

Contexto:

Aunque no es la primera forma estandarizada de organizar información -pero sí la más popular-, las **bases de datos relacionales** (que consisten en tablas y relaciones, y tiene capacidad de **escalar verticalmente**), hacen su primera aparición en los años 70.

Sin embargo, 10 años antes, aparecían las **bases de datos no relacionales**, y aunque aparecieron "primero" el término NoSQL (Not Only SQL) se popularizó en las primeras décadas del 2000.

Esta solución es ampliamente usada en empresas grandes que manejan enormes cantidades de datos -estructurados, semi estructurados y no estructurados- y requieren sistemas que soporten al menos 50 PB sin colapsar.

Este tipo de implementación tiene capacidad de **escalar horizontalmente** (aunque con soluciones tradicionales, cada máquina que se añade al sistema, debe contar con una licencia = costoso).

Además ofrecen una solución al "**impedance mismatch**" ya que, las bbdd relacionales normalmente necesitan transformar los objetos definidos en el sistema, en filas y columnas, lo que choca/rompe características de la POO, como herencia, encapsulación, tipos complejos... Esto no sucede necesariamente en bbdd NO relacionales, ya que la información puede guardarse con "estructuras" que mantienen el formato.

- El ejemplo:
- En una base SQL, tendrías que crear varias tablas (una para persona, otra para dirección, relacionarlas...).
  - En una base NoSQL como MongoDB, guardás ese objeto tal cual como un documento.

```
# Objeto en tu código
persona = {
  "nombre": "Orianna",
  "edad": 25,
  "direccion": {
    "calle": "Falsa 123",
    "ciudad": "Madrid"
  }
}
```

ACID vs. CAP vs. BASE

Para asegurar la integridad, consistencia y/o disponibilidad de los datos, y dependiendo del modelo/sistema/implementación que se use para almacenar y tratar los mismos (bbdd relacionales, bbdd no relacionales, sistemas distribuidos, etc.), existen modelos de garantía:

ACID --> para bbdd relacionales

CAP --> para sistemas distribuidos (como muchos nodos NoSQL interconectados a través de la red)

- C-consistencia
- A-disponibilidad
- P-tolerancia a particionado (el sistema sigue funcionando aunque experimente fallos en la comunicación con otros nodos)

-**Teorema Brewer/CAP**- En un sistema distribuido no pueden existir las 3 garantías al mismo tiempo, solo se pueden garantizar 2, así que hay que escoger.

BASE --> para bbdd no relacionales  
Es la versión ACID de estos sistemas.

- BA**-disponibilidad básica
- S**-estado suave  
(Significa que los **nodos** de estos sistemas **NO** tienen un **estado fijo**, ya que por los propios mecanismos de distribución de la información, la información de un nodo puede cambiar en segundo plano, sin interacción directa del usuario. Esto pasa porque un nodo en otra parte del sistema, se actualiza y propaga la información). (Es una **consecuencia natural** de los **sistemas distribuidos**).
- E**-consistencia eventual (eventualmente los datos se sincronizarán y serán consistentes para todo el sistema)



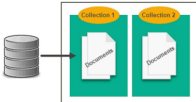
NoSQL:

- Sus características principales:
- No trabaja con modelos relacionales
  - Esta pensado para aplicarse en sistemas distribuidos (clústeres)
  - Se dice que es Schemaless (es decir, que se pueden insertar/guardar datos sin previamente definir alguna estructura para ellos)

- Sus ventajas principales:
- Ya que no requiere esquemas fijos o estrictos, facilita las migraciones, y el desarrollo en general es más ágil
  - Amplio rango de datos a manejar
  - El coste asociado a escalar este tipo de sistemas, suele ser menor en comparación a escalar verticalmente los sistemas tradicionales.
  - Cuenta con garantías de disponibilidad para millones de usuarios y tráfico continuo
  - Es perfecto para nuevos paradigmas de aplicaciones como los modelos implementados por IA's o IoT

Tipos de bbdd NoSQL:

**Bases de datos documentales**



Almacena datos semi-estructurados (suelen ir en formatos JSON, BSON, XML, etc.).  
Permite anidar documentos (óptimo para objetos que vienen de POO).  
Cuenta con indexación selectiva, que significa que aunque el documento es flexible, se pueden crear índices para la búsqueda.

Ejemplos de implementación: MongoDB AWS DynamoDB

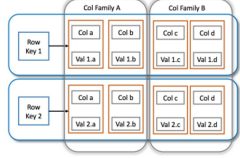
**Bases de datos Clave-Valor**

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2_01/01/2015
K5	3,ZZZ,5623

Consiste en el clásico almacenamiento de parejas de clave-valor, con dos columnas, donde se genera una tabla de *hashes* o diccionarios.

Ejemplos de implementación: Redis AWS DynamoDB

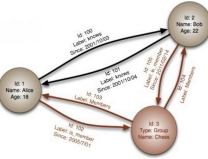
**Bases de datos Columna-Amplia (Wide-Column)**



Aunque suenan similares a las bbdd relacionales, aquí los datos se organizan en columnas (no en filas), lo cual es óptimo para consultas analíticas.  
No cuenta con un esquema fijo, por lo que es muy flexible.  
Y cada columna puede almacenarse por separado en el disco, lo que mejora muchísimo el rendimiento.

Ejemplos de implementación: Canssandra Google Bigtable

**Bases de datos Grafos**



Este modelo logra representar y consultar relaciones entre los datos de manera eficiente usando los nodos (como objetos/entidades) y las aristas (como sus relaciones).

Ejemplos de implementación: Neo4J AWS Neptune

**Polyglot Persistence**

Es la estrategia (que viene de Polyglot programming -usar distintos lenguajes de programación dentro de un mismo sistema, según se requiera-) que sugiere usar para cada parte del sistema la bbdd que mejor se adapte a la necesidad concreta.

Por ejemplo:

- MongoDB para documentos
- Cassandra para analítica
- Neo4j para relaciones
- PostgreSQL para transacciones financieras