Danrui Wang

Accenture

***Comparativa entre Flyway y Liquibase para la Gestión de Bases de Datos en una API REST de Gestión Hospitalaria***

## 1. Introducción

Este documento presenta una comparativa detallada entre dos herramientas populares de migración de bases de datos: **Flyway** y **Liquibase**. Ambas son ampliamente utilizadas en la industria del desarrollo de software para gestionar la evolución de bases de datos, especialmente en proyectos donde es crucial mantener un control riguroso de los cambios en el esquema de datos.

La comparación se realiza en el contexto de una API REST simple desarrollada para gestionar un sistema hospitalario, implementada en dos versiones incrementales. La API está construida utilizando **Java**, **Spring Boot**, **Maven** y **PostgreSQL**, y se enfoca en proporcionar funcionalidades para la gestión de pacientes, historiales médicos y vacunas.

El documento tiene como objetivo evaluar las ventajas y desventajas de Flyway y Liquibase en el contexto de la API desarrollada. Además, busca describir de manera detallada la implementación de migraciones utilizando Flyway y Liquibase, destacando los aspectos técnicos de cada herramienta. Finalmente, ofrece recomendaciones basadas en las pruebas realizadas y en las necesidades específicas del proyecto, con el fin de ayudar en la selección de la herramienta más adecuada para gestionar las migraciones de bases de datos en este entorno.

## 2. Descripción General del Proyecto

## 2.1 Tecnologías Utilizadas

Para el desarrollo de la API se han utilizado las siguientes tecnologías:

* **Java 11 o superior**: Lenguaje de programación base del proyecto.
* **Spring Boot**: Framework para el desarrollo rápido de aplicaciones Java.
* **Maven**: Herramienta de gestión de proyectos y dependencias.
* **PostgreSQL**: Sistema de gestión de bases de datos relacionales.
* **Spring Data JPA**: Abstracción sobre JPA para facilitar las operaciones con la base de datos.
* **Hibernate**: Implementación de JPA utilizada por defecto en Spring Boot.
* **Flyway Migration**: Herramienta para la gestión de migraciones de base de datos.
* **Liquibase Migration**: Alternativa a Flyway para la gestión de migraciones.

## 2.2 Arquitectura de la API

En este apartado se describe el servicio API REST programado para realizar las pruebas con las distintas herramientas. La API sigue una arquitectura RESTful, facilitando la interacción con el sistema a través de endpoints. Se utilizan controladores para manejar las solicitudes HTTP y servicios para encapsular la lógica de negocio. La persistencia de datos se gestiona mediante Spring Data JPA y Hibernate, simplificando las operaciones con la base de datos PostgreSQL.

## 3. Funcionalidades y Endpoints

A continuación, se detallan las funcionalidades ofrecidas por la API y los endpoints asociados en cada versión.

## 3.1 Versión 1: Gestión de Pacientes

## 3.1.1 Funcionalidades

* **Crear Paciente**: Añadir nuevos pacientes al sistema.
* **Listar Pacientes**: Obtener una lista de todos los pacientes registrados.
* **Actualizar Paciente**: Modificar la información de un paciente existente.
* **Eliminar Paciente**: Eliminar un paciente del sistema.

## 3.1.2 Endpoints

* GET /api/v1/paciente: Retorna una lista de todos los pacientes.
* POST /api/v1/paciente: Crea un nuevo paciente con los datos proporcionados en el cuerpo de la solicitud.
* PUT /api/v1/paciente/{id}: Actualiza el paciente con el ID especificado utilizando los datos proporcionados.
* DELETE /api/v1/paciente/{id}: Elimina el paciente con el ID especificado.

## 3.2 Versión 2: Gestión de Historiales Médicos y Vacunas

## 3.2.1 Funcionalidades Adicionales

**Gestionar Historiales Médicos:**

* **Añadir Historial Médico**: Agregar registros médicos a un paciente.
* **Listar Historiales Médicos**: Obtener todos los historiales médicos de un paciente.
* **Obtener Historial Médico**: Recuperar un historial médico específico de un paciente.
* **Eliminar Historial Médico**: Borrar un historial médico de un paciente.

**Gestionar Vacunas:**

* **Añadir Vacuna**: Registrar una vacuna administrada a un paciente.
* **Listar Vacunas**: Obtener todas las vacunas registradas para un paciente.
* **Obtener Vacuna**: Recuperar información de una vacuna específica de un paciente.
* **Eliminar Vacuna**: Borrar el registro de una vacuna de un paciente.

## 3.2.2 Endpoints para Historiales Médicos

* GET /api/v1/paciente/{pacienteId}/historialmedico: Lista todos los historiales médicos de un paciente.
* POST /api/v1/paciente/{pacienteId}/historialmedico: Crea un nuevo historial médico para el paciente especificado.
* GET /api/v1/paciente/{pacienteId}/historialmedico/{historialId}: Obtiene un historial médico específico.
* DELETE /api/v1/paciente/{pacienteId}/historialmedico/{historialId}: Elimina un historial médico específico.

## 3.2.3 Endpoints para Vacunas

* GET /api/v1/paciente/{pacienteId}/vacunas: Lista todas las vacunas de un paciente.
* POST /api/v1/paciente/{pacienteId}/vacunas: Registra una nueva vacuna para el paciente especificado.
* GET /api/v1/paciente/{pacienteId}/vacunas/{vacunaId}: Obtiene información de una vacuna específica.
* DELETE /api/v1/paciente/{pacienteId}/vacunas/{vacunaId}: Elimina una vacuna específica.

## 4. Comparativa entre Flyway y Liquibase

## 4.1 Introducción a las Herramientas de Migración

Las migraciones de bases de datos son esenciales para mantener la coherencia y la integridad de los datos a medida que una aplicación evoluciona. **Flyway** y **Liquibase** son dos herramientas líderes en el manejo de migraciones. A continuación, se presenta una comparación detallada de ambas en el contexto de este proyecto.

## 5. Implementación con Flyway

## 5.1 Descripción de Flyway

Flyway es una herramienta de migración de bases de datos ampliamente utilizada que permite gestionar el esquema de la base de datos de forma incremental. Flyway realiza un seguimiento de las migraciones aplicadas y las organiza de manera secuencial, aplicando los cambios necesarios en el orden correcto cada vez que se inicia una nueva migración.

A continuación, se detalla cómo funciona Flyway y cómo se configuran los archivos de migración SQL.

## 5.1.1 Funcionamiento General de Flyway

Flyway se basa en el uso de archivos de migración que contienen scripts SQL o código Java para aplicar cambios en el esquema de la base de datos. Cada archivo de migración tiene un nombre específico que sigue una convención que Flyway utiliza para identificar el orden en el que deben aplicarse los cambios.

Cuando se ejecuta Flyway (mediante un comando o automáticamente a través de una integración con un framework como Spring Boot), compara las migraciones ya aplicadas con las nuevas migraciones disponibles y ejecuta aquellas que aún no se han aplicado. Flyway lleva un registro en una tabla especial llamada flyway\_schema\_history dentro de la base de datos, donde almacena información sobre las migraciones que se han ejecutado, como la versión, descripción, fecha de ejecución, y estado.

## 5.1.2 Convenciones de Nombres de Archivos en Flyway

Flyway identifica y ejecuta los archivos de migración basándose en el nombre del archivo. Por esta razón, es esencial seguir la convención de nombres de archivos al crear nuevas migraciones. Los archivos deben tener un nombre con el siguiente formato: ***V<version\_number>\_\_<description>.sql***

* **V**: Indica que se trata de una migración "versionada".
* **<version\_number>**: Es el número de versión de la migración. Flyway aplica las migraciones en orden ascendente según este número, que puede incluir puntos decimales (por ejemplo, V1, V2.1).
* **\_\_**: Es un separador entre el número de versión y la descripción.
* **<description>**: Es una breve descripción de la migración, que debe estar en formato "snake\_case" (palabras separadas por guiones bajos). No debe contener espacios ni caracteres especiales.

## 5.1.3 Ubicación de los Archivos de Migración

Por defecto, Flyway espera que los archivos de migración SQL se ubiquen en el directorio db/migration dentro del classpath del proyecto. En un proyecto **Spring Boot**, este directorio normalmente se encuentra en *src/main/resources/db/migration*.

## 5.1.4 Principios de Migración en Flyway

Flyway sigue algunos principios clave:

1. **Migración Lineal**: Las migraciones deben ser aplicadas de forma secuencial. Si Flyway detecta una brecha en la secuencia de versiones (por ejemplo, si se salta de V2 a V4), lanzará un error.
2. **Inmutabilidad**: Una vez aplicada una migración, no debe ser modificada. Si es necesario revertir o modificar una migración ya aplicada, se debe crear una nueva migración para realizar los cambios necesarios.
3. **Control de Versiones**: Flyway mantiene un control estricto de las versiones de los cambios mediante la tabla flyway\_schema\_history, lo que permite conocer qué migraciones se han aplicado, en qué orden y en qué momento.

## 5.1.5 Integración con Spring Boot

Flyway se integra automáticamente con Spring Boot. Cuando se incluye la dependencia de Flyway en el proyecto, al iniciar la aplicación, Spring Boot ejecuta automáticamente las migraciones disponibles en el directorio de migración (db/migration).

## 5.2 Configuración del Proyecto

Se ha creado un proyecto utilizando **Spring Initializer** con las siguientes dependencias:

* **Spring Web**
* **Spring Data JPA**
* **PostgreSQL Driver**
* **Flyway Migration**
* **Lombok**

La estructura del proyecto permite una integración sencilla de Flyway para gestionar las migraciones de la base de datos.

## 5.3 Migraciones Implementadas

## 5.3.1 Versión 1: Inicialización del Esquema

En el directorio db/migration, se ha creado el archivo V1\_\_initializing\_schema.sql con el siguiente contenido:

|  |
| --- |
| *CREATE SEQUENCE paciente\_sequence START WITH 1 INCREMENT BY 1;*  *CREATE TABLE paciente (*  *id BIGINT PRIMARY KEY DEFAULT NEXTVAL('paciente\_sequence'),*  *dni VARCHAR(50) NOT NULL,*  *nombre VARCHAR(100) NOT NULL,*  *email VARCHAR(100) NOT NULL,*  *fecha\_nacimiento DATE NOT NULL*  *);* |

Este script crea una secuencia para generar IDs únicos y una tabla paciente con los campos necesarios.

## 5.3.2 Versión 2: Extensión del Esquema

Para la segunda versión, se ha creado el archivo V2\_\_initializing\_schema.sql con el siguiente contenido:

|  |
| --- |
| *CREATE SEQUENCE historial\_medico\_sequence START WITH 1 INCREMENT BY 1;*  *CREATE SEQUENCE vacuna\_sequence START WITH 1 INCREMENT BY 1;*  *CREATE TABLE historial\_medico (*  *id BIGINT PRIMARY KEY DEFAULT NEXTVAL('historial\_medico\_sequence'),*  *descripcion VARCHAR(255) NOT NULL,*  *fecha DATE NOT NULL,*  *paciente\_id BIGINT NOT NULL,*  *CONSTRAINT fk\_paciente\_historial FOREIGN KEY (paciente\_id) REFERENCES paciente (id) ON DELETE CASCADE*  *);*  *CREATE TABLE vacuna (*  *id BIGINT PRIMARY KEY DEFAULT NEXTVAL('vacuna\_sequence'),*  *vacuna VARCHAR(255) NOT NULL,*  *fecha DATE NOT NULL,*  *paciente\_id BIGINT NOT NULL,*  *CONSTRAINT fk\_paciente\_vacuna FOREIGN KEY (paciente\_id) REFERENCES paciente (id) ON DELETE CASCADE*  *);* |

Este script añade las tablas historial\_medico y vacuna, incluyendo las secuencias necesarias y las claves foráneas que relacionan estas tablas con paciente.

## 5.4 Gestión de Rollbacks con Flyway

Flyway no ofrece soporte nativo para **rollbacks** en su edición **Community**. Sin embargo, es posible implementar una estrategia de reversión utilizando migraciones de "deshacer" que simulan el rollback de una operación anterior. A continuación, veremos cómo aplicar esta técnica en el contexto del proyecto.

## 5.4.1 Estrategias para Rollbacks

Existen dos estrategias principales para manejar rollbacks en Flyway en proyectos que utilizan la edición Community:

* **Migraciones de Reversión Simulada**: Crear scripts SQL que reviertan los cambios realizados por una migración anterior.
* **Flyway Undo (Solo para ediciones Pro y Enterprise)**: Flyway Pro y Enterprise ofrecen una funcionalidad avanzada para realizar rollbacks mediante migraciones undo, que permiten revertir automáticamente los cambios de migraciones anteriores.

En este proyecto, debido al uso de la edición Community de Flyway, se implementará la primera estrategia: **migraciones de reversión simulada**.

## 5.4.2 Migraciones de reversión simulada

Para revertir una migración en Flyway, se puede crear un nuevo archivo de migración SQL que ejecute las operaciones inversas a las que se realizaron en la migración original.

**Ejemplo: Reversión de la Migración del Historial Médico**

1. **Migración original**: En la segunda versión de nuestro esquema, hemos creado la tabla historial\_medico utilizando el archivo V2\_\_initializing\_schema.sql:

|  |
| --- |
| *CREATE SEQUENCE historial\_medico\_sequence START WITH 1 INCREMENT BY 1;*  *CREATE TABLE historial\_medico (*  *id BIGINT PRIMARY KEY DEFAULT NEXTVAL('historial\_medico\_sequence'),*  *descripcion VARCHAR(255) NOT NULL,*  *fecha DATE NOT NULL,*  *paciente\_id BIGINT NOT NULL,*  *CONSTRAINT fk\_paciente\_historial FOREIGN KEY (paciente\_id) REFERENCES paciente (id) ON DELETE CASCADE*  *);* |

1. **Crear migración de reversión**: Para revertir esta migración, podemos crear un archivo de migración llamado V3\_\_drop\_historial\_medico\_table.sql que elimine la tabla y la secuencia asociada:

|  |
| --- |
| DROP TABLE IF EXISTS historial\_medico;  DROP SEQUENCE IF EXISTS historial\_medico\_sequence; |

1. **Aplicar la migración**: Al aplicar este nuevo archivo de migración, Flyway ejecutará las instrucciones y eliminará la tabla historial\_medico junto con su secuencia.
2. **Verificar el estado de las migraciones**: Después de ejecutar la migración, podemos verificar el estado de las mismas usando el comando “./flyway info”. Este comando mostrará un historial de todas las migraciones ejecutadas, incluyendo la eliminación de la tabla historial\_medico.

## 5.4.3 Uso de Flyway Undo (Opcional, Pro/Enterprise Edition)

Para proyectos que utilicen la edición Pro o Enterprise de Flyway, existe una funcionalidad llamada **Flyway Undo** que facilita el rollback automático de migraciones.

1. **Crear archivos de undo**: En lugar de escribir manualmente las migraciones de reversión, podemos crear un archivo de undo con el prefijo U. Para la migración que crea la tabla historial\_medico, el archivo de undo sería U2\_\_drop\_historial\_medico\_table.sql:

|  |
| --- |
| DROP TABLE IF EXISTS historial\_medico;  DROP SEQUENCE IF EXISTS historial\_medico\_sequence; |

1. **Aplicar undo**: Para deshacer la última migración aplicada, podemos ejecutar el comando “./flyway -pro undo”. Este comando revertirá automáticamente la última migración aplicada y actualizará el estado en el historial de Flyway.
2. **Verificar el estado**: Al ejecutar “./flyway -pro info”, se verá un registro detallado de las migraciones deshechas, con una columna que muestra si cada migración es reversible. 

## 5.4.4 Consideraciones

* **Migraciones Fallidas**: Si una migración falla a mitad de camino, Flyway no podrá revertir automáticamente los cambios. En estos casos, es importante crear scripts de rollback adicionales para garantizar que el sistema pueda volver a un estado estable.
* **Control de Versiones**: Es recomendable usar etiquetas y versiones en Flyway para facilitar la gestión de rollbacks y asegurar que las migraciones puedan ser revertidas correctamente.

## 5.5 Ventajas de Flyway

* **Simplicidad**: Es fácil de configurar y utilizar, lo que permite una rápida integración en proyectos existentes.
* **Convención sobre Configuración**: Reduce la complejidad al seguir convenciones estándar para la gestión de migraciones.
* **Integración con Herramientas de Construcción**: Se integra fácilmente con Maven, Gradle y otras herramientas de CI/CD.
* **Rendimiento**: Aplicación rápida de migraciones, ideal para proyectos con requisitos de despliegue ágiles.
* **Compatibilidad Multiplataforma**: Soporta una amplia gama de sistemas de gestión de bases de datos.

## 5.6 Desventajas de Flyway

* **Menor Flexibilidad**: Al estar centrado en scripts SQL, puede ser menos adaptable en entornos donde se prefieren definiciones en otros formatos.
* **Gestión de Conflictos**: En entornos con múltiples desarrolladores o ramas, la gestión de versiones puede ser más complicada.

## 6. Implementación con Liquibase

## 6.1 Descripción de Liquibase

Liquibase es una herramienta de migración de bases de datos que permite controlar y versionar cambios en el esquema de manera declarativa. A diferencia de Flyway, Liquibase soporta múltiples formatos para definir migraciones, como XML, YAML, JSON y SQL, proporcionando mayor flexibilidad y opciones avanzadas para gestionar cambios complejos.

## 6.1.1 Funcionamiento General de Liquibase

Liquibase utiliza archivos llamados **changelogs** para definir las migraciones de la base de datos. Cada changelog contiene uno o más **changesets**, que representan cambios atómicos en la base de datos (como crear una tabla, modificar una columna, etc.). Al igual que Flyway, Liquibase mantiene un registro de los changesets aplicados en una tabla de control (databasechangelog), lo que permite aplicar nuevas migraciones de forma incremental.

## 6.1.2 Formatos de Definición de Migraciones

Liquibase admite varios formatos para definir migraciones:

* **XML**: Formato más utilizado y flexible, que permite aprovechar todas las características de Liquibase.
* **YAML**: Formato legible y conciso, útil para configuraciones más simples.
* **JSON**: Alternativa similar a YAML, con soporte para estructuras de datos complejas.
* **SQL**: Permite escribir migraciones directamente en SQL, con etiquetas especiales para controlar el changeset.

## 6.1.3 Integración con Spring Boot

Liquibase se integra fácilmente con Spring Boot. Al incluir la dependencia de Liquibase en el proyecto, Spring Boot detecta automáticamente los changelogs y aplica las migraciones al iniciar la aplicación. La configuración se realiza a través del archivo application.properties o application.yml.

## 6.2 Configuración del Proyecto

Se ha creado un proyecto similar al utilizado con Flyway, pero incorporando la dependencia de **Liquibase Migration** en lugar de Flyway. A continuación, se detallan los cambios realizados.

## 6.2.1 Dependencias Incluidas

En el archivo pom.xml, se ha añadido la dependencia de Liquibase:

|  |
| --- |
| <dependency>  <groupId>org.liquibase</groupId>  <artifactId>liquibase-core</artifactId>  </dependency> |

Además, se ha añadido el plugin de Maven para ejecutar tareas de Liquibase:

|  |
| --- |
| <plugin>  <groupId>org.liquibase</groupId>  <artifactId>liquibase-maven-plugin</artifactId>  <configuration>  <propertyFile>src/main/resources/application.properties</propertyFile>  <propertyFileWillOverride>true</propertyFileWillOverride>  </configuration>  </plugin> |

## 6.2.2 Configuración en application.properties

Se han añadido las siguientes propiedades:

|  |
| --- |
| spring.liquibase.enabled=true  spring.liquibase.change-log=classpath:db/changelog/changelog-master.xml |

Estas propiedades indican a Spring Boot que utilice Liquibase y especifique la ubicación del changelog maestro.

## 6.3 Migraciones Implementadas

Las migraciones se han definido utilizando archivos XML en el directorio db/changelog.

## 6.3.1 Versión 1: Inicialización del Esquema

|  |
| --- |
| db/changelog/changelog-v1.0.xml |
| <?xml version="1.0" encoding="UTF-8"?>  <databaseChangeLog  xmlns="http://www.liquibase.org/xml/ns/dbchangelog"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd">  <changeSet id="1" author="danrui">  <createSequence sequenceName="paciente\_sequence" startValue="1" incrementBy="1"/>  <createTable tableName="paciente">  <column name="id" type="BIGINT">  <constraints primaryKey="true" nullable="false"/>  </column>  <column name="dni" type="VARCHAR(50)">  <constraints nullable="false"/>  </column>  <column name="nombre" type="VARCHAR(100)">  <constraints nullable="false"/>  </column>  <column name="email" type="VARCHAR(100)">  <constraints nullable="false"/>  </column>  <column name="fecha\_nacimiento" type="DATE">  <constraints nullable="false"/>  </column>  </createTable>  <addDefaultValue tableName="paciente" columnName="id" defaultValueSequenceNext="paciente\_sequence"/>  </changeSet>  </databaseChangeLog> |

Este changelog crea la secuencia paciente\_sequence y la tabla paciente con los campos necesarios.

## 6.3.2 Versión 2: Extensión del Esquema

|  |
| --- |
| db/changelog/changelog-v2.0.xml |
| <?xml version="1.0" encoding="UTF-8"?>  <databaseChangeLog  xmlns="http://www.liquibase.org/xml/ns/dbchangelog"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd">  <changeSet id="2" author="danrui">  <createSequence sequenceName="historial\_medico\_sequence" startValue="1" incrementBy="1"/>  <createSequence sequenceName="vacuna\_sequence" startValue="1" incrementBy="1"/>  <createTable tableName="historial\_medico">  <column name="id" type="BIGINT">  <constraints primaryKey="true" nullable="false"/>  </column>  <column name="descripcion" type="VARCHAR(255)">  <constraints nullable="false"/>  </column>  <column name="fecha" type="DATE">  <constraints nullable="false"/>  </column>  <column name="paciente\_id" type="BIGINT">  <constraints nullable="false"/>  </column>  </createTable>  <addForeignKeyConstraint baseTableName="historial\_medico"  baseColumnNames="paciente\_id"  referencedTableName="paciente"  referencedColumnNames="id"  onDelete="CASCADE"  constraintName="fk\_paciente\_historial"/>  <sql>ALTER TABLE historial\_medico ALTER COLUMN id SET DEFAULT NEXTVAL('historial\_medico\_sequence');</sql>  </changeSet>  <changeSet id="3" author="danrui">  <createTable tableName="vacuna">  <column name="id" type="BIGINT">  <constraints primaryKey="true" nullable="false"/>  </column>  <column name="vacuna" type="VARCHAR(255)">  <constraints nullable="false"/>  </column>  <column name="fecha" type="DATE">  <constraints nullable="false"/>  </column>  <column name="paciente\_id" type="BIGINT">  <constraints nullable="false"/>  </column>  </createTable>  <addForeignKeyConstraint baseTableName="vacuna"  baseColumnNames="paciente\_id"  referencedTableName="paciente"  referencedColumnNames="id"  onDelete="CASCADE"  constraintName="fk\_paciente\_vacuna"/>  <sql>ALTER TABLE vacuna ALTER COLUMN id SET DEFAULT NEXTVAL('vacuna\_sequence');</sql>  </changeSet>  </databaseChangeLog> |

Este changelog crea las secuencias historial\_medico\_sequence y vacuna\_sequence, las tablas historial\_medico y vacuna, y establece las relaciones con la tabla paciente.

## 6.3.3 Changelog Maestro

|  |
| --- |
| **Archivo:** db/changelog/changelog-master.xml |
| <?xml version="1.0" encoding="UTF-8"?>  <databaseChangeLog  xmlns="http://www.liquibase.org/xml/ns/dbchangelog"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd">  <include file="db/changelog/changelog-v1.0.xml"/>  <include file="db/changelog/changelog-v2.0.xml"/>  </databaseChangeLog> |

Este archivo sirve como punto de entrada para Liquibase y lista los changelogs que deben ejecutarse en orden.

## 6.4 Gestión de Rollbacks con Liquibase

Liquibase ofrece soporte nativo para realizar rollbacks, permitiendo revertir cambios aplicados a la base de datos de forma segura y controlada.

## 6.4.1Rollback de Changesets

Cada changeSet en Liquibase puede incluir instrucciones de rollback específicas. Si no se proporcionan, Liquibase intenta generar automáticamente el rollback inverso.

|  |
| --- |
| Ejemplo |
| <changeSet id="2" author="danrui">  <!-- Definiciones del changeSet -->  ...  <rollback>  <dropTable tableName="historial\_medico"/>  <dropSequence sequenceName="historial\_medico\_sequence"/>  </rollback>  </changeSet> |

## 6.4.2 Comandos para Realizar Rollbacks

Liquibase permite ejecutar rollbacks basados en:

* **Tag**: Revertir hasta una etiqueta específica.
* **Count**: Revertir un número específico de cambios.
* **Fecha/Hora**: Revertir hasta un punto en el tiempo.

|  |
| --- |
| Ejemplo de comando Maven: |
| mvn liquibase:rollback -Dliquibase.rollbackCount=1 |

Este comando revertirá el último changeSet aplicado.

## 6.4.3 Consideraciones

* **Seguridad**: Al definir rollbacks explícitos, se asegura que la reversión será precisa y sin pérdida de datos.
* **Complejidad**: Es importante considerar el impacto en los datos existentes al realizar rollbacks, especialmente en entornos de producción.

## 6.5 Ventajas de Liquibase

* **Flexibilidad en Formatos**: Soporta XML, YAML, JSON y SQL para definir migraciones.
* **Soporte Nativo para Rollbacks**: Facilita la reversión de cambios con comandos integrados.
* **Control Detallado**: Permite especificar condiciones, contextos y etiquetas para aplicar migraciones en entornos específicos.
* **Gestión Avanzada de Conflictos**: Maneja eficazmente situaciones con múltiples desarrolladores y ramas de código.
* **Integración con Herramientas**: Compatible con diversas herramientas de CI/CD y plugins de Maven y Gradle.

## 6.7 Desventajas de Liquibase

* **Curva de Aprendizaje**: Requiere tiempo para familiarizarse con su sintaxis y capacidades avanzadas.
* **Complejidad en Configuración**: Puede ser más complejo de configurar inicialmente en comparación con Flyway.
* **Rendimiento**: La interpretación de los changelogs puede ser más lenta que la ejecución directa de scripts SQL.

## 7. Tabla Comparativa

|  |  |  |
| --- | --- | --- |
| **Característica** | **Flyway** | **Liquibase** |
| **Facilidad de Uso** | Alta | Media |
| **Formatos Soportados** | SQL, Java | XML, YAML, JSON, SQL |
| **Soporte para Rollbacks** | Limitado(Community Edition) | Completo |
| **Flexibilidad** | Menos | Mayor |
| **Configuración** | Sencilla | Moderadamente Compleja |
| **Integración** | Fácil con herramientas comunes | Requiere más configuración inicial |
| **Gestión de Conflictos** | Básica | Avanzada |
| **Curva de Aprendizaje** | Baja | Alta |

## 8. Recomendaciones Según el Proyecto

## 8.1 Uso de Flyway

* **Proyectos Pequeños o Medianos**: Ideal para estructuras de bases de datos sencillas.
* **Rápida Implementación**: Equipos que buscan una solución rápida y con mínima configuración.
* **Entornos con Menos Complejidad**: Cuando no se requieren funcionalidades avanzadas de migración o rollbacks detallados.

## 8.2 Uso de Liquibase

* **Proyectos Grandes y Complejos**: Adecuado para esquemas de bases de datos complejos que requieren un control detallado.
* **Necesidad de Rollbacks Seguros**: Cuando es crucial poder revertir cambios de manera confiable.
* **Equipos con Experiencia**: Cuando el equipo está dispuesto a invertir tiempo en aprender y configurar la herramienta para aprovechar sus capacidades avanzadas.
* **Entornos Multidesarrollador**: Proporciona mejor gestión de conflictos y fusiones en ramas de código.

## 9. Conclusión

La elección entre Flyway y Liquibase depende de las necesidades específicas del proyecto y las preferencias del equipo de desarrollo.

* **Flyway** es ideal para proyectos que requieren una herramienta sencilla y eficiente para manejar migraciones básicas sin necesidad de funcionalidades avanzadas. Su facilidad de uso y configuración lo hacen adecuado para equipos que buscan una solución rápida.
* **Liquibase** ofrece funcionalidades avanzadas que pueden ser más adecuadas para proyectos con requerimientos más complejos en la gestión de migraciones. Su soporte nativo para rollbacks y múltiples formatos de migración proporciona mayor flexibilidad y control.

## 10. Anexo

## 10.1 Configuración del proyecto flyway

|  |
| --- |
| Application.properties |
| spring.datasource.url=jdbc:postgresql://localhost:5432/Hospital spring.datasource.username=postgres spring.datasource.password=Hola spring.jpa.hibernate.ddl-auto=none spring.jpa.show-sql=true spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect spring.jpa.properties.hibernate.format\_sql=true server.error.include-message=*always* |

## 10.2 Configuración del proyecto liquibase

|  |
| --- |
| Application.properties |
| spring.datasource.url=jdbc:postgresql://localhost:5432/Hospital  spring.datasource.username=postgres  spring.datasource.password=Hola  spring.jpa.hibernate.ddl-auto=none  spring.jpa.show-sql=true  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect  spring.jpa.properties.hibernate.format\_sql=true  server.error.include-message=always  spring.liquibase.enabled=true  spring.liquibase.change-log=classpath:db/changelog/changelog-master.xml |

## 10.3 Estructura del código

La aplicación está organizada en módulos clave que se corresponden con las entidades del dominio: **Paciente**, **HistorialMedico** y **Vacuna**. Cada uno de estos módulos sigue una arquitectura típica de capas, que incluye controladores, servicios y repositorios para manejar las operaciones de la base de datos.

1. **Entidades**:
   * **Paciente**: Representa un paciente, con atributos como id, dni, nombre, email, fechaNacimiento y relaciones con las entidades **HistorialMedico** y **Vacuna**.
   * **HistorialMedico**: Define un registro médico asociado a un paciente, con campos como descripcion, fecha, y una relación con **Paciente**.
   * **Vacuna**: Representa una vacuna administrada a un paciente, con atributos como vacuna, fecha, y una relación con **Paciente**.
2. **Controladores**:
   * **PacienteController**, **HistorialMedicoController**, **VacunaController**: Manejan las solicitudes HTTP y definen los endpoints para las operaciones CRUD sobre pacientes, historiales médicos y vacunas.
3. **Servicios**:
   * **PacienteService**, **HistorialMedicoService**, **VacunaService**: Encapsulan la lógica de negocio y gestionan las operaciones de manipulación de datos, delegando las operaciones de acceso a datos a los repositorios.
4. **Repositorios**:
   * **PacienteRepository**, **HistorialMedicoRepository**, **VacunaRepository**: Interfases que extienden de JpaRepository para proporcionar acceso a las operaciones de base de datos de forma declarativa, simplificando las consultas.
5. **Configuración**:
   * **PacienteConfig**: Clase de configuración que usa CommandLineRunner para inicializar datos en la base de datos al arrancar la aplicación.

## 10.4 Plugins de Maven para Flyway y Liquibase

En el proyecto, se han añadido los siguientes **plugins de Maven** para poder ejecutar comandos relacionados con las migraciones de la base de datos a través de Flyway y Liquibase respectivamente.

## 10.4.1 Plugin de Flyway

Se ha añadido el **plugin de Flyway** en el archivo pom.xml para gestionar las migraciones de la base de datos mediante comandos de Maven. La configuración especifica la conexión a la base de datos PostgreSQL y permite ejecutar comandos como mvn flyway:migrate, mvn flyway:clean, entre otros.

|  |
| --- |
| <plugin>  <groupId>org.flywaydb</groupId>  <artifactId>flyway-maven-plugin</artifactId>  <version>9.22.0</version>  <configuration>  <url>jdbc:postgresql://localhost:5432/Hospital</url>  <user>postgres</user>  <password>Hola</password>  <cleanDisabled>false</cleanDisabled>  </configuration>  </plugin> |

## 10.4.2 Plugin de Liquibase

También se ha agregado el **plugin de Liquibase** en el archivo pom.xml para ejecutar migraciones a través de comandos de Maven. Este plugin permite ejecutar comandos como mvn liquibase:update, mvn liquibase:rollback, entre otros, utilizando los changelogs definidos en Liquibase.

|  |
| --- |
| <plugin>  <groupId>org.liquibase</groupId>  <artifactId>liquibase-maven-plugin</artifactId>  <version>4.27.0</version>  <configuration>  <url>jdbc:postgresql://localhost:5432/Hospital</url>  <username>postgres</username>  <password>Hola</password>  <changeLogFile>classpath:db/changelog/changelog-master.xml</changeLogFile>  <driver>org.postgresql.Driver</driver>  </configuration>  </plugin> |