**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**

**Liquibase Community Dosya Kurulumu:**

<?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-4.20.xsd">  
</databaseChangeLog>

**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>

**Branch and File Naming Standards**

When creating branches used with Liquibase, it is best practice to use the **same naming standards** as your application code and be consistent with these standards.

**Common naming standards for branches may include:**

* Feature name.
* Ticket number assigned to the feature or defect.

Another best practice is to **standardize filenames**. Using specific filenames that relate to the object being updated can help guarantee uniqueness and readability.

**Usig Liquibase Cli Comments**

[**https://learn.liquibase.com/unit/view/id:3462**](https://learn.liquibase.com/unit/view/id:3462)

* **The update-sql command.**
* **The update command.**
* **The history command.**

**The syntax used to add a comment:**

* **XML:** <comment>My comment goes here</comment>
* **SQL:** --comment: My comment goes here
* **JSON:** "comment": "My comment goes here",
* **YAML:** comment: My comment goes here

**XML Example**

<changeSet id="1" author="nvoxland" context="test1">  
 <comment>**Jira-1234**</comment>  
 <createTable tableName="test\_table">  
 <column name="id" type="int"/>  
 </createTable>  
 </changeSet>

**SQL Example:**

--liquibase formatted sql  
--changeset example:1  
--comment: **Jira-1234**  
create table test\_table (id int);

**JSON Example:**

"changeSet": {  
 "id": "1",  
 "author": "nathan.voxland",  
 "comment": **"Jira-1234",**  
 "changes": [  
....

**YAML Example:**

- changeSet:  
 id: 1  
 author: your.name  
 comment: **Jira-1234**  
 changes:  
 - createTable:  
 tableName: person  
....

**Viewing  Changeset Comments**

There are two ways comments can be viewed:

* Running the db-doc command.
* Directly viewing the comment in the changelog.

Komutları normal termialer (cmd) yazcan

**To enter the username and password in the CLI, use the following syntax format:**

liquibase --changelog-file=<yourchangelog.xml> --username=<name> --password=<password> command

**Komut formatı (şifre vs başka yerde tutmadıysan):**

liquibase --url=jdbc:postgresql://localhost:5432/databaseadı --username=username --password=şifre –istediğinkomut

**Database Diff Commands**

Liquibase provides **two** diff commands so users can easily **compare** databases to find missing objects, see whether specific changes were made to the database, and identify any unexpected items in the database:

* **diff:** Used to compare two databases of the same, or different types to each other.
* **diff-changelog:** Used to create a changelog file to synchronize multiple databases.

liquibase --outputFile=mydiff.txt --username=<USERNAME> --password=<PASSWORD> --referenceUsername=<USERNAME>

--referencePassword=<PASSWORD> diff

The --outputFile=mydiff.txt attribute will save the output to a file. If a file is not specified, you will see the output in your command line. The username and password attributes are not required for systems that use an alternate means of authentication.

**Database Snapshot Commands**

There are **two** Liquibase commands that provide users with a snapshot of the database:

* **snapshot:** Used to capture the current state of the database.
* **generate-changelog:** Used to create a changelog file that describes how to re-create the current state of the database.

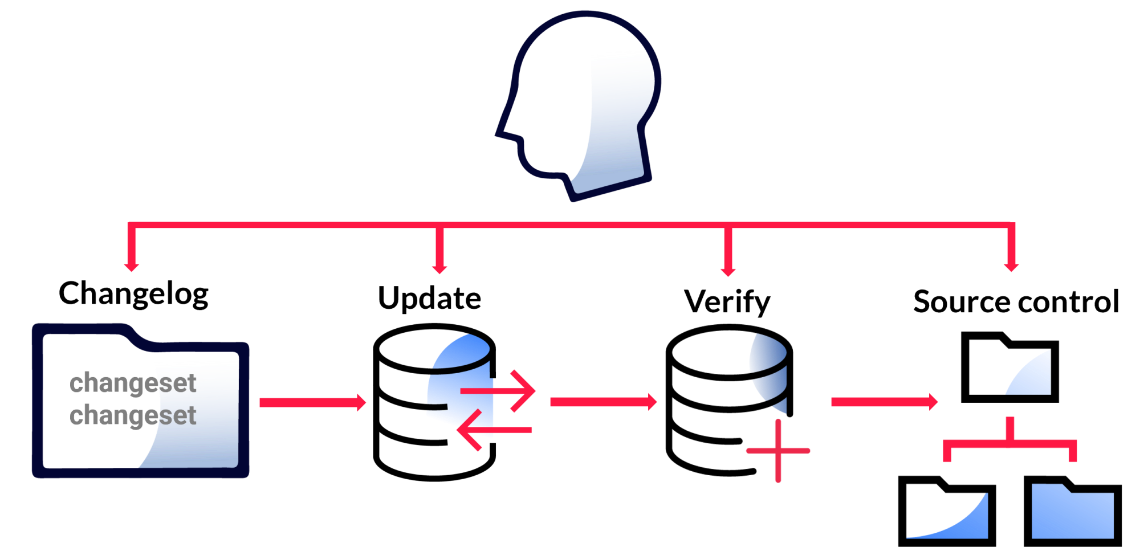
 These commands **do not** modify the database. The minimum required parameters to run these commands are the url, username, and password. The **generate-changelog** command also requires the path to the changelog file.

**liquibase snapshot**

This command generates a snapshot of the current state of the database and stores it for later comparison. It is often used in situations where you need to capture a baseline for future comparisons.

**Example:**

liquibase snapshot --url=jdbc:postgresql://localhost:5432/mydb --username=postgres --password=my\_password --changelog-file=master-changelog.xml --snapshot-name=my\_snapshot

**What Does Standard Liquibase Developer Do?**

**Changlog:**

**The four steps include:**

* **Adding** a changeset(s) to the changelog.
* **Running** the database update command.
* **Verifying** the results of the database update.
* **Saving** the changelog to source control.

**Workflow Review**

**Below is a recap of adding a changeset to a changelog:**

1. Using a text editor, open the changelog.

2. Under the changelog header, add the changeset tag with the author name and id attributes.

3. Define the change in the changeset such as a drop column or create table.

4. Add the changeset information such as the column information and table names with the appropriate changeset closing tags.

5. (optional) Add another changeset.

**Update:**

**Workflow Review**

**Below is an example workflow:**

1. In the CLI run the update-sql command:

**liquibase --changelog-file=yourchangelog.xml update-sql**

**Replace <yourchangelog.xml> with the name of the changelog to be used.**

2. **Inspect** the update-sql command output and correct identified issues in **yourchangelog.xml**.

3. In the CLI run the update command.

**liquibase --changelog-file=yourchangelog.xml update**

**Replace <yourchangelog.xml> with the name of the changelog to be used.**

4. **Review** the update command output to ensure the database changes were executed successfully.

**Output Example**

liquibase update  
Starting Liquibase at 14:59:27 (version 4.xx #3391 built at 20yy-07-06 12:07+0000)  
Liquibase Version: x.xx  
Liquibase Community x.xxby Liquibase  
Running Changeset: commented.postgres.sql::1645123377463-1::voxland  
Running Changeset: commented.postgres.sql::1645123377463-2::voxland  
**Liquibase command 'update' was executed successfully.**

**The Liquibase version number (x.xx) and year (yy) are replaced with the user's environment information.**

**Verify:**

This step in the developer workflow **verifies** that changes were properly executed on the database after running the Liquibase **update command**. Database update verification can be accomplished using **several different methods** including using specific Liquibase commands and/or direct inspection of the database.

**Using Liquibase Commands**

Run the **history** command to see a list of **deployed** changesets:

**liquibase history**

Run the **status --verbose**command to see a list of **undeployed** changesets:

**liquibase status --verbose**

Run the **diff** command to verify database changes were **applied** to the database, or to see if there are **missing or unexpected changes** detected between two databases:

**liquibase diff <primary database connection properties> <reference database or snapshot connection properties>**

**For more detailed information on the above commands, refer to the Liquibase Core Commands section of this course.**

**Direct Inspection**

Use your native database management tool or a database-independent tool like DBeaver to inspect the database.

**Source Control:**

Liquibase is designed to work with source code version control systems similar to Git or SVN.  When changes are complete and approved, you can **commit** your changes to the main branch of your repository. The changes are now ready for deployment.

A benefit of storing application code and database changes in **one location** is that users will know which version of the code changes go with which version of the database changes.

**Workflow Review**

**Below is an example workflow using Git as the source control tool:**

**1.** **Get** the latest version of your changelog(s) from your version control system.

**2.** **Modify** the changelog as required for the task you are completing.

**3.** **Test** your changes locally.

**4**. **Obtain** approval for the changelog modifications using your internal process.

**5.** **Commit** the changes. If the changes were developed in a branch take the appropriate steps to merge them into the main branch of your source code repository.

**To learn more about using Liquibase with source control, sign up for the [Liquibase and the CI/CD Process](https://learn.liquibase.com/catalog/info/id:128?__hstc=164782368.20d0072ff77e78011a42ae0d101d92f6.1657542707344.1659462646245.1659464789274.74&__hssc=164782368.109.1659464789274&__hsfp=2378282809" \t "_blank) course when you have completed this course.**

**Organize Changelogs by Release**

The root changelog file operates as the **primary** changelog that references other changelogs. These other changelogs are referred to as **nested** changelogs.

**The key is to organize changelogs into smaller files that match the software development process used in your organization.**

One way to organize changelogs is by **release.** This directory structure makes it easy to see which changes were made for each release. One advantage to organizing changelogs by release is that the changelogs **do not grow** over time since they only contain changes for a **single** **release**.

Another advantage to this strategy is that changesets can be re-ordered as needed within each changelog.

**In the below example, we are using a root changelog to organize multiple changelogs by release:**

com/  
 example/  
 db/  
 changelog/  
 **db-changelog-root.xml**  
 releases/  
 db.changelog-01.00.xml  
 db.changelog-01.01.xml  
 db.changelog-02.00.xml  
 src/  
 DatabasePool.java  
 AbstractDAO.java

In the above example, the **db-changelog-root.xml** changelog file will **automatically** include the changelogs for each release in alpha-numeric order. When using this format, it is recommended to **avoid** using single-digit release numbering and add one or more zero's before the version number. This will **maintain** the expected order of the changelogs.

Nested changelogs are referenced by the root changelog using **two specialized tags** and can be used with SQL, XML, JSON, and YAML changelog formats:

* The **include tag** - references a **single changelog** file. All changesets in the referenced nested changelog will be executed in the order they appear in the file.
* The **includeAll** **tag** - references a **directory** where one or more changelog files can be found. Changelogs are processed in alphabetical order according to the filename.

**Include:**

<?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"  
 xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"  
 xmlns:pro="http://www.liquibase.org/xml/ns/pro"  
 xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  
 http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd  
 http://www.liquibase.org/xml/ns/dbchangelog-ext http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd  
 http://www.liquibase.org/xml/ns/pro http://www.liquibase.org/xml/ns/pro/liquibase-pro-latest.xsd">  
 **<include file="com/example/db/changelog/releases/db.changelog-01.00.sql"/>  
 <include file="com/example/db/changelog/releases/db.changelog-01.01.sql"/>  
 <include file="com/example/db/changelog/releases/db.changelog-02.00.xml"/>**  
</databaseChangeLog>

**IncludeAll:**

<?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"  
 xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"  
 xmlns:pro="http://www.liquibase.org/xml/ns/pro"  
 xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog  
 http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd  
 http://www.liquibase.org/xml/ns/dbchangelog-ext http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd  
 http://www.liquibase.org/xml/ns/pro http://www.liquibase.org/xml/ns/pro/liquibase-pro-latest.xsd">  
 **<includeAll file="com/example/db/changelog/releases"/>**  
</databaseChangeLog>

**Rollback Strategy Tips**

It is important to state that a **fix forward** approach is the recommended best practice but there may be instances when rolling back a change is the logical option.

**When creating a rollback strategy, consider the following:**

* Fix forward by introducing a **new changeset** that addresses the issue safely.
* Consider writing changesets when possible using Liquibase **Change Types** that auto-generate rollback statements. However, be aware that certain Change Types will undo the change but not preserve the data.
* Use the Liquibase **rollback <tag>**  in the changeset when you want to override the default rollback approach. The tag can also be used to provide instructions for Change Types that do not have an associated automatic rollback.
* **Validate** your rollback scripts in a development environment early in the process.

Certain Change Types support **auto-generated** rollback statements. Below are a few examples:

* addColumn
* createIndex
* createSequence
* createView

**Validating a Rollback**

To ensure safe and predictable rollbacks, test the rollback in a development environment. A complete test cycle should include:

Deploy the changes to the database then validate that they were deployed.

Roll back the changes, validate the changes were undone, and the database was brought back to the previous state.

Redeploy all changes to the database.

The final step of redeploying the changes to the database is to verify that the rollback did not miss changes that would impact a future deployment.

**Database Security**

* Use a **secrets management tool** to manage credentials.
* Use automation tools to **pass credentials** to Liquibase.
* **Limit** direct database access.

It is **strongly recommended to not store credentials in the Liquibase properties file**. Instead, use a secrets management tool or other types of secure locations to manage credentials.

This eliminates the need for individuals to have direct access to the database and provides traceable history including the event and the contents of the change.

**Liquibase works with secrets management tools such as:**

* HashiCorp Vault
* AWS Secrets Manager
* CyberArk
* Azure Key Vault

**Using a CI/CD Tool to Store Credentials**

Another option is to use an **automation tool** to store credentials. As an example, Jenkins can store certain types of credentials used for pipeline projects:

* Credentials are stored in an **encrypted form** on the Jenkins instance and handled in the pipeline project with credential IDs.
* This **minimizes exposure** of the actual credentials to Jenkins users and hinders the ability to copy functional credentials from one Jenkins instance to another.