**Spring Boot com Liquibase**

Qualquer aplicativo que dependa de um banco de dados se beneficiará de uma ferramenta de gerenciamento de banco de dados e, para projetos empresariais, é essencial. As duas soluções mais populares são Liquibase e Flyway.

Não vou me aprofundar nas diferenças entre eles: ambos são bem parecidos com algumas distinções que tornam o Liquibase mais atraente para mim pessoalmente (como a geração de diff, por exemplo). Em vez disso, vou guiá-lo pela configuração do Liquibase com um projeto Spring Boot e descrever duas maneiras de criar changelogs: manualmente e automaticamente.

Também fornecerei um breve resumo da estrutura e dos recursos do Liquibase para que você tenha uma melhor compreensão dos principais conceitos ao concluir o tutorial.

**Estrutura e comandos básicos do Liquibase**

O Liquibase permite que você evolua seu esquema de banco de dados usando scripts SQL ou formatos agnósticos de dados, como XML, JSON ou YAML. Scripts agnósticos de dados são especialmente úteis quando você precisa gerenciar o mesmo esquema em vários bancos de dados.

O fluxo de trabalho do Liquibase é baseado nos seguintes conceitos principais:

* Os logs de alterações são os arquivos usados ​​para monitorar as alterações aplicadas ao banco de dados.
* Changesets são conjuntos de alterações aplicadas ao banco de dados. Eles são incluídos no changelog giles.
* Os tipos de alteração são operações independentes do banco de dados especificadas nos conjuntos de alterações e usadas para atualizar o banco de dados (por exemplo, adicionar uma coluna).
* As tags do changelog ajudam a controlar quando e em qual banco de dados as alterações são feitas.

Existem três tipos de tags de changelog:

* Tags de pré-condição são adicionadas aos changelogs ou conjuntos de alterações e usadas para controlar sua execução com base no estado do banco de dados.
* As tags de contexto controlam se determinados conjuntos de alterações são executados em tempo de execução. Eles são adicionados aos conjuntos de alterações na forma de expressões e, em seguida, especificados com um --context-filterem tempo de execução, e apenas os conjuntos de alterações que correspondem a um filtro especificado são implantados.
* As tags de rótulo são semelhantes às tags de contexto, a diferença é que elas são adicionadas aos conjuntos de alterações como rótulos e especificadas em tempo de execução com uma --label-filteropção.

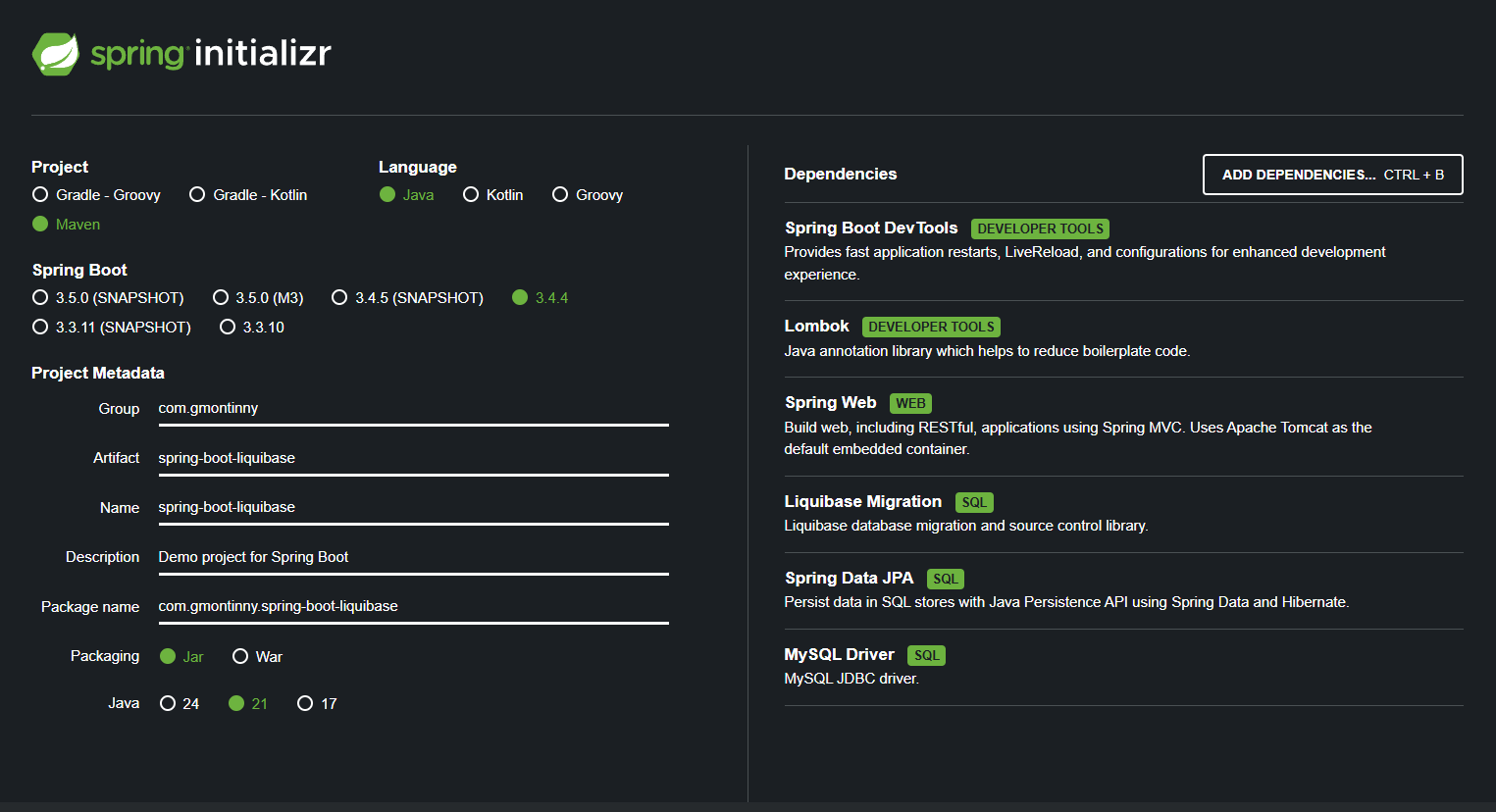
Além disso, existem cerca de 40 comandos divididos em seis categorias:

* **Init** para iniciar o Liquibase no seu projeto,
* **Update**para executar alterações não implantadas,
* **Rollback**para desfazer alterações,
* **Inspection**para comparar diferenças entre bancos de dados,
* **Change** para obter o status das alterações,
* **Utility**para gerenciar arquivos Liquibase.

**Crie um projeto Spring Boot**

Primeiro, vamos criar um projeto totalmente novo via [Spring Initializr](https://start.spring.io/) . Selecione Java, Maven, a versão estável mais recente do Spring Boot (a minha é 3.4.4), Java versão 17. Altere o grupo e o nome do projeto como desejar e adicione as seguintes dependências:

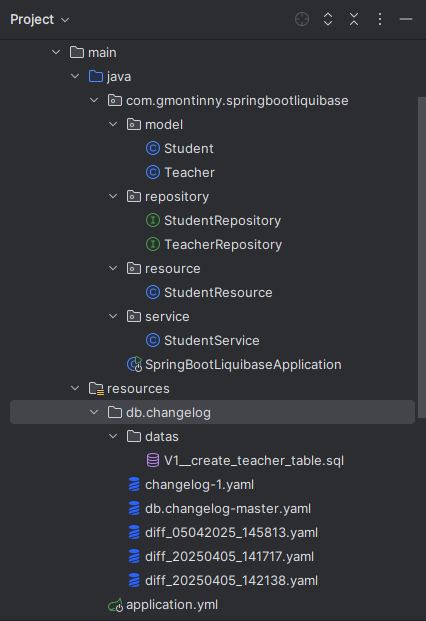
* Lombok,
* Spring Web,
* Spring Data JPA,
* MySQL Driver,
* Liquibase Migration,
* Dev Tools



Configuração do arquivo *pom.xml*:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>3.4.4</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.gmontinny</groupId>  <artifactId>spring-boot-liquibase</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>spring-boot-liquibase</name>  <description>spring-boot-liquibase</description>  <url/>  <licenses>  <license/>  </licenses>  <developers>  <developer/>  </developers>  <scm>  <connection/>  <developerConnection/>  <tag/>  <url/>  </scm>  <properties>  <java.version>17</java.version>  <user.language>en</user.language>  <user.country>US</user.country>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.liquibase</groupId>  <artifactId>liquibase-core</artifactId>  </dependency>   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <scope>runtime</scope>  <optional>true</optional>  </dependency>  <dependency>  <groupId>com.mysql</groupId>  <artifactId>mysql-connector-j</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <optional>true</optional>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <annotationProcessorPaths>  <path>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </path>  </annotationProcessorPaths>  </configuration>  </plugin>  <!-- Maven Properties Plugin para data atual -->  <plugin>  <groupId>org.codehaus.mojo</groupId>  <artifactId>build-helper-maven-plugin</artifactId>  <version>3.4.0</version>  <executions>  <execution>  <id>add-current-date</id>  <goals>  <goal>timestamp-property</goal>  </goals>  <phase>validate</phase>  <configuration>  <name>current.datetime</name>  <pattern>ddMMyyyy\_HHmmss</pattern>  </configuration>  </execution>  </executions>  </plugin>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <excludes>  <exclude>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </exclude>  </excludes>  </configuration>  </plugin>  <!-- Liquibase Maven Plugin -->  <plugin>  <groupId>org.liquibase</groupId>  <artifactId>liquibase-maven-plugin</artifactId>  <version>4.31.1</version>  <configuration>  <changeLogFile>src/main/resources/db/changelog/db.changelog-master.yaml</changeLogFile>  <diffChangeLogFile>src/main/resources/db/changelog/diff\_${current.datetime}.yaml</diffChangeLogFile>  <url>jdbc:mysql://localhost:3306/db\_test?createDatabaseIfNotExist=true&amp;allowPublicKeyRetrieval=true&amp;useSSL=false&amp;useUnicode=true&amp;characterEncoding=UTF-8&amp;serverTimezone=UTC</url>  <username>gmontinny</username>  <password>1203</password>  <referenceUrl>hibernate:spring:com.gmontinny?dialect=org.hibernate.dialect.MySQL8Dialect</referenceUrl>  <driver>com.mysql.cj.jdbc.Driver</driver>  </configuration>  <dependencies>  <dependency>  <groupId>org.liquibase.ext</groupId>  <artifactId>liquibase-hibernate6</artifactId>  <version>4.31.1</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  <version>3.4.4</version>  </dependency>  </dependencies>  </plugin>  </plugins>  </build>  </project> |

**Arquitetura do projeto**



Agora vamos criar uma docker-compose para o banco mysql.

Cria o seguinte arquivo *docker-compose-mysql.yaml*

|  |
| --- |
| version: '3.9' services:  db:  image: mysql:latest  container\_name: mysql-container  restart: always  environment:  MYSQL\_ROOT\_PASSWORD: 1203 *# Substitua pelo valor desejado* MYSQL\_DATABASE: db\_test *# Nome do banco de dados inicial* MYSQL\_USER: gmontinny *# Nome do usuário* MYSQL\_PASSWORD: 1203 *# Senha do usuário* ports:  - "3306:3306"  volumes:  - db\_data:/var/lib/mysql  volumes:  db\_data: |

Depois execute com o comando:

|  |
| --- |
| *docker-compose up -d* |

Atribuir configuração do arquivo *application.yml*

|  |
| --- |
| spring:  application:  name: spring-boot-liquibase   datasource:  url: jdbc:mysql://localhost:3306/db\_test?createDatabaseIfNotExist=true&allowPublicKeyRetrieval=true&useSSL=false&useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  username: gmontinny  password: 1203  driver-class-name: com.mysql.cj.jdbc.Driver   jpa:  hibernate:  ddl-auto: update  properties:  hibernate:  dialect: org.hibernate.dialect.MySQL8Dialect   liquibase:  change-log: classpath:/db/changelog/db.changelog-master.yaml  server:  port: 8080 |

O próximo passo é criar uma classe Entity simples, um repositório e um RestController.

O código para a classe Entity chamada *Student* :

|  |
| --- |
| @Entity @Table(name = "students") @AllArgsConstructor @NoArgsConstructor @Getter @Setter public class Student implements Serializable {   @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  @Column(name = "id", nullable = false)  private Long id;    @Column(name = "first\_name", nullable = false)  private String firstName;    @Column(name = "last\_name", nullable = false)  private String lastName;    @Column(name = "email", nullable = false)  private String email;  } |

Agora criar um interface repository para Student:

|  |
| --- |
| public interface StudentRepository extends JpaRepository<Student, Long> { } |

Cria uma classe service chamado *StudentService:*

|  |
| --- |
| @RequiredArgsConstructor @Service public class StudentService {  private final StudentRepository repository;   */\*\*  \* Método para buscar todos os estudantes utilizando programação funcional  \*  \* @return Lista de estudantes  \*/* public List<Student> findAll() {  return repository.findAll()  .stream()  .collect(Collectors.*toList*());  }  } |

Criar um RestController chamado StudentResource

|  |
| --- |
| @RequiredArgsConstructor @RestController public class StudentResource {  private final StudentService service;   @GetMapping("/students")  List<Student> findAll() {  return service.findAll();  }  } |

**Criar arquivos de changelog do Liquibase**

Já passou da hora de adicionarmos um script Liquibase para gerenciar nosso banco de dados.

Como adicionamos a dependência Liquibase ao criar o projeto, o Spring Boot adicionou automaticamente os diretórios *db/changelog* necessários aos *recursos* . Se você estiver adicionando Liquibase ao projeto existente, você mesmo terá que criá-los.

No diretório *db/changelog* , crie um arquivo *db.changelog-master.yaml* com o seguinte conteúdo:

|  |
| --- |
| databaseChangeLog:  - include:  file: db/changelog/changelog-1.yaml |

Às vezes, os changesets são adicionados diretamente ao arquivo de changelog mestre. No entanto, a melhor prática é usá-lo como uma coleção de links para changelogs com as mudanças reais.

Neste caso, apontamos para o arquivo *changelog-1.yaml* . Vamos criá-lo no mesmo diretório *db/changelog* e preenchê-lo com o seguinte conteúdo:

|  |
| --- |
| databaseChangeLog:  - changeSet:  id: 1  author: Giovanny  preConditions:  - onFail: MARK\_RAN  not:  tableExists:  tableName: students  changes:  - createTable:  tableName: students  columns:  - column:  name: id  type: BIGINT  autoIncrement: true  constraints:  primaryKey: true  nullable: false  - column:  name: first\_name  type: varchar(250)  constraints:  nullable: false  - column:  name: last\_name  type: varchar(250)  constraints:  nullable: false  - column:  name: email  type: varchar(250)  constraints:  nullable: false  - changeSet:  id: 2  author: Giovanny  changes:  - insert:  tableName: students  columns:  - column:  name: first\_name  value: "Giovanny"  - column:  name: last\_name  value: "Montinny"  - column:  name: email  value: "gmontinny@gmail.com" |

Vamos analisar isso mais de perto.

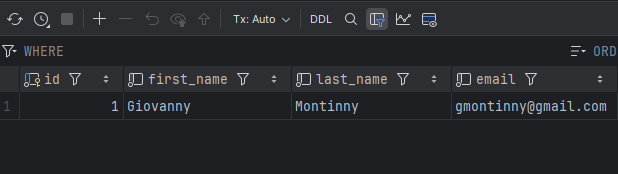
O arquivo contém dois changesets: um para criar a tabela e outro para preenchê-la com dados de teste. Cada changeset tem um id exclusivo. Observe que é uma boa prática executar apenas uma alteração por changeset para facilitar o rollback.

O primeiro conjunto de alterações contém uma seção preConditions que informa ao Liquibase para marcar o conjunto de alterações como executado com sucesso sem executá-lo se a tabela já existir.

Note que o script para criar uma tabela e preenchê-la com dados é agnóstico em relação aos dados. Independentemente do banco de dados que você estiver usando, o Liquibase gerará o script de acordo.

O segundo conjunto de alterações preenche nossa tabela com dados de amostra.

Execute o aplicativo. Você deve ver o banco de dados recém-criado db\_test na ferramenta de administração de banco de dados que você usa. O BD contém três tabelas, duas das quais são específicas do Liquibase e criadas automaticamente.



**Gerar changelogs automaticamente usando o Liquibase diff**

Suponha que você queira alterar o esquema do seu banco de dados: ajuste colunas existentes e adicione novas. Você pode adicionar alterações aos changelogs manualmente ou aproveitar o recurso diff do Liquibase.

O recurso diff nos permite comparar dois bancos de dados ou um banco de dados e novas entidades de persistência. Para usá-lo, você precisa adicionar um liquibase-maven-plugin e uma extensão Liquibase Hibernate ao seu pom.xml:

|  |
| --- |
| <!-- Liquibase Maven Plugin --> <plugin>  <groupId>org.liquibase</groupId>  <artifactId>liquibase-maven-plugin</artifactId>  <version>4.31.1</version>  <configuration>  <changeLogFile>src/main/resources/db/changelog/db.changelog-master.yaml</changeLogFile>  <diffChangeLogFile>src/main/resources/db/changelog/diff\_${current.datetime}.yaml</diffChangeLogFile>  <url>jdbc:mysql://localhost:3306/db\_test?createDatabaseIfNotExist=true&amp;allowPublicKeyRetrieval=true&amp;useSSL=false&amp;useUnicode=true&amp;characterEncoding=UTF-8&amp;serverTimezone=UTC</url>  <username>gmontinny</username>  <password>1203</password>  <referenceUrl>hibernate:spring:com.gmontinny?dialect=org.hibernate.dialect.MySQL8Dialect</referenceUrl>  <driver>com.mysql.cj.jdbc.Driver</driver>  </configuration>  <dependencies>  <dependency>  <groupId>org.liquibase.ext</groupId>  <artifactId>liquibase-hibernate6</artifactId>  <version>4.31.1</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  <version>3.4.4</version>  </dependency>  </dependencies> </plugin> |

Além de uma configuração para formatação de data e hora:

|  |
| --- |
| <!-- Maven Properties Plugin para data atual --> <plugin>  <groupId>org.codehaus.mojo</groupId>  <artifactId>build-helper-maven-plugin</artifactId>  <version>3.4.0</version>  <executions>  <execution>  <id>add-current-date</id>  <goals>  <goal>timestamp-property</goal>  </goals>  <phase>validate</phase>  <configuration>  <name>current.datetime</name>  <pattern>ddMMyyyy\_HHmmss</pattern>  </configuration>  </execution>  </executions> </plugin> |

Há muita coisa acontecendo na configuração do plugin:

* changeLogFile a ponta para nosso registro de alterações principal.
* diffChangeLogFile a ponta para o arquivo changelog que será gerado como resultado da comparação do banco de dados. O nome do arquivo contém um timestamp para facilitar o rastreamento de versões.
* url a ponta para o banco de dados de destino que usamos para comparação.
* referenceURL a ponta para o pacote no aplicativo onde nossas entidades JPA residem. Ele também especifica Hibernate Physical Naming Strategy e Hibernate Implicit naming Strategy para evitar incompatibilidades de API.

Agora, vamos atualizar a classe Student:

|  |
| --- |
| @Column(name = "email\_address", nullable = false) private String email;  @Column(name = "phone\_number") private int phoneNumber; |

Liquibase:diff usa classes compiladas do diretório */target* para comparação, então precisamos reconstruir o projeto. Execute o seguinte comando:

|  |
| --- |
| mvn clean install liquibase:diff -DskipTests=true |

Um arquivo com conteúdo semelhante deve aparecer no diretório *db/changelog* com formatação de *diff\_05042025\_145813.yaml*

Como você pode ver, o Liquibase detectou todas as alterações que introduzimos na classe Student.

**Gerando Linquibase através de uma arquivo .sql**

Para esse processo vamos criar um pasta no seguinte endereço:

|  |
| --- |
| db\changelog\dastas |

Depois vamos criar um arquivo com o seguinte nome *V1\_\_create\_teacher\_table.sql*

|  |
| --- |
| CREATE TABLE teacher (  id BIGINT NOT NULL AUTO\_INCREMENT PRIMARY KEY,  name VARCHAR(100) NOT NULL,  email VARCHAR(150) UNIQUE NOT NULL,  subject VARCHAR(100) NOT NULL,  created\_at TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,  updated\_at TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* ON UPDATE *CURRENT\_TIMESTAMP* ); |

Agora vamos alterar o arquivo *db.changelog-master.yaml*.

|  |
| --- |
| databaseChangeLog:  - include:  file: db/changelog/changelog-1.yaml  - changeSet:  id: create-teacher-table  author: Giovanny  changes:  - sqlFile:  path: db/changelog/datas/V1\_\_create\_teacher\_table.sql  relativeToChangelogFile: false |

Vamos criar a classe *Teacher*

|  |
| --- |
| @Entity @Table(name = "teacher") @AllArgsConstructor @NoArgsConstructor @Getter @Setter public class Teacher implements Serializable {  @Id  @GeneratedValue(strategy = GenerationType.*IDENTITY*)  @Column(name = "id", nullable = false)  private Long id;   @Column(name = "name", nullable = false)  private String name;   @Column(name = "email", nullable = false)  private String email;   // Coluna 'subject'  @Column(name = "subject", nullable = false)  private String subject;   // Colunas 'created\_at' e 'updated\_at'  @Column(name = "created\_at", updatable = false)  private LocalDateTime createdAt;   @Column(name = "updated\_at")  private LocalDateTime updatedAt;   @PrePersist  protected void onCreate() {  this.createdAt = LocalDateTime.*now*();  this.updatedAt = LocalDateTime.*now*();  }   @PreUpdate  protected void onUpdate() {  this.updatedAt = LocalDateTime.*now*();  }  } |

E agora o repository chamado *TeacherRepository*

|  |
| --- |
| public interface TeacherRepository extends JpaRepository<Teacher, Long> {} |

Agora é só executar novamente o comando:

|  |
| --- |
| mvn clean install liquibase:diff -DskipTests=true |

**Conclusão**

Como você pode ver, o Liquibase é uma ferramenta poderosa para migrar o esquema de banco de dados do seu aplicativo Java e manter o controle de todas as alterações no projeto. Você pode criar changelogs manualmente ou confiar no liquibase:diff para gerá-los automaticamente em cenários simples, o que economiza muito tempo.

**Referrências**

https://contribute.liquibase.com/extensions-integrations/directory/integration-docs/springboot/