# Liquibase: Database Scheme Management

Here's a comprehensive guide on **how to use Liquibase** in a Spring Boot application to manage your database schema changes effectively:

---

## 1. Add Liquibase Dependency

To get started, you need to include the Liquibase dependency in your project. If you're using **Maven**, add this to your `pom.xml`:

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

For **Gradle**, add this to your `build.gradle`:

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

This brings the Liquibase core library into your project.

---

## 2. Configure Liquibase

Liquibase uses a **changelog file** to manage database schema changes. By default, Spring Boot looks for a file named `db/changelog/db.changelog-master.yaml`, `db/changelog/db.changelog-master.xml`, or `db/changelog/db.changelog-master.sql` in the classpath. You can customize this location by adding a property to your `application.properties` (or `application.yml`):

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

This tells Spring Boot where to find your changelog file.

---

### 3. Create a Changelog File

The changelog file defines the changes you want to apply to your database. You can write it in formats like XML, YAML, or SQL. Here's an example of an **XML changelog** file located at `src/main/resources/db/changelog/db.chan`

```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>
```

This example creates a `users` table with three columns: `id`, `username`, and `email`. Each `<changeSet>` represents a set of changes to apply.

---

### 4. Run Your Spring Boot Application

When you start your Spring Boot application, Liquibase automatically: - Reads the changelog file. - Checks which changesets have already been applied (tracked in a table called `DATABASECHANGELOG`). - Executes any new changesets against your database.

No additional code is needed—Spring Boot's auto-configuration handles this for you.

---

### 5. Customize Liquibase (Optional)

You can tweak Liquibase's behavior using properties in `application.properties`. Here are some common options:

```
spring.liquibase.enabled=true          # Enable or disable Liquibase
spring.liquibase.drop-first=false      # Drop the database before applying changes (use with caution)
spring.liquibase.contexts=dev,prod     # Run changesets only in specific contexts
```

These settings allow you to adapt Liquibase to your environment or workflow.

---

## 6. Leverage Advanced Features

Liquibase offers powerful features to enhance schema management: - **Contexts and Labels**: Control which changesets run in specific environments (e.g., `dev` vs. `prod`). - **Preconditions**: Ensure certain conditions (e.g., a table exists) are met before applying a changeset. - **Rollbacks**: Define how to undo a changeset if needed. - **Refactoring**: Use built-in support for complex database changes (e.g., renaming columns).

For example, adding a context to a changeset:

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

This changeset only runs in the `dev` context.

---

## 7. Test with an In-Memory Database

For testing, pair Liquibase with an in-memory database like H2. Add the H2 dependency:

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

Then configure your `application.properties` for testing:

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

Liquibase will apply your changes to the in-memory database during tests.

---

**8. Version Control Your Changelog**

Store your changelog files in version control (e.g., Git) alongside your application code. This ensures your database schema evolves in sync with your codebase, making deployments and rollbacks more predictable.

---

**Summary**

To use Liquibase in a Spring Boot application: 1. Add the Liquibase dependency. 2. Configure the changelog file location. 3. Create a changelog file with your database changes. 4. Run your application—Liquibase handles the rest.

Liquibase is a robust tool for managing database schema changes in a structured, version-controlled way, seamlessly integrated with Spring Boot.