

# Introducción a NoSQL



# NoSQL - Introducción

## NoSQL

Término utilizado para referirse a BD que no siguen los principios de los RDBMS

# NoSQL - Introducción

## NoSQL

Término utilizado para referirse a BD que no siguen los principios de los RDBMS

- Sistemas diseñados para lograr alta velocidad y escalabilidad (Google, Facebook, Yahoo, Twitter, etc.)
- Usualmente NO respetan propiedades ACID de transacciones

# NoSQL - Introducción

## NoSQL

Término utilizado para referirse a BD que no siguen los principios de los RDBMS

- Sistemas diseñados para lograr alta velocidad y escalabilidad (Google, Facebook, Yahoo, Twitter, etc.)
- Usualmente NO respetan propiedades ACID de transacciones

## Propiedades ACID

- **Atomicity (Atomicidad):** Transacción como unidad atómica. Las operaciones de una transacción se ejecutan en su totalidad o no se ejecuta ninguna.
- **Consistency preservation (Preservación de Consistencia):** Si la transacción se ejecuta completamente sin interferencia de otra transacción, entonces mantiene a la BD de un estado consistente a otro también consistente.
- **Isolation (Aislamiento):** La ejecución de una transacción no debe interferir con la de otra transacción que se ejecute de manera simultánea. Una transacción debe aparecer ser ejecutada como si lo hiciera de forma aislada a las otras. Incluso si varias de ellas son ejecutadas a la vez.
- **Durability (Durabilidad):** Los cambios aplicados a la BD por una transacción commitada deben persistir en la BD. Estos cambios no se pueden perder a causa de ningún fallo.



# NoSQL - Introducción (Cont.)

## Problemas RDBMS

- Múltiples subsistemas funcionando: **subsistema de recovery, control de concurrencia, control de restricciones de integridad**, etc.
- Baja Performance
- Baja Disponibilidad
- Ejemplo: Amazon (Página 10 del libro Vaish-*Getting Started with NoSQL*, Packt Publishing, 2013)

# NoSQL - Introducción (Cont.)

## Problemas RDBMS

- Múltiples subsistemas funcionando: **subsistema de recovery, control de concurrencia, control de restricciones de integridad**, etc.
- Baja Performance
- Baja Disponibilidad
- Ejemplo: Amazon (Página 10 del libro Vaish-*Getting Started with NoSQL*, Packt Publishing, 2013)

## Solución

Ir más allá de esquema ACID . . .

# NoSQL - Introducción (Cont.)

## Propiedades BASE

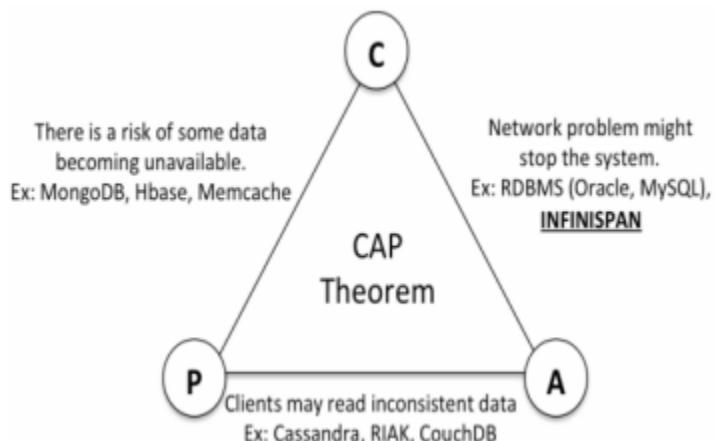
- **Basic Availability:** Cada solicitud garantiza una respuesta: ejecución exitosa o fallida.
- **Soft-state:** El estado del sistema puede cambiar con el tiempo, a veces sin ninguna entrada (por consistencia eventual).
- **Eventual consistency:** La BD puede ser momentáneamente inconsistente pero será consistente con el tiempo.

# NoSQL - Introducción (Cont.)

## CAP

Eric Brewer, 2000, conjetura que “Un sistema distribuído no puede garantizar las siguientes propiedades simultáneamente”

- **Consistency (Consistencia):** Todos los nodos ven los mismos datos al mismo tiempo.
- **Availability (Disponibilidad):** Se garantiza que cada petición a un nodo recibe una confirmación de si ha sido o no resuelta satisfactoriamente.
- **Partition tolerance (Tolerancia a Partición):** El sistema continúa trabajando a pesar de un mensaje perdido o de una falla parcial.



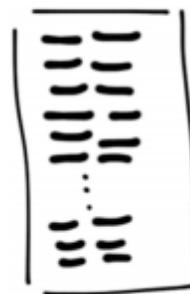
**Transacciones bancarias  
Bolsa de Valores**

# NoSQL - Introducción (Cont.)

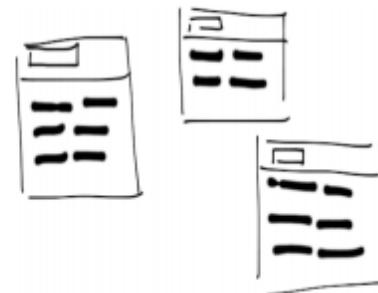
## ¿Por qué utilizar NoSQL?

- **Representación de datos libre de esquemas.** No hay que pensar demasiado para definir una estructura y se puede seguir evolucionando en el tiempo (agregando nuevos campos o anidando datos).
- **Tiempo de desarrollo.** Al ser la representación de datos más adecuada para el problema, se pueden evitar ciertos JOINs extremadamente complejos.
- **Velocidad.** En muchos casos es posible dar respuesta en el orden de los milisegundos en vez de cientos de milisegundos. Esto es beneficioso en celulares y otros dispositivos con conexión intermitente.
- **Planificar escalabilidad.** La aplicación puede ser bastante elástica y manejar picos repentinos de carga.

# NoSQL - Taxonomía



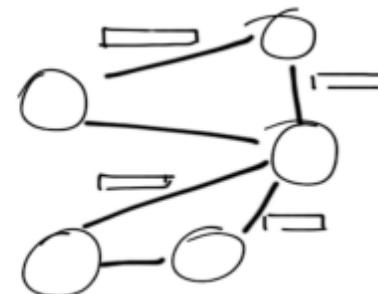
**Clave/Valor**



**Columnas**



**Documentos**



**Grafos**

# NoSQL - Taxonomía - Clave/Valor

## Clave/Valor

Vinculan una clave con un valor (similar a Diccionario, Hash, Map)

## Características

- Sólo se permite operar a través de la clave
- Se puede almacenar cualquier tipo de datos en el campo valor. Delega a la aplicación determinar el tipo del dato.

	Key	Value
Image name	image-12345.jpg	Binary image file
Web page URL	http://www.example.com/my-web-page.html	HTML of a web page
File path name	N:/folder/subfolder/myfile.pdf	PDF document
MD5 hash	9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
REST web service call	view-person?person-id=12345&format=xml	<Person><id>12345</id></Person>
SQL query	SELECT PERSON FROM PEOPLE WHERE PID='12345'	<Person><id>12345</id></Person>

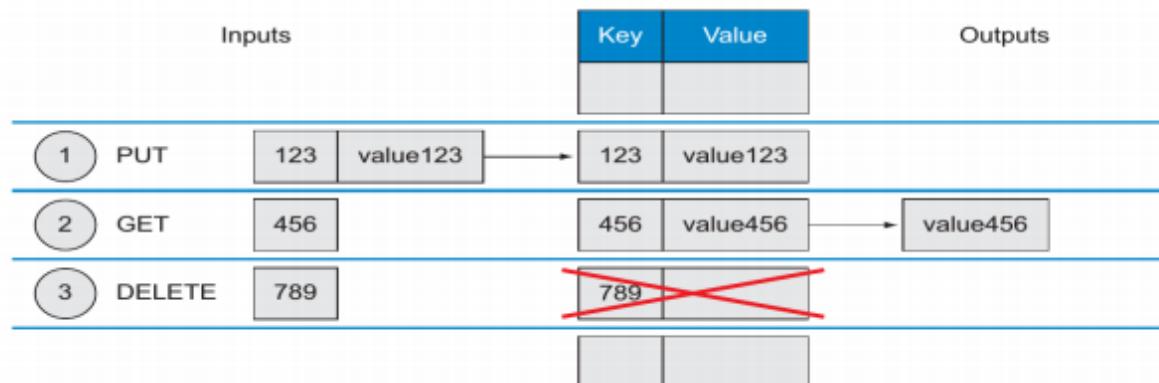
Figure 4.2 Sample items in a key-value store. A key-value store has a key that's associated with a value. Keys and values are flexible. Keys can be image names, web page URLs, or file path names that point to values like binary images, HTML web pages, and PDF documents.

Figura de McCreary/Kelly-Making Sense of NoSQL, Manning, 2014

# NoSQL - Taxonomía - Clave/Valor (Cont.)

## Operaciones principales

- put
- get
- delete



**Figure 4.5** The key-value store API has three simple commands: **put**, **get**, and **delete**. This diagram shows how the **put** command inserts the input key "123" and value "value123" into a new key-value pair; the **get** command presents the key "456" and retrieves the value "value456"; and the **delete** command presents the key "789" and removes the key-value pair.

Figura de McCreary/Kelly-Making Sense of NoSQL, Manning, 2014

# NoSQL - Taxonomía - Clave/Valor (Cont.)

## Comentarios finales

- Escala: No es necesario revisar todas las entradas para consultar un valor
- Ideal para el almacenamiento de datos multimedia
- Ejemplos:
  - Redis (en memoria, with dump or command-log persistence)
  - Memcached (en memoria)
  - MemcacheDB (basado en Memcached)
  - Berkley DB
  - Voldemort (implementación open source de Amazon Dynamo)
- Memcached y Redis permiten la caducidad de las claves, después de lo cual la entrada es desalojada de la BD.
- A veces, se pueden generar claves inteligentemente (UUID Identificador Único Universal) y consultar por un rango de claves.
- Redis permite traer datos que coincidan con cierto patrón de la clave. (Si bien tiene  $\mathcal{O}(n)$ , la constante es bastante baja.)

# NoSQL - Taxonomía - Column Store

## Column Store

Almacenan la información en columnas (a diferencia de las BD orientadas a filas que almacenan los datos en forma de fila)

## Ejemplo

EmployeeID	FirstName	LastName	Age	Salary
SM1	Anuj	Sharma	45	10000000
MM2	Anand		34	5000000
T3	Vikas	Gupta	39	7500000
E4	Dinesh	Verma	32	2000000

Figura de Vaish-*Getting Started with NoSQL*, Packt Publishing, 2013

## Orientado a Filas

SM1,Anuj,Sharma,45,10000000  
MM2,Anand,,34,5000000  
T3,Vikas,Gupta,39,7500000  
E4,Dinesh,Verma,32,2000000

## Orientado a Columnas

SM1,MM2,T3,E4  
Anuj,Anand,Vikas,Dinesh  
Sharma,,Gupta,Verma,  
45,34,39,32  
10000000,5000000,7500000,2000000



# NoSQL - Taxonomía - Column Store (Cont.)

## Ventajas

- Calcular máximo. Mínimo, promedio, etc. en grande volúmenes de datos tiene muy buena performance. ¿Por qué?
- Mismo caso para aplicar la misma operación (actualización, etc.) a la misma columna.

## Ejemplos

- MonetDB
- Vertica
- SybaseIQ

# NoSQL - Taxonomía - Column “Family” Store

## Column “Family” Store

Similar a key-value, pero utilizan fila y columna como clave para los datos.

## Aproximación

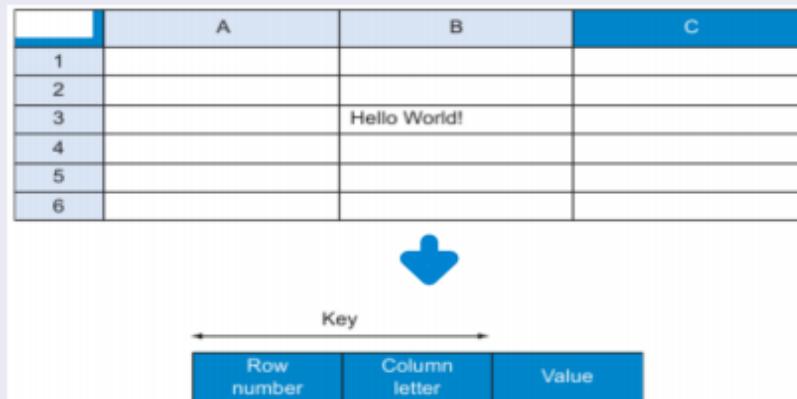


Figura de McCreary/Kelly-Making Sense of NoSQL, Manning, 2014

- A diferencia de los RDBMS, no es necesario ingresar valores para todas las columnas.
- Escala muy bien cuando la matriz posee muchas celdas vacías (a diferencia de los RDBMS).

# NoSQL - Taxonomía - Column “Family” Store (Cont.)

## Más precisamente ...

Key				
Row-ID	Column family	Column name	Timestamp	Value

**Figure 4.20** The key structure in column family stores is similar to a spreadsheet but has two additional attributes. In addition to the column name, a column family is used to group similar column names together. The addition of a timestamp in the key also allows each cell in the table to store multiple versions of a value over time.

Figura de McCreary/Kelly-*Making Sense of NoSQL*, Manning, 2014

- Agrupar columnas permite obtener varias de ellas con sólo mencionar el grupo.
- Timestamp permite almacenar múltiples versiones del valor a través del tiempo

## Ventajas

- No utilizan JOINs → escala muy bien en sistemas distribuidos.
- Capacidad de replicar datos en múltiples nodos de la red. La ausencia de JOINs permite almacenar cualquier parte de la matriz en equipos remotos. Si el servidor que contiene parte de los datos se cae, otros equipos pueden ofrecer los datos.

## Ejemplos

El principal ejemplo es BigTable de Google, que inspiró a: Apache Cassandra y HBase.

# NoSQL - Taxonomía - Documentos

## Documentos

- Almacenamiento de datos semi-estructurados (XML, JSON, etc.).
- Cada registro correspondientes a los documentos pueden tener campos diferentes.
- Los registros pueden o no seguir un esquema específico (como las definiciones de tabla de RDBMS).

## Ventajas/Desventajas

- “A favor/encontra”: BD típicamente no soportan esquemas ni la validación de un documento contra un esquema.
- A diferencia de **clave/valor** y **columnas**, que sólo pueden obtener los valores utilizando la clave, en **documentos** se pueden realizar consultas sobre los valores almacenados en los documentos. Se suelen realizar índices sobre los valores.

## Ejemplos

- |              |                    |         |
|--------------|--------------------|---------|
| ● MongoDB    | ● Lotus Notes      | ● Redis |
| ● CouchDB    | ● Apache Cassandra | ● BaseX |
| ● Jackrabbit | ● Terrastore       |         |



# NoSQL - Taxonomía - Documentos (Cont.)

## Ejemplos

### Documento Empleado 1

```
{  
    "EmployeeID": "SM1",  
    "FirstName" : "Anuj",  
    "LastName"  : "Sharma",  
    "Age"        : 45,  
    "Salary"     : 10000000  
}
```

### Documento Empleado 2

```
{  
    "EmployeeID": "MM2",  
    "FirstName" : "Anand",  
    "Age"        : 34,  
    "Salary"     : 50000000  
    "Address"   : {  
        "Line1" : "123, 4th Street",  
        "City"  : "Bangalore",  
        "State" : "Karnataka"  
    },  
    "Projects" : [ "nosql-migration"  
                  , "top-secret-007" ]  
}
```



# NoSQL - Taxonomía - Documentos (Cont.)

## Ejemplos

### Documento Empleado 1

```
{  
    "EmployeeID": "SM1",  
    "FirstName" : "Anuj",  
    "LastName"  : "Sharma",  
    "Age"       : 45,  
    "Salary"    : 10000000  
}
```

### Documento Empleado 2

```
{  
    "EmployeeID": "MM2",  
    "FirstName" : "Anand",  
    "Age"       : 34,  
    "Salary"    : 50000000  
    "Address"   : {  
        "Line1" : "123, 4th Street",  
        "City"  : "Bangalore",  
        "State" : "Karnataka"  
    },  
    "Projects" : [ "nosql-migration"  
                  , "top-secret-007" ]  
}
```

### Documento Oficina 1

```
{  
    "LocationID"     : "Bangalore-SDC-BTP-CVRN",  
    "RegisteredName": "ACME Software Development Ltd"  
    "RegisteredAddress" : {  
        "Line1": "123, 4th Street",  
        "City" : "Bangalore",  
        "State": "Karnataka"  
    }  
}
```

## Observar

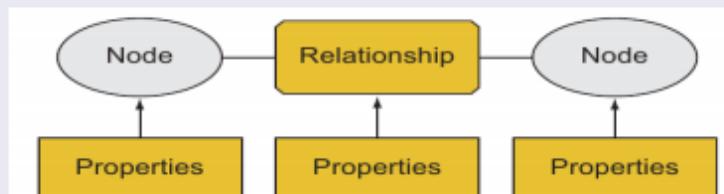
Documentos de empleados tienen campos similares, pero el de oficina no. Sin embargo pueden convivir en la misma BD (debido a la ausencia de esquemas).



# NoSQL - Taxonomía - Grafos

## Grafos

Permiten almacenar relaciones y trabajar con ellas de manera muy eficiente.



**Figure 4.10** A graph store consists of many node-relationship-node structures. Properties are used to describe both the nodes and relationships.

Figura de McCreary/Kelly-Making Sense of NoSQL, Manning, 2014

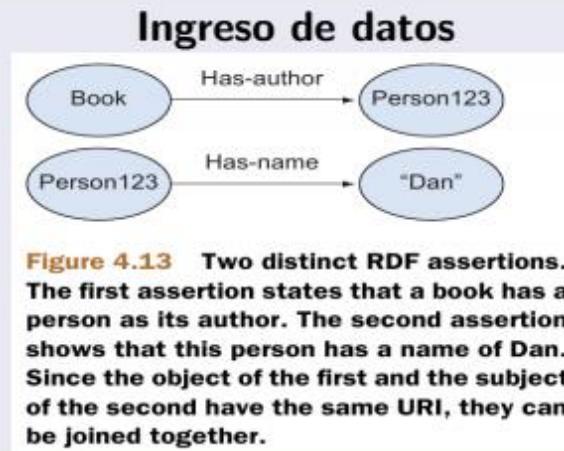
- Nodos suelen representar entidades (personas, organizaciones, números de teléfono, páginas web, equipos de una red, o células de un organismo vivo).
- Las relaciones pueden ser consideradas como las conexiones entre estas entidades.

## Ventajas

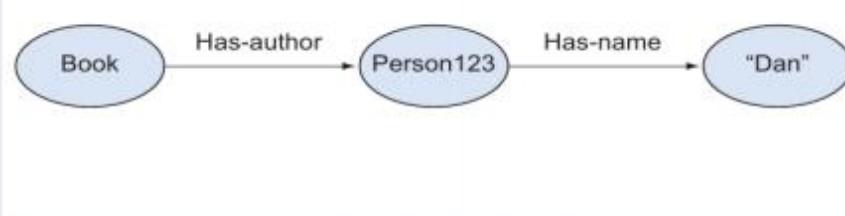
Muy útiles para el modelado de redes sociales, problemas basados en reglas, etc.

# NoSQL - Taxonomía - Grafos (Cont.)

## Ejemplo



**Consulta: El autor del libro, ¿se denomina “Dan”?**



**Figure 4.14** How two distinct RDF assertions can be joined together to create a new assertion. From this graph you can answer yes to the question, “Does this book have any author that has the name “Dan”?”

Figuras de McCreary/Kelly-Making Sense of NoSQL, Manning, 2014

# NoSQL - Taxonomía - Grafos (Cont.)

## Otras consultas

- ¿Cuál es el camino más corto entre dos nodos en un grafo?
- ¿Qué nodos tienen nodos vecinos con determinadas propiedades?
- Dados dos nodos en un grafo, ¿cuán similares son sus nodos vecinos?
- ¿Cuál es el grado promedio de los nodos?

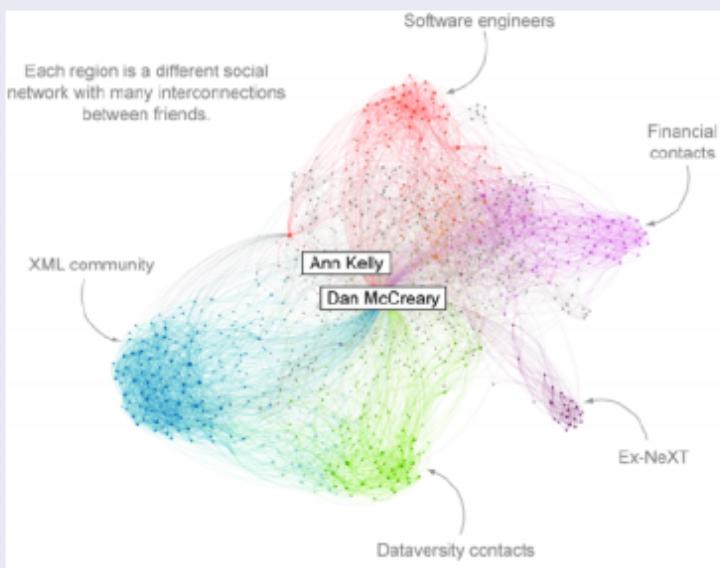
## Ventajas/Desventajas

- En **RDBMS** JOINs son costosos en términos de latencia: muchas E/S en disco. En **grafos** es más rápido debido a la naturaleza pequeña de cada nodo y la capacidad de mantenerlos en memoria RAM (en ppio. no requiere E/S a disco).
- Difícil de escalar en varios servidores debido a la estrecha conexión entre nodos. Los datos pueden ser replicados en varios servidores para mejorar el rendimiento de lecturas/consultas; las escrituras y consultas que abarcan múltiples servidores puede ser un problema complejo a la hora de implementar una solución.
- Una posible solución es almacenar los datos en una BD orientada a **documentos** y las relaciones en una orientada a **grafos**.
- Una consulta puede retornar un conjunto de nodos y relaciones para crear una imagen de visualización en pantalla



# NoSQL - Taxonomía - Grafos (Cont.)

## Ejemplo



**Figure 4.15** A social network graph generated by the LinkedIn InMap system. Each person is represented by a circle, and a line is drawn between two people that have a relationship. People are placed on the graph based on the number of connections they have with all the other people in the graph. People and relationships are shaded the same when there's a high degree of connectivity between the people. Calculating the placement of each person in a social network map is best performed by an in-memory graph traversal program.

Figura de McCreary/Kelly-*Making Sense of NoSQL*, Manning, 2014

## Sistemas

- Neo4j
- FlockDB  
(Twitter)

# NoSQL - Taxonomía - Múltiples

## Múltiples

Sistemas que abarcan múltiples tipos.

## Ejemplos

- **OrientDB:** Abarca Documentos, Clave-Valor y Grafos. Sitio web oficial:  
<http://www.orientdb.org>
- **ArangoDB:** Documentos, Clave-Valor y Grafos. Sitio web oficial:  
<http://www.arangodb.org>
- **Aerospike:** Un híbrido entre RDBMS y NoSQL. Abarca RDBM, Documentos, Clave-Valor y Grafos. Código fuente:  
<https://github.com/JakSprats/Alchemy-Database>

# NoSQL - Guía Visual (incompleta ...)

## Guía Visual

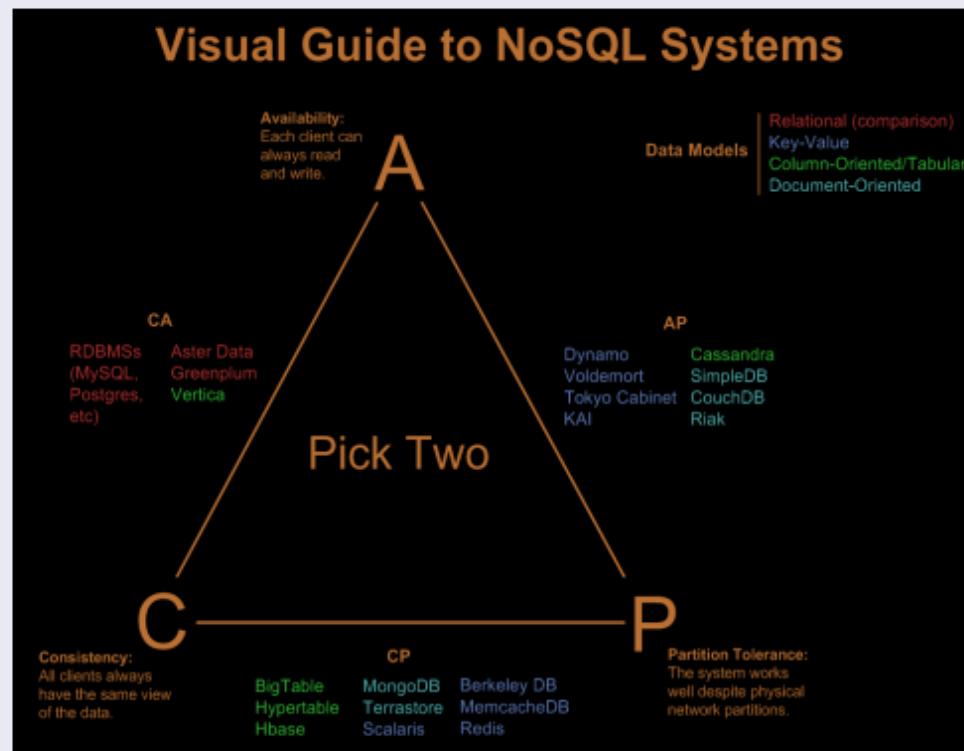


Figura de McCreary/Kelly-Making Sense of NoSQL, Manning, 2014

# NoSQL - Map-Reduce - Concepto

## Map-Reduce

MapReduce es un marco de trabajo para el procesamiento en paralelo de grandes volúmenes de datos en varios equipos (nodos).

## Funcionamiento

- La operación **map** tiene un nodo principal, que divide una operación en subpartes y distribuye cada operación a otro nodo para su procesamiento,
- y **reduce** es el proceso donde el nodo maestro junta los resultados de los otros nodos y las combina en respuesta al problema original.

# NoSQL - Map-Reduce - Idea Programación Funcional

## Idea de Programación Funcional

La idea proviene de programación funcional

## Ejemplo de Programación Funcional

- **map** aplica una función a cada elemento de una lista. Ejemplo: Si la función a aplicar es *duplicate* y la lista es [1, 2, 3, 4], al aplicar dicha función utilizando **map**, devuelve [2, 4, 6, 8] (la lista original no se altera)
- Notar que es altamente paralelizable.
- **reduce** (o *fold*) es una función de agrupamiento o acumulación de los elementos de la lista. Ejemplo: Si la función es *suma* y la lista es la generada por el **map** previo ([2,4,6,8]), entonces el **reduce** devuelve 20.

# NoSQL - Map-Reduce - Idea BD

## Idea en BD

Misma idea, pero aplicado a set de datos

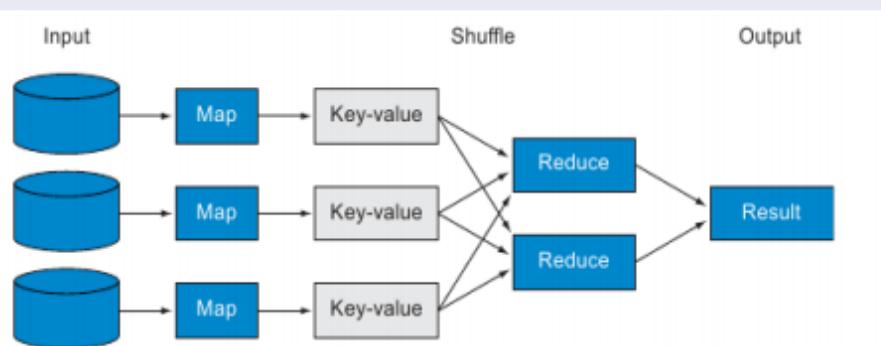
## Ejemplo en BD

En una BD orientada a clave/valor:

- **map** aplica una función a cada par (clave,valor), generando una nueva colección.
- **reduce** aplica una función de agregación (ej: suma) a la colección generada por **map**, para devolver el resultado final.

# NoSQL - Map-Reduce - Ejemplo

## Ejemplo



**Figure 6.10** The basics of how the map and reduce functions work together to gain linear scalability over big data transforms. The map operation takes input data and creates a uniform set of key-value pairs. In the shuffle phase, which is done automatically by the MapReduce framework, key-value pairs are automatically distributed to the correct reduce node based on the value of the key. The reduce operation takes the key-value pairs and returns consolidated values for each key. It's the job of the MapReduce framework to get the right keys to the right reduce nodes.

Figura de McCreary/Kelly-*Making Sense of NoSQL*, Manning, 2014

- ① **map** recupera los datos de la BD y los transforma en una colección de operaciones que pueden ser ejecutados independientemente en distintos procesadores.
- ② La salida de los **map**, son pares (clave,valor).
- ③ Como siguiente fase, **reduce**, utiliza los pares como entrada, ejecuta la operación asignada y retorna un resultado.

# NoSQL - Map-Reduce - Problemas

## Problema I

¿Qué pasa si los datos de origen se encuentran en tres o más nodos? ¿Se mueven los datos entre nodos?

Si se quiere ser eficiente, la respuesta es NO.

Entonces, se debe tener en cuenta en qué nodo debe ejecutarse la función **map/reduce**.

## Problema II

¿Qué pasa si en medio de la operación falla alguno de los **map / reduce**?

¿Es necesario reiniciar todo el trabajo completo o se puede asignar sólo la parte que falló a otro nodo?

# NoSQL - Sharding

## Problema

A medida que la cantidad de datos aumenta, puede llegar un momento en que se alcanza la capacidad máxima del sistema.  
Surge la necesidad de particionar los datos.

## Sharding

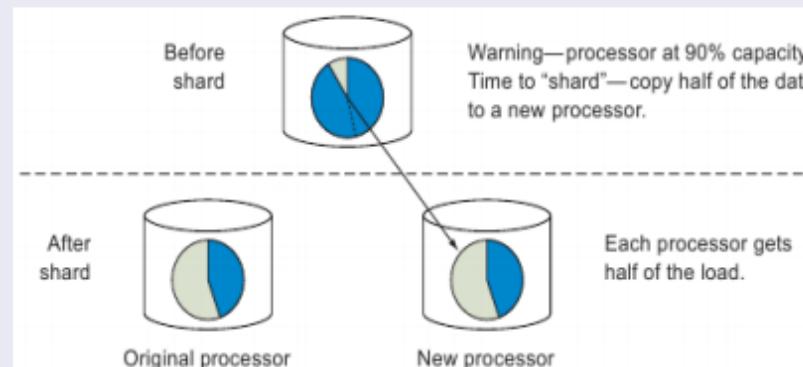
Fragmentar la BD en fragmentos denominados **shards** y distribuirlos a través de los servidores disponibles. Conceptualmente, los **shards** comparten esquemas y colectivamente representan el total del dataset.

## Funcionamiento

- En el pasado: Dar de baja el sistema. Actualmente: Se puede realizar con el sistema en funcionamiento.
- Algunos sistemas permiten realizarlo de manera automática y otros de manera manual.

# NoSQL - Sharding - Visualmente

## Gráficamente



**Figure 2.9** Sharding is performed when a single processor can't handle the throughput requirements of a system. When this happens you'll want to move the data onto two systems that each take half the work. Many NoSQL systems have automatic sharding built in so that you only need to add a new server to a pool of working nodes and the database management system automatically moves data to the new node. Most RDBMSs don't support automatic sharding.

Figura de McCreary/Kelly-Making Sense of NoSQL, Manning, 2014

# NoSQL - Sharding - Ejemplo

## Ejemplo

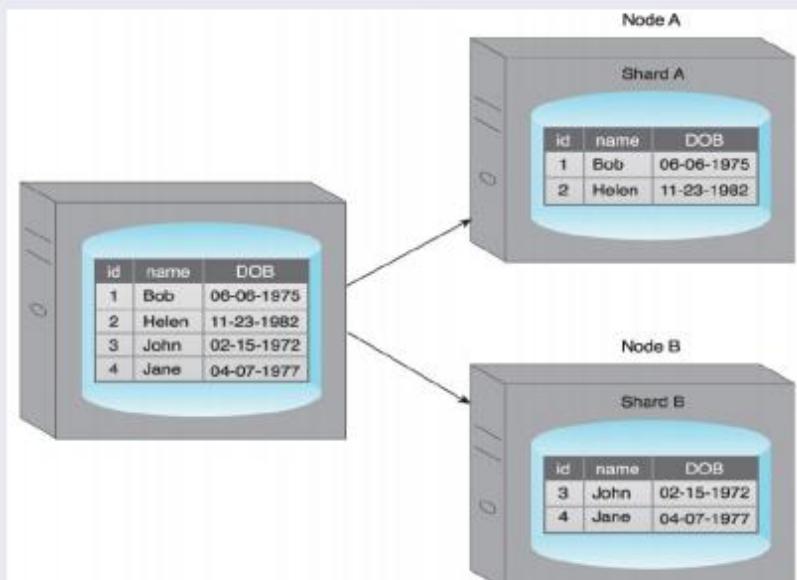


Figure 5.5 An example of sharding where a dataset is spread across Node A and Node B, resulting in Shard A and Shard B, respectively.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

# NoSQL - Sharding - Ejemplo (Cont.)

## En la práctica

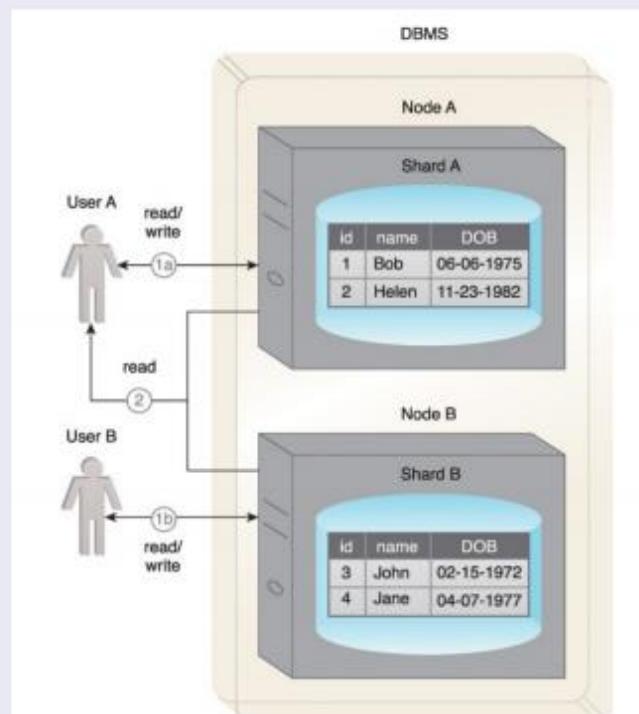


Figure 5.6 A sharding example where data is fetched from both Node A and Node B.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

# NoSQL - Sharding - Ventajas/Desventajas

## Ejemplos BD de cuentas de usuarios

Posibles criterios para fragmentar:

- Nombres de usuarios: A-N en un servidor y O-Z en otro
- Origen: división Continental
- Otra: Al azar

## Problemas

- ¿Si el usuario cambia de nombre?
- ¿Si usuario se muda? Usuarios cercanos, ¿tienden a tener relaciones más estrechas? ¿Qué sucede, entonces, con determinados horarios (noche vs. día)?  
¿sobrecarga de determinados servidores?

## Ventajas/Desventajas

- (+) Tolerancia parcial ante caída de nodos (sólo se pierden los datos del nodo afectado)
- (-) Consultas que involucran varios nodos pueden afectar negativamente la performance



# NoSQL - Réplicas

## Problema

A medida que aumenta la cantidad de servidores, aumenta la probabilidad de falla.

## Replicación

Almacenamiento de múltiples copias de la BD, cada una de ellas conocidas como **réplicas**.

## Métodos de implementación

- Master-Slave
- Peer-to-Peer

## Funcionamiento: Gráficamente

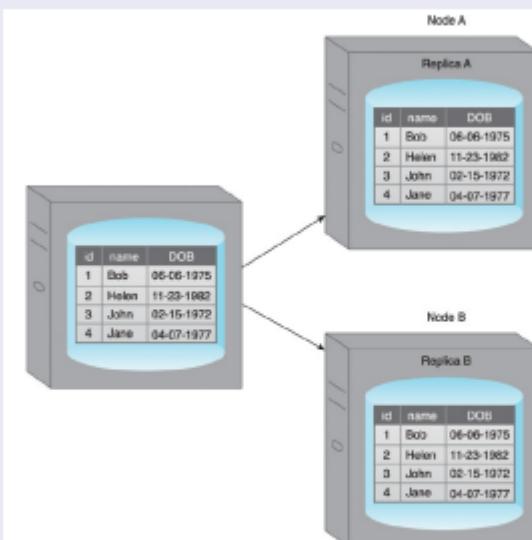


Figure 5.7 An example of replication where a dataset is replicated to Node A and Node B, resulting in Replica A and Replica B.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

# NoSQL - Réplicas - Master-Slave

## Funcionamiento: Gráficamente

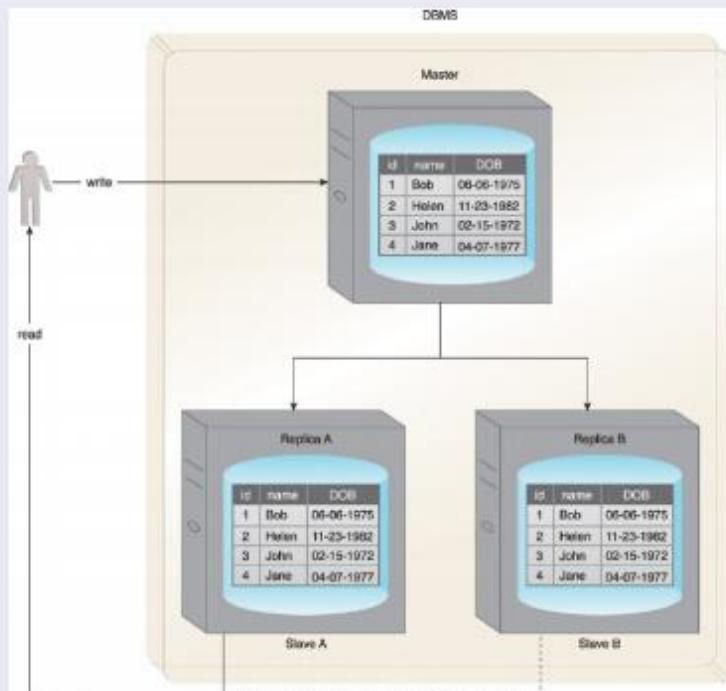


Figure 5.8 An example of master-slave replication where Master A is the single point of contact for all writes, and data can be read from Slave A and Slave B.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

## Funcionamiento

- Escrituras (insert, delete, update) en el nodo Master.
- Lecturas sobre cualquier nodo Slave.

## Ventajas/Desventajas

- (+) Ideal para escenarios de uso intensivo de lecturas
- (+) Si el nodo Master falla, se puede continuar las lecturas
- (+) Nodo Slave puede ser configurado como backup y tomar el rol de Master en caso de fallo

# NoSQL - Réplicas - Master-Slave (Cont.)

## Problema: Gráficamente

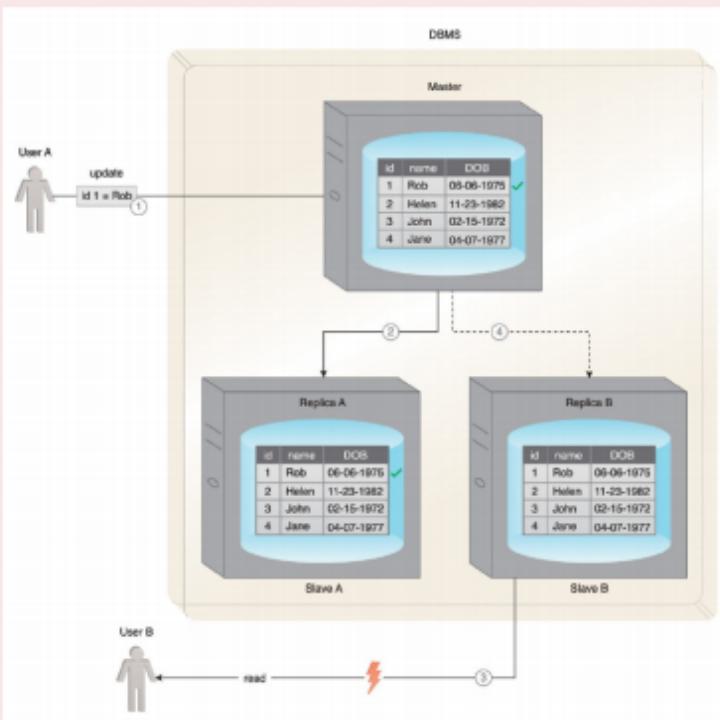


Figure 5.9 An example of master-slave replication where read inconsistency occurs.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

## Problema

### Lectura inconsistente

### Possible Solution

Sistema de votación donde una lectura es consistente  $\Leftrightarrow$  la mayoría de los nodos contienen la misma versión del registro.

Contra: Implementación requiere de un sistema de comunicación entre nodos Slaves rápido y confiable.

# NoSQL - Réplicas - Peer-to-Peer

## Funcionamiento: Gráficamente

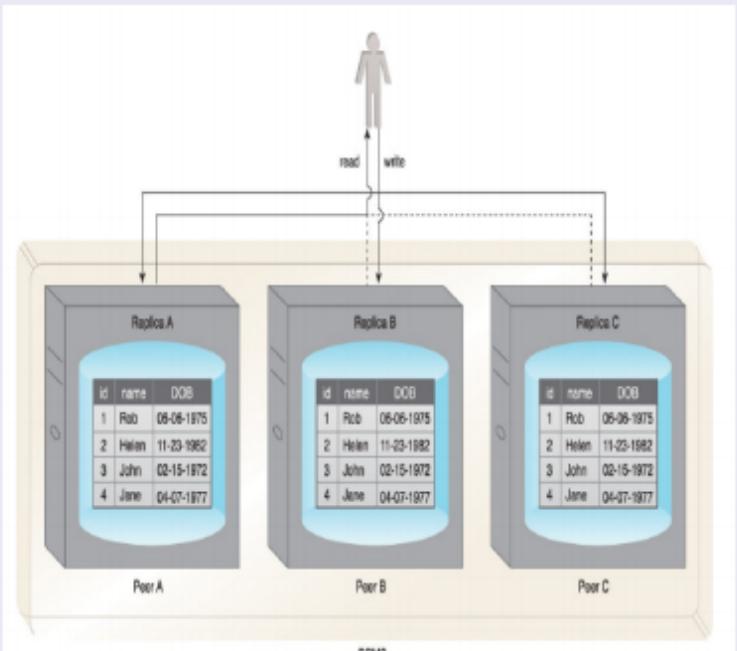


Figure 5.10 Writes are copied to Peers A, B and C simultaneously. Data is read from Peer A, but it can also be read from Peers B or C.

## Funcionamiento

Todos los nodos (denominados **peers**) posseen el mismo nivel de jerarquía y son capaces de manejar tanto las lecturas como las escrituras.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

# NoSQL - Réplicas - Peer-to-Peer (Cont.)

## Problemas: Gráficamente

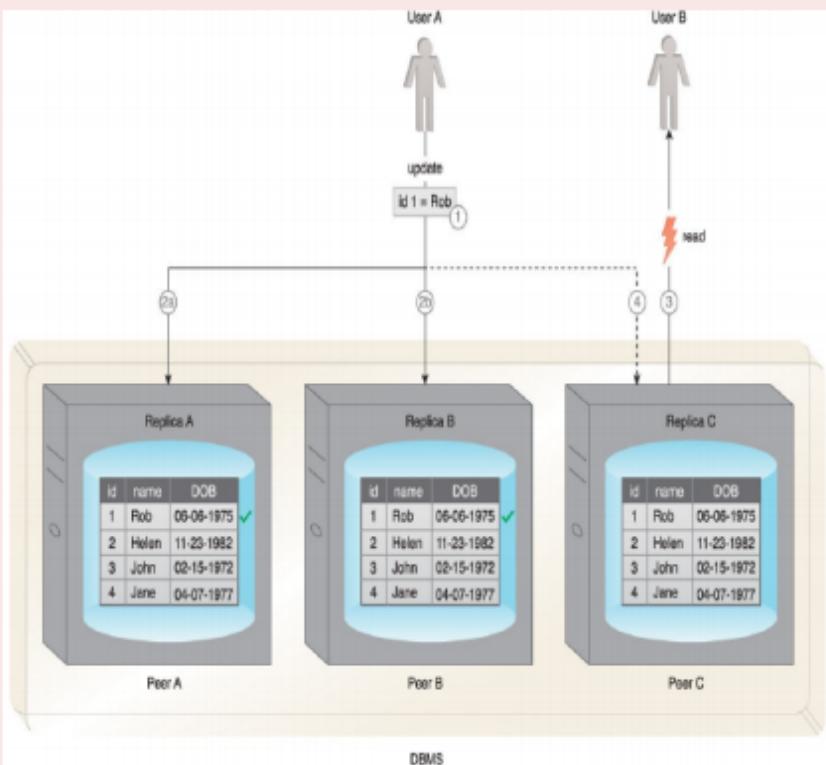


Figure 5.11 An example of peer-to-peer replication where an inconsistent read occurs.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

## Problema

### Lectura inconsistente

#### Estrategia de concurrencia Pesimista

- Estrategia proactiva.
- Utiliza **locking de registro**.
- Va en contra de la **disponibilidad**. El registro que está siendo actualizado permanece no disponible hasta que locks son eliminados.

#### Estrategia de concurrencia Optimista

- Estrategia reactiva.
- No utiliza **locking**.
- Permite inconsistencias sabiendo que eventualmente se llegará a un estado consistente luego de que TODAS las actualizaciones se propaguen.

# NoSQL - Réplicas - Peer-to-Peer (Cont.)

## Solución Optimista

- Peers pueden permanecer inconsistentes por un tiempo hasta alcanzar consistencia. Sin embargo, BD permanece disponible dado que no existen **locks**.
- reads pueden ser inconsistentes durante un tiempo. Mientras algunos peers completaron la actualización, otros están pendientes de hacerlo. Sin embargo, los **reads** eventualmente serán consistentes cuando la actualización alcance a TODOS los nodos.
- Para asegurar consistencia, se puede implementar un sistema de votación al igual que en el esquema Master-Slave

# NoSQL - Sharding + Réplicas

## Recordando . . .

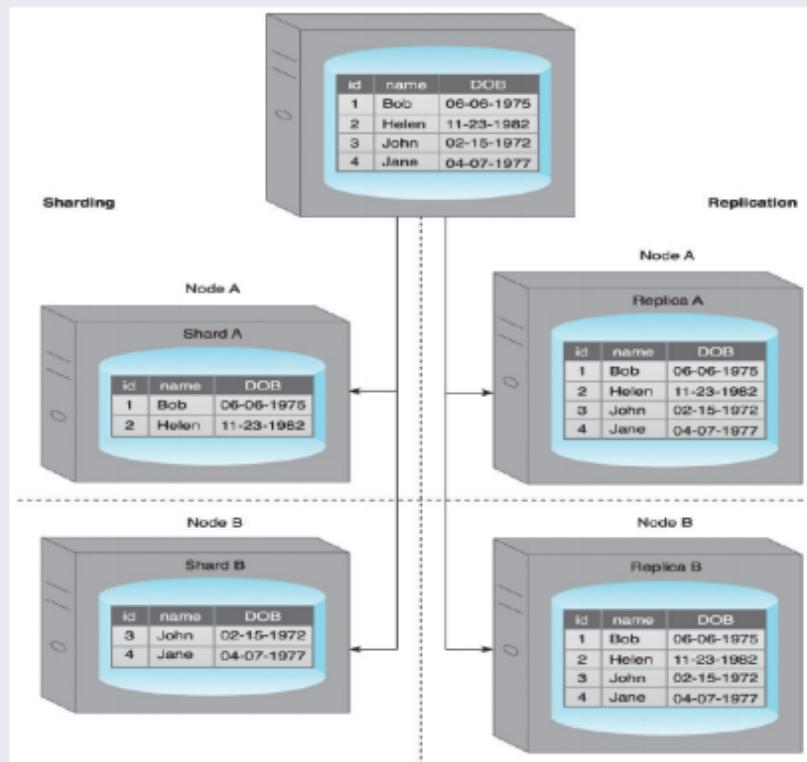


Figure 5.12 A comparison of sharding and replication that shows how a dataset is distributed between two nodes with the different approaches.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

## Ventajas

- Mejorar la limitada tolerancia a fallos, ofrecida por sharding
- Aprovechar beneficios del incremento de disponibilidad y escalabilidad de las Réplicas.

## Combinaciones

- Sharding + Replicación Master-Slave.
- Sharding + Replicación Peer-to-Peer.

# NoSQL - Sharding + Réplicas - Master/Slave

## Funcionamiento

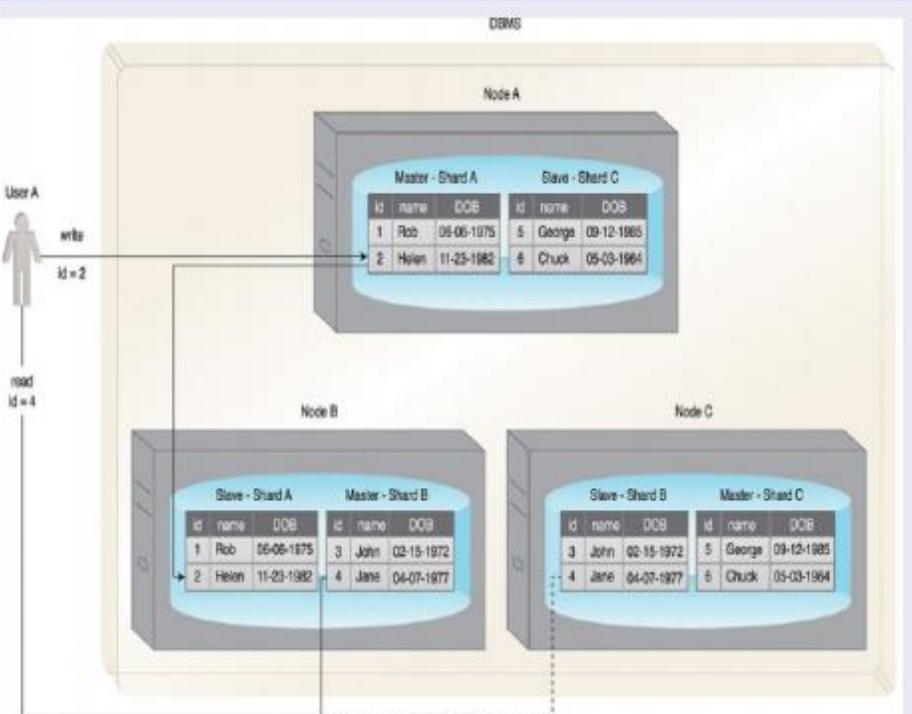


Figure 5.13 An example that shows the combination of sharding and master-slave replication.

Figura de Erl/Khattak/Buhler-Big Data Fundamentals, Prentice Hall, 2016

## Funcionamiento

### Esquema de Shard-Master y Shards-Slaves

## Ventajas/Desventajas

- Consistencia de escrituras mantenida por el Shard-Master.
- Si falla el Shard-Master, impacta en las operaciones de escritura.
- Réplicas en Shard-Slaves mejoran escalabilidad y proveen tolerancia a fallos en operaciones de lectura.

# NoSQL - Sharding + Réplicas - Peer-to-Peer

## Funcionamiento

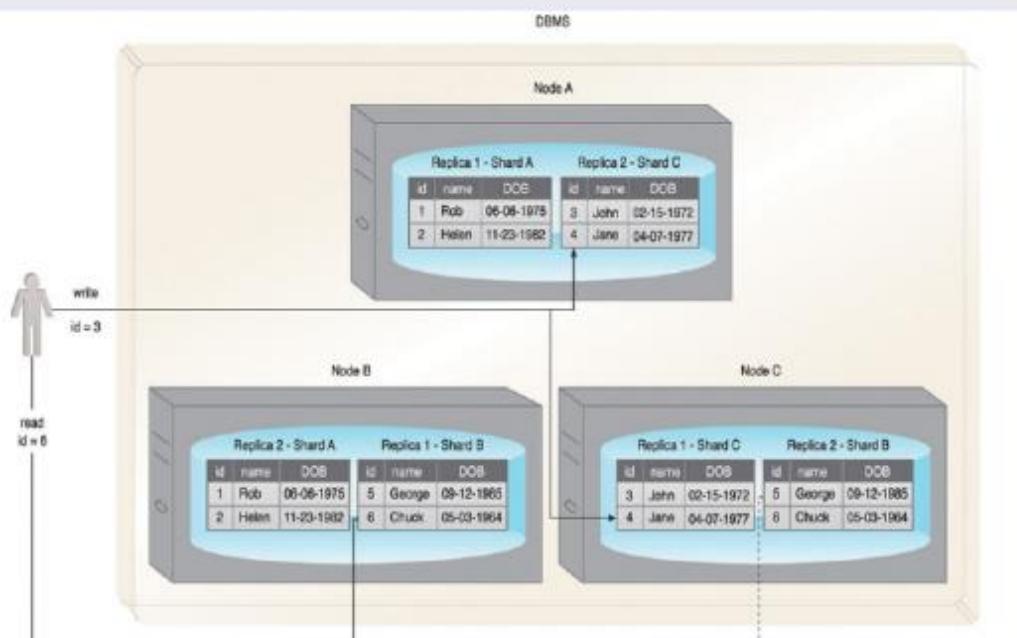


Figure 5.14 An example of the combination of sharding and peer-to-peer replication.

Figura de Erl/Khattak/Buhler-*Big Data Fundamentals*, Prentice Hall, 2016

## Funcionamiento

Cada Shard se replica en múltiples peers.

## Ventajas/Desventajas

- Este esquema incrementa escalabilidad y tolerancia a fallos.
- Como no hay Master, no existe único punto de falla por lo tanto es tolerante a fallos tanto de operaciones de lectura como escritura.

# NoSQL - Sharding + Réplicas - Balance de Carga

## Aumento de eficiencia

Muchos NoSQL utilizan “commodity processors” (CPs), donde cada uno almacena parte del conjunto de datos. Al realizar un query sobre todos los datos es más eficiente enviar el query a los CPs y que se procese en cada uno de ellos, que traer todos los datos al nodo central y procesar el query en este último.