Informe Final: Sistema de Gestión y Recomendación de Viajes

Trabajo Práctico Integrador - Bases de Datos NoSQL

Autor: Dario Mecha | **Fecha:** Octubre 2025

Índice de Contenidos

1. Introducción

El presente trabajo integra tres tecnologías NoSQL —MongoDB, Neo4j y Redis— para desarrollar un **Sistema de Gestión y Recomendación de Viajes**. El objetivo principal fue modelar distintos tipos de información y operaciones que reflejan un entorno de gestión turística real, aprovechando las fortalezas de cada base de datos.

El sistema se ha diseñado para cumplir con los siguientes propósitos:

- **MongoDB:** Utilizado para almacenar datos descriptivos y persistentes como usuarios, destinos, hoteles, actividades y reservas confirmadas. Su flexibilidad de esquema es ideal para la variedad de atributos de estas entidades.
- **Neo4j:** Empleado para modelar relaciones complejas y realizar recomendaciones basadas en vínculos sociales (amigos) y de comportamiento (viajes previos) entre usuarios y destinos.
- **Redis:** Implementado para manejar información volátil y de acceso rápido en memoria, como sesiones de usuarios conectados, búsquedas recientes en caché y reservas no confirmadas que expiran tras un tiempo.

El desarrollo se realizó íntegramente en **Python**, utilizando el entorno de **JupyterLab**. La integración con las bases de datos se logró mediante sus respectivos controladores (`pymongo`, `neo4j-driver`, `redis-py`). Los resultados de las consultas y análisis se visualizaron directamente en las celdas del Notebook, complementados en algunos casos con librerías de visualización como `matplotlib` y `seaborn`.

2. Estructura del Proyecto

El proyecto está organizado en una estructura de directorios clara y modular, facilitando la gestión de datos, la lógica de negocio y la ejecución de procesos. A continuación, se detalla la jerarquía de archivos y sus responsabilidades.

Jerarquía de Archivos


viajes-db
notebooks
Carga de Datos.ipynb
Consultas.ipynb
Informe Final.ipynb
fuentes
usuarios.csv
destinos.csv
hoteles.csv
actividades.csv
reservas.csv
usuarios_relaciones.csv
src
mongo.py
neo4j.py
redis.py
utils.py
db_connections.py
constants.py
generador_datos.py
docker-compose.yml

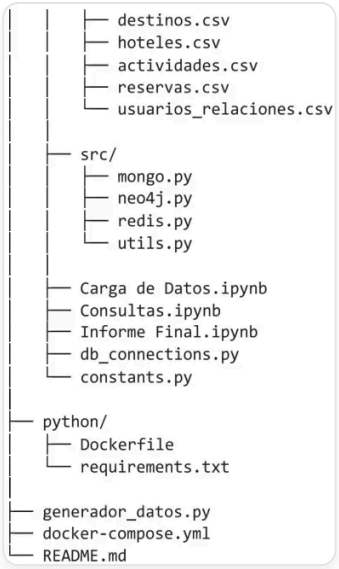
Dockerfile

requirements.txt

README.md

Componentes Principales

- **Generación de Datos:** El script `generador_datos.py` utiliza la librería **Faker** para crear datos sintéticos realistas de usuarios, destinos, hoteles, actividades y reservas. Estos datos se almacenan en archivos CSV dentro de la carpeta `/fuentes`, sirviendo como la carga inicial para las bases de datos.
- **Procesamiento y Lógica:** Los módulos especializados en la carpeta `/src` (`mongo.py`, `neo4j.py`, `redis.py`) encapsulan la lógica específica para interactuar con cada base de datos, promoviendo un código más limpio y modular.
- **Carga de Datos:** El notebook `Carga de Datos.ipynb` orquesta un pipeline que lee los archivos CSV, transforma los datos según sea necesario y los distribuye a la base de datos correspondiente (MongoDB, Neo4j o Redis).
- **Consultas Integradas:** El notebook `Consultas.ipynb` demuestra el poder del sistema al combinar consultas a las tres bases de datos para obtener información completa y resolver casos de uso complejos, como la recomendación de viajes.



Visualización de la estructura y componentes del proyecto.

3. Modelado de Datos

Para el funcionamiento del sistema se adoptó una arquitectura de **persistencia políglota** (Polyglot Persistence), utilizando tres bases de datos NoSQL con fines distintos y complementarios.

Base de Datos	Rol Principal	Tipo de Modelo
MongoDB	Almacena entidades y registros persistentes (estado maestro del sistema).	Documental
Neo4j	Representa relaciones complejas y permite recomendaciones basadas en grafos.	Grafos
Redis	Gestiona datos temporales, de sesión y en caché para un acceso ultrarrápido.	Clave-Valor

Arquitectura Polyglot Persistence

> *Esta arquitectura consiste en utilizar diferentes tipos de bases de datos dentro de una misma aplicación, eligiendo la tecnología más adecuada según la naturaleza de los datos y las operaciones requeridas. En lugar de depender de una única base de datos para resolver todas las necesidades, se adopta una combinación que aprovecha las fortalezas de cada modelo. Esta estrategia refleja una tendencia moderna en el desarrollo de sistemas de información, donde la especialización y la interoperabilidad generan soluciones más eficientes, escalables y cercanas a los escenarios del mundo real.*

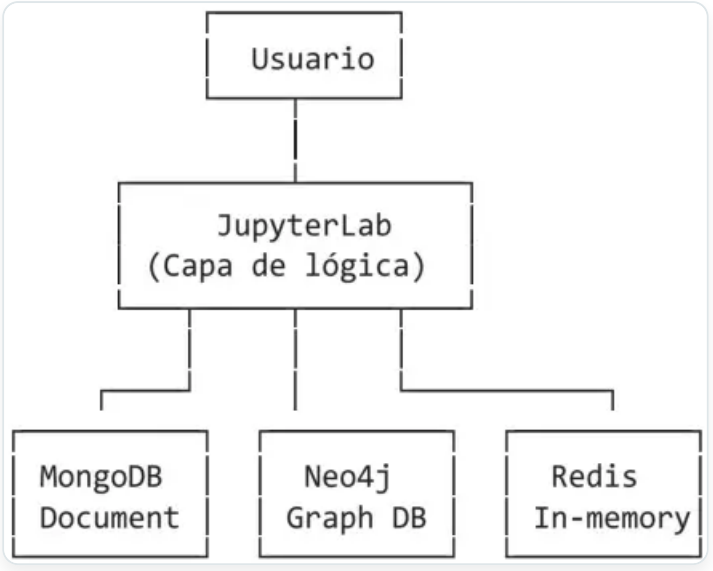


Diagrama conceptual de la arquitectura de persistencia políglota implementada.

Esquemas por Base de Datos

MongoDB: Almacén de Documentos

MongoDB actúa como la fuente de verdad para los datos persistentes. Las colecciones principales son:

Colección: `usuarios`

```
{
    "usuario_id": 1,
```

```
    "nombre": "Dario",
    "apellido": "Micheli",
    "email": "email@gmail.com",
    "telefono": "+34843181960"
}
```

Colección: `destinos`

```
{
    "destino_id": 1,
    "provincia": "Buenos Aires",
    "ciudad": "La Plata",
    "pais": "Argentina",
    "tipo": "cultural",
    "precio_promedio": 50000
}
```

Colección: `actividades`

```
{
    "actividad_id": 1,
    "nombre": "Culipatin",
    "tipo": "deportiva",
    "ciudad": "Bariloche",
    "provincia": "Rio Negro",
    "precio": 10000
}
```

Colección: `hoteles`

```
{
    "hotel_id": 1,
    "nombre": "AMAU",
    "ciudad": "La Plata",
    "provincia": "Buenos Aires",
    "precio": 80000,
```

```
    "calificacion": 5,
    "servicios": ["spa", "wifi"]
}
```

Colección: `reservas`

```
{
    "reserva_id": 1,
    "usuario_id": 2,
    "destino_id": 1,
    "hotel_id": 1,
    "fecha_reserva": "2025-07-01",
    "estado": "Confirmada",
    "precio_total": 90000
}
```

Neo4j: Grafo de Relaciones

Neo4j modela las conexiones entre entidades, lo que es fundamental para el motor de recomendaciones.

Nodos

- `(:Usuario {usuario_id})`: Representa a un usuario del sistema.
- `(:Destino {destino_id})`: Representa un destino turístico.

Relaciones

- `(:Usuario)-[:VISITO]->(:Destino)`: Se crea cuando un usuario tiene una reserva confirmada y pasada a un destino.
- `(:Usuario)-[:AMIGO_DE]->(:Usuario)`: Modela las relaciones sociales entre usuarios.

Redis: Caché y Datos Temporales

Redis gestiona datos efímeros para optimizar el rendimiento y manejar estados temporales.

Estructuras de Datos

- **STRING** `usuario:{usuario_id}:sesion`: Almacena el estado de la sesión de un usuario.

```
# Ejemplo: Clave -> Valor
usuario:15:sesion -> "activa"
```

- **STRING** `busqueda:{tipo}:{parametro}`: Almacena en caché los resultados de búsquedas frecuentes para una respuesta instantánea.

```
# Ejemplo: Clave -> Valor (JSON)
busqueda:destinos:ciudad:La Rioja|tipo:Playa -> "
{'destino_id': 17, ...}"
```

- **HASH** `reserva_temp:{reserva_id}`: Guarda los detalles de una reserva que aún no ha sido confirmada. Estas claves suelen tener un TTL (Time To Live) para expirar automáticamente.

```
# Ejemplo: Clave -> Hash
reserva_temp:960 -> {'usuario_id': '18', 'destino_id':
'17', ...}
```

4. Proceso de Carga de Datos

El poblamiento inicial de las bases de datos se realiza a través de un pipeline automatizado que consta de los siguientes pasos:

1. **Lectura de CSVs:** Se utiliza la función `utils.lectura_csv()` para cargar en memoria los datos sintéticos generados y almacenados en la carpeta `/fuentes`.

2. **Inserción en MongoDB:** La función `insertar_en_mongo()` toma los datos cargados y crea las colecciones (`usuarios`, `destinos`, `hoteles`, etc.) en la base de datos documental, estableciendo el estado maestro del sistema.

3. **Creación de Nodos en Neo4j:** Con la función `crear_nodos_neo4j()`, se generan los nodos `:Usuario` y `:Destino` en la base de datos de grafos, utilizando los IDs correspondientes de MongoDB para mantener la consistencia.

4. **Creación de Relaciones en Neo4j:**
    - 5. `crear_relaciones_visito()`: Analiza las reservas con estado "Confirmada" o "Pagada" en MongoDB y crea una relación `[:VISITO]` entre el usuario y el destino correspondiente en Neo4j.
    - 6. `crear_relaciones_usuarios()`: Carga las relaciones de amistad y familiares desde el archivo `usuarios_relaciones.csv` para crear las relaciones `[:AMIGO_DE]` entre nodos `:Usuario`.

7. **Carga en Redis:**
    - 8. `guardar_usuarios_conectados()`: Simula un entorno en tiempo real seleccionando 15 usuarios aleatorios y guardando su estado de sesión como "activa" en Redis.
    - 9. `carga_masiva_reservas_temporales()`: Carga reservas sin un estado final (ej. "en proceso") en Redis como hashes temporales, listas para ser confirmadas o para expirar.

5. Consultas Implementadas

Se implementó un conjunto diverso de consultas para demostrar la capacidad del sistema y la sinergia entre las tres bases de datos. Cada consulta está diseñada para resolver un problema específico del dominio turístico.

N°	Descripción de la Consulta	Base(s) de Datos Utilizada(s)
1.A	Listar usuarios que visitaron "Bariloche".	Neo4j
1.B	Encontrar amigos de un usuario que visitaron un destino que él también visitó.	Neo4j
1.C	Sugerir destinos a un usuario que no haya visitado (ni él ni sus amigos).	Neo4j
1.D	Recomendar destinos basados en los viajes más frecuentes de sus amigos.	Neo4j
1.E	Listar los hoteles disponibles en los destinos recomendados en el punto anterior (1.D).	MongoDB
1.F	Ver las reservas que están actualmente en proceso (no confirmadas).	Redis
1.G	Listar todos los usuarios conectados actualmente en el sistema.	Redis
1.H	Mostrar destinos con un precio promedio inferior a $100,000.	Redis (Caché) y MongoDB
1.I	Mostrar todos los hoteles de "Jujuy".	Redis (Caché) y MongoDB
1.J	Mostrar la cantidad de hoteles de un destino específico.	MongoDB

N°	Descripción de la Consulta	Base(s) de Datos Utilizada(s)
1.K	Mostrar las actividades de "Ushuaia" del tipo "aventura".	Redis (Caché) y MongoDB
1.L	Listar las reservas concretadas de cada usuario.	MongoDB
2.I	Encontrar el destino más visitado.	MongoDB
2.II	Encontrar el hotel más barato.	MongoDB
2.III	Encontrar la actividad más popular.	MongoDB
3.A	Incrementar el precio de todas las actividades de "Tucumán" en un 5%.	MongoDB
3.B	Agregar el servicio de "SPA" al hotel con id=1.	MongoDB
3.C	Eliminar un destino específico.	MongoDB
3.D	Eliminar un usuario específico.	MongoDB
3.E	Eliminar las relaciones `AMIGO_DE` para un usuario específico.	Neo4j

6. Instrucciones de Ejecución

Para replicar el entorno y ejecutar el proyecto, siga los siguientes pasos en orden:

1. **Generar Datos de Prueba:** Ejecute el script `python generador_datos.py`. Este comando creará todos los archivos `.csv` necesarios en la carpeta `/fuentes`.
2. **Cargar los Datos:** Abra el notebook `notebooks/Carga de Datos.ipynb` en un entorno de JupyterLab.

3. **Ejecutar Celdas de Carga:** Ejecute todas las celdas del notebook `Carga de Datos.ipynb` en secuencia. Esto poblará MongoDB, Neo4j y Redis.
4. **Realizar Consultas:** Abra el notebook `notebooks/Consultas.ipynb`.
5. **Ejecutar Celdas de Consulta:** Ejecute las celdas de este notebook para probar las diferentes consultas implementadas y ver los resultados en tiempo real.

7. Análisis Detallado por Tecnología

La elección de cada base de datos no fue arbitraria, sino que respondió a la necesidad de resolver problemas específicos de la manera más eficiente posible. A continuación se justifica el uso de cada tecnología.

¿Por qué MongoDB para Datos Transaccionales y Descriptivos?

- ✓ **Flexibilidad de Esquema:** Ideal para entidades como hoteles, donde los servicios pueden variar (ej. `"servicios": ["wifi", "spa", "pileta"]`). Permite agregar nuevos atributos sin alterar la colección entera.
- ✓ **Consultas de Agregación Eficientes:** MongoDB proporciona un potente framework de agregación que facilita la obtención de estadísticas complejas, como contar documentos (`count_documents()`) o calcular métricas (destino más popular, hotel más barato).
- ✓ **Escalabilidad Horizontal:** MongoDB está diseñado para escalar horizontalmente (sharding), lo que permite distribuir grandes volúmenes de datos a través de múltiples servidores, asegurando el rendimiento a medida que la aplicación crece.

- **Modelo de Documentos Intuitivo:** El formato BSON (similar a JSON) se mapea directamente con los objetos en lenguajes de programación como Python, simplificando el desarrollo.

¿Por qué Neo4j para Relaciones y Recomendaciones?

- **Modelado Intuitivo de Relaciones:** Representar usuarios, destinos y sus conexiones (`AMIGO_DE`, `VISITO`) es natural y directo, reflejando el dominio del problema sin la complejidad de las tablas de unión en SQL.
- **Rendimiento Superior en Travesías:** Las consultas que exploran relaciones (ej. "amigos de amigos") son extremadamente rápidas, ya que Neo4j no depende de JOINs recursivos, sino que sigue punteros directos en el grafo.
- **Potente Lenguaje de Consulta (Cypher):** Cypher es un lenguaje declarativo y visualmente intuitivo, diseñado para expresar patrones de grafos. Por ejemplo, para recomendar destinos basados en amigos:

```
MATCH (u:Usuario {usuario_id: $id})-[:AMIGO_DE]-(amigo)-
[:VISITO]->(d:Destino)
WHERE NOT (u)-[:VISITO]->(d)
RETURN d.ciudad, count(*) AS amigos_visitaron
ORDER BY amigos_visitaron DESC
```

- **Análisis de Red:** Facilita la identificación de usuarios influyentes, destinos populares dentro de un círculo social y la detección de comunidades o clústeres.

¿Por qué Redis para Datos Temporales y Caché?

- **Velocidad Extrema:** Al ser una base de datos en memoria, las operaciones de lectura y escritura se completan en microsegundos, lo que es ideal para casos de uso de baja latencia como el seguimiento de sesiones de usuario.

- **Expiración Automática (TTL):** Redis permite asignar un "tiempo de vida" a las claves. Esto es perfecto para reservas temporales o resultados de búsqueda en caché, que expiran automáticamente sin necesidad de intervención manual.
- **Caché Inteligente:** Almacenar los resultados de consultas frecuentes (ej. hoteles en una ciudad popular) en Redis reduce drásticamente la carga sobre las bases de datos persistentes (MongoDB y Neo4j), mejorando la velocidad de respuesta para el usuario final.
- **Estructuras de Datos Versátiles:** Redis no es solo un simple almacén clave-valor. Ofrece estructuras como Hashes, Lists, Sets y Sorted Sets que son muy útiles para modelar diferentes tipos de datos temporales de manera eficiente.

8. Decisiones Específicas de Diseño

Durante el desarrollo se tomaron varias decisiones clave para optimizar el sistema y simular un comportamiento realista.

- **Separación de Reservas Confirmadas vs. Temporales:** Las reservas confirmadas, que son un registro histórico, se guardan en MongoDB para persistencia y generan una relación `:VISITO` en Neo4j para el grafo de conocimiento. En cambio, las reservas temporales (en proceso) viven únicamente en Redis, ya que son efímeras y no deben contaminar el estado persistente hasta su confirmación.
- **Límites en las Relaciones Sociales:** Para evitar una "explosión de conexiones" y mantener el grafo manejable en este prototipo, se estableció un límite máximo de 8 amigos y 2 familiares por usuario.
- **Filtro de Fecha en la Relación `VISITO`:** La relación `:VISITO` solo se crea si la fecha de la reserva es anterior o igual al día actual

(`fecha_reserva <= hoy`). Esto asegura que la relación represente un viaje que ya ocurrió, lo cual es crucial para la lógica de recomendación.

- ✅ **Manejo de Arrays de Servicios:** Los servicios de los hoteles, almacenados como un string con formato de lista en el CSV (ej. `"['spa','wifi']"`), se convierten a listas reales de Python usando `ast.literal_eval()` antes de insertarlos en MongoDB. Esto permite realizar consultas sobre los elementos del array.

- ✅ **Simulación de Usuarios Conectados:** Se seleccionan 15 usuarios de forma aleatoria para marcarlos como "conectados" en Redis. Esto simula un escenario dinámico y realista de usuarios activos en la plataforma en cualquier momento.

9. Conclusiones y Aprendizajes

El desarrollo de este proyecto permitió integrar tres tecnologías NoSQL con propósitos complementarios, demostrando el valor práctico y la eficiencia de la arquitectura **polyglot persistence**. La experiencia consolidó la comprensión sobre cómo y por qué elegir una herramienta específica para un problema determinado.

MongoDB resultó fundamental para gestionar información estructurada pero flexible, como los perfiles de usuario, los catálogos de destinos y el historial de reservas. Su modelo documental simplificó enormemente el desarrollo y su capacidad de agregación fue clave para obtener métricas de negocio.

Neo4j brilló al representar y consultar relaciones complejas de manera natural. La capacidad de generar recomendaciones basadas en vínculos entre personas y lugares se simplificó drásticamente gracias a su modelo de grafo y al lenguaje Cypher, superando las limitaciones de los modelos relacionales tradicionales para este tipo de tareas.

Redis aportó la velocidad y eficiencia necesarias para manejar datos volátiles. La gestión de sesiones, el cacheo de búsquedas y el manejo de reservas en proceso demostraron cómo una capa de datos en memoria puede optimizar significativamente el rendimiento global del sistema y la experiencia del usuario.

En conjunto, la combinación de estas bases de datos dentro de un mismo entorno de trabajo evidenció cómo cada tecnología resuelve un aspecto diferente del problema: **persistencia (MongoDB), conocimiento (Neo4j) y tiempo real (Redis)**. El proyecto no solo fortaleció la comprensión teórica sobre los distintos modelos de datos NoSQL, sino que también mostró que la integración inteligente de tecnologías diversas es una estrategia poderosa para construir soluciones más escalables, dinámicas y alineadas con los complejos escenarios del mundo digital actual.