**Liquibase Overview**

**Official Resources:**

* GitHub Actions: <https://github.com/liquibase-github-actions>
* Community Forum: <https://forum.liquibase.org>
* YouTube Channel: [Liquibase on YouTube](https://www.youtube.com/channel/UC5qMsRjObu685rTBq0PJX8w)

**What is Liquibase?**

Liquibase is a version control tool for databases. It is used to **track**, **log**, and **manage** schema changes in a database. Changes are executed through command-line operations or integrated into CI/CD pipelines.

**💾 Supported Databases**

Liquibase supports a wide range of relational databases, including:

* Oracle
* PostgreSQL
* SQL Server / MSSQL
* Oracle ATP
* H2
* IBM DB2 on Z
* MariaDB
* MySQL
* SQLite
* Informix
* Apache Derby

**📄 Supported Changelog Languages**

Database schema changes can be written in the following formats:

* **SQL**
* **XML**
* **YAML** *(note: indentation is critical in YAML)*
* **JSON**

**🧱 Table Creation Example in Each Format**

**XML** (most common format with the most examples in Liquibase documentation):

<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:**

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:**

--liquibase formatted sql

--changeset sena:1

CREATE TABLE users (

id BIGINT PRIMARY KEY NOT NULL,

username VARCHAR(50) NOT NULL

);

**Changelogs, Changesets, and Change Types in Liquibase**

Liquibase allows you to define and manage database changes using:

* **Changelogs** (the main file that organizes changesets)
* **Changesets** (individual units of change)
* **Change types** (specific operations like createTable, addColumn, etc.)

**Rollback Support**

Liquibase supports **automatic rollback** for many types of changes, allowing you to safely undo changes if needed.

**Version Control Benefits**

Liquibase integrates seamlessly with version control systems like **Git**, **SVN**, etc. When database changes are added to version control:

* You gain the same benefits as with application code:
  + View change history
  + Track who changed what and when
  + Revert changes if necessary
* Changesets stored in version control allow you to:
  + Understand the current state of the database
  + See which changes have been applied across environments
* Multiple developers can work on different changesets **simultaneously**.
* When merging code, **conflicts are minimized** because each changeset has a unique combination of id and author.

There is no need to adopt a new version control system — Liquibase works perfectly with what you're already using (e.g., Git, SVN).

In short:  
By storing Liquibase changes in a version control system like Git, you can manage your database changes **just like your source code** — with full history, team collaboration, and rollback capability.

**What Does a Changeset Include?**

A typical Liquibase changelog includes:

* A **changelog header**
* One or more **changesets**
* **Changeset attributes** such as id, author, context, and rollback options

**Liquibase Community Changelog File Setup**

When using Liquibase in a community edition project, your XML changelog file typically starts with the following structure:

<?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>

**Changset Examples(XML):**

**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>

**addColumn:**

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

<addColumn tableName="product">

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

</addColumn>

</changeSet>

**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>

**Update:**

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

<update tableName="product">

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

<where>id = 1</where>

</update>

</changeSet>

**dropTable:**

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

<dropTable tableName="product"/>

</changeSet>

**Applicaiton.Properties:**

# Enabling Liquibase:

spring.liquibase.enabled=

# Path to the changelog file:

spring.liquibase.change-log=

# Log level for Liquibase commands:

spring.liquibase.log-level=

# Context used to apply changes only in a specific environment:

spring.liquibase.contexts=

# Database reset (drop) operation:

spring.liquibase.drop-first=

# Character set for SQL files:

spring.liquibase.sql-script-encoding=

# Changelog file to be used for rollback:

spring.liquibase.rollback-file=

# Table name for tracking database changes:

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

# Table name for tracking the lock status of the database changes:

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

# Liquibase command to be executed (e.g., update, rollback):

spring.liquibase.command=

# Liquibase command to reset the `databaseChangeLog` and `databaseChangeLogLock` tables:

spring.liquibase.clearCheckSums=

# JDBC URL for the database (usually used with Spring DataSource):

spring.liquibase.url=

# Database username:

spring.liquibase.username=

# Database password:

spring.liquibase.password=

# JDBC driver used for database connection:

spring.liquibase.driver=

# Version control for the Liquibase change file:

spring.liquibase.change-log-file=

# Ensures Liquibase checks for changes on every application startup:

spring.liquibase.check-changeset=

# Logs both applied and rollback SQL commands for Liquibase:

spring.liquibase.log-sql=

# Date for the rollback version if rollback is required:

spring.liquibase.rollback-version=

# Number of parallel threads used to execute Liquibase operations:

spring.liquibase.parallel=

# Only update schemas that are active during Liquibase operations:

spring.liquibase.schemas=

Liquibase **uses the --kebab-case format** with lowercase letters and a dash (-) in the CLI environment:  
• Log level is --log-level.  
• The output file is --output-file.

**Example:**liquibase [global argument] [command] [command attribute]  
liquibase --changelog-file=dbchangelog.xml [command] [command attribute]

• Using the CLI allows you to apply database changes independently without running the Spring Boot application. However, Liquibase integration with Spring Boot is much easier, as you can configure it through the application.properties file, enabling Liquibase operations to be executed automatically when the application starts.  
• In Spring Boot, you can manage Liquibase's operations using settings in the application.properties or application.yml file, eliminating the need to manually write CLI commands, making it automatic.

Liquibase CLI (Command Line Interface) is a standalone tool, typically used in the following scenarios:

* **Running Liquibase commands independently**: The CLI allows you to execute Liquibase commands directly from the terminal or command line.
* **Running in an isolated environment**: If you're running Liquibase outside of Spring Boot (for instance, solely for database management purposes), you can use the Liquibase CLI.
* **Using in CI/CD pipelines**: Liquibase can be used independently in a CI/CD pipeline (such as Jenkins, GitLab CI, CircleCI, etc.). In this case, CLI commands can be used to apply database changes before running the application.

**Things to keep in mind when writing the changeset id and author:**

* Both the author and id are required because multiple people 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 applied. 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 need to remember the id values used in other files. The combination of id and author only needs to be unique within the current file.
* It is important to avoid duplicating the same author and id combination within the same file.

**Example of correct usage:**

**<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  
....

**View Changeset Comments**

You can add comments within your changeset files in Liquibase to document the purpose and details of changes. These comments can be viewed in two ways:

* **Via the db-doc command**:  
  To generate a documentation file of the changesets and their associated comments:

liquibase db-doc

This will generate a documentation file (HTML by default) that includes all changesets, along with their comments.

* **Directly within the changelog**:  
  You can view comments directly within the changelog XML file. For example:

<changeSet id="1" author="john\_doe">

<!-- This is a comment explaining the change -->

<createTable tableName="example\_table">

<column name="id" type="int"/>

</createTable>

</changeSet>

* + Comments in Liquibase changelog files are added using XML-style comments (<!-- Comment text -->), and they do not affect the execution of the changeset.

**2. Rollback Changesets**

You can rollback changesets using Liquibase CLI, which allows you to undo applied changes.

* **Rollback to a specific tag**:  
  This command reverts all changes made up to a specific tag:

liquibase rollback <tag>

* **Rollback a specific changeset by ID**:  
  Rollback a specific changeset using its ID and author:

liquibase rollbackCount <number of changesets>

* **Rollback to the previous point**:  
  This command rolls back the last changeset applied:

liquibase rollbackToDate <timestamp>

**3. Applying Changesets**

Changesets are executed via the **update** command to apply the changes defined in the changelog.

* **Run changeset**:  
  To apply the changes in the changelog file:

liquibase update

* **Update a specific changeset**:  
  To apply changesets up to a specific changeset ID:

liquibase update --changeset=<id> --author=<author>

**4. Diff Commands (Compare Databases)**

Liquibase offers commands for comparing two databases or generating changelogs based on differences.

* **Compare two databases**:  
  This command compares the target database with a reference database and identifies the differences:

liquibase diff

* **Generate changelog from diff**:  
  Create a changelog file based on the differences between two databases:

liquibase diff-changelog

**5. Database Documentation Command**

Generate a documentation of your Liquibase changesets in HTML format.

* **Generate changelog documentation**:  
  This command creates an HTML file that details the changesets in your changelog:

liquibase db-doc

**6. List All Applied Changesets**

You can view the list of applied changesets in your database.

* **Show all applied changesets**:  
  Display the changesets that have been applied to the database:

liquibase history

**7. Testing Changesets**

Liquibase also offers commands to test changesets before running them.

* **Test a changeset**:  
  Validate changesets without actually applying them to the database:

liquibase validate

**8. Rollback SQL Generation**

You can generate SQL for the rollback of applied changes.

* **Generate rollback SQL**:  
  Liquibase can generate the SQL needed to rollback a changeset:

liquibase rollbackSQL <tag>

**9. View or Change Log Level**

You can set the log level to view detailed information during command execution.

* **Set log level**:  
  Use this command to adjust the verbosity of the log output:

liquibase --log-level=debug update

**10. View Liquibase Status**

Get the status of Liquibase and its changesets in the database.

* **Get database status**:  
  Show the current status of the database with respect to the applied changesets:

liquibase status

**11. Changelog File Path and Additional Parameters**

When running any of these commands, make sure to specify the correct path to your changelog file (if it’s not named dbchangelog.xml).

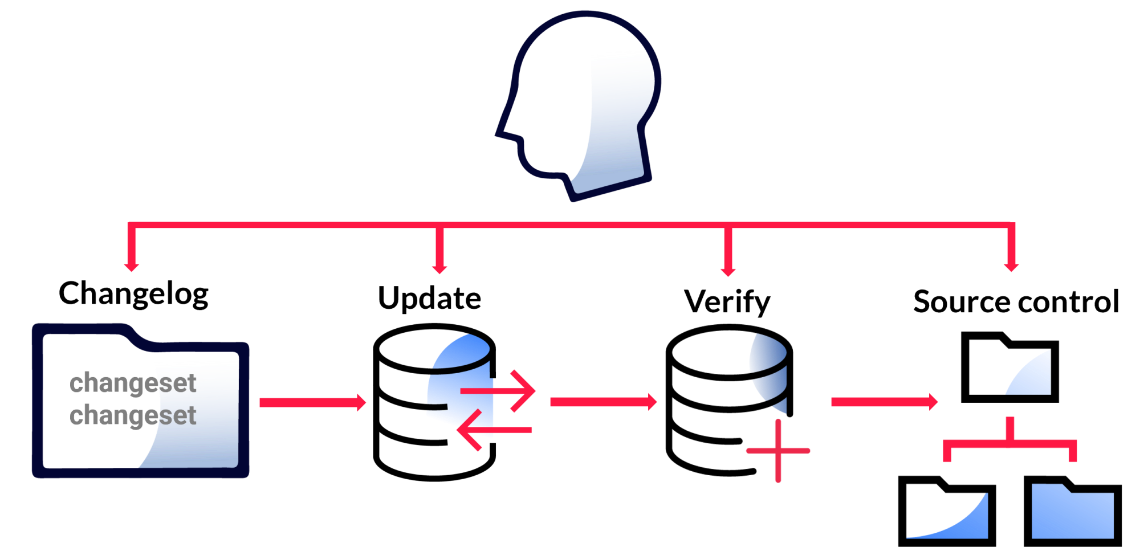
* **Example with a changelog file path**:

liquibase --changelog-file=yourchangelog.xml update

* **Additional options for username and password**:

liquibase --changelog-file=yourchangelog.xml --username=username --password=password update

These commands represent key actions you can perform using **Liquibase CLI** to manage changesets, generate SQL for rollback, compare databases, and more. Each command allows for effective database version control, whether you are applying, testing, or rolling back changes.

**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.

Data created by users when running the application is **outside the scope** of Liquibase and is better managed by other tools. This includes:

* **ETL**-related activities.
* **Large** data loads.
* **Customer** data.

**TÜRKÇE:**

**Liquibase Genel Bakış**  
Resmi Kaynaklar:  
• GitHub Actions: <https://github.com/liquibase-github-actions>  
• Topluluk Forumu: <https://forum.liquibase.org>  
• YouTube Kanalı: Liquibase YouTube'da

**Liquibase Nedir?**  
Liquibase, veritabanları için bir sürüm kontrol aracıdır. Veritabanındaki şema değişikliklerini izlemek, kaydetmek ve yönetmek için kullanılır. Değişiklikler, komut satırı işlemleri aracılığıyla ya da CI/CD hatlarına entegre edilerek gerçekleştirilir.

**💾 Desteklenen Veritabanları**  
Liquibase, aşağıdaki veritabanlarını geniş bir yelpazede destekler:  
• Oracle  
• PostgreSQL  
• SQL Server / MSSQL  
• Oracle ATP  
• H2  
• IBM DB2 on Z  
• MariaDB  
• MySQL  
• SQLite  
• Informix  
• Apache Derby

**📄 Desteklenen Changelog Dilleri**  
Veritabanı şeması değişiklikleri şu formatlarda yazılabilir:  
• SQL  
• XML  
• YAML (Not: YAML'de girinti çok önemlidir)  
• JSON

**🧱 Her Format İçin Tablo Oluşturma Örneği**  
XML (Liquibase dokümantasyonunda en yaygın format ve en fazla örnek bu formatta bulunmaktadır):

<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:**

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:**

--liquibase formatted sql

--changeset sena:1

CREATE TABLE users (

id BIGINT PRIMARY KEY NOT NULL,

username VARCHAR(50) NOT NULL

);

**Liquibase'de Changelog'lar, Changeset'ler ve Değişim Türleri**  
Liquibase, veritabanı değişikliklerini tanımlamanıza ve yönetmenize olanak sağlar:  
• **Changelog'lar** (değişiklikleri düzenleyen ana dosya)  
• **Changeset'ler** (bireysel değişim birimleri)  
• **Değişim türleri** (createTable, addColumn gibi spesifik işlemler)

**Rollback Desteği**  
Liquibase, birçok değişim türü için otomatik rollback (geri alma) desteği sunar, bu da değişiklikleri güvenli bir şekilde geri almanıza olanak tanır.

**Sürüm Kontrolü Avantajları**  
Liquibase, Git, SVN gibi sürüm kontrol sistemleriyle sorunsuz bir şekilde entegre olur. Veritabanı değişiklikleri sürüm kontrolüne eklendiğinde:  
• **Uygulama kodu ile aynı avantajları elde edersiniz:**

* Değişim geçmişini görüntüleme
* Kimin, neyi ve ne zaman değiştirdiğini izleme
* Gerekirse değişiklikleri geri alma  
  • **Sürüm kontrolünde saklanan changeset'ler size şu avantajları sağlar:**
* Veritabanının mevcut durumunu anlama
* Hangi değişikliklerin hangi ortamlarda uygulandığını görme  
  • Birden fazla geliştirici aynı anda farklı changeset'ler üzerinde çalışabilir.  
  • Kod birleştirildiğinde, her changeset'in benzersiz bir id ve yazar kombinasyonuna sahip olduğu için çatışmalar minimize edilir.  
  Yeni bir sürüm kontrol sistemi benimsemenize gerek yoktur — Liquibase, zaten kullandığınız (örneğin Git, SVN) sistemlerle mükemmel çalışır.

**Kısacası:**  
Liquibase değişikliklerini Git gibi bir sürüm kontrol sisteminde saklayarak, veritabanı değişikliklerinizi kaynak kodunuz gibi yönetebilirsiniz — tam geçmiş, takım çalışması ve rollback (geri alma) yetenekleriyle.

**Bir Changeset Neler İçerir?**  
Tipik bir Liquibase changelog'u şunları içerir:  
• Bir changelog başlığı  
• Bir veya daha fazla changeset  
• Changeset özellikleri (id, author, context ve rollback seçenekleri gibi)

**Liquibase Community Changelog Dosyası Yapılandırması**  
Liquibase'i topluluk sürümünde bir projede kullanırken, XML changelog dosyanız genellikle aşağıdaki yapıyla başlar:

<?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:**

**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>

**addColumn:**

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

<addColumn tableName="product">

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

</addColumn>

</changeSet>

**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>

**Update:**

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

<update tableName="product">

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

<where>id = 1</where>

</update>

</changeSet>

**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=

Liquibase, CLI ortamında küçük harfler ve tire (-) ile **kebab-case** formatını kullanır: • Log seviyesi --log-level olarak tanımlanır. • Çıktı dosyası --output-file olarak belirtilir. Örnek:

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

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

* CLI kullanmak, Spring Boot uygulamasını çalıştırmadan bağımsız bir şekilde veritabanı değişikliklerini uygulamanıza olanak sağlar. Ancak, Spring Boot ile Liquibase entegrasyonu çok daha kolaydır, çünkü application.properties dosyasını kullanarak yapılandırma yapılabilir ve uygulama başladığında Liquibase işlemleri otomatik olarak çalıştırılır.
* Spring Boot'ta, application.properties veya application.yml dosyasındaki ayarlarla Liquibase'in işlemleri yönetilebilir, bu da CLI komutlarını manuel olarak yazma gerekliliğini ortadan kaldırır ve işlemleri otomatik hale getirir.
* Liquibase CLI (Komut Satırı Arayüzü), bağımsız bir araçtır ve genellikle şu senaryolarda kullanılır:
* Liquibase komutlarını bağımsız olarak çalıştırmak: CLI, Liquibase komutlarını doğrudan terminal veya komut satırından çalıştırmanıza olanak tanır.
* İzole bir ortamda çalıştırmak: Eğer Liquibase, Spring Boot dışında (örneğin sadece veritabanı yönetimi amacıyla) çalıştırılıyorsa, Liquibase CLI kullanılabilir.
* CI/CD pipeline'larında kullanmak: Liquibase, CI/CD pipeline'ında (örneğin Jenkins, GitLab CI, CircleCI vb.) bağımsız olarak kullanılabilir. Bu durumda, uygulama çalıştırılmadan önce veritabanı değişikliklerini uygulamak için CLI komutları kullanılabilir.

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

* Hem author hem de id gereklidir çünkü aynı id değerini birden fazla kişi kullanabilir.
* Yazarın adı veya yazar için başka bir benzersiz değer kullanılabilir.
* Id etiketi, değişikliklerin hangi sırayla uygulanacağını kontrol etmez. Bu, açıklayıcı bir metin veya changelog içinde benzersiz olan bir sayı olabilir.
* Dosya adı da bir kimlik parçası olduğu için kullanıcı, diğer dosyalarda kullanılan id değerlerini hatırlamak zorunda kalmaz. Id ve author kombinasyonunun yalnızca mevcut dosyada benzersiz olması gerekir.
* Aynı author ve id kombinasyonunun aynı dosya içinde tekrarlanmamasına dikkat edilmelidir.

**Doğru kullanım örneği:**

**<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 ve Dosya İsimlendirme Standartları**

Liquibase ile kullanılan branch'ler oluştururken, uygulama kodunuzla aynı isimlendirme standartlarını kullanmak ve bu standartlara tutarlı bir şekilde bağlı kalmak en iyi uygulamadır. Branch'ler için yaygın isimlendirme standartları şunları içerebilir: • Özellik adı (Feature name). • Özellik veya hata için atanan bilet numarası (Ticket number).

Bir diğer en iyi uygulama, dosya isimlerini standartlaştırmaktır. Güncellenen nesneyle ilgili belirli dosya isimleri kullanmak, benzersizlik ve okunabilirlik sağlamak için yardımcı olabilir.

**Liquibase CLI Yorumları Kullanma**

<https://learn.liquibase.com/unit/view/id:3462>

* update-sql komutu.
* update komutu.
* history komutu.

**Yorum eklemek için kullanılan sözdizimi:**

* 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

</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  
....

**Changelog İçinde Yorum Görüntüleme:**

Yorumlar doğrudan changelog XML dosyasında görülebilir. Örnek:

<changeSet id="1" author="john\_doe">

<!-- Bu, değişikliği açıklayan bir yorumdur -->

<createTable tableName="example\_table">

<column name="id" type="int"/>

</createTable>

</changeSet>

* Liquibase changelog dosyalarına XML tarzı yorumlar (<!-- Yorum metni -->) eklenir ve bu yorumlar changeset'in uygulanmasını etkilemez.

**2. Rollback Changesets (Değişiklikleri Geri Almak):**

Liquibase CLI kullanarak changeset'leri geri alabilirsiniz, bu da uygulanan değişiklikleri geri almanıza olanak tanır.

* **Belirli bir tag'e geri dönmek:** Bu komut, belirli bir tag'e kadar yapılan tüm değişiklikleri geri alır:

liquibase rollback <tag>

* **Belirli bir changeset'i ID ile geri almak:** Belirli bir changeset'i ID ve yazar bilgisiyle geri almak:

liquibase rollbackCount <changeset sayısı>

* **Önceki noktaya geri dönmek:** Bu komut, son uygulanan changeset'i geri alır:

liquibase rollbackToDate <timestamp>

**3. Changeset Uygulama:**

Changeset'ler, changelog dosyasındaki değişiklikleri uygulamak için update komutu ile çalıştırılır.

* **Changeset'i çalıştırmak:** Changelog dosyasındaki değişiklikleri uygulamak için:

liquibase update

* **Belirli bir changeset'i güncellemek:** Belirli bir changeset ID'sine kadar changeset'leri uygulamak için:

liquibase update --changeset=<id> --author=<author>

**4. Diff Komutları (Veritabanlarını Karşılaştırmak):**

Liquibase, iki veritabanını karşılaştırmak veya farklılıklara dayalı olarak changelog dosyaları oluşturmak için komutlar sunar.

* **İki veritabanını karşılaştırmak:** Bu komut, hedef veritabanını bir referans veritabanı ile karşılaştırır ve farkları belirler:

liquibase diff

* **Diff ile changelog oluşturmak:** İki veritabanı arasındaki farklara dayalı olarak bir changelog dosyası oluşturur:

liquibase diff-changelog

**5. Veritabanı Belgeleri Komutu:**

Liquibase değişiklik setlerinizin HTML formatında belgesini oluşturabilirsiniz.

* **Changelog belgesi oluşturmak:** Bu komut, changelog dosyanızdaki changeset'leri detaylandıran bir HTML dosyası oluşturur:

liquibase db-doc

**6. Tüm Uygulanan Changeset'leri Listeleme:**

Veritabanınızdaki uygulanan changeset'lerin listesini görebilirsiniz.

* **Tüm uygulanan changeset'leri göster:** Veritabanınıza uygulanan changeset'leri görüntülemek için:

liquibase history

**7. Changeset'leri Test Etme:**

Liquibase, changeset'leri çalıştırmadan önce test etmenize olanak tanır.

* **Bir changeset'i test etmek:** Changeset'leri, veritabanına uygulamadan doğrulamak için:

liquibase validate

**8. Rollback SQL Üretimi:**

Uygulanan değişikliklerin geri alınması için SQL üretilebilir.

* **Rollback SQL üretmek:** Liquibase, bir changeset'in geri alınması için gereken SQL'i üretebilir:

liquibase rollbackSQL <tag>

**9. Log Seviyesini Görüntüleme veya Değiştirme:**

Komut çalıştırırken ayrıntılı bilgi görmek için log seviyesini ayarlayabilirsiniz.

* **Log seviyesini ayarlamak:** Log çıktısının ayrıntı seviyesini ayarlamak için:

liquibase --log-level=debug update

**10. Liquibase Durumunu Görüntüleme:**

Liquibase ve veritabanındaki changeset'lerin durumunu alabilirsiniz.

* **Veritabanı durumunu almak:** Veritabanınızın mevcut durumunu, uygulanan changeset'lere göre göstermek için:

liquibase status

**11. Changelog Dosya Yolu ve Ek Parametreler:**

Bu komutları çalıştırırken, changelog dosyanızın doğru yolunu belirtmeyi unutmayın (eğer dosyanın adı dbchangelog.xml değilse).

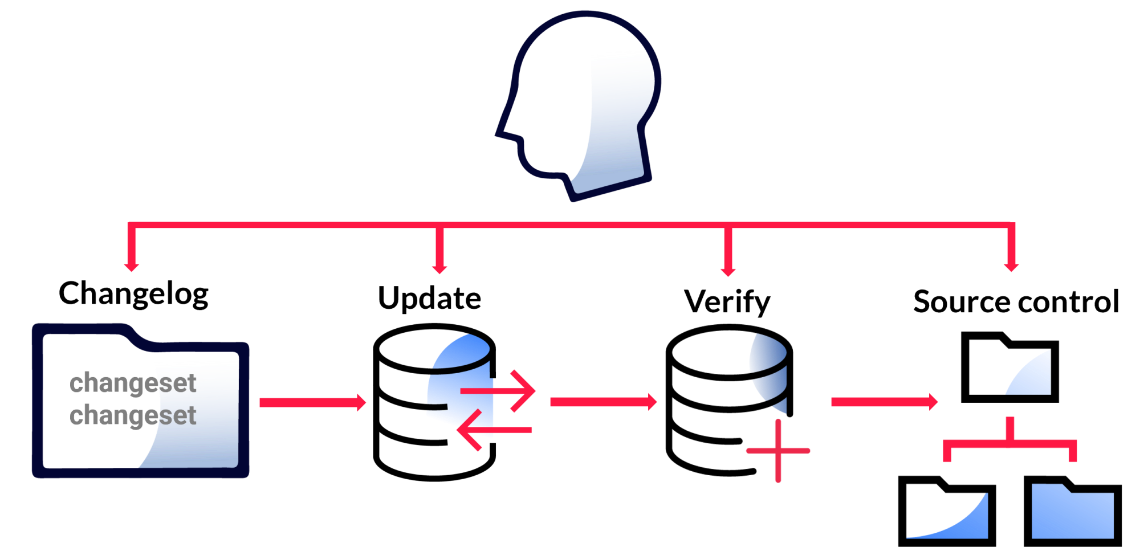
* **Changelog dosya yolu ile örnek:**

liquibase --changelog-file=yourchangelog.xml update

* **Kullanıcı adı ve şifre için ek seçenekler:**

liquibase --changelog-file=yourchangelog.xml --username=username --password=password update

Bu komutlar, Liquibase CLI kullanarak changeset'leri yönetmek, rollback için SQL üretmek, veritabanlarını karşılaştırmak ve daha fazlasını yapmak için kullanılabilecek temel işlemleri temsil etmektedir. Her komut, veritabanı sürüm kontrolünü etkili bir şekilde yönetmenizi sağlar, ister değişiklikleri uyguluyor, test ediyor ya da geri alıyor olun.

**Liquibase Developer Ne Yapar**

**Changelog:**

Dört adım şunları içerir:

* Changelog'a bir veya daha fazla changeset eklemek.
* Veritabanı güncelleme komutunu çalıştırmak.
* Veritabanı güncellemesinin sonuçlarını doğrulamak.
* Changelog'u kaynak kontrol sistemine kaydetmek.

**İş Akışı İncelemesi**

Aşağıda, changelog'a changeset ekleme sürecinin bir özeti bulunmaktadır:

1. Bir metin düzenleyici kullanarak changelog dosyasını açın.
2. Changelog başlığının altına, yazar adı ve id öznitelikleriyle changeset etiketi ekleyin.
3. Changeset içinde yapılacak değişikliği tanımlayın (örneğin, bir sütun silme veya tablo oluşturma).
4. Changeset bilgilerini (sütun bilgileri ve tablo adları gibi) uygun changeset kapanış etiketleriyle ekleyin.
5. (isteğe bağlı) Başka bir changeset ekleyin.

**Güncelleme:**

İş Akışı İncelemesi Aşağıda örnek bir iş akışı bulunmaktadır:

1. CLI'da update-sql komutunu çalıştırın:

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

<yourchangelog.xml> kısmını kullanılacak changelog dosyasının adıyla değiştirin.

1. update-sql komutunun çıktısını inceleyin ve yourchangelog.xml dosyasındaki tespit edilen sorunları düzeltin.
2. CLI'da update komutunu çalıştırın:

liquibase --changelog-file=yourchangelog.xml update

<yourchangelog.xml> kısmını kullanılacak changelog dosyasının adıyla değiştirin.

1. Güncelleme komutunun çıktısını inceleyin ve veritabanı değişikliklerinin başarıyla uygulandığından emin olun.

**Çıktı Örneği:**

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.xx by Liquibase

Running Changeset: commented.postgres.sql::1645123377463-1::voxland

Running Changeset: commented.postgres.sql::1645123377463-2::voxland

Liquibase command 'update' was executed successfully.

Liquibase sürüm numarası (x.xx) ve yıl (yy) kullanıcının ortam bilgileriyle değiştirilmiştir.

**Doğrulama:**

Bu adım, Liquibase güncelleme komutu çalıştırıldıktan sonra veritabanındaki değişikliklerin doğru şekilde uygulandığını doğrular. Veritabanı güncellemesinin doğrulanması, çeşitli Liquibase komutları kullanılarak ve/veya veritabanasının doğrudan incelenmesiyle yapılabilir.

**Liquibase Komutları Kullanarak:**

* Uygulanan changeset'lerin listesini görmek için history komutunu çalıştırın:

liquibase history

* Uygulanmamış changeset'lerin listesini görmek için status --verbose komutunu çalıştırın:

liquibase status --verbose

* Veritabanı değişikliklerinin veritabanına uygulanıp uygulanmadığını doğrulamak veya iki veritabanı arasındaki eksik veya beklenmedik değişiklikleri görmek için diff komutunu çalıştırın:

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

Daha ayrıntılı bilgi için, yukarıdaki komutlarla ilgili Liquibase Core Komutları bölümüne bakınız.

**Doğrudan Denetim:**

Veritabanını incelemek için yerel veritabanı yönetim aracınızı veya DBeaver gibi veritabanından bağımsız bir araç kullanabilirsiniz.

**Kaynak Kontrolü:**

Liquibase, Git veya SVN gibi kaynak kodu sürüm kontrol sistemleriyle çalışacak şekilde tasarlanmıştır. Değişiklikler tamamlandığında ve onaylandığında, değişikliklerinizi depolamanızın ana dalına (main branch) gönderebilirsiniz. Bu değişiklikler artık dağıtıma hazırdır.

Uygulama kodu ve veritabanı değişikliklerini bir arada saklamanın avantajı, kullanıcıların hangi sürümdeki kod değişikliklerinin hangi sürümdeki veritabanı değişiklikleriyle birlikte olduğunu bilmeleridir.

**İş Akışı İncelemesi:**

Aşağıda Git kaynak kontrol aracı kullanılarak yapılacak örnek bir iş akışı bulunmaktadır:

1. Kaynak kontrol sisteminizden changelog dosyanızın en son sürümünü alın.
2. Yaptığınız göreve göre changelog dosyasını gerektiği şekilde değiştirin.
3. Değişikliklerinizi yerel olarak test edin.
4. Changelog değişikliklerinin onayını iç sürecinizle alın.
5. Değişiklikleri commitleyin. Eğer değişiklikler bir dalda (branch) geliştirilmişse, bunları ana dalınıza (main branch) birleştirmek için gerekli adımları atın.

**Çıkış Yapma:**

Liquibase ve kaynak kontrolü hakkında daha fazla bilgi edinmek için, bu kursu tamamladıktan sonra Liquibase ve CI/CD Süreci kursuna kaydolabilirsiniz.

**Çıktıları Yayınlara Göre Düzenlemek**

Kök changelog dosyası, diğer changelog'ları referans alan birincil changelog olarak çalışır. Bu diğer changelog'lar, iç içe geçmiş changelog'lar olarak adlandırılır. Anahtar, changelog dosyalarını, organizasyonunuzda kullanılan yazılım geliştirme süreciyle uyumlu daha küçük dosyalar halinde düzenlemektir.

Changelog'ları sürüme göre düzenlemenin bir yolu, bu dizin yapısını kullanmaktır. Bu yapı, her sürüm için hangi değişikliklerin yapıldığını görmeyi kolaylaştırır. Changelog'ları sürüme göre düzenlemenin bir avantajı, zamanla büyümemeleridir çünkü yalnızca bir sürüm için değişiklik içerirler. Bu stratejinin bir diğer avantajı ise, her changelog içindeki changeset'lerin gerektiğinde sıralanabilmesidir. Aşağıdaki örnekte, sürüme göre organize edilmiş birden fazla changelog için kök changelog kullanıyoruz:

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

Yukarıdaki örnekte, **db-changelog-root.xml** changelog dosyası, her sürüm için olan changelog'ları alfasayısal sırayla otomatik olarak dahil edecektir. Bu formatı kullanırken, tek haneli sürüm numaralarından kaçınılması ve sürüm numarasından önce bir veya daha fazla sıfır eklenmesi önerilir. Bu, changelog'ların beklenen sırayla sıralanmasını sağlar.

İç içe geçmiş changelog'lar, kök changelog tarafından iki özel etiket kullanılarak referans gösterilir ve SQL, XML, JSON ve YAML changelog formatlarıyla kullanılabilir:

* **include etiketi**: Tek bir changelog dosyasını referans alır. Referans verilen iç içe geçmiş changelog'daki tüm changeset'ler, dosyada göründükleri sırayla çalıştırılacaktır.
* **includeAll etiketi**: Bir dizini referans alır ve bu dizinde bir veya daha fazla changelog dosyası bulunabilir. Changelog'lar, dosya adına göre alfabetik sırayla işlenir.

**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>

**Roll Back Stratejisi İpuçları**

İleriye doğru düzeltme yaklaşımının önerilen en iyi uygulama olduğunu belirtmek önemlidir, ancak bazen bir değişikliği geri almanın mantıklı bir seçenek olabileceği durumlar olabilir.  
Bir geri alma stratejisi oluştururken şu hususları göz önünde bulundurun:

* **İleriye doğru düzeltme** yaparak, sorunu güvenli bir şekilde ele alan yeni bir changeset ekleyin.
* **Geri alma ifadelerini otomatik olarak üreten Liquibase Change Types** kullanarak changeset'ler yazmayı düşünün. Ancak, bazı Change Types'ın değişikliği geri alacağını ancak veriyi korumayacağını unutmamalısınız.
* **Liquibase rollback <tag>** etiketini changeset içinde kullanarak, varsayılan geri alma yaklaşımını geçersiz kılmak isteyebilirsiniz. Bu etiket ayrıca, otomatik geri alma ile ilişkilendirilmeyen Change Types için talimatlar sağlamak amacıyla kullanılabilir.
* Geri alma betiklerinizi **geliştirme ortamında** süreç başında doğrulayın.

Bazı Change Types, otomatik olarak oluşturulan geri alma ifadelerini destekler. İşte bazı örnekler:

* **addColumn**
* **createIndex**
* **createSequence**
* **createView**

**Geri Alma Doğrulama**

Geri almanın güvenli ve öngörülebilir olmasını sağlamak için geri alma işlemini geliştirme ortamında test edin. Tam bir test döngüsü şu adımları içermelidir:

1. Veritabanına değişiklikleri dağıtın ve dağıtıldığından emin olun.
2. Değişiklikleri geri alın, değişikliklerin geri alındığını ve veritabanının önceki hale döndüğünü doğrulayın.
3. Tüm değişiklikleri tekrar veritabanına dağıtın. Son adım olan değişiklikleri yeniden dağıtmak, geri almanın gelecekteki dağıtımları etkileyebilecek değişiklikleri atlamadığını doğrulamayı içerir.

**Veritabanı Güvenliği**

* **Kimlik bilgilerini yönetmek için bir gizli yönetim aracı** kullanın.
* **Otomasyon araçları kullanarak Liquibase'e kimlik bilgilerini geçirin.**
* **Veritabanına doğrudan erişimi sınırlayın.**

Kimlik bilgilerini **Liquibase özellikleri dosyasında saklamamanız** şiddetle tavsiye edilir. Bunun yerine, kimlik bilgilerini yönetmek için bir gizli yönetim aracı veya diğer güvenli konumları kullanın. Bu, bireylerin veritabanına doğrudan erişim gereksinimini ortadan kaldırır ve değişikliklerin etkinliğini ve içeriğini izlenebilir hale getirir.

Liquibase, aşağıdaki gizli yönetim araçlarıyla çalışır:

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

**CI/CD Aracı Kullanarak Kimlik Bilgilerini Saklama**

Bir başka seçenek, kimlik bilgilerini saklamak için bir otomasyon aracı kullanmaktır. Örneğin, Jenkins bazı türdeki kimlik bilgilerini pipeline projeleri için saklayabilir:

* Kimlik bilgileri, Jenkins örneğinde şifreli bir şekilde saklanır ve pipeline projesinde kimlik bilgisi ID'leriyle işlenir.
* Bu, gerçek kimlik bilgilerinin Jenkins kullanıcılarına maruz kalmasını en aza indirir ve kimlik bilgilerini bir Jenkins örneğinden diğerine kopyalama yeteneğini engeller.

Uygulama çalıştırıldığında kullanıcılar tarafından oluşturulan veriler, Liquibase'in kapsamı dışındadır ve başka araçlar tarafından daha iyi yönetilir. Bu veriler şunları içerir:

* **ETL ile ilgili faaliyetler.**
* **Büyük veri yüklemeleri.**
* **Customer verisi.**