Esquema Resumen Estructurado

I. Fundamentos y Evolución del Paradigma

- **Definición:** NoSQL ("Not Only SQL") denota sistemas de gestión de datos no relacionales, distribuidos y diseñados para modelos de datos flexibles.
- **Drivers Tecnológicos (Las 3 Vs del Big Data):**
    - **Volumen:** Insuficiencia del escalado vertical (*scale-up*) de RDBMS; necesidad de escalado horizontal (*scale-out*) en *commodity hardware*.
    - **Variedad:** Datos no estructurados/semiestructurados (JSON, logs, grafos) incompatibles con esquemas rígidos.
    - **Velocidad:** Requisitos de ingestión y procesamiento en tiempo real (baja latencia).
- **Evolución Histórica:**
    - 1998 (Strozzi): NoSQL como base de datos sin interfaz SQL.
    - 2009 (Evans): NoSQL como respuesta arquitectónica a la Web 2.0 y el Big Data.

II. Taxonomía de Bases de Datos NoSQL

El documento clasifica los sistemas según su modelo de datos y algoritmos de almacenamiento:

A. Bases de Datos Documentales

- **Unidad de datos:** Documentos autocontenidos (JSON, BSON, XML).
- **Características:** Esquema flexible (*schema-less* o *schema-on-read*), jerarquías anidadas.
- **Ejemplos:**
    - **MongoDB:** Líder de mercado, usa BSON, Sharding y Replica Sets. Fuerte en desarrollo ágil.
    - **Couchbase:** Arquitectura *Memory-First*, lenguaje SQL++ (N1QL), fuerte consistencia eventual configurable.

B. Almacenes Clave-Valor (Key-Value)

- **Unidad de datos:** Diccionario asociativo (Hash Map).
- **Características:** Simplicidad extrema, complejidad $O(1)$ en acceso por clave, opacidad del valor.
- **Ejemplos:**
    - **Redis:** En memoria (in-memory), estructuras de datos complejas (listas, sets, hyperloglogs). Persistencia opcional (RDB/AOF).
    - **DynamoDB:** Gestionada (AWS), *Serverless*, usa SSDs, consistente y altamente escalable.

C. Orientadas a Columnas (Wide-Column / Columnar)

- **Unidad de datos:** Familias de columnas. Almacenamiento físico por columnas, no por filas.
- **Características:** Escrituras eficientes, compresión alta, ideal para agregaciones OLAP y series temporales.
- **Ejemplos:**
    - **Apache Cassandra:** Arquitectura P2P (sin maestro), *tunable consistency*, escritura optimizada vía *Commit Logs* y *MemTables*.
    - **HBase:** Sobre HDFS (Hadoop), consistencia fuerte, ideal para acceso aleatorio a Big Data.

D. Bases de Datos de Grafos

- **Unidad de datos:** Nodos, Aristas (Relaciones) y Propiedades.
- **Características:** Relaciones como ciudadanos de primera clase. Adyacencia libre de índices (*Index-free adjacency*) para recorridos $O(k)$ donde k es el subgrafo vecinal, no el total.
- **Ejemplos:**
    - **Neo4j:** Nativa de grafos, cumplimiento ACID, lenguaje Cypher.

E. Motores de Búsqueda (Search Engines)

- **Base:** Apache Lucene (Índices invertidos).
- **Ejemplos:** Elasticsearch (Stack ELK) y Apache Solr. Optimizados para *full-text search* y análisis, no transaccionalidad.

III. Teorema CAP y Consistencia

En sistemas distribuidos, solo se pueden garantizar 2 de 3 simultáneamente:

1. **C (Consistency):** Todos los nodos ven la misma data al mismo tiempo.
2. **A (Availability):** Garantía de respuesta (incluso con datos obsoletos).
3. **P (Partition Tolerance):** Resiliencia ante cortes de comunicación.
    - *Sistemas CP:* Priorizan consistencia (e.g., HBase, MongoDB por defecto).
    - *Sistemas AP:* Priorizan disponibilidad (e.g., Cassandra, DynamoDB por defecto).

IV. Comparativa Crítica: SQL vs. NoSQL

Característica	SQL (Relacional)	NoSQL (Distribuido)
Esquema	Rígido (*Schema-on-write*)	Flexible (*Schema-on-read*)
Escalabilidad	Vertical (Costosa, límites físicos)	Horizontal (Elástica, *commodity hardware*)
Integridad	ACID (Atomicidad, Consistencia...)	BASE (Basic Availability, Soft-state, Eventual consistency)
Relaciones	JOINs complejos	Desnormalización, anidamiento o grafos explícitos

V. Guía de Selección (Heurísticas)

- **Usar SQL si:** Integridad referencial crítica (financiera), esquema estable, consultas complejas ad-hoc estandarizadas.
- **Usar Documental si:** Prototipado rápido, objetos complejos anidados, CMS.
- **Usar Clave-Valor si:** Caché, sesiones, contadores en tiempo real (baja latencia crítica).
- **Usar Columnar si:** Series temporales, IoT, logs masivos (escritura intensiva).
- **Usar Grafos si:** Detección de fraude, redes sociales, recomendadores (la relación aporta más valor que la entidad).