Comparativa SQL y NoSQL

IT BOARDING

BOOTCAMP

Índice

IT BOARDING
BOOTCAMP

// SQL y NoSQL

El dilema a la hora de comenzar un proyecto, surge en la previa elección de la tecnología a utilizar. Es allí, donde las preguntas se deben orientar hacia cuál es el problema y qué escenario se resolverá.

¿Cuál es mejor? SQL vs NoSQL

➔ Realmente, no existe un tipo de base de datos mejor o peor que otro, más bien, cada uno permite almacenar los mismos datos de diferentes formas, siendo las alternativas diferentes en función de los objetivos de un proyecto.

SQL

id_alumno	edad	calificación
1	12	77
2	12	68
3	11	75

NoSQL

```
[
 {
  “id_alumno”: 1,
  “edad”: 12,
  “calificación”: 77
 },
 {
  “id_alumno”: 2,
  “edad”: 12,
  “calificación”: 68
 },
 {
  “id_alumno”: 3,
  “edad”: 11,
  “calificación”: 75
  }
]
```

Diferencias

SQL y NoSQL

IT BOARDING

BOOTCAMP

Diferencias: SQL vs NoSQL

Modelo - Estructura

SQL

- Son relacionales y tienen esquemas fijos.
- Se estructuran en tablas

NoSQL

- Son más dinámicas, no requieren esquemas fijos
- Pueden ser basadas en: documentos, clave-valor, grafos o columnares.

Diferencias: SQL vs NoSQL

Lenguaje

SQL

- Definen y utilizan un lenguaje de consulta poderoso y estructurado
- Tiene estándares bien definidos

NoSQL

- Tienen un esquema dinámico para datos no estructurados. Los datos se almacenan de muchas formas, permitiendo flexibilidad.
- No posee una definición clara de estándares.

Diferencias: SQL vs NoSQL

Escalabilidad

SQL

➔ Ante aumento de peticiones, escala verticalmente (Se debe mejorar potencia de servidor).

NoSQL

➔ Ante aumento de peticiones, escala horizontalmente (Se añade/distribuye la BBDD en más servidores).

Diferencias: SQL vs NoSQL

Soporte

SQL

- Las BBDD y herramientas son más maduras.
- Existe mayor soporte por parte de proveedores.

NoSQL

- No ofrecen tanta fiabilidad.
- Ofrecen menos garantías y confianza.

Diferencias: SQL vs NoSQL

Performance

SQL

- Tienen que juntar datos en consultas a diferentes tablas, por lo que en gran volúmen se vuelven más lentas.

NoSQL

- En general, tienen menos relaciones, por lo que las hace más veloces.

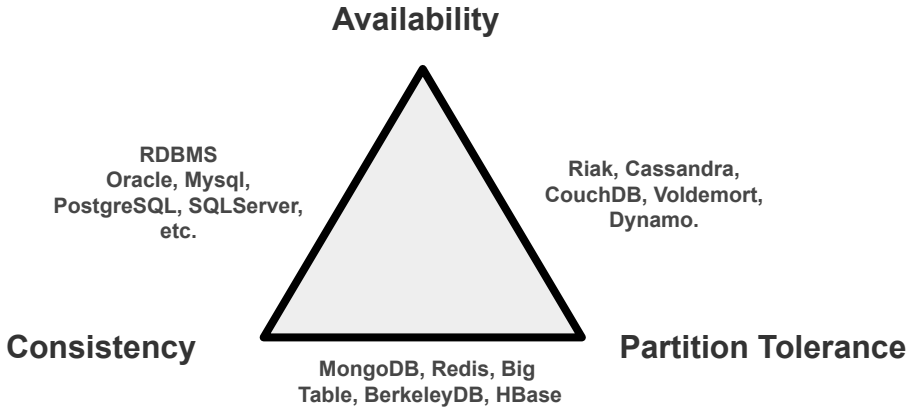
Performance - CAP, BASE y ACID

IT BOARDING

BOOTCAMP

Teorema de CAP, BASE y ACID

- Algunos conceptos que aparecen y se deben entender al hablar de ***performance*** son: **CAP** - **BASE** y **ACID**



Teorema de CAP

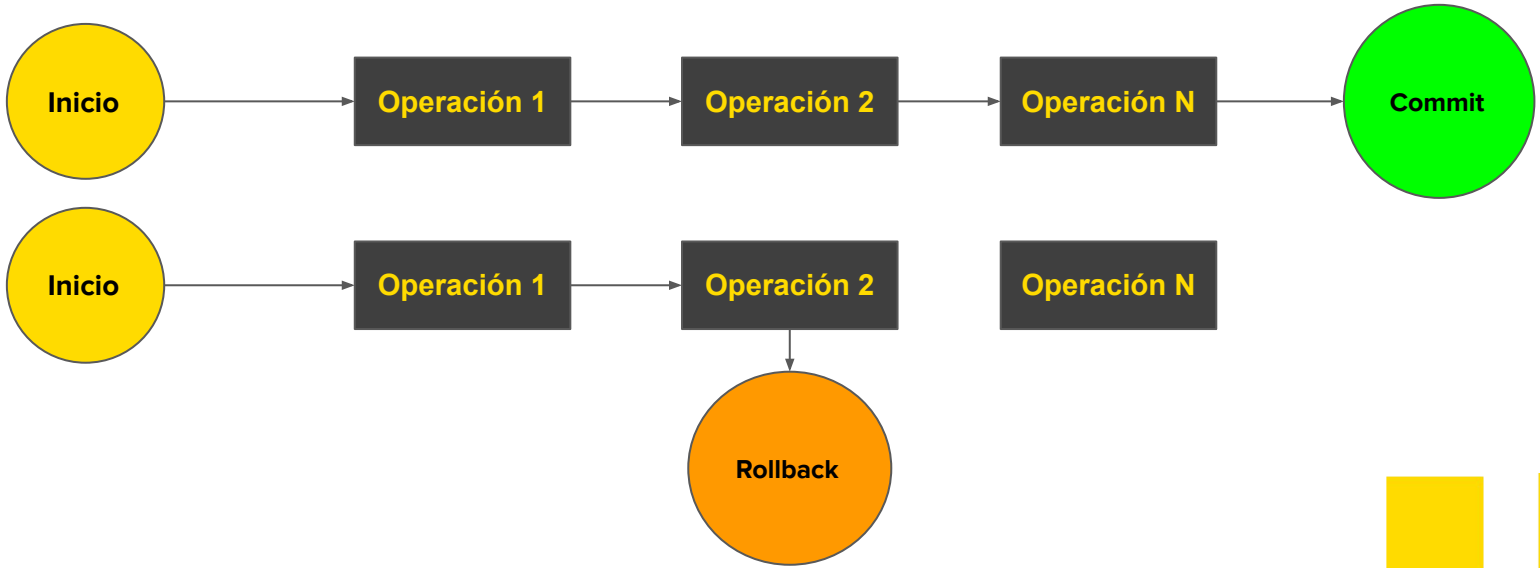
"Según el teorema, un sistema no puede asegurar más de dos de estas tres características simultáneamente."

Propiedades: ACID

- **ACID**: El objetivo de estas propiedades es garantizar las transacciones, aunque de alguna forma, suponga menor rendimiento. Es utilizado en los sistemas gestores de bases de datos relacionales (SQL).
  - **Atomicidad (Atomicity)**: Una transacción debe ser una unidad atómica.
  - **Consistencia (Consistency)**: La ejecución completa de una transacción pasa la BBDD de un estado consistente a otro.
  - **Independencia (Isolation)**: Una transacción no debe interferir con otras que se ejecutan de forma concurrente.
  - **Durabilidad (Durability)**: Los cambios aplicados deben perdurar en la BBDD.

Ejemplo ACID

- Un ejemplo donde se debería respetar las propiedades ACID, podría ser una *transferencia bancaria* que requiere varias operaciones y no puede fallar. Si la transferencia falla, se deberán deshacer todas las operaciones realizadas desde el inicio.

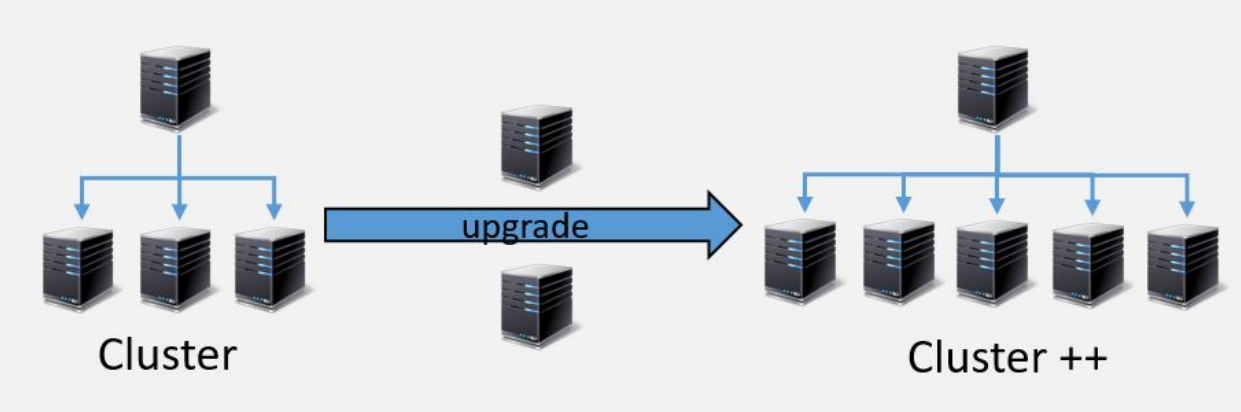


Propiedades: BASE

- Las BBDD NoSQL no son apropiadas para realizar transacciones complejas, de hecho, tienen una mayor tolerancia a los fallos que las SQL.
- **BASE**: El objetivo de estas propiedades es tener mayor escalabilidad y mayor velocidad, disminuyendo exigencias de las garantías que se ofrecen en las transacciones con propiedades ACID.
    - **Basically Available**: Prioridad en la disponibilidad de los datos.
    - **Soft State**: Los datos en las diferentes réplicas no necesitan ser consistentes en todo momento.
    - **Eventually Consistent**: Se asegura la consistencia solo después de que pasa cierto tiempo.

Ejemplo BASE

- Un ejemplo de las propiedades BASE, las podemos ver reflejadas en las grandes virtudes de realizar escalamientos horizontales y poder expandir las BBDD.



"Se gana disponibilidad perdiendo parte de la consistencia, otorgando un estado de los datos más flexible."

Sharding

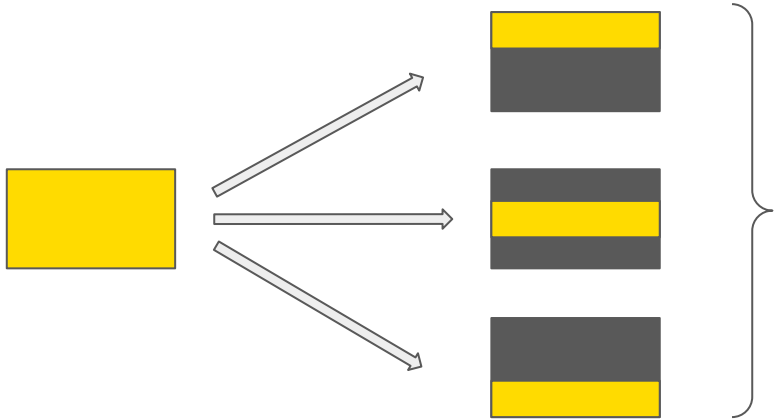
// Fragmentación de datos

IT BOARDING

BOOTCAMP

¿Qué es Sharding?

- Es una forma de **segmentar/fragmentar** los datos de una base de datos de forma **horizontal**, es decir, partir la base de datos principal en varias en bases de datos más pequeñas y repartiendo la información.
- Lo que se consigue es una partición de datos en diferentes bases que tengan cierta homogeneidad, para conseguir una escalabilidad mucho más rápida.



La idea es ***distribuir*** datos que no caben en un solo nodo en un grupo de nodos de base de datos.

Ejemplo Sharding

➔ Si se tiene un conjunto de datos organizado en una tabla grande, se podría separar las filas o columnas en varias tablas más pequeñas.

Tabla original

id_cliente	nombre	apellido	ciudad
1	Lindsey	Stirling	Los Angeles
2	Armin	Van Buuren	New York
3	John	Williams	San Diego
4	Edward	Harris	Miami

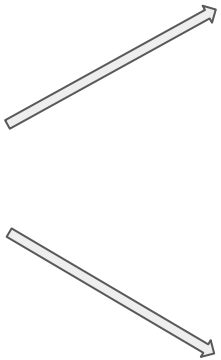


Tabla Horizontal 1

id_cliente	nombre	apellido	ciudad
1	Lindsey	Stirling	Los Angeles
2	Armin	Van Buuren	New York

Tabla Horizontal 2

id_cliente	nombre	apellido	ciudad
3	John	Williams	San Diego
4	Edward	Harris	Miami

¿Por qué se utiliza Sharding?

Cuando se fragmenta una tabla más grande, se puede almacenar los nuevos fragmentos de datos, denominados **fragmentos lógicos**, en varios nodos para lograr una ***escalabilidad horizontal*** y un ***rendimiento mejorado***. Una vez que el fragmento lógico se almacena en otro nodo, se denomina **fragmento físico**.

Algunas ventajas de utilizar sharding:

- ➔ Se pueden lograr mejoras de rendimiento.
- ➔ Facilidad para escalar bases de datos.
- ➔ Permite el procesamiento masivo de forma paralela, aprovechando los clusters.
- ➔ Se puede ofrecer niveles más altos de disponibilidad.

Diferencia entre fragmentar y particionar

- Ambos conceptos tratan de dividir un gran conjunto de datos en subconjuntos más pequeños, pero:
  - **Fragmentar** implica que los datos se **distribuyen** en *varias computadoras*, mientras que la partición no.
  - **Particionar** consiste en **agrupar** subconjuntos de datos dentro de *una sola instancia* de base de datos.

En algunos casos son conceptos que se utilizan como sinónimos, pero realmente hacen referencia a cosas diferentes.

Gracias!

IT BOARDING

BOOTCAMP