1. Introducción

Este documento describe un enfoque estructurado para migrar datos de usuarios y CIS desde una base de datos relacional (MySQL) a una base de datos NoSQL basada en documentos (MongoDB). El objetivo es garantizar una transición eficiente, minimizando el tiempo de inactividad y preservando la integridad de los datos durante el proceso.

---

2. Estrategias de Migración

La migración puede abordarse mediante dos enfoques principales: Big Bang e Incremental. El enfoque Big Bang implica realizar la migración completa en una única ventana de tiempo, lo que permite una implementación rápida pero conlleva un alto riesgo de tiempo de inactividad. Por otro lado, el enfoque Incremental divide el proceso en etapas, como migrar tablas o lotes de datos secuencialmente. Este método reduce el impacto operacional pero introduce complejidad al requerir sincronización constante entre las bases de datos durante la transición.

Para ejecutar la migración, se recomiendan herramientas como AWS Database Migration Service (DMS) para replicación continua, scripts personalizados en Python (usando bibliotecas como PyMySQL y PyMongo) para transformaciones específicas, y el MongoDB Connector for BI para análisis previo de datos.

---

3. Diseño del Esquema en MongoDB

3.1 Colecciones y Documentos

En MongoDB, pueden ser tres colecciones principales: users, topics, e ideas, esta última con votos embebidos para optimizar las consultas.

Ejemplo de Documento en `users`:

```
{
  "_id": ObjectId("6548a3b7d1f94a3d3a7e8f1c"),
  "email": "maria@ejemplo.com",
  "firstName": "María",
```

```
  "lastName": "López",
  "role": "user",
  "createdAt": ISODate("2023-10-01T08:00:00Z"),
  "username": "maria_lopez"
}
```

Ejemplo de Documento en `topics`:

```
{
  "_id": ObjectId("6548a3b7d1f94a3d3a7e8f2d"),
  "title": "Inteligencia Artificial",
  "description": "Soluciones innovadoras en IA",
  "createdAt": ISODate("2023-09-15T10:30:00Z")
}
```

Ejemplo de Documento en `ideas` con Votos Embebidos:

```
{
  "_id": ObjectId("6548a3b7d1f94a3d3a7e8f3e"),
  "title": "Sistema de votación en tiempo real",
  "description": "Implementar un sistema...",
  "topicId": ObjectId("6548a3b7d1f94a3d3a7e8f2d"),
  "votes": [
    {
      "voterId": ObjectId("6548a3b7d1f94a3d3a7e8f1c"),
      "voterType": "user",
      "createdAt": ISODate("2023-10-05T14:20:00Z")
    }
  ],
  "createdAt": ISODate("2023-10-02T09:15:00Z")
}
```

3.2 Decisiones de Modelado

- Votos Embebidos en Ideas: **Almacenar los votos como subdocumentos dentro de `ideas` elimina la necesidad de operaciones de unión (`JOIN`), mejorando el rendimiento al recuperar una idea con sus votos asociados.**
- Referencias para Relaciones Cruzadas: **Campos como `topicId` en `ideas` utilizan `ObjectId` para mantener referencias a otras colecciones, asegurando integridad referencial.**
- Desnormalización Opcional: **Si las consultas frecuentes requieren el título del tema en las ideas, se puede incluir `topicTitle` como campo desnormalizado.**

---

4. Estrategia de Migración

4.1 Extracción y Transformación de Datos

Se emplearán scripts en Python para extraer datos de MySQL, transformarlos al formato de documentos JSON, y cargarlos en MongoDB.

Conexión y Extracción desde MySQL:

```
import pymysql
import pandas as pd
```

Configurar conexión a MySQL

```
conn_mysql = pymysql.connect(
    host='localhost',
    user='root',
    password='*****',
    db='cis_db'
)
```

Extraer datos

```
df_users = pd.read_sql('SELECT * FROM users', conn_mysql)
df_topics = pd.read_sql('SELECT * FROM topics', conn_mysql)
```

```
df_ideas = pd.read_sql('SELECT * FROM ideas', conn_mysql)
df_votes = pd.read_sql('SELECT * FROM votes', conn_mysql)
```

Transformación y Carga en MongoDB:

```
from pymongo import MongoClient
from bson import ObjectId

# Configurar conexión a MongoDB
client = MongoClient('mongodb://localhost:27017')
db = client['cis_mongodb']

# Migrar usuarios
users_collection = db['users']
users_collection.insert_many(df_users.to_dict('records'))

# Migrar topics
topics_collection = db['topics']
topics_collection.insert_many(df_topics.to_dict('records'))

# Migrar ideas con votos embebidos
ideas_collection = db['ideas']
for _, idea in df_ideas.iterrows():
    votes = df_votes[df_votes['IdealId'] == idea['Id']]
    votes_list = []
    for _, vote in votes.iterrows():
        vote_doc = {
            "voterId": ObjectId(vote['VoterId']),
            "voterType": vote['VoterType'],
            "createdAt": vote['CreatedAt']
        }
        votes_list.append(vote_doc)
    idea_doc = {
        "title": idea['Title'],
        "description": idea['Description'],
        "topicId": ObjectId(idea['TopicId']),
        "votes": votes_list,
```

```
        "createdAt": idea['CreatedAt']
    }
    ideas_collection.insert_one(idea_doc)
```

4.2 Validación Post-Migración

Se deberían hacer verificaciones para asegurar la integridad de los datos:

Conteo de Registros:

```
assert users_collection.count_documents({}) == df_users.shape[0], "Error en  usuarios"
assert ideas_collection.count_documents({}) == df_ideas.shape[0], "Error en ideas"
```

Consistencia de Relaciones:

```
    # Verificar que los votos tengan un voterId válido
    for idea in ideas_collection.find():
        for vote in idea['votes']:
            assert users_collection.find_one({"_id": vote['voterId']}) is not None, "Voto sin usuario"
```

---

5. Optimizaciones y Buenas Prácticas

- Índices: **Crear índices en campos de consulta frecuente como `users.email`, `ideas.topicId`, y `topics.title`.**
- Sharding: **Para escalar horizontalmente en caso de un crecimiento masivo de votos o ideas.**
- Agregaciones: **Utilizar pipelines de MongoDB para generar reportes, como "votos por tema" o "actividad de usuarios".**

---

6. Riesgos y Mitigación

- Pérdida de Relaciones: **Validar referencias con scripts automatizados durante la migración.**
- Rendimiento en Consultas: **Perfilar consultas usando `explain()` en MongoDB y ajustar índices según resultados.**

- Tiempo de Inactividad: **Implementar replicación en MongoDB y realizar la migración en horarios de baja actividad.**