Lectura sesión Tecnologías de bases de datos

1. ¿Qué es un Sistema de Base de Datos?

por Kibernum Capacitación S.A.

¿Qué es un Sistema de Base de Datos?

Un sistema de base de datos es un conjunto organizado de datos y un conjunto de programas que permiten almacenar, modificar y extraer información de forma eficiente y segura.

Definición formal:

Es un sistema computacional que permite la gestión estructurada de información, proporcionando funcionalidades para almacenar, consultar, actualizar y borrar datos de forma controlada.

> Ejemplo simple: Piensa en una agenda digital que guarda nombres, teléfonos y correos. Esa agenda tiene un "sistema" que te permite buscar un contacto, agregar uno nuevo o borrar uno antiguo. Eso es, en miniatura, un sistema de base de datos.

Tipos de Sistemas de Base de Datos

Bases de Datos Relacionales (RDBMS)

Estas bases organizan los datos en tablas (filas y columnas), como una hoja de Excel. Cada fila es un registro y cada columna representa un campo o atributo.

Ejemplos de RDBMS:

- MySQL
- PostgreSQL
- Oracle
- Microsoft SQL Server

Caso de uso ideal: Cuando los datos tienen relaciones claras y reglas estrictas, como en un sistema bancario.

Bases de Datos NoSQL

NoSQL significa “Not Only SQL”. Estas bases están diseñadas para almacenar datos no estructurados o semiestructurados, de forma más flexible que las relacionales.

Tipos de NoSQL:

- Documentales (Ej: MongoDB)
- Clave-valor (Ej: Redis)
- Columnar (Ej: Cassandra)
- Grafos (Ej: Neo4j)

Caso de uso ideal: Cuando necesitas escalabilidad o manejar grandes volúmenes de datos que cambian rápido, como redes sociales, catálogos, logs de servidores, etc.

Comparativa y Casos de Uso

Característica	Relacional (RDBMS)	NoSQL
Estructura	Tablas	Flexible (JSON, clave-valor)
Esquema fijo	Sí	No necesariamente
Integridad de datos	Alta (ACID)	Variable (BASE)
Escalabilidad	Vertical (más recursos)	Horizontal (más nodos)
Ejemplo típico	ERP, bancos, e-commerce	Redes sociales, IoT, Big Data

Bases de Datos Documentales – MongoDB

¿Qué es MongoDB?

MongoDB es una base de datos NoSQL que guarda la información como si fuera un archivo JSON. En lugar de tablas y filas como en una base de datos tradicional, MongoDB usa colecciones y documentos.

Ideal para...

- Aplicaciones web modernas
- Tiendas online (e-commerce)
- Blogs, foros y redes sociales
- Cuando los datos son flexibles o cambian mucho

Ventajas principales:

- Permite guardar estructuras complejas (como objetos anidados).
- Muy flexible, no necesitas definir un esquema fijo.
- Se pueden hacer búsquedas avanzadas dentro de los documentos.

Estructura y Uso de MongoDB

Estructura: Documentos en formato JSON

Uso común: E-commerce, blogs, aplicaciones con datos flexibles

```
// Insertar un documento en la colección "usuarios"
db.usuarios.insertOne({
  nombre: "Juan Pérez",
  correo: "juan@gmail.com",
  edad: 30,
  direccion: {
    calle: "Calle Falsa 123",
    ciudad: "Santiago"
  }
});
```

Explicación:

- db.usuarios es como decir "colección de usuarios"
- insertOne agrega un nuevo usuario como un documento JSON
- Este formato permite guardar estructuras anidadas, como la dirección.

Bases de Datos Clave-Valor – Redis

¿Qué es Redis?

Redis (Remote Dictionary Server) es una base de datos clave-valor ultrarrápida que se guarda en memoria. Es como un diccionario gigante, donde cada palabra (clave) tiene un significado (valor).

Ideal para...

- Guardar datos que necesitas rápido (como la memoria RAM 🧠).
- Almacenar resultados temporales como:
  - Sesiones de usuario
  - Contadores
  - Configuraciones
  - Cachés para acelerar sitios web

Estructura: Claves únicas que apuntan a valores

Uso común: Caché, sesiones, contadores, configuración

```
# Guardar un valor
SET nombre "Miguel"

# Obtener el valor guardado
GET nombre
```

Explicación:

- SET guarda un valor bajo una clave específica
- GET recupera el valor asociado a esa clave
- Es extremadamente rápido y muy usado para almacenar información temporal

Bases de Datos Columnar – Cassandra

¿Qué es Cassandra?

Apache Cassandra es una base de datos NoSQL que guarda la información por columnas, en lugar de por filas como las bases de datos tradicionales. Es perfecta para trabajar con grandes cantidades de datos distribuidos, sin perder velocidad ni disponibilidad.

Ideal para...

- Big Data y analítica masiva
- Registros de eventos (logs)
- Datos en tiempo real (como IoT)
- Sistemas con escritura constante y rápida

Ventajas principales:

- Escalable horizontalmente (añades más nodos fácilmente)
- Consulta eficiente de columnas específicas
- Alta disponibilidad y sin punto único de falla

Estructura: Columnas agrupadas, no necesariamente uniformes

Uso común: Big Data, métricas, logs, IoT


```cql
-- Crear una tabla
CREATE TABLE empleados (
  id UUID PRIMARY KEY,
  nombre TEXT,
  cargo TEXT,
  salario DOUBLE
);

-- Insertar un empleado
INSERT INTO empleados (id, nombre, cargo, salario)
VALUES (uuid(), 'Laura', 'Analista', 75000.0);

-- Consultar todos los empleados
SELECT * FROM empleados;
```

- **Explicación**:
- Cassandra usa un modelo similar a SQL pero enfocado en columnas
- Ideal para almacenar grandes volúmenes y acceder a columnas específicas rápidamente

Este fragmento de código en Cassandra CQL crea una tabla llamada empleados con cuatro columnas: id (clave primaria de tipo UUID), nombre, cargo y salario. La estructura es similar al SQL tradicional, pero adaptada al modelo de datos por columnas que utiliza Cassandra. El comando **INSERT INTO** permite agregar un nuevo registro generando automáticamente un identificador único con uuid(), ideal para entornos distribuidos. Finalmente, la instrucción **SELECT * FROM** empleados; recupera todos los datos almacenados en la tabla. Este tipo de operaciones básicas son fundamentales para trabajar con Cassandra, una base de datos NoSQL diseñada para manejar grandes volúmenes de información de manera rápida y escalable

Bases de Datos de Grafos – Neo4j

¿Qué es Neo4j?

Neo4j es una base de datos de grafos, donde la información se organiza como si fuera una red. En lugar de tablas o documentos, usa nodos (entidades) y relaciones (conexiones).

Ideal para...

- Redes sociales
- Sistemas de recomendación
- Análisis de relaciones complejas (como conexiones entre personas o eventos)

Estructura: Nodos (entidades) y relaciones

Uso común: Redes sociales, recomendaciones, análisis de relaciones

```
// Crear nodos persona
CREATE (a:Persona {nombre: "Carlos"})
CREATE (b:Persona {nombre: "Ana"})

// Crear una relación entre ellos
CREATE (a)-[:AMIGO_DE]->(b)

// Consultar amistades
MATCH (p:Persona)-[:AMIGO_DE]->(amigo)
RETURN p.nombre, amigo.nombre
```

Explicación:

- Cada Persona es un nodo con propiedades
- La relación AMIGO_DE conecta dos nodos
- El lenguaje **Cypher** permite consultar relaciones como si dibujaras una red de conexiones

Comparativa Final de Tipos de Bases de Datos NoSQL

Tipo	Ejemplo	Formato	Caso de uso común
Documental	MongoDB	JSON	Datos estructurados y flexibles
Clave-valor	Redis	Clave → Valor	Caché, sesiones, tokens
Columnar	Cassandra	Column Families	Big Data, series de tiempo, métricas
Grafos	Neo4j	Nodos y relaciones	Redes, recomendaciones, grafos sociales

Principios Fundamentales, Teorema de CAP y Transaccionalidad

Principios ACID (RDBMS)

- **A** - Atomicidad: Todo o nada. Si un paso falla, nada se guarda.
- **C** - Consistencia: Los datos siempre cumplen las reglas.
- **I** - Aislamiento: Una transacción no afecta a otra en curso.
- **D** - Durabilidad: Lo que se guarda, permanece incluso tras fallos.

Ejemplo: Si haces una transferencia bancaria, o se hacen ambos movimientos (débito y crédito) o ninguno.

Principios BASE (NoSQL)

BASE es más laxo y está diseñado para sistemas más flexibles y distribuidos.

- **B** - Basically Available: El sistema siempre responde, aunque sea con datos antiguos.
- **S** - Soft State: El estado del sistema puede cambiar con el tiempo, incluso sin entradas.
- **E** - Eventually Consistent: Los datos se sincronizan eventualmente, no al instante.

Ejemplo: En una red social, publicas una foto. La ves tú al instante, pero tu amigo puede verla segundos después.

Teorema de CAP

El **Teorema de CAP**, también llamado **Teorema de Brewer**, dice que **una base de datos distribuida solo puede garantizar dos de estas tres propiedades al mismo tiempo**

- **C** - Consistency: Todos ven la misma versión del dato al mismo tiempo.
- **A** - Availability: El sistema siempre responde, incluso si hay fallos.
- **P** - Partition Tolerance: El sistema sigue funcionando, aunque haya fallos en la red que dividen al sistema.

Visual fácil: Imagina tres globos y solo puedes sostener dos con tus manos. Siempre tendrás que soltar uno.

Aplicación en NoSQL

- Availability + Partition Tolerance (AP): como Cassandra.
- Consistency + Partition Tolerance (CP): como MongoDB en configuración segura.
- Consistency + Availability (CA): no es posible si hay particiones de red (por eso CA puras no existen en sistemas distribuidos reales).

Transaccionalidad en Bases de Datos

Una transacción es una unidad de trabajo que debe ejecutarse completa y correctamente.

En bases de datos relacionales:

- Las transacciones siguen el modelo ACID, garantizando precisión y recuperación ante errores.
- Se controlan con instrucciones como: BEGIN, COMMIT, ROLLBACK.

En bases de datos NoSQL:

- Algunos sistemas NoSQL también permiten transacciones (por ejemplo, MongoDB desde la versión 4.0).
- Pero suelen tener un alcance limitado, y se implementan de forma distinta, priorizando disponibilidad y rendimiento.

Ejemplo ilustrado de transacción

Supón que haces una compra online:

1. Se descuenta tu saldo
2. Se reduce el stock del producto
3. Se genera una orden de compra

Si todo eso ocurre bien, COMMIT. Si algo falla, ROLLBACK, y todo se deshace como si nunca hubiera ocurrido.

Ejemplo ilustrado de transacción – Reserva de hotel

1. → Se verifica la disponibilidad de la habitación
2. → Se bloquea la habitación para evitar que otro la reserve
3. → Se realiza el pago con tarjeta
4. → Se envía una confirmación de reserva al correo del cliente

Si todo sale bien, se ejecuta el COMMIT y la reserva queda registrada. Si algo falla (no hay fondos, error de conexión, etc.), se hace ROLLBACK y todo se revierte, dejando la habitación disponible y sin cobros.