Liquibase: Datenbank-Schema-Management

Hier ist eine umfassende Anleitung zur Verwendung von **Liquibase** in einer Spring Boot-Anwendung, um Ihre Datenbankschemaänderungen effektiv zu verwalten:

---

1. Liquibase-Abhängigkeit hinzufügen

Um loszulegen, müssen Sie die Liquibase-Abhängigkeit in Ihr Projekt einbinden. Wenn Sie **Maven** verwenden, fügen Sie dies zu Ihrer `pom.xml` hinzu:

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

Für **Gradle** fügen Sie dies zu Ihrer `build.gradle` hinzu:

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

Dadurch wird die Liquibase-Kernbibliothek in Ihr Projekt eingebunden.

---

2. Liquibase konfigurieren

Liquibase verwendet eine **Changelog-Datei**, um Datenbankschemaänderungen zu verwalten. Standardmäßig sucht Spring Boot nach einer Datei mit dem Namen `db/changelog/db.changelog-master.yaml`, `db/changelog/db.changelog-master.xml` oder `db/changelog/db.changelog-master.sql` im Klassenspeicher. Sie können diesen Speicherort anpassen, indem Sie eine Eigenschaft zu Ihrer `application.properties` (oder `application.yml`) hinzufügen:

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

Dies teilt Spring Boot mit, wo sich Ihre Changelog-Datei befindet.

---

3. Eine Changelog-Datei erstellen

Die Changelog-Datei definiert die Änderungen, die an Ihrer Datenbank vorgenommen werden sollen. Sie können sie in Formaten wie XML, YAML oder SQL schreiben. Hier ist ein Beispiel für eine **XML-Changelog-Datei**, die sich an `src/main/resources/db/changelog/db.changelog-master.xml` befindet:

```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>
```

Dieses Beispiel erstellt eine `users`-Tabelle mit drei Spalten: `id`, `username` und `email`. Jedes `<changeSet>` stellt einen Satz von Änderungen dar, die angewendet werden sollen.

---

4. Ihre Spring Boot-Anwendung ausführen

Wenn Sie Ihre Spring Boot-Anwendung starten, führt Liquibase automatisch folgende Schritte aus: - Liest die Changelog-Datei. - Überprüft, welche Changesets bereits angewendet wurden (verfolgt in einer Tabelle namens `DATABASECHANGELOG`). - Führt alle neuen Changesets an Ihrer Datenbank aus.

Kein zusätzlicher Code ist erforderlich—Spring Boots Autokonfiguration erledigt dies für Sie.

---

5. Liquibase anpassen (optional)

Sie können das Verhalten von Liquibase mit Eigenschaften in `application.properties` anpassen. Hier sind einige gängige Optionen:

```
spring.liquibase.enabled=true          # Liquibase aktivieren oder deaktivieren
spring.liquibase.drop-first=false      # Datenbank vor dem Anwenden von Änderungen löschen (mit Vorsicht verwe
spring.liquibase.contexts=dev,prod     # Changesets nur in bestimmten Kontexten ausführen
```

Diese Einstellungen ermöglichen es Ihnen, Liquibase an Ihre Umgebung oder Ihren Arbeitsablauf anzupassen.

---

6. Fortgeschrittene Funktionen nutzen

Liquibase bietet leistungsstarke Funktionen zur Verbesserung der Schemaverwaltung: - **Kontexte und Labels**: Steuern Sie, welche Changesets in bestimmten Umgebungen (z. B. `dev` vs. `prod`) ausgeführt werden. - **Vorbedingungen**: Stellen Sie sicher, dass bestimmte Bedingungen (z. B. eine Tabelle existiert) vor dem Anwenden eines Changesets erfüllt sind. - **Rückgänge**: Definieren Sie, wie ein Changeset bei Bedarf rückgängig gemacht werden kann. - **Refactoring**: Nutzen Sie die eingebaute Unterstützung für komplexe Datenbankänderungen (z. B. Umbenennen von Spalten).

Zum Beispiel, ein Kontext zu einem Changeset hinzufügen:

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

Dieses Changeset wird nur im `dev`-Kontext ausgeführt.

---

7. Mit einer In-Memory-Datenbank testen

Zum Testen kombinieren Sie Liquibase mit einer In-Memory-Datenbank wie H2. Fügen Sie die H2-Abhängigkeit hinzu:

```
<dependency>
    <groupId>com.h2database</groupId>
```

```
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Konfigurieren Sie dann Ihre `application.properties` für Tests:

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

Liquibase wird Ihre Änderungen während der Tests auf die In-Memory-Datenbank anwenden.

---

8. Ihre Changelog-Datei in der Versionskontrolle speichern

Speichern Sie Ihre Changelog-Dateien in der Versionskontrolle (z. B. Git) zusammen mit Ihrem Anwendungscode. Dadurch stellt sicher, dass sich Ihr Datenbankschema synchron mit Ihrem Code entwickelt, was Bereitstellungen und Rückgänge vorhersehbarer macht.

Zusammenfassung

Um Liquibase in einer Spring Boot-Anwendung zu verwenden: 1. Fügen Sie die Liquibase-Abhängigkeit hinzu. 2. Konfigurieren Sie den Speicherort der Changelog-Datei. 3. Erstellen Sie eine Changelog-Datei mit Ihren Datenbankänderungen. 4. Starten Sie Ihre Anwendung—Liquibase erledigt den Rest.

Liquibase ist ein robustes Werkzeug zur strukturierten, versionskontrollierten Verwaltung von Datenbankschemaänderungen, das nahtlos in Spring Boot integriert ist.