Gestion des schémas de base de données avec Liquibase

Voici un guide complet sur **comment utiliser Liquibase** dans une application Spring Boot pour gérer efficacement les modifications de schéma de base de données :

---

1. Ajouter la dépendance Liquibase

Pour commencer, vous devez inclure la dépendance Liquibase dans votre projet. Si vous utilisez **Maven**, ajoutez ceci à votre `pom.xml` :

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

Pour **Gradle**, ajoutez ceci à votre `build.gradle` :

```
implementation 'org.liquibase:liquibase-core'
```

Cela intègre la bibliothèque principale de Liquibase dans votre projet.

---

2. Configurer Liquibase

Liquibase utilise un **fichier de journal des modifications** pour gérer les modifications de schéma de base de données. Par défaut, Spring Boot recherche un fichier nommé `db/changelog/db.changelog-master.yaml`, `db/changelog/db.changelog-master.xml` ou `db/changelog/db.changelog-master.sql` dans le classpath. Vous pouvez personnaliser cet emplacement en ajoutant une propriété à votre `application.properties` (ou `application.yml`) :

```
spring.liquibase.change-log=classpath:/db/changelog/db.changelog-master.xml
```

Cela indique à Spring Boot où trouver votre fichier de journal des modifications.

---

3. Créer un fichier de journal des modifications

Le fichier de journal des modifications définit les modifications que vous souhaitez appliquer à votre base de données. Vous pouvez l'écrire dans des formats comme XML, YAML ou SQL. Voici un exemple de **fichier de journal des modifications XML** situé à `src/main/resources/db/changelog/db.changelog-master.xml` :

```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-3.8.xsd">

    <changeSet id="1" author="your-name">
        <createTable tableName="users">
            <column name="id" type="int">
                <constraints primaryKey="true" nullable="false"/>
            </column>
            <column name="username" type="varchar(255)"/>
            <column name="email" type="varchar(255)"/>
        </createTable>
    </changeSet>

</databaseChangeLog>
```

Cet exemple crée une table `users` avec trois colonnes : `id`, `username` et `email`. Chaque `<changeSet>` représente un ensemble de modifications à appliquer.

---

4. Exécuter votre application Spring Boot

Lorsque vous démarrez votre application Spring Boot, Liquibase fait automatiquement : - Lit le fichier de journal des modifications. - Vérifie quels jeux de modifications ont déjà été appliqués (suivis dans une table appelée `DATABASECHANGELOG`). - Exécute tout nouveau jeu de modifications contre votre base de données.

Aucun code supplémentaire n'est nécessaire—l'auto-configuration de Spring Boot s'en charge pour vous.

---

5. Personnaliser Liquibase (Optionnel)

Vous pouvez ajuster le comportement de Liquibase en utilisant des propriétés dans `application.properties`. Voici quelques options courantes :

```
spring.liquibase.enabled=true          # Activer ou désactiver Liquibase
spring.liquibase.drop-first=false      # Supprimer la base de données avant d'appliquer les modifications (uti
spring.liquibase.contexts=dev,prod     # Exécuter les jeux de modifications uniquement dans des contextes spéc
```

Ces paramètres vous permettent d'adapter Liquibase à votre environnement ou à votre flux de travail.

---

6. Utiliser des fonctionnalités avancées

Liquibase propose des fonctionnalités puissantes pour améliorer la gestion des schémas : - **Contextes et étiquettes** : Contrôlez quels jeux de modifications s'exécutent dans des environnements spécifiques (par exemple, `dev` vs. `prod`). - **Préconditions** : Assurez-vous que certaines conditions (par exemple, une table existe) sont remplies avant d'appliquer un jeu de modifications. - **Annulations** : Définissez comment annuler un jeu de modifications si nécessaire. - **Refactorisation** : Utilisez le support intégré pour des modifications de base de données complexes (par exemple, renommer des colonnes).

Par exemple, ajouter un contexte à un jeu de modifications :

```
<changeSet id="2" author="your-name" context="dev">
    <addColumn tableName="users">
        <column name="created_at" type="timestamp"/>
    </addColumn>
</changeSet>
```

Ce jeu de modifications ne s'exécute que dans le contexte `dev`.

---

7. Tester avec une base de données en mémoire

Pour les tests, associez Liquibase à une base de données en mémoire comme H2. Ajoutez la dépendance H2 :

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Ensuite, configurez votre `application.properties` pour les tests :

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.liquibase.change-log=classpath:/db/changelog/db.changelog-master.xml
```

Liquibase appliquera vos modifications à la base de données en mémoire pendant les tests.

---

8. Contrôle de version de votre journal des modifications

Stockez vos fichiers de journal des modifications dans un contrôle de version (par exemple, Git) aux côtés de votre code d'application. Cela garantit que votre schéma de base de données évolue en synchronisation avec votre base de code, rendant les déploiements et les annulations plus prévisibles.

Résumé

Pour utiliser Liquibase dans une application Spring Boot : 1. Ajoutez la dépendance Liquibase. 2. Configurez l'emplacement du fichier de journal des modifications. 3. Créez un fichier de journal des modifications avec vos modifications de base de données. 4. Exécutez votre application—Liquibase s'occupe du reste.

Liquibase est un outil robuste pour gérer les modifications de schéma de base de données de manière structurée et contrôlée par version, intégrée de manière transparente avec Spring Boot.