Módulo 4.3: Bases de datos no Relacionales

MSc. Carlos Cedeño

Profundizando en Bases de Datos No Relacionales (NoSQL)

Las bases de datos NoSQL nacieron de la necesidad de empresas como Google, Amazon y Facebook de manejar un volumen, velocidad y variedad de datos (las "3 V" del Big Data) que el modelo relacional tradicional no podía gestionar de manera eficiente. Para IoT, donde estas tres características son la norma, entender NoSQL es fundamental.

1. La Filosofía NoSQL: Flexibilidad y Escala

A diferencia del rígido mundo de las tablas y esquemas predefinidos, la filosofía NoSQL se basa en dos conceptos clave:

- **Esquema Flexible (o Inexistente):** No necesitas definir cada columna y su tipo de dato antes de empezar. Puedes insertar un "documento" de un dispositivo con tres campos (temperatura, humedad, batería) y, justo después, insertar otro de un modelo más nuevo con cinco campos (añadiendo `co2` y `gps`). La base de datos lo acepta sin problemas. Esto otorga una agilidad inmensa al desarrollo, especialmente en proyectos de IoT donde los tipos de sensores y los datos que envían evolucionan constantemente.

- **Escalabilidad Horizontal:** En lugar de hacer un servidor más grande y caro (escalabilidad vertical), los sistemas NoSQL están diseñados para escalar añadiendo más servidores más económicos a un clúster (escalabilidad horizontal). Esto permite un crecimiento casi ilimitado y distribuye la carga, lo que es esencial para manejar millones de dispositivos conectados.

2. Los Modelos de Datos NoSQL: Un Abanico de Opciones

NoSQL no es un solo tipo de base de datos, sino una familia de sistemas. Cada modelo está optimizado para un tipo de problema específico.

a) Bases de Datos Documentales (Document Stores)

Este es el tipo de NoSQL más popular y un excelente punto de partida. Almacena la información en **documentos**, que suelen ser objetos JSON (o BSON, su equivalente binario). Un documento contiene toda la información de una entidad, de forma jerárquica.

- **Estructura:** Piensa en un objeto de programación. Un solo documento puede contener pares clave-valor, arrays y otros documentos anidados.

```
// Ejemplo de documento para un dispositivo en MongoDB
{
  "_id": "60d5ec49e7b2e7e3f8b0e6a1",
  "dispositivo_id": "TEMP-SEN-0042",
  "modelo": "AgriSensor-Pro",
  "ubicacion": {
    "coordenadas": [-78.4678, -0.1807],
    "sector": "Invernadero C"
  },
  "firmware_version": "v2.1.3",
  "estado": "activo",
  "ultimas_lecturas": [
    {"timestamp": "2025-07-15T10:00:00Z", "temperatura": 25.2, "humedad": 65},
    {"timestamp": "2025-07-15T10:01:00Z", "temperatura": 25.3, "humedad": 64.8}
  ]
}
```

- **¿Por qué en IoT?** Es perfecto para almacenar el estado completo y los datos recientes de un dispositivo en una sola lectura. La flexibilidad del esquema permite que diferentes dispositivos envíen datos distintos sin requerir cambios en la base de datos.
- **Líder del Mercado: MongoDB**.

b) Bases de Datos de Series Temporales (Time-Series Databases)

Son un tipo especializado de NoSQL y **el más importante para la telemetría en IoT**. Están ultra-optimizadas para manejar datos indexados por tiempo.

- **Estructura:** Están diseñadas para una ingesta masiva de datos (escrituras a muy alta velocidad) y para realizar consultas sobre rangos de tiempo de forma extremadamente eficiente.

- **Características Clave en IoT:**
  - **Alta Compresión:** Comprimen los datos de manera muy eficiente, reduciendo los costos de almacenamiento.
  - **Políticas de Retención (Retention Policies):** Permiten eliminar automáticamente datos antiguos (ej. borrar datos con más de un año de antigüedad).
  - **Downsampling (Submuestreo):** Pueden procesar y agregar datos automáticamente. Por ejemplo, tomar datos de cada minuto y crear promedios por hora para el almacenamiento a largo plazo, descartando los datos de alta granularidad más antiguos.
- **Líderes del Mercado: InfluxDB**, **TimescaleDB** (una extensión de PostgreSQL que le da superpoderes de series de tiempo).

c) Bases de Datos Clave-Valor (Key-Value Stores)

Es el modelo más simple. Cada elemento es un par que consiste en una clave única y un valor. El "valor" puede ser cualquier cosa: una cadena de texto, un número, un objeto JSON, una imagen.

- **Estructura:** `clave -> valor`

```
"dispositivo:0042:estado"  -> "activo"
"dispositivo:0042:bateria" -> "98%"
```

- **¿Por qué en IoT?** Por su velocidad extrema. Son ideales para almacenar datos que cambian constantemente y necesitan ser accedidos al instante, como el estado en tiempo real de un dispositivo, cachés de datos o para gestionar sesiones de usuario.
- **Líder del Mercado: Redis**.

d) Bases de Datos Columnares (Column-Family Stores)

Estas bases de datos almacenan datos en columnas en lugar de filas. Son extremadamente eficientes para consultas analíticas sobre grandes volúmenes de datos.

- **Estructura:** Imagina una tabla con miles de millones de filas y cientos de columnas. Si solo quieres analizar una columna (ej. `temperatura`), una base de datos tradicional tendría que leer todas las filas completas. Una base de datos columnar solo lee los datos de la columna `temperatura`, haciéndola muchísimo más rápida para este tipo de trabajo.
- **¿Por qué en IoT?** Perfectas para analítica de Big Data. Cuando quieres calcular la temperatura promedio de todos tus sensores en el último mes, una base de datos columnar es la herramienta ideal.
- **Líder del Mercado: Apache Cassandra**.

3. El Teorema CAP: El Trilema de los Sistemas Distribuidos

El Teorema CAP es un principio fundamental que establece que en un sistema de computación distribuido es imposible garantizar simultáneamente las tres siguientes propiedades:

1. **Consistencia (Consistency):** Todos los nodos del clúster ven los mismos datos al mismo tiempo. Una lectura siempre devolverá la escritura más reciente.
2. **Disponibilidad (Availability):** El sistema siempre está operativo y responde a las solicitudes (aunque la respuesta pueda no ser la versión más actualizada de los datos).
3. **Tolerancia a Particiones (Partition Tolerance):** El sistema continúa funcionando incluso si la comunicación entre los nodos se interrumpe (hay una "partición" en la red).

En la práctica, la tolerancia a particiones (P) es obligatoria en cualquier sistema distribuido real. Por lo tanto, los diseñadores de bases de datos deben **elegir entre consistencia (CP) o disponibilidad (AP)**.

- **Bases de datos CP (Consistencia + Tolerancia a Particiones):** Prefieren dar un error o no responder antes que devolver datos incorrectos. MongoDB y Redis pueden configurarse en este modo.
- **Bases de datos AP (Disponibilidad + Tolerancia a Particiones):** Prefieren responder siempre, aunque sea con datos ligeramente desactualizados. Cassandra es un claro ejemplo.

Para IoT, la elección depende del caso de uso: ¿Es más grave que un sensor no pueda escribir su dato (baja disponibilidad) o que un dashboard muestre una temperatura de hace 2 segundos en lugar de la actual (baja consistencia)?

5. Consultas en el Mundo NoSQL

No existe un lenguaje universal como SQL. Cada base de datos tiene su propia forma de ser consultada, generalmente a través de una API.

- **MongoDB** usa MQL (Mongo Query Language), que tiene una sintaxis basada en JSON: `db.dispositivos.find({ "estado": "activo" })`.
- **InfluxDB** tiene InfluxQL (similar a SQL) y Flux (un lenguaje de scripting de datos más potente).
- **Redis** utiliza comandos simples como `GET dispositivo:0042:estado`.

El enfoque no está en `JOIN`s complejos, sino en recuperar un agregado de datos completo (como un documento) que ya contiene toda la información necesaria.