Evaluación Modular - Módulo 4

Nombre: Carlos Saldivia Susperreguy

1. Comparativa de Tecnologías de Bases de Datos

a) **Tabla Comparativa:**

Criterio	Relacionales (RDBMS)	No Relacionales (NoSQL)
Modelo de datos	Tablas con filas y columnas	Flexible (documentos, clave-valor, grafos, columnas)
Ejemplo de motor	PostgreSQL, MySQL	MongoDB, Cassandra, Neo4j, Redis
Cumplimiento de ACID	Completo	Limitado o eventual
Flexibilidad del esquema	Fijo y predefinido	Flexible o sin esquema
Escalabilidad principal	Vertical (más recursos al servidor)	Horizontal (más servidores)

b) **Teorema de CAP:**

El teorema de CAP establece que en un sistema distribuido solo se pueden garantizar dos de estas tres propiedades simultáneamente: Consistency, Availability y Partition Tolerance.

Este teorema influye directamente en la elección de bases de datos NoSQL según las prioridades del proyecto:

- Si se requiere consistencia estricta (sistemas financieros), se priorizará Consistency sobre Availability;
- Si se necesita alta disponibilidad (redes sociales), se optará por sistemas que privilegien Availability y Partition Tolerance, aceptando consistencia eventual.

2. Optimización en RDBMS

a) Buenas prácticas para optimización SQL:

- Crear índices estratégicos: Implementar índices en columnas utilizadas frecuentemente en WHERE, JOIN y ORDER BY
- Evitar SELECT *: Seleccionar únicamente las columnas necesarias para reducir transferencia de datos

b) Caso:

```
SELECT * FROM pedidos WHERE cliente_id = 1001 AND estado = 'entregado';
```

Índice recomendado:

```
CREATE INDEX idx_cliente_estado ON pedidos(cliente_id, estado);
```

Justificación: Este índice compuesto optimiza ambas condiciones del WHERE, permitiendo que el motor de base de datos filtre eficientemente por cliente_id primero y luego por estado, reduciendo significativamente el tiempo de búsqueda.

3. Diseño Lógico en NoSQL

a) **Modelo seleccionado:** Document-Oriented (MongoDB)

b) **Justificación y esquema:**

c) Es ideal porque los artículos científicos tienen estructura heterogénea con campos variables (algunos artículos pueden tener múltiples autores, diferentes tipos de archivos adjuntos, campos opcionales como DOI).
MongoDB permite almacenar toda la información relacionada en un solo documento, facilitando consultas rápidas y escalabilidad horizontal.

d) Ejemplo de cómo almacenarías un artículo


```
{
  "_id": "articulo123",
  "titulo": "Un enfoque estadístico para la ingeniería de datos",
  "autores": [
    {
      "nombre": "Mario Inostroza",
      "afiliacion": "Universidad de Kibernum"
    },
    {
      "nombre": "Carlos Saldivia",
      "afiliacion": "Instituto Nacional de Estadística"
    }
  ],
  "palabras_clave": ["estadística", "ingeniería de datos"],
  "año": 2023,
  "tipo_publicacion": "revista",
  "archivo": {
    "url": "https://repositorio.universidad.cl/articulo123.pdf",
    "formato": "PDF"
  }
}
```

4. MongoDB

a) Insertar documento:

```
db.productos.insertOne({
  nombre: "EcoBotella",
  categoria: "reciclaje",
  stock: 500
})
```

b) Consultar productos con stock mayor a 100:

```
db.productos.find({ stock: { $gt: 100 } })
```

c) Actualizar stock de "EcoBotella":

```
db.productos.updateOne(
  { nombre: "EcoBotella" },
  { $set: { stock: 300 } }
)
```

5. Cassandra

a) **Dos ventajas técnicas frente a RDBMS:**

- **Escalabilidad horizontal automática**: Cassandra puede escalar agregando nodos sin interrupciones, distribuyendo la carga automáticamente
- **Alta disponibilidad sin punto único de fallo**: Arquitectura peer-to-peer donde todos los nodos son iguales, eliminando puntos únicos de fallo

b) **Keyspace en Cassandra:**

Un keyspace es el equivalente a una base de datos en Cassandra. Es un contenedor lógico donde se crean y almacenan las tablas. Su función principal es definir la estrategia de replicación y el factor de replicación, determinando cuántas copias de cada dato se almacenarán en el clúster y cómo se distribuirán entre los nodos.

c) **Declaración de tabla para ventas:**

```
CREATE TABLE ventas (
    id_venta UUID PRIMARY KEY,
    fecha timestamp,
    cliente text,
    total decimal
);
```

6. DynamoDB

a) Operación para insertar el ítem:
PutItem: Esta operación inserta o reemplaza un ítem completo.
Consulta

```
aws dynamodb put-item \
  --table-name Productos \
  --item '{
    "producto_id": {"S": "A123"},
    "nombre": {"S": "Panel Solar"},
    "categoria": {"S": "energia"},
    "precio": {"N": "129990"},
    "stock": {"N": "30"}
  }'
```

b) Consulta por categoría y condición de eficiencia:
Operación: Query con índice secundario global (GSI)
Condición necesaria: Para que la búsqueda por categoría sea eficiente, se debe crear un Global Secondary Index (GSI) donde "categoria" sea la partition key. Sin este índice, tendríamos que usar Scan (operación costosa que recorre toda la tabla).

Crear Tabla con GSI:

```
aws dynamodb create-table \
  --table-name Productos \
  --attribute-definitions \
    AttributeName=producto_id,AttributeType=S \
    AttributeName=categoria,AttributeType=S \
  --key-schema \
    AttributeName=producto_id,KeyType=HASH \
  --global-secondary-indexes \
    IndexName=CategoriaIndex,KeySchema=[{AttributeName=categoria,KeyType=HASH}],Projection={ProjectionType=ALL},ProvisionedThroughput={ReadCapacityUnits=5,WriteCapacityUnits=5} \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

Consulta:

```
aws dynamodb query \
  --table-name Productos \
  --index-name CategoriaIndex \
  --key-condition-expression "categoria = :cat" \
  --expression-attribute-values '{
    ":cat": {"S": "energia"}
  }'
```