**Liquibase Software**

<https://github.com/liquibase-github-actions>

<https://forum.liquibase.org/>

<https://www.youtube.com/channel/UC5qMsRjObu685rTBq0PJX8w>

Veritabanına yapılan değişiklikleri loglamak ve yönetmek için kullanılır.

Komutlarla çalışır.

**Şu veritabanlarında kullanılabilir:**

* Oracle
* Postgresql
* SQL Server/MSSQL
* OracleATP
* H2
* DB2 on Z
* Maria DB
* MySQL
* SQLite
* Informix
* Apache Derby

**Veritabanı şema değişiklikleri şu dillerde yazılabilir:**

* SQL
* XML
* YAML (YAML'de girinti (indentation) çok hassas bir konu)
* JSON

**Bu dillerin her birinde tablo oluşturma kodu şu şekilde:**

**XML(En yaygın ve Liquibase belgelerinde en çok örneği olan):**

<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="sena">

<createTable tableName="users">

<column name="id" type="BIGINT">

<constraints primaryKey="true" nullable="false"/>

</column>

<column name="username" type="VARCHAR(50)">

<constraints nullable="false"/>

</column>

</createTable>

</changeSet>

</databaseChangeLog>

**YAML(Okunabilirliği yüksek, son yıllarda popüler):**

databaseChangeLog:

- changeSet:

id: 1

author: sena

changes:

- createTable:

tableName: users

columns:

- column:

name: id

type: BIGINT

constraints:

primaryKey: true

nullable: false

- column:

name: username

type: VARCHAR(50)

constraints:

nullable: false

**JSON:**

{

"databaseChangeLog": [

{

"changeSet": {

"id": "1",

"author": "sena",

"changes": [

{

"createTable": {

"tableName": "users",

"columns": [

{

"column": {

"name": "id",

"type": "BIGINT",

"constraints": {

"primaryKey": true,

"nullable": false

}

}

},

{

"column": {

"name": "username",

"type": "VARCHAR(50)",

"constraints": {

"nullable": false

}

}

}

]

}

}

]

}

}

]

}

**SQL(Sadece SQL bilen ekipler için uygun ama Liquibase'in güçlü özellikleri (rollback, platform bağımsızlık vb.) sınırlı kalır):**

--liquibase formatted sql

--changeset sena:1

CREATE TABLE users (

id BIGINT PRIMARY KEY NOT NULL,

username VARCHAR(50) NOT NULL

);

Changelog, changeset veya change type object oluşturulabilir

Liquibase has **rollback support**

* Liquibase kullanıcıları, SQL kodlarını bir versiyon kontrol sistemine (örneğin Git) eklediklerinde, uygulama kodunda olduğu gibi aynı avantajlardan yararlanır.

Yani kodun geçmişini görebilir, kim ne zaman ne değiştirmiş takip edebilir, değişiklikleri geri alabilir vs.

* Liquibase changesetleri (veritabanı değişikliklerini tanımlayan küçük dosyalar) versiyon kontrol sisteminde tutulduğunda:
  + - Veritabanının hangi durumda olduğu anlaşılır.
    - Hangi değişikliklerin uygulandığı izlenebilir.
* Birden fazla geliştirici aynı anda değişiklikler üzerinde çalışabilir.
* Kod birleştirme (merge) zamanı geldiğinde, çakışma (conflict) riski düşer çünkü her changeset benzersizdir (özellikle id ve author alanı sayesinde).
* Liquibase'i kullanmak için farklı bir versiyon kontrol sistemine geçmeye gerek yok.  
  Zaten kullandığın Git, SVN gibi sistemlerle birlikte çalışır.

Yani: Liquibase ile yapılan veritabanı değişikliklerini Git gibi bir sistemde saklayarak, kodda olduğu gibi sürüm kontrolü yapabilir, ekip içinde düzenli bir şekilde çalışabilir ve geçmişi takip edebilirsin.

Chageset şunları içerir:

**The changelog header, changesets, and changeset attributes**

**Changeset Örnekleri(XML):**

**Tablo Oluşturma (createTable)**

<changeSet id="1" author="sena">

<createTable tableName="product">

<column name="id" type="BIGINT">

<constraints primaryKey="true" nullable="false"/>

</column>

<column name="name" type="VARCHAR(100)">

<constraints nullable="false"/>

</column>

<column name="price" type="DECIMAL(10,2)"/>

<column name="created\_at" type="TIMESTAMP"/>

</createTable>

</changeSet>

**Var Olan Tabloya Kolon Ekleme (addColumn)**

<changeSet id="2" author="sena">

<addColumn tableName="product">

<column name="stock" type="INT" defaultValueNumeric="0"/>

</addColumn>

</changeSet>

**Veri Ekleme (insert)**

<changeSet id="3" author="sena">

<insert tableName="product">

<column name="id" value="1"/>

<column name="name" value="Laptop"/>

<column name="price" value="999.99"/>

<column name="created\_at" valueDate="2025-03-10T12:00:00"/>

</insert>

</changeSet>

**Veri Güncelleme (update)**

<changeSet id="4" author="sena">

<update tableName="product">

<column name="price" value="899.99"/>

<where>id = 1</where>

</update>

</changeSet>

**Tablo Silme (dropTable)**

<changeSet id="6" author="sena">

<dropTable tableName="product"/>

</changeSet>

**Applicaiton.Properties:**

**En yaygın:**

#Liquibase'i etkinleştirme:

spring.liquibase.enabled=

#Changelog dosyasının yolu:

spring.liquibase.change-log=

#Liquibase komutunun log seviyesi:

spring.liquibase.log-level=

#Değişikliklerin sadece belirli bir ortamda uygulanması için kullanılan context:

spring.liquibase.contexts=

#Veritabanı sıfırlama (drop) işlemi:

spring.liquibase.drop-first=

#SQL dosyalarının karakter seti:

spring.liquibase.sql-script-encoding=

#Rollback için kullanılacak changelog dosyası:

spring.liquibase.rollback-file=

#Veritabanı değişikliklerini takip eden tablo adı:

spring.liquibase.database-change-log-table=

#Veritabanı değişikliklerini takip eden kilit tablosu adı:

spring.liquibase.database-change-log-lock-table=

#Liquibase komutunun çalıştırılması (örneğin: update, rollback):

spring.liquibase.command=

**Extra:**

#Liquibase'in databaseChangeLog ve databaseChangeLogLock tablolarını sıfırlama işlemi:

spring.liquibase.clearCheckSums=

#Veritabanı için Liquibase'in JDBC URL'si (Genellikle Spring DataSource ile birlikte kullanılır):

spring.liquibase.url=

#Veritabanı kullanıcı adı:

spring.liquibase.username=

#Veritabanı şifresi:

spring.liquibase.password=

#Veritabanı bağlantısı için kullanılan JDBC driver:

spring.liquibase.driver=

#Liquibase tarafından kullanılan değişiklik dosyasının versiyon kontrolü:

spring.liquibase.change-log-file=

#Liquibase'in her uygulama başlatıldığında değişiklikleri tekrar kontrol etmesini sağlar:

spring.liquibase.check-changeset=

#Liquibase'in her iki yönde (başlangıç ve geri alma) değiştirilmiş SQL komutlarını loglaması:

spring.liquibase.log-sql=

#Liquibase'in rollback yapılacaksa, kullanılan rollback versiyonunun tarihi:

spring.liquibase.rollback-version=

#Liquibase işlemlerini paralel olarak yürütmek için kullanılan paralel iş parçacığı sayısı:

spring.liquibase.parallel=

#Yalnızca Liquibase işleminde aktif olan şemaları güncellemek:

spring.liquibase.schemas=

CLI içersine komutlar yazabilirsin.

**Liquibase uses the --kebab-case format of lowercase letters and a dash (-) in the CLI environment:**

* Log level is**--log-level.**
* The output file is **--output-file.**

Ör:

liquibase **[global argument]** [command] [command attribute]

liquibase **--changelog-file=dbchangelog.xml** [command] [command attribute]

* **CLI**'yi kullanmak, **Spring Boot** uygulamasını çalıştırmadan bağımsız bir şekilde veritabanı değişikliklerini uygulamanıza olanak sağlar. Ancak, Spring Boot ile **Liquibase** entegrasyonu zaten çok daha kolaydır ve **application.properties** üzerinden ayar yaparak, uygulama başlatıldığında Liquibase işlemleri otomatik olarak yapılır.
* Spring Boot'ta, **application.properties** veya **application.yml** dosyasındaki ayarlarla **Liquibase**'in çalışmasını yönetebilirsiniz. Bu, **CLI** komutlarını manuel olarak yazmanıza gerek kalmadan otomatik hale gelir.

Liquibase CLI (Command Line Interface) bağımsız bir araçtır ve genellikle aşağıdaki durumlar için kullanılır:

* **Bağımsız olarak Liquibase komutlarını çalıştırmak**: CLI, doğrudan terminal veya komut satırından Liquibase komutlarını çalıştırmanıza olanak tanır.
* Bağımsız bir ortamda çalıştırma: Eğer Liquibase'i Spring Boot dışındaki bir ortamda (örneğin, sadece veritabanı yönetimi yapmak amacıyla) çalıştırıyorsanız, Liquibase CLI'yi kullanabilirsiniz.
* CI/CD Pipeline'larında kullanmak: Liquibase'i bir CI/CD pipeline'ında (Jenkins, GitLab CI, CircleCI vb.) bağımsız olarak kullanabilirsiniz. Bu durumda, uygulama çalıştırılmadan önce veritabanı değişikliklerini devreye almak için CLI komutları kullanılabilir.

**Changeset id ve author yazarken dikkat edilmesi gerekenler:**

* The author and id are required since more than one person could use the same id value.
* The author's name or another unique value for the author can be used.
* The id tag does not control the order in which changes are run. It can be descriptive or a number that is unique within the changelog.
* The file name is also part of the identifier so the user does not have to remember the id's used in other files. The id and author combinations only need to be unique in the current file.
* It is important to avoid duplicating author and id combinations in the same file.

**Örnek doğru kullanım:**

**<changeSet id="1" author="nvoxland">** <addColumn tableName="person">   
 <column name="username" type="varchar(8)"/>   
 </addColumn>   
 </changeSet>   
 **<changeSet id="2" author="nvoxland">** <addLookupTable   
 existingTableName="person" existingColumnName="state"   
 newTableName="state" newColumnName="id" newColumnDataType="char(2)"/>   
 </changeSet>