# Informe Técnico: Proyecto GreenScape

David Michel Batista  Barbaro Yoel Martínez González

Asignatura: Bases de Datos
Grupo: D_211

## 1. Introducción

El presente informe detalla las decisiones técnicas, arquitectónicas y de diseño adoptadas por nuestro equipo para la implementación de las funcionalidades avanzadas de la plataforma `GreenScape`. El enfoque principal ha sido la ingeniería de datos híbrida, combinando la robustez de una base de datos relacional (`MySQL`) para transacciones comerciales y estructura de usuarios, con la flexibilidad de una base de datos documental (`MongoDB`) para gestión de contenido no estructurado.

## 2. Comparación de Diseños para Conversaciones en Comentarios

Para abordar el requerimiento de permitir conversaciones anidadas en los comentarios, implementamos y evaluamos dos enfoques distintos. A continuación, presentamos nuestro análisis comparativo, incluyendo los resultados de las pruebas de rendimiento.

### 2.1. Modelo Relacional (MySQL)

**Implementación:** Modificamos la tabla `Comentar` para incluir una columna `IDPadre` que actúa como una clave foránea recursiva referenciando a la misma tabla.

- **Consulta:** Para reconstruir un hilo completo, utilizamos **CTEs Recursivas** (`WITH RECURSIVE`). Esto nos permite iterar desde el comentario raíz hacia abajo a través de todos los niveles de profundidad.
- **Integridad:** Configuramos `ON DELETE CASCADE` en la clave foránea `IDPadre`.

**Ventajas:**

1. **Integridad Referencial:** El motor de base de datos garantiza que no existan comentarios huérfanos. Si se borra un padre, la cascada elimina automáticamente a los hijos.
2. **Eficiencia Teórica en Escritura:** Insertar un nuevo comentario es una operación atómica simple (`INSERT`) que no requiere cálculos previos complejos. **No obstante, en nuestras pruebas, el costo transaccional del `COMMIT` resultó en un tiempo de escritura mayor que el del modelo no relacional.**

**Desventajas:**

1. **Complejidad de Lectura:** A medida que la profundidad del hilo crece, la consulta recursiva se vuelve computacionalmente costosa, ya que el motor debe realizar múltiples `JOIN` internos en tiempo de ejecución, reflejado en nuestro tiempo de prueba.
2. **Escalabilidad:** En escenarios de alto tráfico de lectura (típico de redes sociales), la recursividad constante puede saturar el servidor de base de datos.

## 2.2. Modelo No Relacional (MongoDB)

**Implementación:** Diseñamos un modelo híbrido enriquecido. Cada documento de comentario almacena su `IDPadre`, pero decidimos desnormalizar datos clave añadiendo dos campos adicionales: `IDRaiz` (el ID del comentario que inició el hilo) y `Nivel` (profundidad en el árbol).

- **Consulta:** La recuperación del hilo no requiere recursividad. Realizamos una única consulta `find({ 'IDRaiz': ... })` y ordenamos los resultados en la capa de aplicación (`Python`) utilizando un algoritmo topológico.
- **Escritura:** Antes de insertar, realizamos una lectura del padre para heredar el `IDRaiz` y calcular el `Nivel + 1`. Esta operación, a pesar de su complejidad lógica, **resultó ser significativamente más rápida en la práctica que la operación transaccional de** `MySQL`.

**Ventajas:**

1. **Velocidad de Lectura Extrema:** Recuperar un hilo de 100 comentarios requiere una sola consulta indexada por `IDRaiz`, evitando las uniones recursivas. Nuestro tiempo de prueba lo confirma, siendo ideal para la carga rápida de la interfaz de usuario.
2. **Velocidad de Escritura Empírica:** A pesar de la lógica adicional requerida (lectura del padre y cálculo del `IDRaiz`/`Nivel`), el diseño de `MongoDB`, que evita el `COMMIT` transaccional pesado de `SQL`, demostró ser superior para la inserción de nuevos comentarios.
3. **Flexibilidad:** Podemos añadir metadatos al comentario (reacciones anidadas, tipos de contenido) sin alterar el esquema global.

**Desventajas:**

1. **Integridad Lógica:** La integridad referencial depende de la lógica de la aplicación. Si se borra un comentario en `MongoDB`, debemos encargarnos manualmente de borrar sus descendientes o el hilo quedaría corrupto.

## 2.3. Conclusión y Selección

Tras realizar pruebas de rendimiento (utilizando el módulo `6_#_MySQL_vs_Mongo_Comentarios.py` que desarrollamos), concluimos que el **Modelo No Relacional (`MongoDB`)** es más apropiado para este escenario específico de `GreenScape`.

**Razón:** En una red social, la proporción de lecturas frente a escrituras suele ser muy alta (muchos usuarios leen los comentarios, pocos escriben). La optimización de lectura que ofrece `MongoDB` mediante el campo `IDRaiz`, junto con el rendimiento inesperadamente superior en escritura, supera por amplio margen las desventajas de gestionar la integridad referencial en la aplicación, ofreciendo una mejor experiencia de usuario final.

# 3. Justificación del Modelo de Documentación Jerárquica de Plantas

Para el sistema de documentación de plantas, nos enfrentamos al reto de almacenar información con estructura variable (certificados, guías de riego, análisis de suelo) asociada a una entidad principal. Decidimos utilizar **MongoDB** basándonos en los siguientes criterios:

## 3.1. Estructura Variable y Polimorfismo

Las fichas técnicas de las plantas no son uniformes. Un Çertificado Fitosanitario"tiene campos como `fecha_inspeccion` y `validez`, mientras que un .ªnálisis de Suelo"tiene `ph_optimo` y `drenaje`.

- En un modelo relacional, esto hubiera requerido múltiples tablas (`Planta_Certificados`, `Planta_Suelos`, etc.) o una tabla con muchas columnas nulas.
- En `MongoDB`, utilizamos el patrón de **Documentos Embebidos**. Creamos una colección `plantas` donde cada documento contiene un array `DocumentosSecundarios`. Cada objeto dentro de este array puede tener un esquema totalmente diferente sin generar conflictos.

### 3.2. Consultas Atómicas

Al visualizar la documentación de una planta en la aplicación, el usuario necesita ver *toda* la información simultáneamente.

- Nuestro diseño permite recuperar la Ficha Técnica principal y todos sus documentos secundarios (independientemente de su tipo y cantidad) en **una sola operación de lectura**. Esto reduce drásticamente la latencia de red en comparación con realizar múltiples `SELECT` o `JOIN`s complejos en `SQL`.

### 3.3. Jerarquía Natural

La información botánica es intrínsecamente jerárquica (Planta → Ficha → Documentos Específicos → Detalles). El formato `JSON` de `MongoDB` mapea esta estructura de forma natural, simplificando el código de la aplicación (`Python`).

## 4. Otras Consideraciones Técnicas Relevantes

Durante el desarrollo del proyecto, tomamos decisiones de diseño adicionales para asegurar la robustez y cumplir con las restricciones técnicas:

1. **Uso de Controladores Nativos (Sin ORM):** Cumpliendo estrictamente con las restricciones, evitamos el uso de herramientas como `SQLAlchemy`. Desarrollamos nuestras propias clases envoltorios (`DatabaseConnector` para `mysql-connector` y `MongoConnector` para `pymongo`).
2. **Sincronización de Datos (Migración):** Para facilitar la transición y comparación entre `SQL` y `NoSQL`, implementamos scripts de migración (`setup_comments_to_mongo.py`). Decidimos mantener los mismos identificadores (`IDCom` de `MySQL` se convierte en `_id` en `MongoDB`). Esto fue crucial para asegurar que la integridad de los datos se mantuviera al pasar de un entorno relacional a uno documental, permitiendo pruebas comparativas fidedignas.