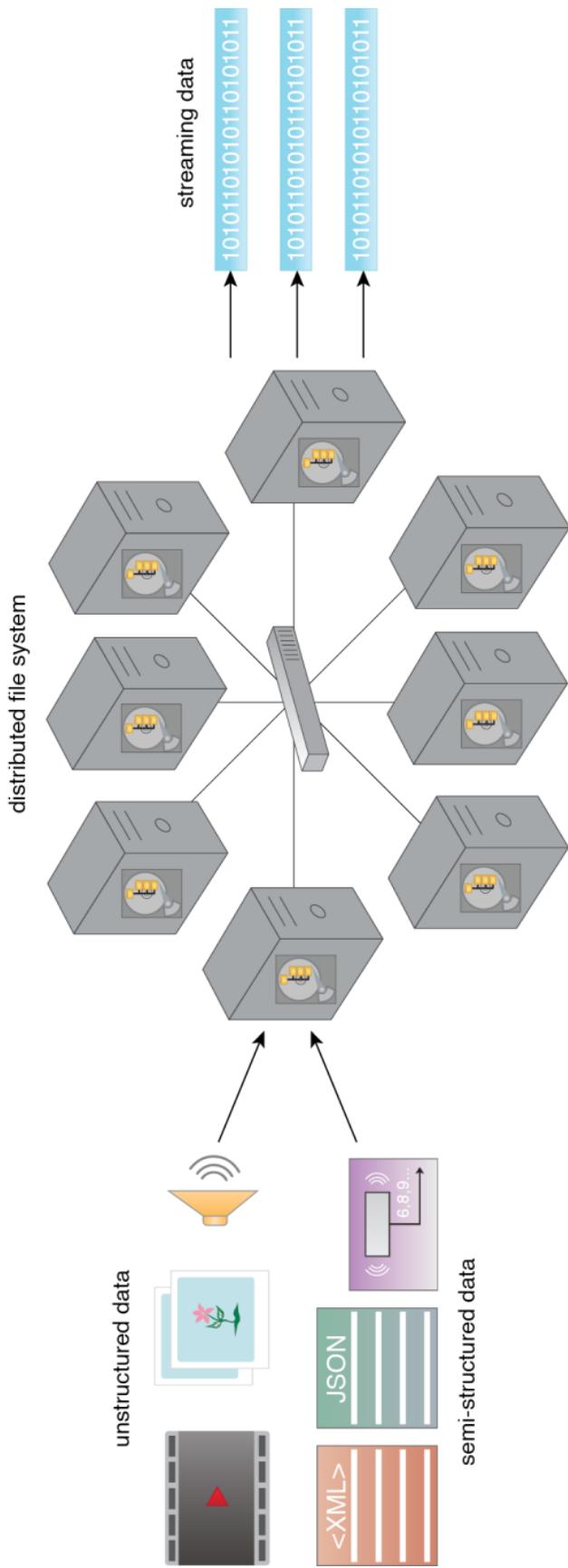


Figura 7.27 – Sistema de archivos distribuido con acceso a datos en modo de streaming sin operaciones aleatorias de lectura ni escritura.

## **Dispositivo de almacenamiento en disco: base de datos**

En los entornos de solución de Big Data, predominan tres tipos de tecnología de bases de datos para el almacenamiento en disco:

- **bases de datos relacionales o sistemas de gestión de bases de datos relacionales (RDBMS, por sus siglas en inglés)**
- **bases de datos no relacionales o no solo SQL (NoSQL)**
- **NewSQL**

En la siguiente sección se analiza la idoneidad de estas tres clases de bases de datos como dispositivos de almacenamiento de Big Data.

### **RDBMS**

Los sistemas de gestión de bases de datos relacionales (RDBMS, por sus siglas en inglés) son ideales para el manejo de cargas de trabajo transaccionales que combinan pequeñas cantidades de datos con operaciones aleatorias de lectura y escritura. Los RDBMS **son compatibles con las características ACID** y, como tal, **generalmente están restringidos a un solo nodo**. Por este motivo, los RDBMS **no ofrecen ningún tipo de redundancia por defecto ni tolerancia a errores**.

Por lo general, las bases de datos relacionales deben ser actualizadas a fin de gestionar grandes volúmenes de datos que son recibidos a un ritmo rápido. Los RDBMS **utilizan la escalabilidad vertical, no la horizontal, que es una estrategia de escalabilidad más costosa y desorganizada**. Esto hace que los RDBMS no sean tan adecuados para el almacenamiento de datos a largo plazo.

Tenga en cuenta que algunas bases de datos relacionales, por ejemplo IBM DB2 pureScale, Sybase ASE Cluster Edition, Oracle Real Application Clusters (RAC) y Microsoft Parallel Data Warehouse (PDW) pueden ser ejecutadas en clusters (Figura 7.28). Sin embargo, los clusters de estas bases de datos siguen utilizando el almacenamiento compartido, lo que puede ser un punto único de error.

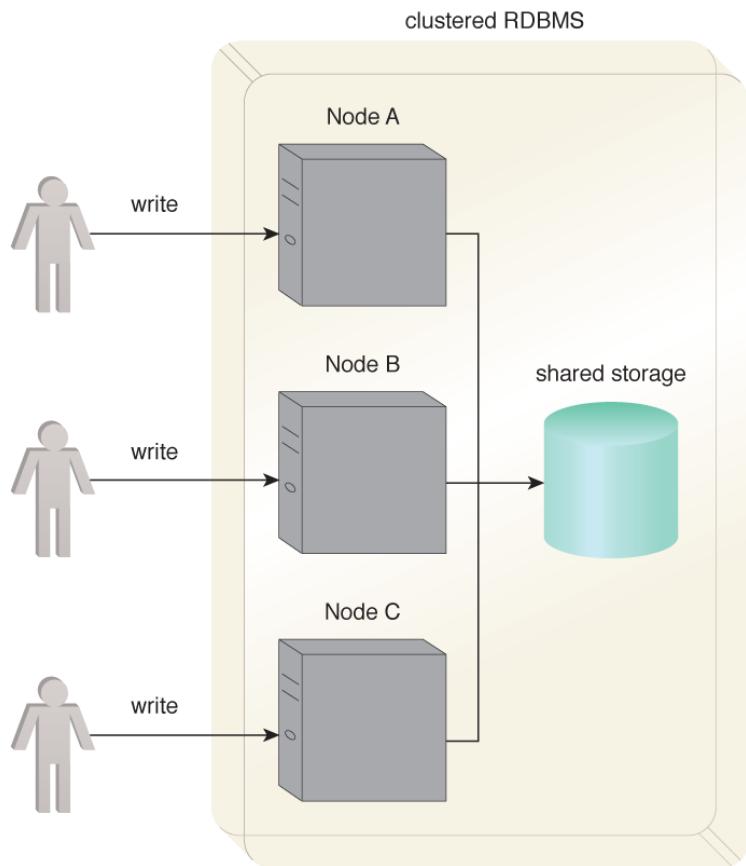


Figura 7.28 – Una base de datos relacional en clusters utiliza una arquitectura de almacenamiento compartido, la cual es un posible punto único de fallo que afecta la disponibilidad de la base de datos.

En cuanto al sharding, las bases de datos relacionales deben ser **particionadas manualmente**, en su mayoría usando **lógica de aplicación**. Esto quiere decir que el cliente (la lógica de aplicación) debe saber qué shard consultar con el fin de obtener los datos necesarios. Esto complica aún más el procesamiento de datos cuando se requieren datos de múltiples shards.

La Figura 7.29 ilustra los siguientes pasos:

1. Un usuario escribe un registro (id = 2).
2. La lógica de aplicación determina en qué shard debe ser escrito.
3. El registro es enviado al shard determinado por la lógica de aplicación.
4. El usuario lee un registro (id = 4) y la lógica de aplicación determina qué shard contiene los datos.
5. Los datos son leídos y devueltos a la aplicación.
6. La aplicación le devuelve el registro al usuario.

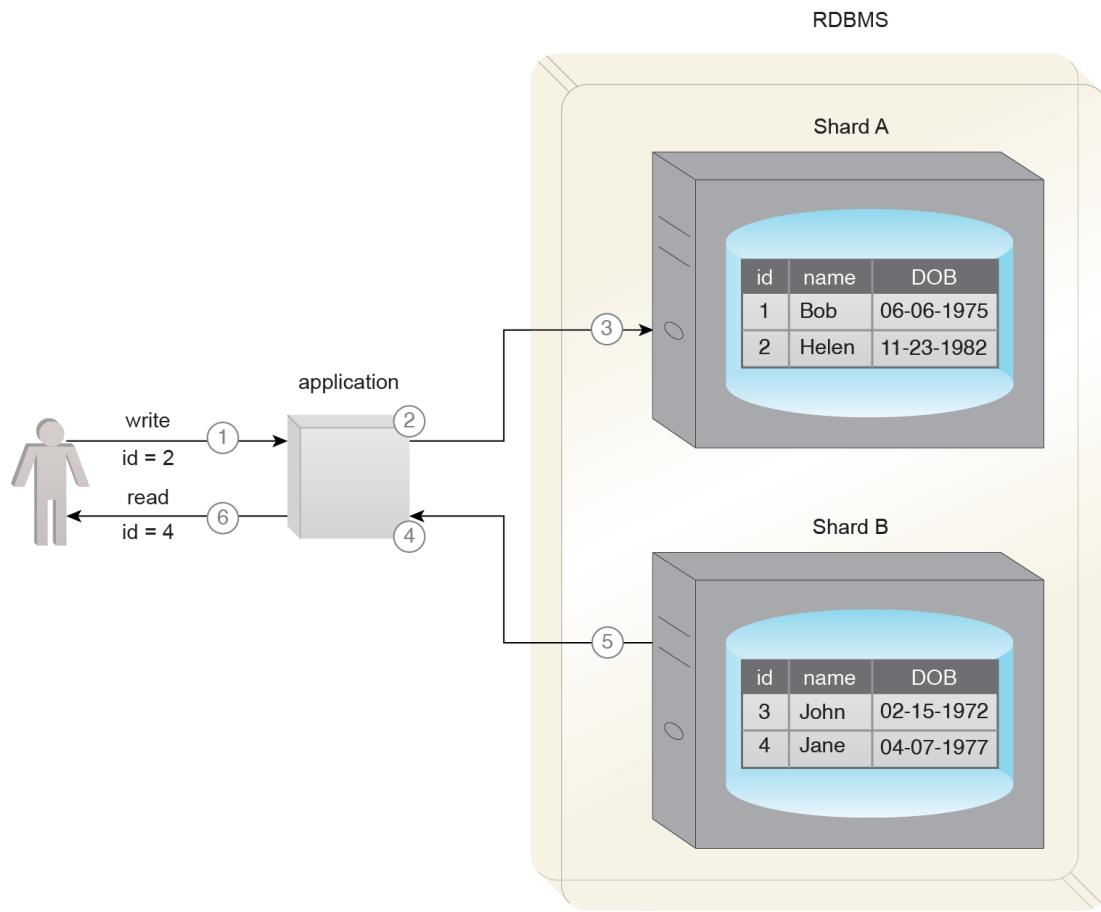


Figura 7.2 – Ejemplo de una base de datos relacional que es particionada (sharding) manualmente mediante la lógica de aplicación.

La Figura 7.30 ilustra los siguientes pasos:

1. Un usuario solicita múltiples registros (**id = 1, 3**) y se utiliza la lógica de aplicación para determinar qué shards deben ser leídos.
2. La lógica de aplicación determina que los shards A y B deben ser leídos.
3. Los datos son leídos y combinados por la aplicación.
4. Por último, los datos son devueltos al usuario.

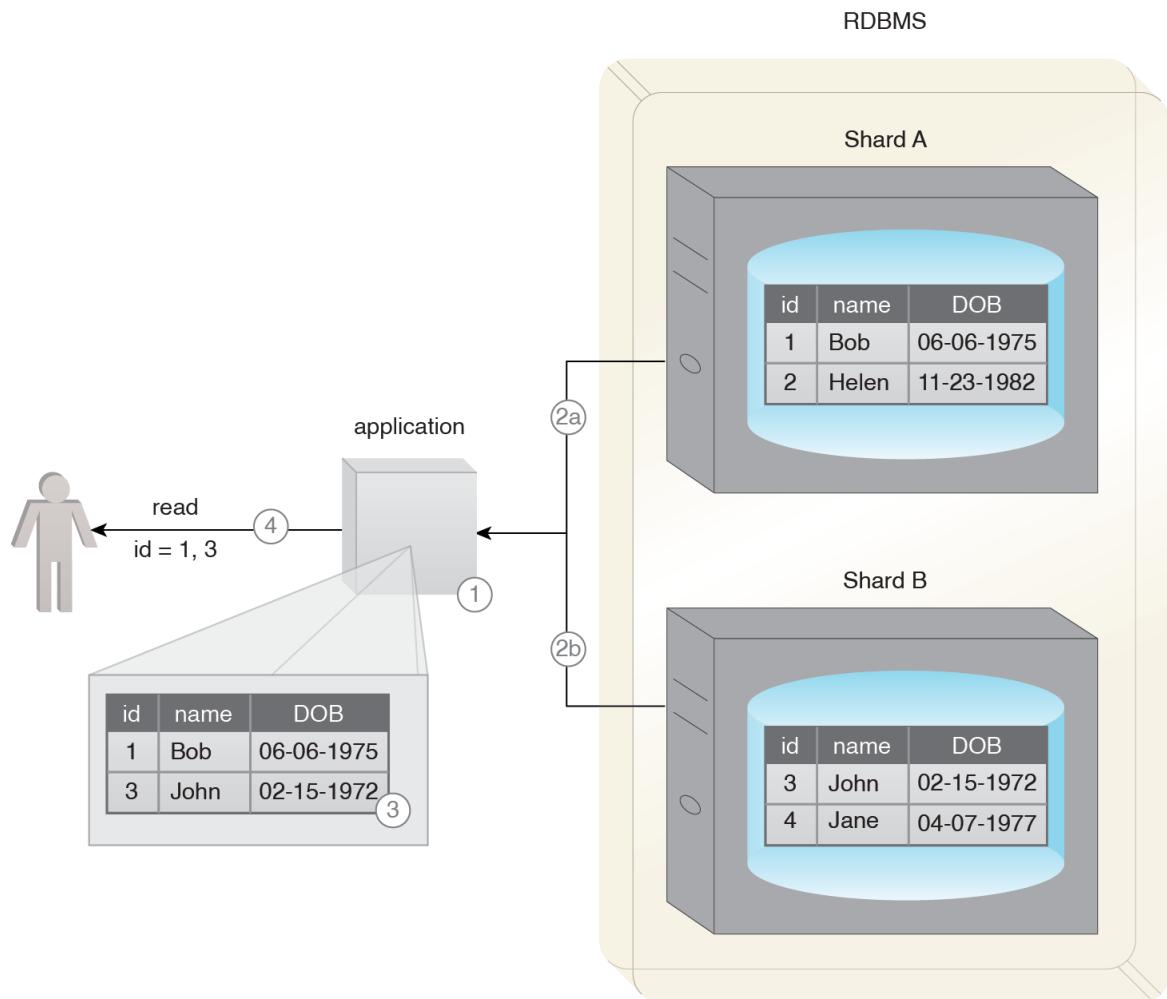


Figure 7.30 – Ejemplo del uso de la lógica de aplicación.

Por lo general, las bases de datos relacionales exigen que los datos se ajusten a un esquema. Como resultado, no es posible almacenar datos semiestructurados o sin estructurar cuyo esquema no sea conocido o cambie constantemente. Además, **al momento del ingreso o actualización de los datos se valida la conformidad del esquema, lo que genera sobrecarga y latencia.**

Esta latencia es lo que convierte a las bases de datos relacionales en una elección poco adecuada para almacenar datos altamente veloces, ya que se necesita un dispositivo de almacenamiento de bases de datos altamente disponible y con capacidad de escribir datos rápidamente. A causa de sus deficiencias, un RDBMS tradicional no es útil como dispositivo de almacenamiento para un entorno de solución de Big Data.

## Dispositivo de almacenamiento en disco: NoSQL

El término *No Solo SQL* (*NoSQL*) se refiere a la tecnología utilizada para desarrollar bases de datos no relacionales de nueva generación con alta escalabilidad y tolerancia a errores.



NoSQL Database

Figura 7.31 – Símbolo utilizado para representar una base de datos NoSQL.

## Dispositivo de almacenamiento en disco: características de NoSQL

A continuación se enumeran algunas de las características principales de los dispositivos de almacenamiento NoSQL que los diferencian de los RDBMS tradicionales. Esta lista debe ser considerada únicamente como una guía general, ya que no todos los dispositivos de almacenamiento NoSQL cuentan con estas características.

- **Modelo de datos sin esquema:** pueden existir datos sin procesar.
- **Escalabilidad horizontal en lugar de escalabilidad vertical:** se añaden más nodos, a medida que sea necesario, en lugar de reemplazar el nodo actual con uno más grande y de mejor rendimiento.
- **Alta disponibilidad:** la tecnología incorporada basada en clusters brinda tolerancia a errores por defecto.
- **Menores costos operativos:** plataformas incorporadas de código abierto y sin costos de licenciamiento, que pueden ser implementadas en hardware básico.
- **Consistencia a largo plazo:** es posible que las operaciones de lectura a través de múltiples nodos no sean consistentes inmediatamente después de una operación de escritura. Sin embargo, con el tiempo, todos los nodos tendrán un estado de consistencia.
- **Compatible con BASE, no con ACID:** la compatibilidad con las características BASE requiere que una base de datos mantenga alta disponibilidad en caso de un fallo de red o de un nodo, pero no es necesario que la base de datos tenga un estado de consistencia cada vez que ocurre una actualización. La base de datos puede tener un estado no consistente o flexible hasta que finalmente obtiene consistencia. Como resultado, teniendo en cuenta el teorema CAP, los dispositivos de almacenamiento NoSQL generalmente son AP o CP.
- **Acceso a los datos determinado por API:** el acceso a los datos generalmente está respaldado por solicitudes basadas en API, incluyendo API RESTful, mientras que algunas implementaciones también brindan una capacidad de consulta similar a SQL.
- **Autosharding y replicación:** para ser compatible con la escalabilidad horizontal y brindar alta disponibilidad, un dispositivo de almacenamiento NoSQL automáticamente utiliza técnicas

de sharding y replicación, en las cuales los datasets son particionados horizontalmente y luego copiados a múltiples nodos.

- **Almacenamiento en caché integrado:** elimina la necesidad de una capa de almacenamiento distribuida en caché de terceros, como Memcached.
- **Soporte distribuido para consultas:** los dispositivos de almacenamiento NoSQL mantienen un comportamiento de consultas consistente a través de múltiples shards.
- **Persistencia políglota:** el uso del dispositivo de almacenamiento NoSQL no exige retirar los RDBMS tradicionales. De hecho, ambos pueden ser utilizados al mismo tiempo, y por consiguiente, es compatible con la persistencia políglota (un enfoque de datos persistentes usando distintos tipos de tecnologías de almacenamiento). Esto es útil para el desarrollo de sistemas que requieren datos estructurados y datos sin estructurar.
- **Basada en datos agregados:** a diferencia de las bases de datos relacionales que son más eficaces con datos completamente normalizados, los dispositivos de almacenamiento NoSQL almacenan datos desnormalizados agregados (una entidad que contiene datos combinados, a menudo anidados, de un objeto) y por lo tanto se elimina la necesidad de joins y asignaciones entre objetos de aplicación y los datos almacenados en la base de datos. Tenga en cuenta que los dispositivos de almacenamiento de bases de datos de grafos (presentados más adelante) no están basados en datos agregados.

## **Dispositivo de almacenamiento en disco: justificación de NoSQL**

La aparición de dispositivos de almacenamiento NoSQL puede ser atribuida principalmente a las características de volumen, velocidad y variedad de los datasets de Big Data.

### **Volumen**

El requisito de almacenamiento de volúmenes cada vez más grandes de datos exige el uso de bases de datos altamente escalables y que reduzcan los costos, para que así la empresa sea competitiva. Los dispositivos de almacenamiento NoSQL cumplen este requisito al brindar capacidad de escalabilidad y utilizar servidores básicos de bajo costo. Además, no hay ningún tipo de costo de licenciamiento, ya que las bases de datos NoSQL generalmente siguen el modelo de desarrollo de Código Abierto.

### **Velocidad**

El veloz flujo de entrada de datos necesita bases de datos con capacidad de escribir datos con acceso rápido. Los dispositivos de almacenamiento NoSQL permiten ejecutar operaciones de escritura rápidas sin validación contra estructura, en lugar de utilizar el principio de validación contra estructura. Con su alta disponibilidad, los dispositivos de almacenamiento NoSQL pueden garantizar que no ocurra la latencia de escritura debido a fallas de nodos o de red.

### **Variedad**

Es necesario que un dispositivo de almacenamiento maneje distintos tipos de formatos de datos, incluidos documentos, correos electrónicos, imágenes y videos, así como datos incompletos. Los dispositivos de almacenamiento NoSQL pueden almacenar estos distintos

tipos de formatos de datos semiestructurados y sin estructurar. Al mismo tiempo, los dispositivos de almacenamiento NoSQL pueden almacenar datos sin esquemas y datos incompletos con la capacidad adicional de hacer cambios en el esquema a medida que evoluciona el modelo de datos de los datasets. En otras palabras, las bases de datos NoSQL son compatibles con la *evolución del esquema*.

## Dispositivo de almacenamiento en disco: tipos de NoSQL

Los dispositivos de almacenamiento NoSQL pueden dividirse en cuatro tipos principales, según la forma en que almacenan los datos, como se muestra en las Figuras 7.32 a 7.35:

- llave-valor (key-value)
- documento
- basado en columnas
- grafo

key	value
631	John Smith, 10.0.30.25, Good customer service
365	100101011101101111011101010110101001110011010
198	<CustomerId>32195</CustomerId><Total>43.25</Total>

Figura 7.32 – Ejemplo de almacenamiento NoSQL de tipo llave-valor (key-value).



Figura 7.33 – Ejemplo de almacenamiento NoSQL de tipo documento.

studentId	personal details	address	modules history
821	FirstName: Cristie LastName: Augustin DoB: 03-15-1992 Gender: Female Ethnicity: French	Street: 123 New Ave City: Portland State: Oregon ZipCode: 12345 Country: USA	Taken: 5 Passed: 4 Failed: 1
742	FirstName: Carlos LastName: Rodriguez MiddleName: Jose Gender: Male	Street: 456 Old Ave City: Los Angeles Country: USA	Taken: 7 Passed: 5 Failed: 2

Figura 7.34 – Ejemplo de almacenamiento NoSQL de tipo basado en columnas.

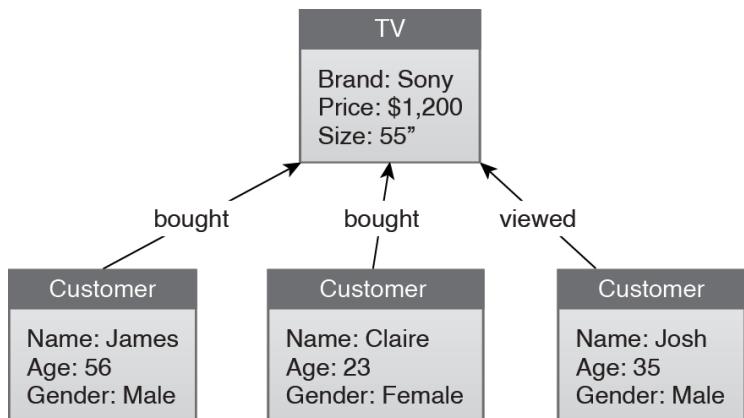


Figura 7.35 – Ejemplo de almacenamiento NoSQL de tipo grafo.

## NoSQL: Llave-valor (key-value)

Los dispositivos de almacenamiento del tipo Llave-valor (key-value) almacenan los datos como parejas de una llave y su valor correspondiente, que funcionan como tablas hash. La tabla es una lista de valores, en la que cada valor está identificado con una llave. El valor no es visible para la base de datos, y en esencia es almacenado como un BLOB. El valor almacenado puede ser cualquier dato agregado, desde datos de un sensor hasta videos.

Únicamente se pueden buscar los valores por medio de las llaves, ya que la base de datos no tiene en cuenta los detalles de los datos agregados almacenados. No es posible hacer actualizaciones parciales. **Las actualizaciones son operaciones de eliminación o inserción.**

Por lo general, los dispositivos de almacenamiento de tipo Llave-valor (key-value) no actualizan ningún índice, y por consiguiente, las operaciones de escritura son bastante rápidas. **Los dispositivos de almacenamiento de tipo Llave-valor (key-value) basados en un modelo sencillo de almacenamiento son altamente escalables.**

Ya que las llaves son el único medio a través del cual se puede recuperar la información, generalmente la llave está anexada al tipo de valor que es guardado, para así recuperarlo fácilmente. Por ejemplo, 123\_sensor1.

A fin de que los datos almacenados tengan una estructura, la mayoría de dispositivos de almacenamiento de tipo Llave-valor (key-value) ofrecen colecciones o sectores de almacenamiento (buckets; por ejemplo, tablas) dentro de los cuales se pueden organizar las parejas de Llave-valor (key-value), como se muestra en la Figura 7.36. Algunas implementaciones admiten la compresión de valores para reducir así el espacio de almacenamiento. Sin embargo, esto genera latencia al momento de realizar operaciones de lectura, ya que los datos deben ser descomprimidos primero.

key	value
631	John Smith, 10.0.30.25, Good customer service
365	101011010101101010111010110101011010101110101110
198	<CustomerId>32195</CustomerId><Total>43.25</Total>

The diagram shows three rows of a table with columns 'key' and 'value'. Row 1: key 631, value 'John Smith, 10.0.30.25, Good customer service' with an arrow pointing to 'text'. Row 2: key 365, value '101011010101101010111010110101011010101110101110' with an arrow pointing to 'image'. Row 3: key 198, value '<CustomerId>32195</CustomerId><Total>43.25</Total>' with an arrow pointing to 'XML'.

Figura 7.36 – Ejemplo de datos organizados en parejas Llave-valor (key-value).

Un dispositivo de almacenamiento de tipo Llave-valor (key-value) es ideal si:

- es necesario almacenar datos sin estructurar
- es necesario ejecutar operaciones de lectura o escritura de alto rendimiento
- el valor puede ser identificado únicamente por medio de la llave
- el valor es una entidad independiente y no depende de otros valores
- generalmente, los valores tienen una estructura relativamente sencilla o son binarios
- los patrones de consulta son sencillos, e incluyen únicamente operaciones de inserción, selección y eliminación

- los valores almacenados son manipulados a nivel de la aplicación

Un dispositivo de almacenamiento de tipo llave-valor (key-value) no es adecuado si:

- las aplicaciones requieren buscar o filtrar (filtering) los datos utilizando los atributos del valor almacenado
- existen relaciones entre distintas entradas de llave-valor (key-value)
- se debe actualizar un grupo de valores de llaves en una sola operación
- es necesario manipular múltiples llaves en una sola operación
- es necesario mantener la consistencia entre distintos valores
- es necesario actualizar atributos individuales del valor

Entre los ejemplos de dispositivos de almacenamiento de tipo llave-valor (key-value) están Riak, Redis y Amazon DynamoDB.

## Lectura

En la sección *Bases de datos de tipo llave-valor (key-value)*, en las páginas 81 a 88 del libro *Resumen de NoSQL* incluido en este módulo, encontrará un análisis más detallado sobre los dispositivos de almacenamiento de tipo llave-valor (key-value).

## NoSQL: documento

Los dispositivos de almacenamiento de tipo documento también almacenan los datos como parejas llave-valor (key-value). Sin embargo, **a diferencia de los dispositivos de almacenamiento llave-valor (key-value), el valor almacenado es un documento que puede tener una estructura anidada compleja**; por ejemplo, una factura, tal como lo ilustra la Figura 7.37. El documento puede ser codificado utilizando un esquema de codificación basado en texto, como XML o JSON, o utilizando un esquema de codificación binario, como BSON (JSON binario).

Al igual que los dispositivos de almacenamiento llave-valor (key-value), la mayoría de los dispositivos de almacenamiento de tipo documento cuentan con colecciones o sectores de almacenamiento (buckets; por ejemplo, tablas) en los cuales se pueden organizar las parejas de llave-valor (key-value). Las siguientes son las principales diferencias entre los dispositivos de almacenamiento de tipo documento y los dispositivos de almacenamiento de tipo llave-valor (key-value):

- los dispositivos de almacenamiento de tipo documento tienen en cuenta el valor
- el valor almacenado es autodescriptivo; se puede deducir el esquema a partir de la estructura del valor
- una operación de selección puede referirse a un campo al interior del valor agregado
- una operación de selección puede recuperar una parte del valor agregado

- existe compatibilidad con las actualizaciones parciales; se puede actualizar un subconjunto del valor agregado
- por lo general existe compatibilidad con los índices que aceleran las búsquedas

Cada documento puede tener un esquema distinto. Es posible almacenar documentos de distintos tipos, o documentos del mismo tipo que tengan menos o más campos entre sí. Se pueden añadir campos a un documento luego de la inserción inicial, lo que refleja una compatibilidad flexible con esquemas.

Cabe señalar que los dispositivos de almacenamiento de tipo documento no se limitan a almacenar datos que tengan forma de documentos, como un archivo XML, sino que **también pueden ser usados para almacenar cualquier dato agregado que esté compuesto por una colección de campos que tienen un esquema plano o anidado.**

Un dispositivo de almacenamiento de tipo documento es ideal si:

- se van a almacenar datos semiestructurados orientados a documentos compuestos por esquemas planos o anidados
- es necesaria la evolución de los esquemas, ya que no se conoce la estructura del documento o es probable que esta cambie
- las aplicaciones requieren una actualización parcial de los datos agregados almacenados como un documento
- se deben realizar búsquedas en distintos campos de los documentos
- se van a almacenar objetos de dominio, como clientes, en forma de objetos
- los patrones de consulta implican operaciones de inserción, selección, actualización y eliminación

Un dispositivo de almacenamiento de tipo documento no es adecuado si:

- es necesario actualizar múltiples documentos como parte de una sola transacción
- se deben realizar operaciones que requieren joins entre múltiples documentos o datos normalizados de almacenamiento
- es necesario aplicar los esquemas para obtener un diseño consistente de consulta, ya que la estructura del documento puede cambiar en medio de ejecuciones sucesivas de consultas, lo que haría necesario que se reestructure la consulta
- el valor almacenado no es autodescriptivo
- se deben almacenar datos binarios

Entre los ejemplos de dispositivos de almacenamiento de tipo documento están MongoDB, CouchDB y Terrastore.

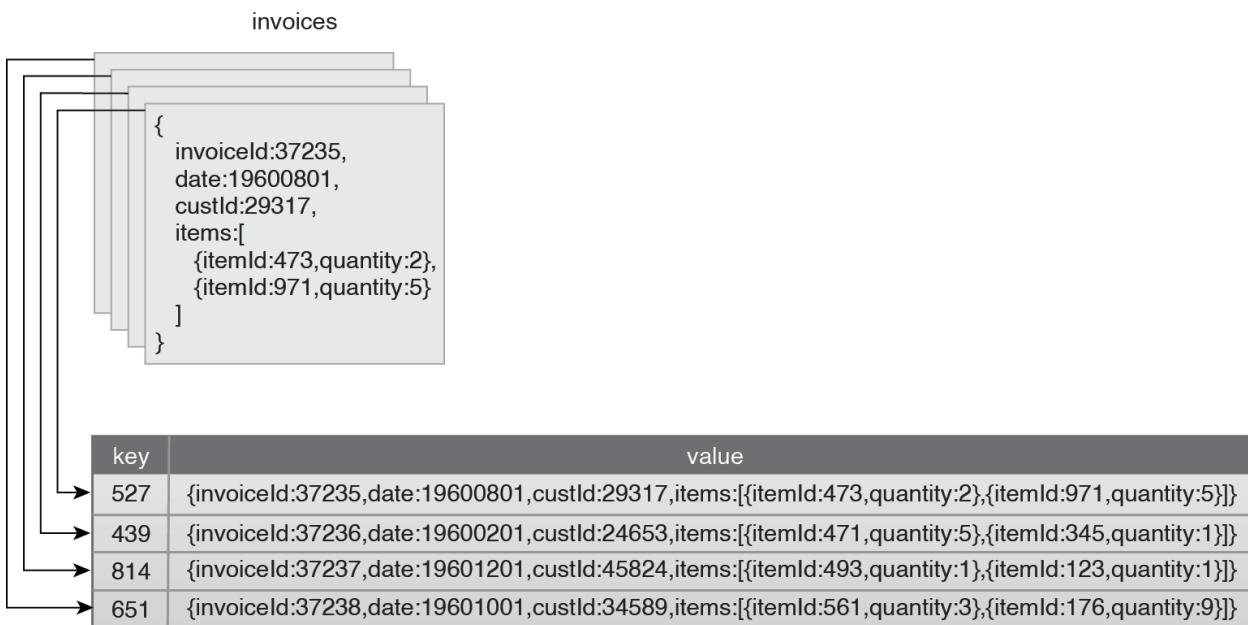


Figura 7.37 – Ejemplo de un valor almacenado como un documento con una estructura anidada compleja.

## Lectura

En la sección *Bases de datos de tipo documento*, en las páginas 89 a 98 del libro *Resumen de NoSQL* incluido en este módulo, encontrará un análisis más detallado sobre los dispositivos de almacenamiento de tipo documento.

## NoSQL: basado en columnas

Los dispositivos de almacenamiento basados en columnas almacenan datos de forma similar a un RDBMS, pero **agrupan las columnas relacionadas en una fila**, lo que genera **familias de columnas** (Figura 7.38). Cada columna en si puede estar compuesta por un conjunto de columnas relacionadas, a lo que se le conoce como **supercolumna**.

Cada supercolumna puede contener un número arbitrario de columnas relacionadas que generalmente son recuperadas o almacenadas como una unidad. Cada fila está compuesta por múltiples familias de columnas y puede tener un conjunto de columnas distinto, lo que refleja una **compatibilidad flexible con esquemas**. Cada fila está identificada por una **clave de fila**.

Los dispositivos de almacenamiento basados en columnas brindan **acceso rápido a los datos** con **capacidad de lectura/escritura aleatoria**, y almacenan distintas columnas-familias en archivos físicos separados, lo que ayuda en gran medida a acelerar las consultas, ya que solo se hacen búsquedas en las columnas-familias requeridas.

Algunos dispositivos de almacenamiento basados en columnas son compatibles con la **compresión selectiva basada en columnas**. No comprimir las columnas que pueden ser utilizadas en búsquedas puede acelerar las consultas, ya que no es necesario comprimir la columna objetivo para realizar la búsqueda. La mayoría de implementaciones son compatibles con el control de versiones de datos, mientras que algunas son compatibles con un tiempo de

expiración específico después del cual las columnas configuradas son eliminadas automáticamente.

Un dispositivo de almacenamiento basado en columnas es ideal si:

- se necesita la capacidad de escritura/lectura aleatoria en tiempo real, y los datos que serán almacenados tienen una estructura definida
- los datos representan una estructura tabular, cada fila está compuesta de un gran número de columnas y existen grupos anidados de datos interrelacionados
- se requiere la compatibilidad con la evolución del esquema, ya que se pueden añadir o eliminar columnas sin que se generen tiempos de inactividad en el sistema
- casi siempre se accede a la vez a ciertos campos, y las búsquedas deben ser ejecutadas usando valores de campos
- se requiere usar el almacenamiento eficazmente cuando los datos están compuestos por filas incompletas, ya que las bases de datos basadas en columnas únicamente asignan espacio de almacenamiento si una columna existe para una fila (si no hay columnas, no se asigna espacio)
- los patrones de consulta implican operaciones de inserción, selección, actualización y eliminación

Un dispositivo de almacenamiento basado en columnas no es adecuado si:

- se requiere acceso a datos relacionales; por ejemplo, joins
- se requiere compatibilidad transaccional con ACID
- se deben almacenar datos binarios
- se deben ejecutar consultas que sean compatibles con SQL
- es probable que los patrones de consulta cambien frecuentemente, lo que inicia la correspondiente restructuración de las columnas; por ejemplo, durante el desarrollo de la prueba de concepto

Algunos ejemplos son Cassandra, HBase y Amazon SimpleDB.

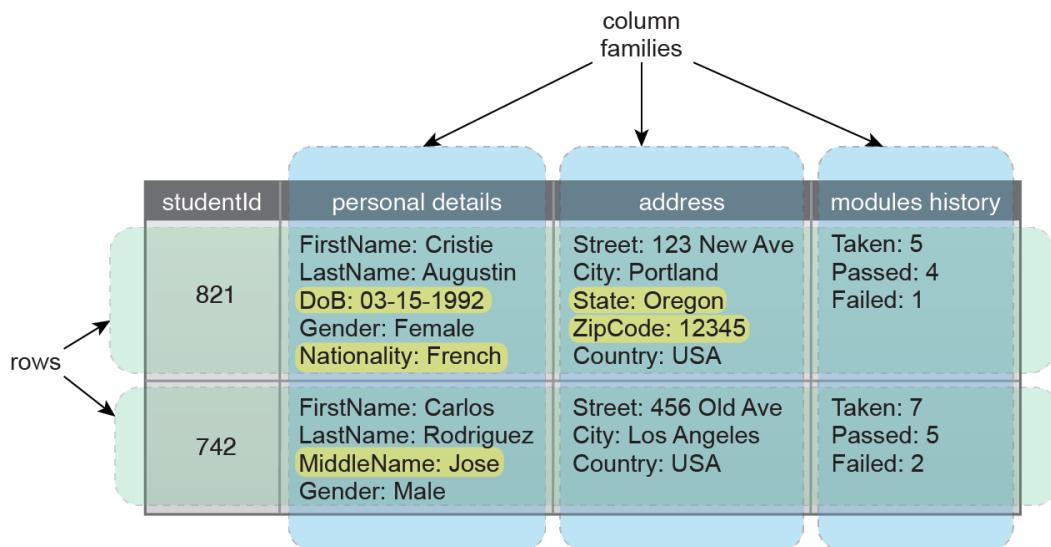


Figura 7.38 – Las columnas resaltadas representan la funcionalidad de esquema admitida por las bases de datos basadas en columnas, en la que cada fila puede tener un conjunto distinto de columnas.

## Lectura

En la sección *Almacenamiento basado en columnas*, en las páginas 99 a 109 del libro *Resumen de NoSQL* incluido en este módulo, encontrará un análisis más detallado sobre los dispositivos de almacenamiento basados en columnas.

## NoSQL: grafo

Los dispositivos de almacenamiento de tipo grafo son utilizados para **dar continuidad a las entidades interconectadas**. A diferencia de otros dispositivos de almacenamiento NoSQL en los cuales se hace énfasis en la estructura de las entidades, los dispositivos de almacenamiento de tipo grafo hacen énfasis en el almacenamiento de los links entre las entidades (Figura 7.39).

**Las entidades son almacenadas como nodos** (no confundirlos con los *nodos de cluster*), también llamados vértices, mientras que **los links entre las entidades son almacenados como bordes**. En la jerga de RDBMS, cada nodo puede ser considerado como una **fila**, mientras que un borde indica un **join**.

Entre los nodos puede existir más de un tipo de link por medio de múltiples bordes. Cada nodo puede tener datos de atributo como parejas llave-valor (key-value); por ejemplo, un nodo de cliente con atributos de ID, nombre y edad.

Cada borde tiene sus propios **datos de atributo como parejas llave-valor (key-value)**, los cuales pueden ser utilizados para filtrar (filtering) aún más los resultados de la consulta. Usar múltiples bordes es semejante a definir múltiples llaves externas en una RDBMS. Por lo general, las consultas implican la búsqueda de nodos interconectados con base en los atributos del nodo o del borde, lo que comúnmente se conoce como **recorrido de nodos**. Los bordes pueden ser **unidireccionales** o **bidireccionales**, lo que determina la dirección de recorrido del nodo.

Normalmente, los dispositivos de almacenamiento de tipo grafo proporcionan consistencia al cumplir con las características de ACID.

El grado de utilidad de un dispositivo de almacenamiento de tipo grafo depende de la cantidad y de los tipos de bordes definidos entre los nodos. Entre más bordes haya y más diversos sean, se podrán manejar consultas más variadas. Como resultado, **es importante capturar de forma rigurosa los tipos de relaciones que existen entre los nodos**. Esto no solo es cierto para los escenarios actuales de uso, sino también para el análisis exploratorio de datos (Data Analysis).

Los dispositivos de almacenamiento de tipo grafo generalmente permiten que se añadan nuevos tipos de nodos sin hacer cambios a la base de datos. También permiten que se definan links adicionales entre los nodos a medida que aparecen nuevos tipos de relaciones o nodos en la base de datos.

Un dispositivo de almacenamiento de tipo grafo es ideal si:

- se deben almacenar las entidades interconectadas
- se consultan las entidades con base en el tipo de relación que tienen entre sí, en lugar de sus atributos
- se buscan grupos de entidades interconectadas
- se buscan distancias entre las entidades en términos de distancia de recorrido del nodo
- se realiza la minería de datos (Data Mining) con el fin de hallar patrones

Un dispositivo de almacenamiento de tipo grafo no es adecuado si:

- se requiere actualizar una gran cantidad de atributos de nodos o de bordes, ya que esto implica buscar nodos o bordes, lo cual es una operación costosa comparada con el recorrido de nodos
- las entidades tienen una gran cantidad de atributos o datos anidados; en este caso, es mejor almacenar entidades sencillas en un dispositivo de almacenamiento de tipo grafo, al tiempo que el resto de los datos de atributos son almacenados en un dispositivo de almacenamiento NoSQL que no sea de tipo grafo
- se requiere almacenamiento binario; de otro modo las consultas basadas en la selección de atributos de nodo/borde dominan las consultas de recorrido de nodos

Algunos ejemplos incluyen Neo4J, Infinite Graph y OrientDB.

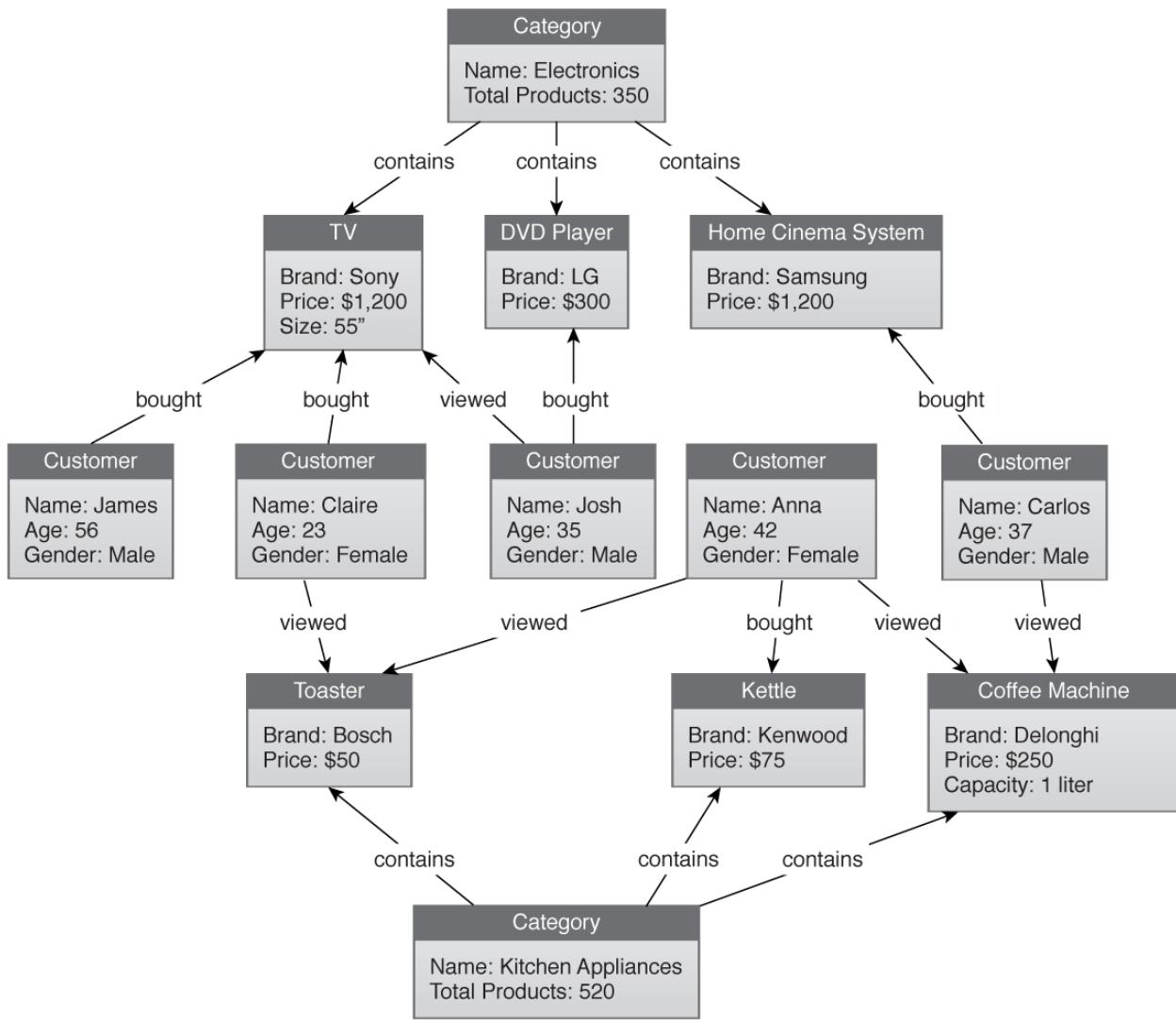


Figura 7.39 – Ejemplo de un dispositivo de almacenamiento de tipo grafo que muestra las relaciones entre las distintas entidades.

## Lectura

En la sección *Bases de datos de tipo grafo*, en las páginas 111 a 121 del libro *Resumen de NoSQL* incluido en este módulo, encontrará un análisis más detallado sobre los dispositivos de almacenamiento de tipo grafo NoSQL.

## NewSQL

Los dispositivos de almacenamiento de tipo NoSQL cuentan con alta escalabilidad, disponibilidad, tolerancia a errores y son muy rápidos para las operaciones de lectura y escritura. Sin embargo, no proporcionan la misma compatibilidad de transacciones y consistencia que los RDBMS compatibles con ACID. Siguiendo el modelo BASE, los dispositivos de almacenamiento NoSQL proporcionan una consistencia a largo plazo en lugar de inmediata, y por lo tanto podrían estar en una etapa de inconsistencia hasta que alcancen el estado de consistencia. Como resultado, no pueden ser utilizados para implementar sistemas de transacciones a gran escala.

Los dispositivos de almacenamiento de tipo NewSQL **combinan las propiedades de ACID de las RDBMS con la escalabilidad y la tolerancia a errores que ofrecen los dispositivos de almacenamiento NoSQL**. Estas bases de datos generalmente son compatibles con la sintaxis compatible con SQL para las operaciones de definición y manipulación de datos, y a menudo usan un modelo de datos relacionales para el almacenamiento de datos.

Las bases de datos de tipo NewSQL pueden ser utilizadas para desarrollar sistemas OLTP con grandes volúmenes de transacciones; por ejemplo, un banco. También pueden ser utilizadas en los análisis en tiempo real; por ejemplo, análisis operativos, ya que algunas implementaciones están basadas en memoria.

Comparado con un dispositivo de almacenamiento de tipo NoSQL, un dispositivo de almacenamiento de tipo NewSQL **permite una transición más fácil de un RDBMS tradicional a una base de datos altamente escalable, debido a que es compatible con SQL**. Algunos ejemplos incluyen VoltDB, FoundationDB, NuoDB e InnoDB.

## Ejercicio 7.2: seleccione el dispositivo de almacenamiento correcto

Identifique correctamente cuál sería el tipo de dispositivo de almacenamiento ideal en cada una de las siguientes oraciones (tenga en cuenta que no todos los dispositivos de almacenamiento son utilizados):

- en disco
  - en memoria
  - sistema de archivos distribuido
  - RDBMS
  - llave-valor (key-value)
  - documento
  - basado en columnas
  - grafo
1. Jane necesita almacenar una gran cantidad de archivos del circuito cerrado de televisión. Debido a la baja calidad del video, todos los archivos serán procesados, uno después de otro, utilizando una biblioteca de software de mejoramiento de video. ¿Qué tipo de dispositivo de almacenamiento puede utilizar Jane para garantizar la máxima capacidad de procesamiento de lectura y un procesamiento rápido de los archivos de video?

---
  2. Kerry está diseñando una aplicación de Big Data que debe almacenar grandes cantidades de archivos XML. Cada archivo XML representa una entidad aparte, compuesta por múltiples secciones, cada una con subcampos. Se deben recuperar y actualizar distintas secciones del archivo XML como parte del flujo de trabajo (Workflow) de procesamiento. ¿Qué tipo de dispositivo de almacenamiento NoSQL es el más adecuado para los requisitos de almacenamiento de datos de Kerry?

---
  3. Roger planea reemplazar la base de datos relacional con una base de datos NoSQL para almacenar los datos de sesión de los usuarios de una popular tienda online. Las sesiones de los usuarios son identificadas por medio de la ID y de una marca de hora, y almacenan los datos específicos de la aplicación, que son exclusivamente anexados. Los datos de sesión de los usuarios son analizados, lo que requiere que los datos de la sesión de cada uno de los usuarios sean agrupados. ¿Qué tipo de dispositivo de almacenamiento NoSQL puede ser utilizado en este caso?

- 
4. Mike está a cargo del diseño de una base de datos para almacenar activos físicos que se encuentran distribuidos geográficamente por el país. Cada activo tiene un conjunto básico de atributos, como ID, tipo y fecha de fabricación. Asimismo, cada activo está conectado físicamente con varios otros activos. Mike conoció que la base de datos será usada intensamente por los ingenieros con el fin de hallar activos que están conectados con otros, así como para determinar la distancia entre dos activos. ¿Qué tipo de dispositivo de almacenamiento puede usar Mike para satisfacer los requisitos de consulta de los ingenieros?
- 
5. John está diseñando una aplicación web que almacena distintas piezas de información relacionadas con cada cliente, como la información personal del cliente (incluyendo dirección e información de tarjeta de crédito), el historial de compras del cliente y comentarios publicados en la página web para diferentes productos. John quiere ser capaz de buscar clientes usando sus nombres para poder actualizar los registros de los clientes. Igualmente, él espera que se ejecuten varios tipos de análisis de los clientes. Uno de los análisis que determina los sentimientos de los clientes requiere acceso interactivo a texto en los comentarios, además de buscar los comentarios hechos por cada cliente. ¿Qué tipo de base de datos NoSQL puede utilizar John para facilitar el acceso a cada campo y permitir que los datos de los comentarios sean recuperados rápidamente?
- 

*Las respuestas al ejercicio se encuentran al final de este cuadernillo.*

## Notas

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---





## **Notas / Bocetos**

# Características del motor de procesamiento de Big Data

Procesar datasets de Big Data no es lo mismo que procesar small data, debido principalmente a la introducción de formatos de datos semiestructurados y sin estructurar, y a la gran cantidad de datos involucrados. Small data hace referencia a los datos que están estructurados en su mayoría —si no completamente— y que son procesados por las aplicaciones empresariales y almacenados en bases de datos relacionales.

Debido a las características únicas de Big Data, el procesamiento de datasets de Big Data establece determinados requisitos para los motores de procesamiento. Esta sección aborda algunas de las características fundamentales que son esenciales para formular un conjunto tecnológico y elegir los frameworks de software adecuados para el procesamiento de Big Data.

A continuación se presentan las siguientes características del motor de procesamiento:

- procesamiento de datos distribuidos/paralelos
- procesamiento de datos sin esquema
- soporte para múltiples cargas de trabajo
- escalabilidad lineal
- redundancia y tolerancia a errores
- bajo costo

## Procesamiento de datos distribuidos/paralelos

Debido a la característica de volumen de Big Data y tal como se estableció en la sección *Dispositivos de almacenamiento en disco*, los datasets de Big Data **generalmente se almacenan haciendo uso de las tecnologías distribuidas** (sistema de archivos distribuido o base de datos NoSQL).

Las características propias de un dispositivo de almacenamiento distribuido requieren un motor de procesamiento que pueda procesar datos sin necesidad de transferirlos desde su lugar de almacenamiento hacia un recurso de informática, como sí sucede con el procesamiento de datos distribuidos. Con el fin de maximizar la característica de valor de Big Data, es necesario utilizar un modelo de procesamiento basado en el **principio de dividir un problema difícil en tantas partes como sea necesario**, al igual que sucede con el procesamiento de datos paralelos.

## Procesamiento de datos sin esquema

Los datasets de Big Data se presentan en múltiples formatos (característica de variedad) los cuales podrían no hacer parte de ningún esquema, especialmente en el caso de los datos sin estructurar. Los esquemas pueden cambiar con el tiempo, con el fin de acomodarse a las necesidades empresariales cambiantes, o simplemente debido a una actualización de software.

De la misma forma, es posible que en el futuro se deba incluir un mayor número de fuentes de datos con esquemas desconocidos.

Al no hacer parte de ningún modelo de datos en particular, los datasets de Big Data requieren un procesamiento flexible para que puedan ser procesados sin necesidad de ser almacenados en un modelo de datos en particular cuando están en su estado sin procesar.

## **Soporte para múltiples cargas de trabajo**

Los datasets de Big Data se presentan en dos formas; **gruesos** (característica de volumen) y/o **rápidos** (característica de velocidad). Muchas veces, es posible procesar los datos offline y por lotes (Batch Processing), como sucede con la generación de reportes urgentes. En otros casos, los resultados pueden solicitarse en tiempo real, como con el procesamiento de una señal de GPS. En estos casos se generan cargas de trabajo con procesamientos opuestos: transaccional y por lotes (Batch Processing).

Para lograr el máximo valor de los datasets de Big Data, **es necesario que la plataforma de procesamiento subyacente sea compatible tanto con las cargas de trabajo transaccionales como con las de lote**. Es posible lograrlo con un solo motor de procesamiento, o también pueden ser necesarios varios motores de procesamiento. A pesar de que, para obtener el máximo valor de los datasets de Big Data es ideal tener la capacidad de procesar datos en tiempo real (transaccional) y por lotes (Batch Processing), puede que no sea necesario o no se pueda proporcionar compatibilidad para ambas modalidades.

Generalmente, el montaje de un entorno de solución de procesamiento por lotes (Batch Processing) de Big Data es más sencillo y más económico que el montaje de una solución de procesamiento en tiempo real de Big Data. Por esto, aumentar la compatibilidad para el procesamiento de múltiples cargas de trabajo debe estar determinado por un objetivo empresarial, e involucrar análisis cuidadosos de rentabilidad.

## **Escalabilidad lineal**

Debido a las características de volumen y velocidad de Big Data, la demanda de procesamiento puede aumentar considerablemente con mayores volúmenes de datos que son recibidos rápidamente. Dar soporte a un entorno de procesamiento distribuido con capacidades de procesamiento paralelo requiere un motor de procesamiento que pueda ofrecer un rendimiento estable mientras el volumen de datos crece.

El procesamiento de datasets de Big Data requiere un **motor de procesamiento con alta escalabilidad que pueda ser escalado linealmente**. En el contexto del procesamiento, la escalabilidad lineal significa que al añadir más nodos de procesamiento se obtiene un incremento proporcional en el rendimiento.

La Inteligencia de negocios (BI) en tiempo real y la analítica pueden aprovechar dicho entorno de procesamiento escalable linealmente para entregar respuestas más rápidas que comprendan operaciones complejas en un dataset completo. La escalabilidad lineal se logra, generalmente, **usando una estrategia de escalabilidad horizontal, puesto que es un método sencillo, ininterrumpido y económico para aumentar la capacidad de procesamiento**.

## **Redundancia y tolerancia a errores**

Un entorno de procesamiento de datos altamente distribuidos con capacidades de procesamiento de datos paralelos generalmente involucra una arquitectura complicada. Con una arquitectura de procesamiento de escalabilidad horizontal —que por defecto involucra un gran número de nodos y componentes de red—, aumentan las posibilidades de una falla parcial del sistema.

Puesto que una falla del sistema en medio de una tarea distribuida de larga duración sería perjudicial para el cumplimiento de los objetivos analíticos, **un motor de procesamiento debe ofrecer tolerancia a errores, para que así una falla parcial no inhabilite todo el sistema y no haya la necesidad de que el procesamiento de datos tenga que comenzar nuevamente desde el principio.**

La tolerancia a errores generalmente es proporcionada por medio de la redundancia. Si se cuenta con recursos de procesamiento redundantes, el sistema puede seguir funcionando incluso si ocurre una falla parcial. La escalabilidad horizontal es bastante útil en este caso, pues la redundancia generalmente se puede aumentar simplemente añadiendo más recursos de procesamiento.

## **Bajo costo**

Al inicio de una iniciativa de Big Data, es posible que el costo de la implementación de un entorno de procesamiento distribuido altamente escalable, que involucre unos cuantos nodos y equipo de red, no sea tan alto. Sin embargo, con el tiempo, al aumentar el volumen de datos y los tipos y frecuencia de la analítica que es ejecutada, **la necesidad de un mayor número de recursos de procesamiento se puede traducir en altos costos de TI**, lo que involucra tanto software como hardware. Esto puede ser contraproducente para el propósito original de la iniciativa de Big Data, el cual en la mayoría de los casos es ayudar a la empresa a entregar un mayor valor, disminuir costos, encontrar nuevas fuentes de ingresos o establecer nuevas ofertas de servicios.

Por esta razón, es muy importante utilizar un motor de procesamiento de bajo costo, cuyos costos puedan reducirse a un mínimo a medida que aumenta la demanda por recursos de procesamiento. Implementar **un software de código abierto en un hardware básico** ayuda a minimizar los costos.

Otro aspecto relacionado con mantener los costos bajos es **la capacidad del motor de procesamiento para aprovechar el Cloud Computing**. La naturaleza flexible y por demanda de Cloud Computing ayuda a evitar cualquier inversión directa de capital, sumada a una configuración más rápida del entorno de solución de procesamiento de datos.

### Ejercicio 7.3: encuentre el término correspondiente para cada enunciado

Responda las siguientes preguntas completando los espacios en blanco con uno de los términos a continuación:

- procesamiento de datos sin esquema
  - soporte para múltiples cargas de trabajo
  - bajo costo
  - redundancia y tolerancia a errores
  - procesamiento de datos distribuidos/paralelos
  - escalabilidad lineal
1. ¿Qué característica del motor de procesamiento requiere rendimiento estable frente a un gran volumen de datos recibidos a alta velocidad?  
\_\_\_\_\_
  2. ¿Qué característica del motor de procesamiento da soporte a los modelos de datos cambiantes y permite el procesamiento de datos en su forma original sin la necesidad de llevar a cabo ninguna transformación en el modelo de datos?  
\_\_\_\_\_
  3. ¿Qué característica del motor de procesamiento fomenta el uso de software de código abierto y Cloud Computing?  
\_\_\_\_\_
  4. ¿Qué característica del motor de procesamiento permite el procesamiento de datos tanto por lotes (Batch Processing) como en tiempo real (transaccional)?  
\_\_\_\_\_

5. ¿Qué característica del motor de procesamiento permite que haya funcionalidad cuando se presentan fallas en el sistema?

---

6. ¿Qué característica del motor de procesamiento permite el procesamiento de grandes cantidades de datos en la fuente sin necesidad de transferirlos desde su lugar de almacenamiento hasta un recurso de informática?

---

*Las respuestas al ejercicio se encuentran al final de este cuadernillo.*

## Notas

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---





## **Notas / Bocetos**

# Procesamiento fundamental de Big Data

Una mirada rápida a las características del motor de procesamiento indica que la mayoría de ellas son funciones de la arquitectura informática subyacente sobre la cual se basa la plataforma de Big Data resultante. El procesamiento de grandes cantidades de datos no es un fenómeno nuevo, y existen diferentes arquitecturas de procesamiento de datos a gran escala. **El procesamiento de Big Data requiere un entorno distribuido que sea capaz de procesar datos en paralelo, lo cual es una característica soportada por la arquitectura en cluster.**

## Procesamiento de Big Data: cluster

Como se presentó en el Módulo 2, un cluster es un grupo de nodos conectados entre sí por una red que procesa tareas en paralelo. Un cluster es una red de nodos centralizada en la que cada nodo es responsable de una subtarea de un problema mayor (Figura 7.40). **Los clusters permiten el procesamiento de datos distribuidos.** Idealmente, un cluster comprende nodos básicos de bajo costo que en conjunto ofrecen una capacidad de procesamiento mayor con una redundancia y tolerancia a errores integrada, pues está compuesto por nodos separados físicamente.

Los clusters son **altamente escalables**, y dan soporte a la escalabilidad horizontal con un mejor rendimiento lineal. Además, ofrecen una implementación ideal del entorno para un motor de procesamiento, puesto que los grandes datasets se pueden dividir en datasets más pequeños y luego ser procesados paralelamente de forma distribuida.

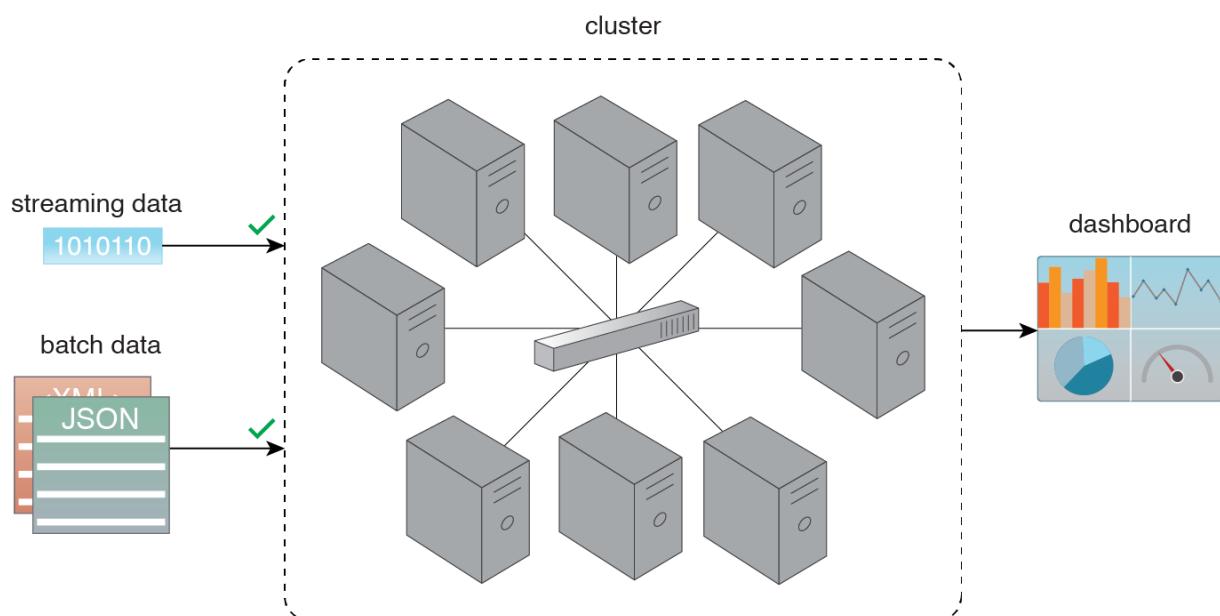


Figura 7.40 - Un cluster puede ser utilizado tanto por un motor de procesamiento en tiempo real como por un motor de procesamiento por lotes (Batch Processing), tales como Spark y MapReduce, respectivamente.

En la sección anterior, se abordó un conjunto de características del motor de procesamiento, y previamente en esta sección, se había establecido que es necesario que un entorno de procesamiento de Big Data esté basado en una arquitectura en cluster que soporte la escalabilidad lineal al procesamiento de datos distribuidos/paralelos. A continuación se explica cómo se usa la arquitectura en cluster para el procesamiento de Big Data. En la arquitectura en cluster, los datasets de Big Data pueden ser procesados **por lotes** o **en tiempo real** usando un **motor de procesamiento por lotes (Batch Processing)** o un **motor de procesamiento en tiempo real**, respectivamente.

### **Procesamiento de Big Data: modo por lotes (Batch Processing)**

En el modo por lotes, **los datos se procesan offline en lotes donde el tiempo de respuesta puede ser de minutos u horas**. Los datos primero son guardados en el disco, y luego son procesados. El modo por lotes generalmente involucra el procesamiento de varios datasets grandes, ya sea individualmente o combinados, enfocándose principalmente en las características de volumen y variedad de los datasets de Big Data.

La mayoría del procesamiento de Big Data se hace por lotes (Batch Processing). Es relativamente sencillo, fácil de configurar y tiene menor costo comparado con el modo en tiempo real. La Inteligencia de negocios (BI) estratégica, la analítica predictiva/prescriptiva y las operaciones de ETL generalmente hacen uso del modo por lotes.

### **Procesamiento de Big Data: modo en tiempo real**

En el modo en tiempo real, **los datos son procesados en la memoria al momento de ser capturados y antes de ser guardados en el disco**. El tiempo de respuesta generalmente varía entre unos pocos segundos hasta poco menos de un minuto. El modo en tiempo real se enfoca en la característica de velocidad de los datasets de Big Data.

Al hablar de procesamiento de Big Data, el procesamiento en tiempo real también se denomina procesamiento de flujo de eventos (ESP), pues los datos llegan continuamente (flujo) o por intervalos (eventos). Los datos del evento o flujo generalmente son pequeños, pero su naturaleza continua hace que el resultado sean datasets muy grandes.

Otro término relacionado, el “**modo interactivo**”, se ubica dentro de la categoría de tiempo real. El modo interactivo generalmente se refiere al procesamiento de consultas en tiempo real. La Inteligencia de negocios (BI) y la analítica operativa generalmente hacen uso del modo en tiempo real.

#### **NOTA**

Es importante tener en cuenta que, aunque las dos modalidades de procesamiento aquí presentadas están basadas en un contexto donde los datos primero son guardados o no en el disco, el simple hecho de que los datos estén almacenados en el disco no significa que no puedan ser procesados en tiempo real o prácticamente en

tiempo real.

El procesamiento de datos, luego de que estos hayan sido guardados en el disco, también puede hacer referencia al *procesamiento de consultas*, que es un modelo de procesamiento caracterizado por el tiempo de respuesta.

El procesamiento de consultas por lotes hace referencia a las consultas que no ofrecen un resultado inmediato. Por otra parte, el procesamiento de consultas en tiempo real hace referencia a las consultas que generan un resultado inmediato.

## Notas

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

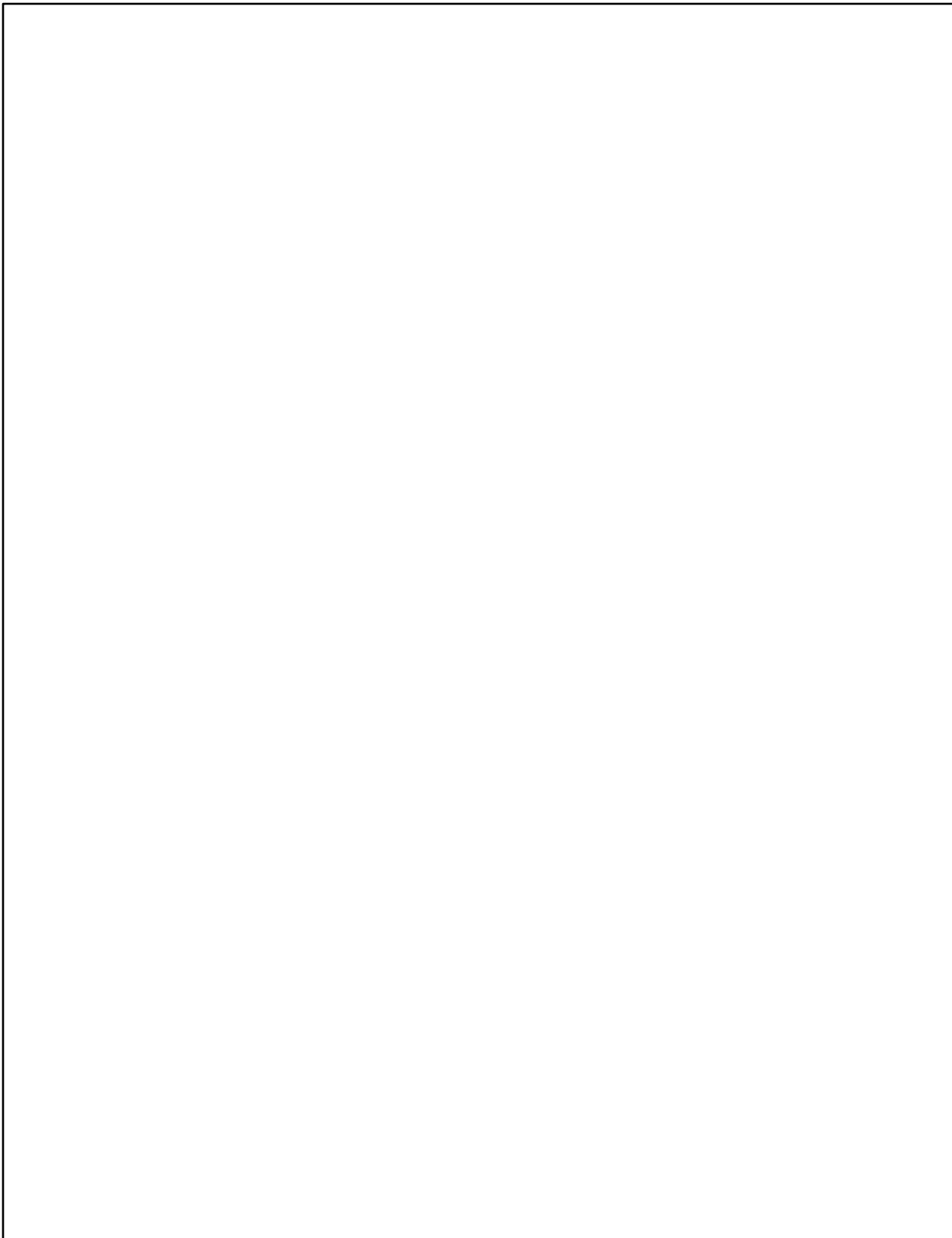
---

---





## **Notas / Bocetos**



## Presentación del motor de procesamiento de MapReduce

MapReduce es una implementación muy utilizada del mecanismo de motor de procesamiento por lotes (Batch Processing). Es un **motor de procesamiento altamente confiable y escalable basado en el principio de dividir un problema difícil en tantas partes como sea necesario**, con tolerancia a errores y redundancia integrados. Este motor divide un problema muy grande en un conjunto de problemas más pequeños que son más fáciles y rápidos de solucionar, y está basado en la informática distribuida y paralela. MapReduce es un **motor de procesamiento enfocado en el modo por lotes que se usa para procesar grandes datasets por medio del procesamiento paralelo implementado en clusters de hardware básico**.



Figura 7.41 – Símbolo utilizado para representar un motor de procesamiento.

MapReduce **no requiere que los datos de entrada hagan parte de ningún modelo de datos en particular**. Por lo tanto, puede ser usado para procesar datasets sin esquema. Un dataset se divide en varias partes más pequeñas, y en cada parte se realizan operaciones de forma independiente y paralela. Los resultados de todas las operaciones son combinados para llegar a una respuesta. Debido a la sobrecarga de coordinación involucrada, el motor de procesamiento MapReduce **generalmente solo soporta cargas de trabajo por lotes**. MapReduce se basa en el trabajo de investigación hecho por Google sobre este tema que fue publicado a principios del año 2000.

El motor de procesamiento MapReduce funciona de forma diferente al paradigma de procesamiento distribuido tradicional. Tradicionalmente, el procesamiento de datos requiere que estos pasen del nodo de almacenamiento al nodo de procesamiento, donde se ejecuta el algoritmo de procesamiento como tal, lo que funciona bastante bien para datasets pequeños. Sin embargo, **en el caso de los datasets más grandes, mover los datos generalmente implica una sobrecarga mayor que el procesamiento de los datos como tal**.

Con MapReduce, es el algoritmo de procesamiento de datos el que se transfiere hacia los nodos que almacenan los datos. El algoritmo de procesamiento de datos es ejecutado en paralelo en estos nodos, **eliminando la necesidad de transferir primero los datos**. Esto no solo ahorra ancho de banda de la red, sino que además genera una gran reducción en el tiempo de procesamiento de grandes datasets, puesto que procesar grupos más pequeños de datos en paralelo es mucho más rápido.

## Conceptos de MapReduce

Una sola ejecución de procesamiento del motor de procesamiento MapReduce se conoce como un trabajo de MapReduce. Cada trabajo de MapReduce está compuesto por una tarea de mapeo y una de reduce. Cada tarea está compuesta por diferentes etapas:

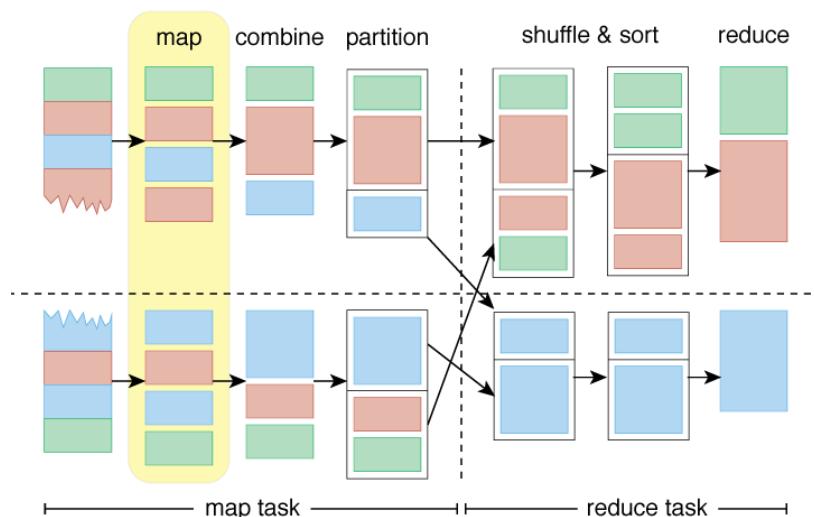
### Tarea de mapeo

- **Mapear**
- **Combinar** (opcional)
- **Dividir**

### Tarea de reduce

- **Mezclar y clasificar**
- **Reducir**

## MapReduce: mapear



La primera etapa de MapReduce se conoce como mapeo: en esta etapa, **el archivo del dataset se divide en varias partes más pequeñas**. Cada parte se analiza desde sus registros de constitución como un par llave-valor (key-value). La llave generalmente es la posición ordinal del registro, y el valor es el registro como tal. Por ejemplo, (234 el cielo es azul).

Posteriormente, los pares llave-valor (key-value) analizados de cada división son enviados a una **función de mapeo** o mapeador, con una función de mapeo por división. La función de mapeo funciona con una lógica definida por el usuario. Generalmente, cada división contiene múltiples pares llave-valor (key-value) y el mapeador es ejecutado una vez para cada par llave-valor (key-value) en la división.

El mapeador procesa cada par llave-valor (key-value) según la lógica definida por el usuario y además genera como resultado un par llave-valor (key-value). La **llave de salida** puede ser la misma que la llave de entrada o un valor de subcadena del valor de entrada, u otro objeto serializable definido por el usuario. Asimismo, el **valor de salida** puede ser el mismo que el valor de entrada o un valor de subcadena del valor de entrada, u otro objeto serializable definido por el usuario.

Cuando se hayan procesado todos los registros de la división, el resultado es una lista de pares llave-valor (key-value) en los que pueden existir varios pares llave-valor (key-value) para la misma llave. Se debe tener en cuenta que para un par llave-valor (key-value) de entrada, es posible que un mapeador no genere ningún par llave-valor (key-value) de salida (filtrado (filtering)), o que genere varios pares llave-valor (key-value) (demultiplexing). La etapa de mapeo se puede resumir con la ecuación descrita en la Figura 7.42.

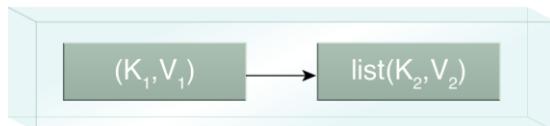
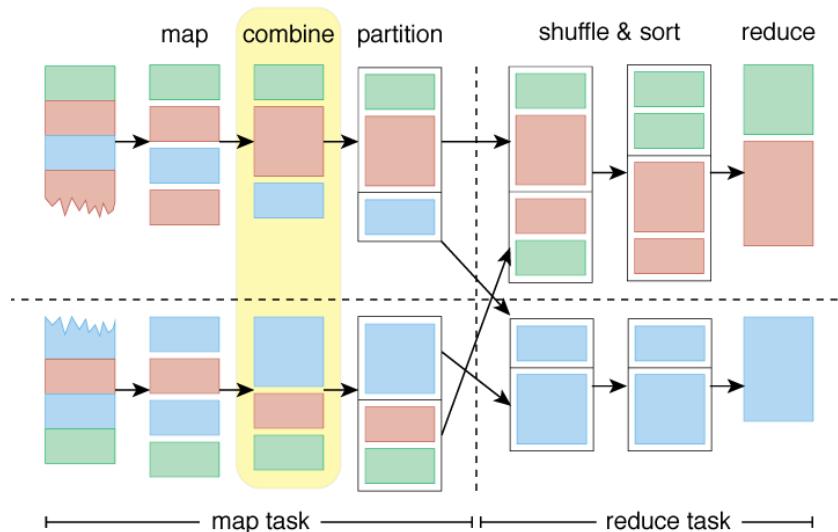


Figura 7.42 - Resumen de la etapa de mapeo.

### MapReduce: combinar



Generalmente, el resultado de la función de mapeo se maneja directamente por medio de la función reduce. Sin embargo, **las tareas de mapeo y reduce generalmente son ejecutadas en distintos nodos**. Esto requiere que los datos se muevan entre mapeadores y reductores, lo cual puede consumir una cantidad importante de ancho de banda y contribuye directamente con los tiempos de latencia del procesamiento.

Con datasets más grandes, el tiempo empleado para mover los datos entre las etapas de mapeo y reduce puede sobrepasar el tiempo efectivo de procesamiento de las tareas de mapeo y reduce. Por esta razón, el motor MapReduce ofrece una **función opcional de combinación** (combinador) que **resume los resultados del mapeador antes de que sean procesados por el reductor**.

Un combinador es básicamente una función reduce que agrupa localmente los resultados del mapeador en el mismo nodo que este. La función reduce se puede usar como una función de combinación, o se puede usar una función definida por el usuario.

El motor MapReduce combina todos los valores de una llave determinada desde el resultado del mapeador, creando varios pares llave-valor (key-value) como entradas para el combinador, donde la llave no se repite y el valor existe como una lista de todos los valores correspondientes para dicha llave. **La etapa de combinación solo es una etapa de mejora, y por consiguiente, es posible que el motor MapReduce no haga uso de ella.**

Cabe señalar que un combinador debe **únicamente ser especificado cuando su uso no afecte el resultado neto**, pues realizar una acción en una parte del grupo puede generar resultados diferentes a que si se realizara la misma acción en todo el grupo. Por ejemplo, una función de combinación encontrará el número mayor o menor, pero no encontrará el promedio de todos los números. La etapa de combinación se puede resumir con la ecuación descrita en la Figura 7.43.

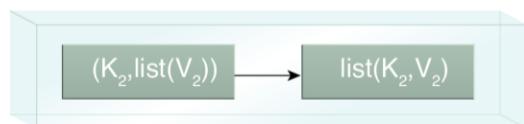
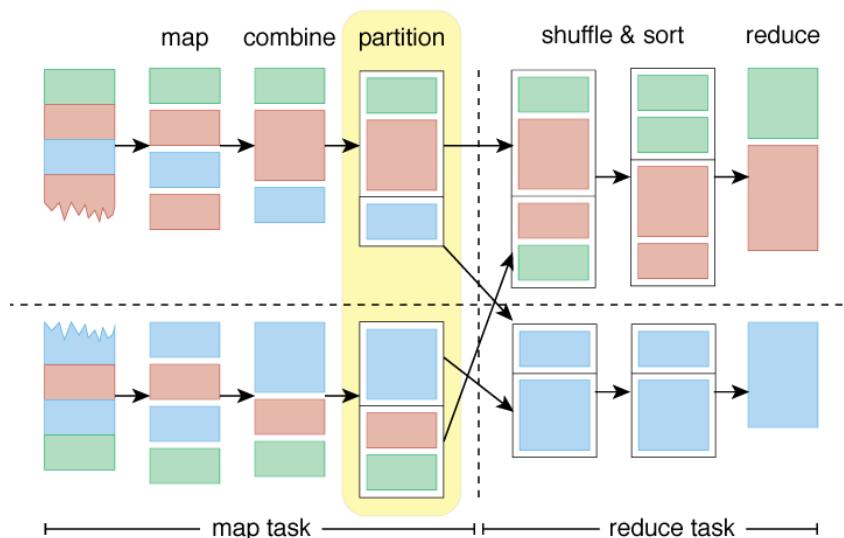


Figura 7.43 - Resumen de la etapa de combinación.

## MapReduce: dividir



En esta etapa, si hay más de un reductor involucrado, un particionador divide el resultado del mapeador o combinador (si este ha sido especificado y usado por el motor de MapReduce) en particiones entre las instancias de reductores. **El número de particiones es igual al número de reductores.**

Aunque cada partición contiene diferentes pares llave-valor (key-value), todos los registros de una llave en particular están dentro de la misma partición. El motor MapReduce garantiza una distribución aleatoria y proporcional entre los reductores, mientras se asegura de que todas las llaves iguales de varios mapeadores estén en la misma instancia de reductor.

Dependiendo de la naturaleza del trabajo, algunos reductores pueden recibir una mayor cantidad de pares llave-valor (key-value) que otros. Como resultado de esta carga de trabajo irregular, algunos reductores finalizarán antes que otros. En general, esto es menos eficiente y genera mayores tiempos de ejecución de trabajo que si el trabajo hubiera sido distribuido uniformemente entre los reductores. Esto se puede corregir personalizando la lógica de división, para así garantizar una distribución equitativa de los pares llave-valor (key-value).

La función de partición es la última etapa de la tarea de mapeo, en la que se genera el índice del reductor al cual se debe enviar una partición en particular. La etapa de división se puede resumir con la ecuación descrita en la Figura 7.44.

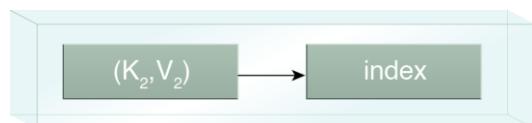
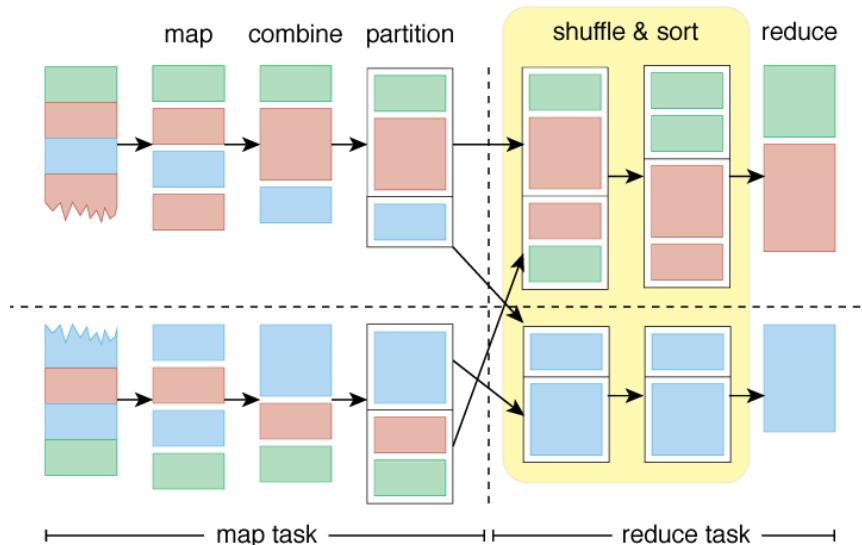


Figura 7.44 - Resumen de la etapa de división.

### MapReduce: mezclar y clasificar



Durante la primera etapa de la tarea de reduce, los resultados de todos los particionadores se copian desde la red a los nodos, ejecutando la tarea de reduce. Esto se conoce como **mezcla**. La lista de llave-valor (key-value) de salida de cada particionador puede contener las mismas llaves varias veces.

Posteriormente, el motor de MapReduce **agrupa y clasifica automáticamente los pares llave-valor (key-value) de acuerdo a las llaves** para que el resultado incluya una lista clasificada de todas las llaves de entrada y que sus valores **aparezcan con las mismas llaves correspondientes**. Es posible personalizar la forma en la que las llaves son agrupadas y clasificadas.

Después, el motor de MapReduce combina cada grupo de llaves antes de que la función reduce procese el resultado de la mezcla y clasificación. Esta combinación crea un par llave-valor (key-value) único por grupo, en el cual la llave es la llave del grupo y el valor es la lista de todos los valores del grupo. Esta etapa se puede resumir con la ecuación descrita en la Figura 7.45.

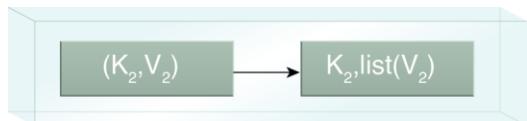
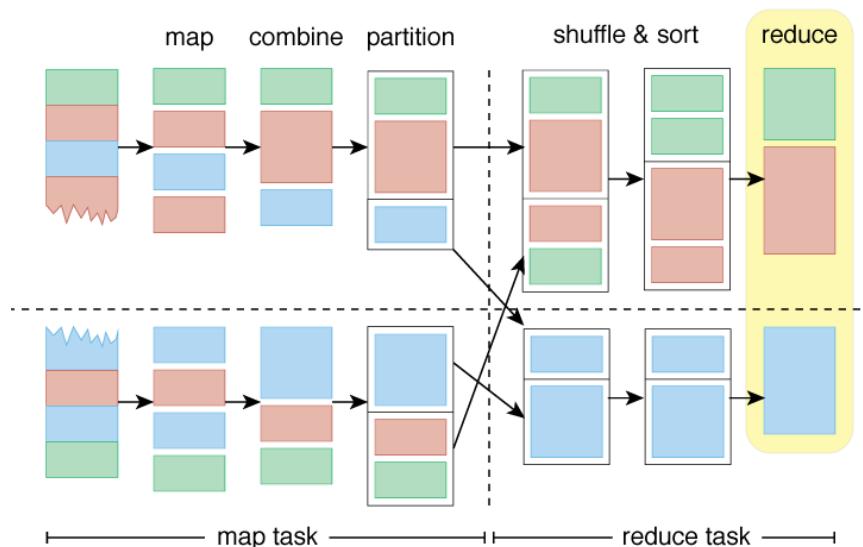


Figura 7.45 - Resumen de la etapa de mezcla y clasificación.

## MapReduce: reducir



La etapa final de la tarea de reduce es la reducción. Dependiendo de la lógica definida por el usuario y que haya sido especificada en la función reduce (reductor), **el reductor resumirá aún más la entrada o generará un resultado sin hacer ningún cambio**. Por consiguiente, para cada par llave-valor (key-value) que es recibido por un reductor, se procesa la lista de valores almacenada en la parte de valor del par y se escribe un nuevo par llave-valor (key-value).

La llave de salida puede ser la misma que la llave de entrada o un valor de subcadena del valor de entrada, u otro objeto serializable definido por el usuario. El valor de salida puede ser el mismo que el valor de entrada o un valor de subcadena del valor de entrada, u otro objeto serializable definido por el usuario.

Cabe señalar que, al igual que un mapeador, para un par llave-valor (key-value) de entrada, un reductor puede no generar ningún par llave-valor (key-value) de salida (filtrado (filtering)), o puede generar varios pares llave-valor (key-value) (demultiplexing). **Los pares llave-valor (key-value) generados por un reductor se combinan para formar el resultado final.**

value) son el resultado del reductor, y se escriben posteriormente como un archivo separado — un archivo por reductor.

Para ver el resultado completo del trabajo de MapReduce, se deben combinar todas las partes del archivo. Es posible personalizar la cantidad de reductores. También es posible que haya un trabajo de MapReduce sin ningún reductor, por ejemplo cuando se realiza el filtrado (filtering).

Tenga en cuenta que la firma de salida (tipos llave-valor (key-value)) de la función de mapeo debe coincidir con la firma de entrada (tipos llave-valor (key-value)) de la función reduce/de combinación. La etapa de reduce se puede resumir con la ecuación descrita en la Figura 7.46.

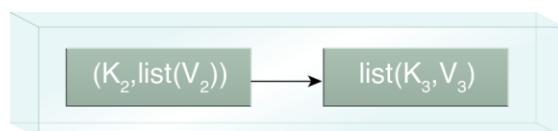


Figura 7.46 - Resumen de la etapa de reduce.

## MapReduce: ejemplo

La Figura 7.47 ilustra los siguientes pasos:

1. a, b. La entrada (sales.txt) es dividida en dos partes.
2. a, b. Dos tareas de mapeo que se ejecutan en dos nodos diferentes (Nodo A y Nodo B) extraen los productos y la cantidad de los correspondientes registros de división en paralelo. El resultado de cada función de mapeo es un par llave-valor (key-value) en el cual el producto es la llave, mientras que la cantidad es el valor.
3. a, b. El combinador luego realiza una suma a nivel local de las cantidades de los productos.
4. a, b. Como solo hay una tarea de reduce, no se realiza partición.
5. El resultado de ambas tareas de mapeo es copiado a un tercer nodo (Nodo C) que ejecuta la etapa de mezcla como parte de la tarea de reduce.
6. La etapa de clasificación luego agrupa todas las cantidades del mismo producto como una sola lista.
7. Al igual que el combinador, la función reduce luego suma las cantidades de cada producto único para crear un resultado.

NOTA
Tenga en cuenta que algunos dispositivos de almacenamiento NoSQL ofrecen por defecto compatibilidad con MapReduce para el procesamiento por lotes (Batch Processing). Aprovechando la implementación en cluster de los

dispositivos de almacenamiento NoSQL, el motor de procesamiento MapReduce distribuye el procesamiento de solicitudes a través de varios nodos.

Esto se logra generalmente mediante el suministro de modelos de mapeo y reduce que el usuario debe implementar mediante la API correspondiente de los dispositivos de almacenamiento NoSQL.

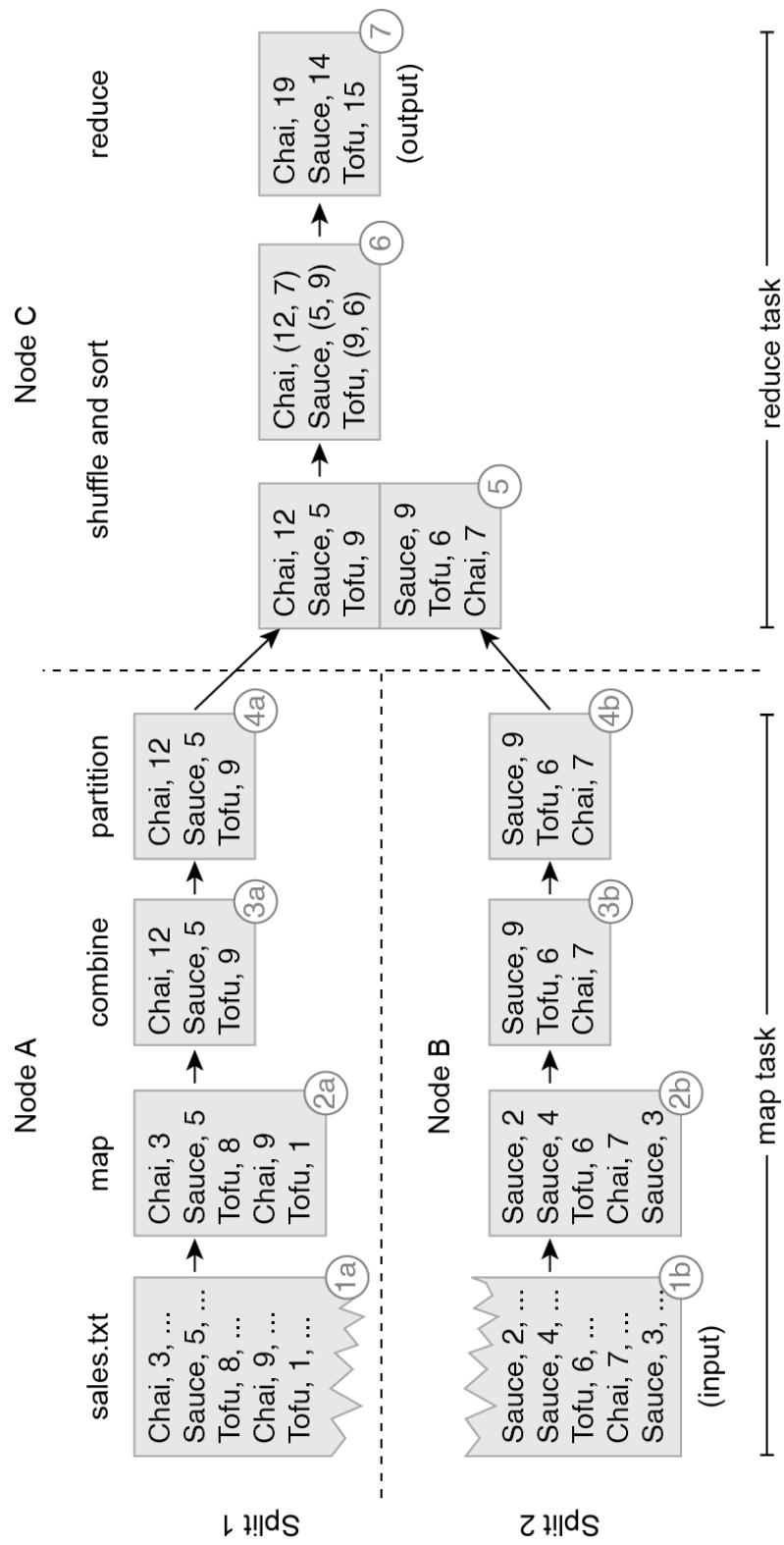


Figura 7.47 – Ejemplo de MapReduce.

### Ejercicio 7.4: organice las etapas de MapReduce

Organice las etapas de MapReduce a continuación en el orden correcto:

mezclar y clasificar

1. \_\_\_\_\_

dividir

2. \_\_\_\_\_

mapear

3. \_\_\_\_\_

reducir

4. \_\_\_\_\_

combinar

5. \_\_\_\_\_

*Las respuestas al ejercicio se encuentran al final de este cuadernillo.*

## Notas

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---





## **Notas / Bocetos**

# Diseño fundamental del algoritmo de MapReduce

A diferencia de los modelos de programación tradicionales, MapReduce sigue un modelo de programación distinto. Para entender cómo se pueden diseñar o adaptar los algoritmos según el modelo de programación de MapReduce, primero debemos explicar su principio de diseño.

Cómo se explicó antes, MapReduce trabaja con base en el principio de dividir un problema difícil en tantas partes como sea necesario. Sin embargo, es importante entender la semántica de este principio en el contexto de MapReduce. El principio de dividir un problema difícil en tantas partes como sea necesario se puede alcanzar generalmente al usar uno de los siguientes enfoques:

- paralelismo de tareas
- paralelismo de datos

## Paralelismo de tareas

El paralelismo de tareas se refiere a la **paralelización del procesamiento de datos, al dividir una tarea en subtareas y ejecutar cada subtarea en un procesador por separado**, generalmente en un nodo separado dentro de un cluster (Figura 7.48). Cada subtarea generalmente ejecuta un **algoritmo diferente** usando su propia copia de los mismos o distintos datos como entrada, en paralelo. Generalmente, el resultado de las diferentes subtareas se combina para obtener el conjunto final de resultados.

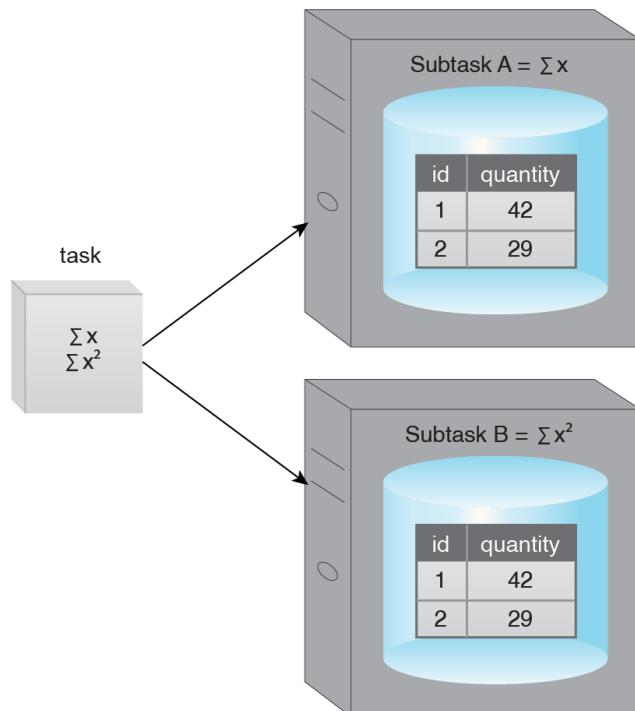


Figura 7.48 - Una tarea se divide en dos subtareas, Subtarea A y Subtarea B, que luego son ejecutadas en dos nodos diferentes dentro del mismo dataset.

## Paralelismo de datos

El paralelismo de datos se refiere a la **parallelización del procesamiento de datos**, al dividir un dataset en varios subdatasets y procesar cada uno de estos subdatasets en paralelo (Figura 7.49). Distintos subdatasets son distribuidos entre diferentes nodos y luego son procesados usando el **mismo algoritmo**. Generalmente, el resultado de los diferentes subdatasets procesados se combina para obtener el conjunto final de resultados.

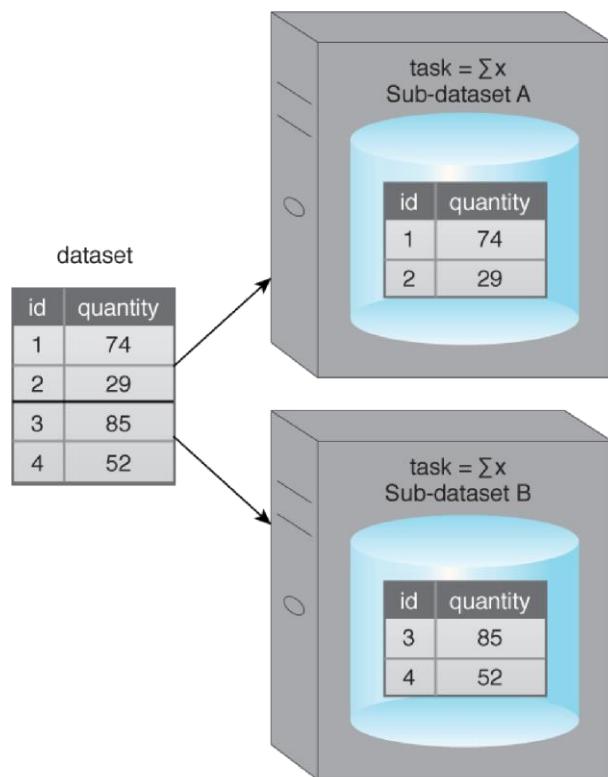


Figura 7.49 - Un dataset se divide en dos subdatasets, Subdataset A y Subdataset B, que luego son ejecutados en dos nodos diferentes usando la misma función.

## Diseño del algoritmo de MapReduce

Dentro de los entornos Big Data, en una unidad de datos generalmente se debe ejecutar varias veces la misma tarea (por ejemplo, un registro), en cuyo caso el dataset completo es distribuido entre distintas ubicaciones debido a su gran tamaño. MapReduce aborda esta necesidad al usar el **enfoque de paralelismo de datos**, en el cual los datos son divididos en varias partes, y cada división es procesada por su propia instancia de la función de mapeo, la cual contiene la misma lógica de procesamiento que las demás funciones de mapeo.

La mayor parte del desarrollo algorítmico tradicional sigue un enfoque secuencial en el que las operaciones que involucran los datos son realizadas una después de la otra, de tal manera que la operación siguiente depende de la operación que la precede.

En MapReduce, las operaciones se dividen entre las funciones mapeo y reduce. Las tareas de mapeo y reduce son independientes, y por ello, se ejecutan por separado. Además, cada instancia de una función de mapeo o reduce se ejecuta independiente de otras instancias.

Por lo general, las firmas de las funciones en el desarrollo algorítmico tradicional no están limitadas. En MapReduce, las firmas de las funciones de mapeo y reduce se limitan a un conjunto de pares llave-valor (key-value) que son la única forma en la que una función de mapeo se comunica con una función reduce. Además, la lógica en la función de mapeo depende de la forma en que se analizan los registros, lo que además depende de cómo está compuesta una unidad lógica de datos dentro del dataset.

Por ejemplo, cada línea en un archivo de texto generalmente representa un solo registro. Sin embargo, es posible que un conjunto de dos o más líneas efectivamente representen un solo registro (Figura 7.50). Además, la lógica de la función reduce depende del resultado de la función de mapeo, en especial de cuáles llaves son el resultado de la función de mapeo, pues la función reduce recibe una sola llave con una lista consolidada de sus valores. Se debe tener en cuenta que en algunos casos, por ejemplo en la extracción de textos, puede no ser necesaria la función reduce.

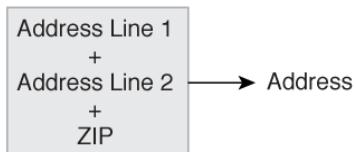


Figura 7.50 - Ejemplo de tres líneas que representan un solo registro.

### Diseño del algoritmo de MapReduce: consideraciones

Las consideraciones fundamentales para desarrollar un algoritmo de MapReduce se pueden resumir de la siguiente forma:

- Lógica algorítmica relativamente simple, de forma tal que se pueda obtener el resultado requerido aplicando la misma lógica a diferentes partes de un dataset en paralelo, y luego agregando los resultados de alguna forma.
- Disponibilidad del dataset, de forma distribuida particionada a través de un cluster, para que las distintas funciones de mapeo puedan procesar diferentes subconjuntos de dataset en paralelo.
- Comprensión de la estructura de datos dentro del dataset, para poder elegir una unidad de datos (un solo registro) importante.
- Dividir la lógica algorítmica en funciones de mapeo y reduce, para que la lógica en la función de mapeo no dependa totalmente del dataset, pues solo se dispone de los datos dentro de una división.
- Emitir la llave correcta a partir de la función de mapeo, junto con todos los datos requeridos —como los valores—, ya que la lógica de la función reduce solo puede procesar los valores

que hayan sido emitidos como parte de los pares llave-valor (key-value) de la función de mapeo.

- **Emitir la llave correcta a partir de la función reduce**, junto con los datos requeridos —como los valores—, ya que el resultado de cada función reduce se convierte en el resultado final del algoritmo de MapReduce.

### Ejercicio 7.5: complete los espacios en blanco

1. El framework de MapReduce se basa en el principio de \_\_\_\_\_.
2. El \_\_\_\_\_ y el \_\_\_\_\_ son los dos enfoques usados generalmente para lograr el principio de dividir un problema difícil en tantas partes como sea necesario.
3. El \_\_\_\_\_ se refiere a la paralelización del procesamiento de datos al dividir una tarea en subtareas y ejecutar cada subtarea en un procesador por separado, generalmente en un nodo separado dentro de un cluster.
4. El \_\_\_\_\_ se refiere a la paralelización del procesamiento de datos al dividir un dataset en varios subdatasets y procesar cada uno de estos subdatasets en paralelo.
5. El framework de MapReduce aborda la necesidad de una ejecución repetida de la misma tarea en datos distribuidos al usar el enfoque de \_\_\_\_\_.
6. En MapReduce, la lógica en la función reduce depende del resultado de la función \_\_\_\_\_.
7. El framework de MapReduce requiere que el dataset sea \_\_\_\_\_ a través del cluster para que múltiples funciones \_\_\_\_\_ puedan procesar los subdatasets en paralelo.

8. Con el algoritmo de MapReduce, la lógica en la función de mapeo no debe depender del dataset completo, pues solo se dispone de los datos dentro de una

\_\_\_\_\_.

*Las respuestas al ejercicio se encuentran al final de este cuadernillo.*

## Notas

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---





## **Notas / Bocetos**

# Respuestas a los ejercicios

## Respuestas al ejercicio 7.1.

1. **Sharding** es el proceso de particionar horizontalmente un gran dataset en un grupo de datasets más pequeños y manejables llamados **shards**, distribuidos entre múltiples nodos.
2. En el sharding, los **patrones de consulta** deben ser tomados en cuenta, de tal forma que los shards no se conviertan en cuellos de botella en términos del rendimiento.
3. La **replicación** crea múltiples copias de un dataset, conocidas como **rélicas**, y las almacena en varios nodos.
4. Los dos métodos para implementar la replicación incluyen la replicación **maestro-esclavo** y la replicación **peer-to-peer**.
5. En la replicación **maestro-esclavo**, el nodo **maestro** es el único punto de contacto para todas las operaciones de escritura, mientras que los datos pueden ser leídos desde cualquier nodo **esclavo**.
6. En la replicación **peer-to-peer** no hay nodos **maestro ni esclavo**, y todos los nodos, llamados **peers**, operan al mismo nivel.
7. El sharding y la replicación se pueden combinar para mejorar la limitada **tolerancia a errores** que ofrece el sharding, y al mismo tiempo, beneficiarse de la mayor **disponibilidad y escalabilidad** que ofrece la replicación.
8. El teorema CAP establece que un sistema de archivos distribuido solo puede proporcionar dos de las tres propiedades, que son **consistencia, disponibilidad y tolerancia al particionado**.
9. En el contexto del teorema CAP, las bases de datos relacionales proporcionan **consistencia y disponibilidad**.
10. En el contexto del diseño de bases de datos, el acrónimo BASE representa **disponibilidad todo el tiempo, estado flexible y consistencia a largo plazo**.

### **Respuestas al ejercicio 7.2.**

1. sistema de archivos distribuido
2. documento
3. llave-valor
4. grafo
5. basado en columnas

### **Respuestas al ejercicio 7.3.**

1. escalabilidad lineal
2. procesamiento de datos sin esquema
3. bajo costo
4. soporte para múltiples cargas de trabajo
5. redundancia y tolerancia a errores
6. procesamiento de datos distribuidos/paralelos

### **Respuestas al ejercicio 7.4.**

1. mapear
2. combinar
3. dividir
4. mezclar y clasificar
5. reducir

## **Respuestas al ejercicio 7.5.**

1. El framework de MapReduce se basa en el principio de **dividir un problema difícil en tantas partes como sea necesario**.
2. El **parallelismo de tareas** y el **parallelismo de datos** son los dos enfoques usados generalmente para lograr el principio de dividir un problema difícil en tantas partes como sea necesario.
3. El **parallelismo de tareas** se refiere a la parallelización del procesamiento de datos al dividir una tarea en subtareas y ejecutar cada subtarea en un procesador por separado, generalmente en un nodo separado dentro de un cluster.
4. El **parallelismo de datos** se refiere a la parallelización del procesamiento de datos al dividir un dataset en varios subdatasets y procesar cada uno de estos subdatasets en paralelo.
5. El framework de MapReduce aborda la necesidad de una ejecución repetida de la misma tarea en datos distribuidos al usar el enfoque de **parallelismo de datos**.
6. En MapReduce, la lógica en la función reduce depende del resultado de la función **de mapeo**.
7. El framework de MapReduce requiere que el dataset sea **particionado** a través del cluster para que múltiples funciones de **mapeo** puedan procesar los subdatasets en paralelo.
8. Con el algoritmo de MapReduce, la lógica en la función de mapeo no debe depender del dataset completo, pues solo se dispone de los datos dentro de **una sola división**.

## **Examen B90.07**

El curso que acaba de finalizar corresponde al Examen B90.07, que es un examen oficial del Programa Profesional Certificado de Ciencias de Big Data (BDSCP).

**PEARSON VUE**

Este examen se puede tomar en los Centros de Examinación de Pearson VUE en todo el mundo, o a través de Pearson VUE Proctoring Online, que le permite tomar exámenes desde su casa o estación de trabajo y contar con supervisión en vivo. Si desea obtener más información, visite las siguientes páginas web:

[www.bigdatascienceschool.com/exams/](http://www.bigdatascienceschool.com/exams/)

[www.pearsonvue.com/arcitura/](http://www.pearsonvue.com/arcitura/)

[www.pearsonvue.com/arcitura/op/ \(Online Proctoring\)](http://www.pearsonvue.com/arcitura/op/)

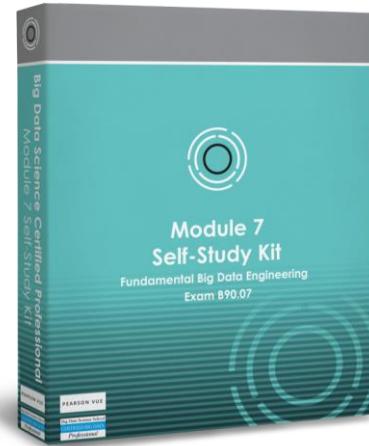
## **Kit de autoaprendizaje del Módulo 7**

Para este Módulo se encuentra disponible un kit oficial de autoaprendizaje de BDSCP, el cual le ofrece materiales y recursos de estudio adicionales, incluyendo una guía separada de autoaprendizaje, CD de audiotutoría y tarjetas de memorización.

Tenga en cuenta que las versiones de este kit de autoaprendizaje están disponibles con y sin un cupón para el Examen B90.07 de Pearson VUE.

Si desea obtener más información, visite las siguientes páginas web:

[www.bigdataselfstudy.com](http://www.bigdataselfstudy.com)



# Información y recursos de contacto

## Comunidad de AITCP

Únase a la creciente comunidad de Profesionales Certificados en TI de Arcitura (Arcitura IT Certified Professional, AITCP), conectándose mediante las plataformas oficiales de social media: LinkedIn, Twitter, Facebook y YouTube.

Los links de social media y de la comunidad están disponibles en:

- [www.arcitura.com/community](http://www.arcitura.com/community)
- [www.servicetechbooks.com/community](http://www.servicetechbooks.com/community)



## Información general del programa

Si desea conocer información general acerca del programa de BDSCP y los requisitos de certificación, visite la siguiente página web:

[www.bigdatascienceschool.com](http://www.bigdatascienceschool.com) y [www.bigdatascienceschool.com/matrix/](http://www.bigdatascienceschool.com/matrix/)

## Información general acerca de los Módulos del curso y los kits de autoaprendizaje

Si desea conocer información general acerca de los Módulos del curso BDSCP y los kits de autoaprendizaje, visite las siguientes páginas web:

[www.bigdatascienceschool.com](http://www.bigdatascienceschool.com) y [www.bigdataselfstudy.com](http://www.bigdataselfstudy.com)

## Inquietudes acerca del examen de Pearson VUE

Si desea conocer información relacionada con la presentación de los exámenes de BDSCP en los centros de examinación de Pearson VUE o mediante la supervisión online de Pearson VUE, visite las siguientes páginas web:

[www.pearsonvue.com/arcitura/](http://www.pearsonvue.com/arcitura/)  
[www.pearsonvue.com/arcitura/op/](http://www.pearsonvue.com/arcitura/op/) (Online Proctoring)

## Programación de talleres dirigidos al público y guiados por instructores

Si desea conocer la más reciente programación de los talleres de BDSCP guiados por instructores que están abiertos al público, visite la siguiente página web:

[www.bigdatascienceschool.com/workshops](http://www.bigdatascienceschool.com/workshops)

## **Talleres privados guiados por instructores**

Los entrenadores certificados pueden realizar los talleres directamente en sus instalaciones, con la opción de supervisión de exámenes en el sitio. Si desea saber más acerca de las opciones y tarifas, envíe un correo electrónico a la siguiente dirección:

info@arcitura.com

o llame a la línea

1-800-579-6582

## **Convertirse en un entrenador certificado**

Si usted está interesado en alcanzar el rango de Entrenador Certificado para este o cualquier otro curso o programa de Arcitura, puede obtener más información visitando la siguiente página web:

[www.arcitura.com/trainerdevelopment/](http://www.arcitura.com/trainerdevelopment/)

## **Inquietudes generales sobre BDSCP**

En caso de que tenga otras preguntas relacionadas con este Curso, o cualquier otro Módulo, Examen o Certificación que haga parte del programa BDSCP, envíe un correo electrónico a la siguiente dirección:

info@arcitura.com

o llame a la línea

1-800-579-6582

## **Notificaciones automáticas**

Si desea que se le notifique automáticamente sobre cambios o actualizaciones al programa BDSCP y a los sitios de recursos relacionados, envíe un mensaje de correo electrónico en blanco a la siguiente dirección:

notify@arcitura.com

## **Retroalimentación y comentarios**

Ayúdenos a mejorar este curso. Envíe su retroalimentación o comentarios a la dirección de correo electrónico:

info@arcitura.com