Actividad 1 - Módulo 4

Nombre: Carlos Saldivia Susperreguy

1. Exploración y Conceptualización

¿Qué es un sistema de base de datos?

Un sistema de base de datos es un conjunto organizado de datos junto con los programas que permiten almacenar, modificar y extraer información de forma eficiente y segura. Es un sistema computacional que facilita la gestión estructurada de información, proporcionando funcionalidades para crear, consultar, actualizar y eliminar datos de manera controlada.

Tipos de sistemas de bases de datos

Bases de Datos Relacionales (RDBMS)

Las bases de datos relacionales organizan la información en tablas con filas y columnas, similar a una hoja de cálculo. Cada fila representa un registro único y cada columna un atributo específico. Las tablas se relacionan entre sí mediante claves primarias y foráneas, creando un modelo estructurado y predecible.

Características principales:

- Estructura rígida y predefinida (esquema fijo)
- Uso del lenguaje SQL para consultas
- Garantías de integridad referencial
- Transacciones ACID completas

Bases de Datos NoSQL

NoSQL significa "Not Only SQL" y agrupa diferentes tipos de bases de datos diseñadas para manejar datos no estructurados o semiestructurados de manera más flexible que las relacionales.

Tipos principales:

- **Documentales**: Almacenan datos como documentos JSON (ej: MongoDB)
- **Clave-valor**: Estructura simple de diccionario (ej: Redis)
- **Columnar**: Organizan datos por columnas (ej: Cassandra)
- **Grafos**: Representan relaciones como redes (ej: Neo4j)

Principio ACID en bases de datos relacionales

ACID es un conjunto de propiedades que garantizan transacciones seguras y confiables en bases de datos relacionales:

- **Atomicidad**: Una transacción es una unidad indivisible - o se ejecuta completamente o no se ejecuta en absoluto
- **Consistencia**: Los datos siempre cumplen las reglas de integridad definidas
- **Aislamiento**: Las transacciones concurrentes no interfieren entre sí
- **Durabilidad**: Los cambios confirmados permanecen incluso ante fallos del sistema

Principio BASE en bases de datos NoSQL

BASE es un modelo más flexible diseñado para sistemas distribuidos:

- **Basically Available**: El sistema siempre responde, aunque sea con datos no actualizados
- **Soft State**: El estado del sistema puede cambiar con el tiempo sin intervención externa
- **Eventually Consistent**: Los datos se sincronizan eventualmente, no instantáneamente

Teorema CAP

El Teorema CAP establece que en un sistema distribuido solo se pueden garantizar dos de estas tres propiedades simultáneamente:

- **Consistency**: Todos los nodos ven la misma versión de los datos al mismo tiempo
- **Availability**: El sistema siempre responde a las solicitudes
- **Partition Tolerance**: El sistema continúa operando aunque haya fallos de red

¿Qué es la transaccionalidad en bases de datos?

La transaccionalidad es la capacidad de agrupar varias operaciones en una unidad de trabajo que se ejecuta completamente o no se ejecuta en absoluto. Una transacción garantiza que el sistema pase de un estado consistente a otro estado consistente.

Proceso típico:

1. **BEGIN**: Inicia la transacción
2. **Operaciones**: Se ejecutan las acciones requeridas
3. **COMMIT**: Confirma todos los cambios si todo es exitoso
4. **ROLLBACK**: Revierte todos los cambios si algo falla

2. Representación Comparativa

Aspecto	*RDBMS*	*NoSQL*
Estructura de datos	Tablas con filas y columnas	Flexible (JSON, clave-valor, grafos)
Escalabilidad	Vertical (más recursos al servidor)	Horizontal (más servidores)
Transacciones	ACID completo	Limitadas o eventualmente consistentes
Esquema	Fijo y predefinido	Flexible o sin esquema
Consultas	SQL estandarizado	APIs específicas o lenguajes propios
Integridad	Referencial estricta	Responsabilidad de la aplicación
Uso común	Sistemas financieros, ERP, CRM	Aplicaciones web, Big Data, IoT
Ejemplos	PostgreSQL, MySQL, Oracle	MongoDB, Redis, Cassandra, Neo4j
Principios teóricos	ACID	BASE
Consistencia	Inmediata	Eventual
Disponibilidad	Puede verse afectada	Prioridad alta
Rendimiento	Optimizado para consultas complejas	Optimizado para operaciones simples

3. Ejemplos Ilustrados

Ejemplo 1: Transacción en SQL (PostgreSQL)

```sql
-- Estructura de la tabla
CREATE TABLE cuentas (
    id INT PRIMARY KEY,
    nombre VARCHAR(100),
    saldo DECIMAL(10,2)
);

-- Transacción: Transferir 100 desde cuenta 1 a cuenta 2
START TRANSACTION;

UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;

UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2;

-- Si ambas operaciones fueron exitosas:
COMMIT;

-- Si ocurre un error en cualquier paso:
-- ROLLBACK;
```

Ejemplo 2: Inserción de documento en MongoDB

```javascript
// Insertar un usuario con información anidada
db.usuarios.insertOne({
  nombre: "Carlos Saldivia",
  email: "carlos.saldivia@kibernum.cl",
  edad: 30,
  direccion: {
    calle: "Calle falsa 1234",
    ciudad: "Santiago",
    pais: "Chile"
  },
});
```

Reflexión Final

¿En qué tipo de proyecto usarías una base de datos relacional?

Utilizaría una base de datos relacional en proyectos que requieren transacciones críticas, integridad de datos estricta y relaciones complejas entre entidades. Por ejemplo, en un sistema bancario, una plataforma de e-commerce, un sistema de gestión hospitalaria o un ERP empresarial.

¿En qué contexto optarías por una NoSQL?

Optaría por NoSQL cuando necesito escalabilidad horizontal, manejo de grandes volúmenes de datos, o cuando la estructura de datos es variable. Cuando la velocidad de desarrollo y la flexibilidad son más importantes que la consistencia inmediata. Por ejemplo, en una aplicación de redes sociales o un sistema de logs y métricas.

¿Qué papel juega el teorema CAP en estas decisiones?

El teorema CAP es fundamental para entender las limitaciones y compromisos de cada tecnología. Me ayuda a decidir qué propiedades son más críticas para mi aplicación específica. Nos dice que no existe una solución perfecta y que debo evaluar según los requisitos del proyecto.

Si necesito consistencia estricta (como en sistemas financieros), elegiré tecnologías que prioricen Consistency sobre Availability. Si requiero alta disponibilidad (como en redes sociales), optaré por sistemas que privilegien Availability y Partition Tolerance, aceptando consistencia eventual.