Implementación de Programación Concurrente en Bases de Datos NoSQL

Programación Concurrente

Ingeniería Informática

Silvia Lorero Salor

CONTENTS

1. ENUNCIADO

1.1. OBJETIVOS DE LA PRÁCTICA

En esta práctica, nuestro objetivo será obtener una comprensión práctica de cómo aplicar los principios de la programación concurrente en un entorno de base de datos NoSQL. Implementaremos diferentes estrategias de programación concurrente y evaluaremos su rendimiento al manejar grandes volúmenes de datos.

En la era actual de Big Data, las organizaciones generan y consumen cantidades enormes de información diariamente. Con tanta información disponible, las bases de datos tradicionales SQL pueden resultar ineficientes para manejar volúmenes de datos tan masivos y estructuras de datos complejas. Por lo tanto, las bases de datos NoSQL se han convertido en una elección popular debido a su escalabilidad y flexibilidad. Sin embargo, la eficiencia de estas bases de datos puede mejorarse aún más con el uso de la programación concurrente.

1.2. DETALLES DE LA TAREA

Para la base de datos NoSQL, puedes optar por utilizar MongoDB, que es una base de datos orientada a documentos, o Apache Cassandra, que es una base de datos orientada a columnas. Ambas opciones son open-source y tienen amplia documentación disponible.

En cuanto a los datos, puedes utilizar conjuntos de datos públicos disponibles en recursos como Kaggle, Google Dataset Search o el Portal de Datos Abiertos del Gobierno. Sin embargo, elige un conjunto de datos que sea lo suficientemente grande para que sea relevante el uso de programación concurrente.

Podrías optar por un conjunto de datos relacionados con transacciones de comercio electrónico, tweets de Twitter, datos meteorológicos históricos, registros de sensores

de IoT, etc. Recuerda, cuanto más grande y complejo sea el conjunto de datos, más interesantes serán los desafíos al implementar la programación concurrente.
Para la implementación de la programación concurrente, puedes usar herramientas y bibliotecas como ExecutorService y Future en Java, o ThreadPoolExecutor en Python. Estas herramientas te permiten gestionar la creación de hilos, su ejecución y la recolección de resultados de manera eficiente.
En cuanto a las pruebas de rendimiento, existen varias herramientas que puedes usar para monitorear y registrar el rendimiento de tu aplicación, como JMeter, Gatling o incluso las herramientas de monitoreo integradas en tu IDE.

1.3. MÁS DETALLES

Configuración inicial: Inicia configurando una base de datos NoSQL (puedes elegir entre una base de datos clave-valor, orientada a documentos, orientada a columnas o una base de datos de grafos).
Modelado de datos: Modela un conjunto de datos para tu base de datos NoSQL elegida. Los datos deben ser lo suficientemente complejos para requerir el uso de programación concurrente (por ejemplo, un número significativo de registros o una estructura de datos compleja).
Implementación de programación concurrente: Implementa la programación concurrente para realizar operaciones en tu base de datos. Esto puede implicar el uso de múltiples hilos o procesos para manejar las operaciones de lectura y escritura, la implementación de bloqueos y otras estrategias de control de concurrencia, o el uso de bibliotecas de programación concurrente de alto nivel.
Pruebas y evaluación de rendimiento: Realiza pruebas de rendimiento en tu implementación. ¿Cómo se compara el rendimiento de tu implementación con y sin programación concurrente? Considera el tiempo de ejecución, el uso de recursos y cualquier otro factor de rendimiento relevante.
Informe de la práctica: Escribe un informe que documente tu enfoque, los desafíos que encontraste y cómo los resolviste, los resultados de tus pruebas de rendimiento y cualquier hallazgo o conclusión interesante que hayas obtenido de la práctica.

2. PROBLEMAS

2.1. PRINCIPALES DIFICULTADES

Primordialmente, como es muy parecido al anterior proyecto, lo único que tuve que hacer fue hacer un par de cambios, es decir, que hasta cambiamos de SQL a NoSQL(MongoDB). A partir de ahí cuando ya lo descargué, tuve un par de errores.
Cuando ya creé la base de datos dentro del IntelliJ y buscaba en el navegador con el localhost:8080, se me abría sin ningún tipo de error y si añadía por ejemplo un libro, se me actualizaba correctamente en la base de datos creada en el IntelliJ. Pero cuando intentaba abrirlo en la aplicaciçon de MongoDB Server, me saltaba un error, dando a entender que no se podía conectar. Finalmente, lo pude solucionar.
Además cabe destacar que como era muy parecida a la práctica anterior, no tuve ninguna complicación más.

3. RESUMEN

3.1. INTRODUCCIÓN

Primordialmente, la realización, de este trabajo lo hice a través de Bootify como se indicaba en clase. A partir de ahí, fui escogiendo el lenguaje, en este caso Java y Maven. A continuación, fui importando todas las dependencias correspondientes para que funcionase el proyecto, como por ejemplo, Docker. También escogemos en la base de datos que queremos hacerlo, en este caso escogí MongoDB. A continuación, lo descargamos y hacemos unos cambios en el docker-compose.yml, como se muestra:

```
spring:
  data:
    mongodb:
      uri: mongodb://localhost:27017/GestionBancosMongo
      auto-index-creation: true
  mvc:
    format:
      date: yyyy-MM-dd
      date-time: yyyy-MM-dd'T'HH:mm:ss
      time: HH:mm:ss
  docker:
    compose:
      lifecycle-management: start-only
springdoc:
  pathsToMatch: /api/**
server:
```

Figure 3.1: Archivo docker-compose.yml

3.2. CREACIÓN DE CLASES

Las entidades que fueron creadas para este proyecto son las siguientes: Biblioteca creada como súper clase, y las complementarias, es decir, Bibliotecario, Lector, Prestamo y Libro. En este trabajo, eliminé las relaciones que había entre clases. En esta imagen se puede observar de una manera más clara, el UML:

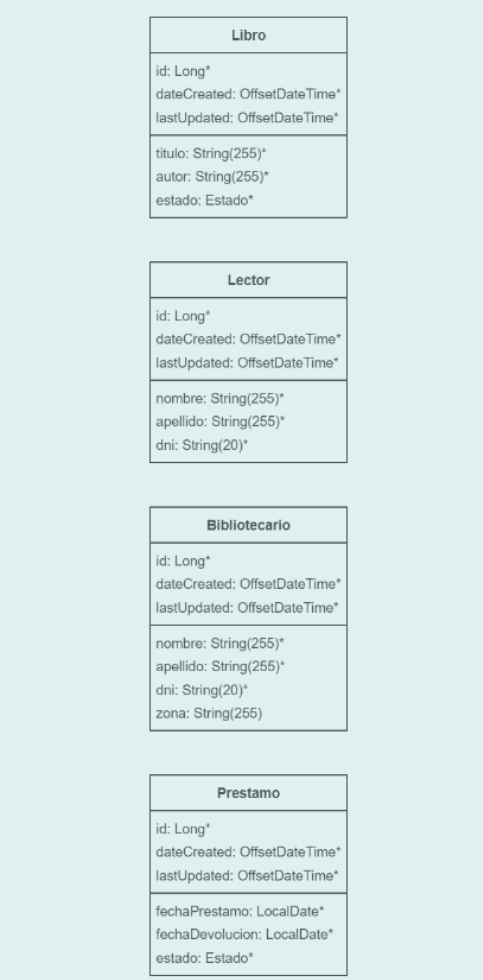

Figure 3.2: UML.

3.3. PROGRAMA

Una vez importado el proyecto, lo abro en IntelliJ y lo primero que hago es cambiar los archivos de docker-compose.yml y seguidamente el application.yml.En este último añadí al final lo siguiente: server:port: 8080 . A continuación, abro Docker y ejecutó para que se me inicialice con SpringBoot y posteriormente abrir el navegador web con el localhost:8080 para comprobar que todo funciona. También tengo que tener abierto MondoDB Server para corroborar que se me actualiza correctamente la base de datos, como se osberva a continuación:

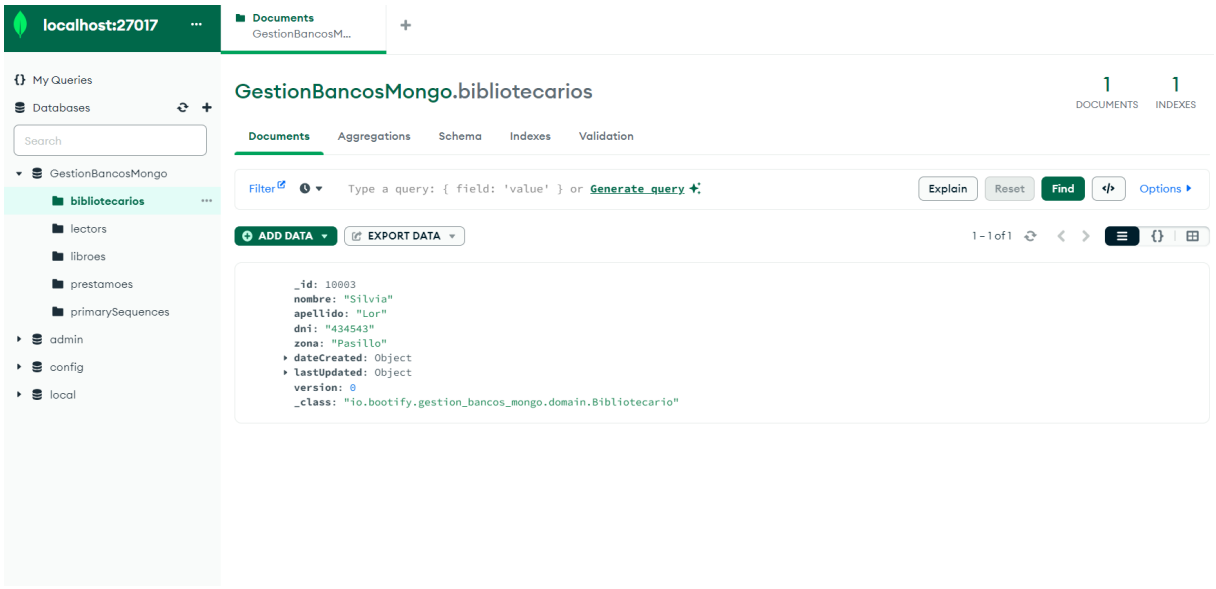

Figure 3.3: Captura de MongoDB.

4. CONCLUSIÓN

4.1. CONCLUSIONES FINALES

Para finalizar todo lo aprendido en este proyecto, cabría destacar la aplicación de Docker, así como Spring Boot aunque ya implementamos algún proyecto antes que este. También cabría destacar, como el uso de una base de datos NoSQL(MongoDB) en el que creabamos una en el propio IntelliJ para que se pudiera ver si se guardaban los datos y se actualizaban correctamente.
Una vez importado de Bootify, el proceso fue mucho más sencillo puesto que no me salían tantos errores que empezandólo de cero y además al ser el segundo proyecto es mucho más fácil puesto que ya tienes conocimientos anteriores.
Para concluir, he aprendido nuevos conocimientos a partir de esta práctica puesto que además estuve investigando por mi cuenta de la razón de todos los fallos una vez inicializaba SpringBoot, como comenté anteriormente.

5. BIBLIOGRAFÍA

5.1. VIDEO ANTES DE CONOCER BOOTIFY

https://www.youtube.com/watch?v=eWbGV3LLwVQ

5.2. BOOTIFY

https://bootify.io/

5.3. MONGODB

https://www.freecodecamp.org/espanol/news/como-empezar-a-utilizar-mongodb-en-10-minutos/

5.4. DOCKER

https://github.com/Camelcade/Perl5-IDEA/issues/2126