**Database migration tools**

**Migration:**

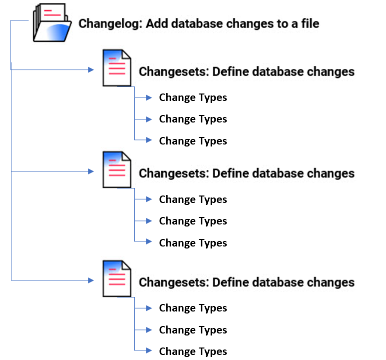
Migrations help transition database schemas from their current state to a new desired state, whether that involves adding tables and columns, removing elements, splitting fields, or changing types and constraints. Migrations manage incremental, often reversible, changes to data structures in a programmatic way.

**Tools:**

Flyway and Liquibase

**Liquibase:**

Liquibase is also an open-source tool used for managing database changes. It is a database schema change management solution that enables you to revise and release database changes faster and safer from development to production. It offers a more flexible and feature-rich approach compared to Flyway. Liquibase uses XML, YAML, or JSON formatted change logs to define the database changes, allowing for a more expressive and customizable migration process. Liquibase supports a wide range of databases, including Oracle, MySQL, PostgreSQL, SQL Server, and many others.



**Changelog:**

Changelog sequentially lists all changes made to your database. This ledger helps Liquibase audit your database and execute any changes that are not yet applied. You can store and version your changelog in any source control tool.

Liquibase deploys the changes you specify in your changelog to your database. You can specify which changelog you want to use with the --changelog-file argument in your CLI, liquibase.properties as environment variables

**Changeset:**

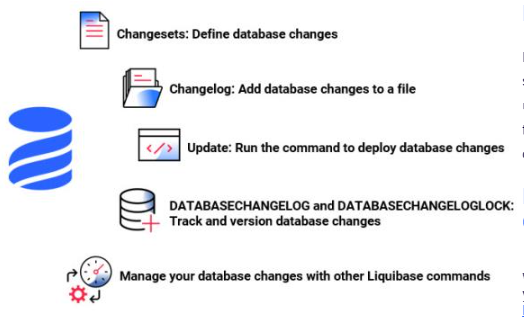
Database changes have the format of changeset. Changesets contain Change types, which are types of operations to apply to the database, such as adding a column or primary key. Context, label and precondition changelog tags help precisely control when a database change is made and to which database environment it is deployed.

A changeset is the basic unit of change in Liquibase. You store all your changesets in your Changelog. Your changesets contain Change Type that specify what each change does, like creating a new table or adding a column to an existing table.



**Liquibase properties file:**

To set the connection between Liquibase with your database, you need the database connection information and parameters. Liquibase includes a properties file to store database connection information and parameters that rarely change.



When changes are deployed, it creates two tables in your database: DATABASECHANGELOG and DATABASECHANGELOGLOCK.

The DATABASECHANGELOG table tracks deployed changes so that you have record. Liquibase compares the changesets in the changelog file with the DATABASECHANGELOG tracking table and deploys only new changesets.

DATABASECHANGELOGLOCK prevents multiple instances of Liquibase from updating the database at the same time. The table manages access to the DATABASECHANGELOG table during deployment and ensures only one instance of Liquibase is updating the database.

**Liquibase commands:**

UPDATE

ROLLBACK

SNAPSHOT

DIFF

STATUS

UTILITY

**Example of Liquibase Migration:**

Using Liquibase, the migration for creating the "users" table would involve creating a change log file, typically in XML format, as follows:



Liquibase provides various other features such as rollback support, precondition checks, and the ability to create complex database changes using change sets.

**Flyway:**

Flyway extends DevOps to your databases to accelerate software delivery and ensure quality code.  Flyway updates a database from one version to the next using migrations. We can write migrations either in SQL with database-specific syntax, or in Java for advanced database transformations.

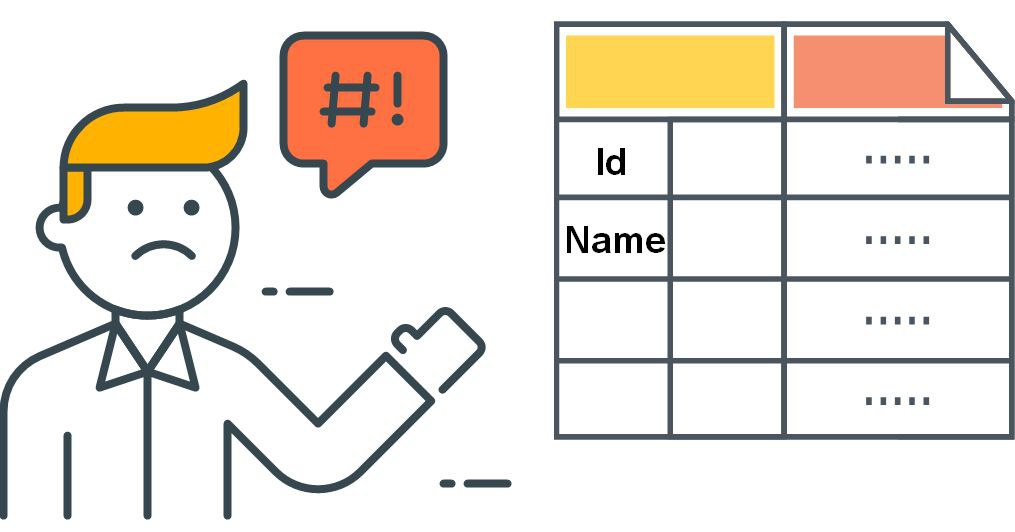
Migrations can either be versioned or repeatable. Within a single migration run, repeatable migrations are always applied last, after pending versioned migrations have been executed.

Repeatable migrations are applied in order of their description. For a single migration, all statements are run within a single database transaction.

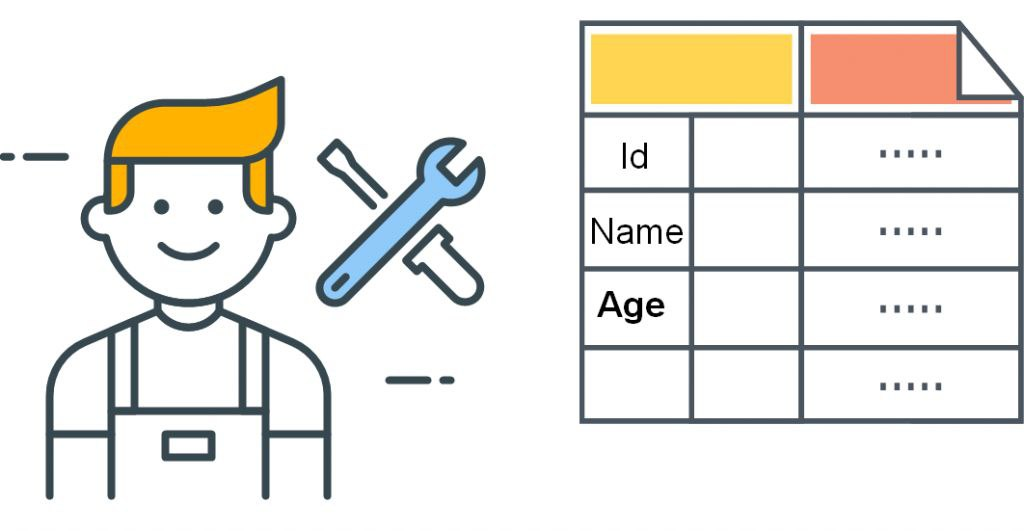
John wants to create a database with information about his businesses customers. He has chosen to use FlyWay to handle his database migrations.



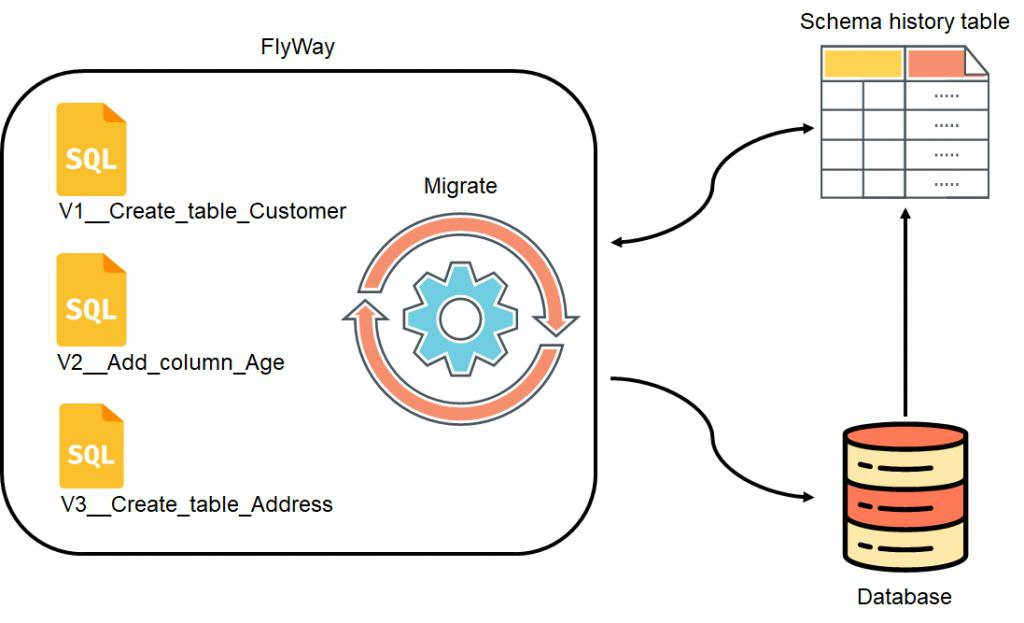
At first, he creates a script to add his customer table to the database, but after running the migration he realizes that he forgot to add information about the age of his customers.



As this column is crucial for John’s database design, he needs to add it. To achieve this, he creates a new script that adds the age column to the customer table, before running a new migration.



John is now happy with his customer table. And John can continue with developing his database.

On the technical side, the story translates to the following diagram. 

Each time John created a migration he ran the the migration process in FlyWay. As a result, FlyWay compared all migration files with the history table to identify changes not yet applied to the database.

**Flyway Commands:**

MIGRATE

CLEAN

INFO

VALIDATE

UNDO

BASELINE

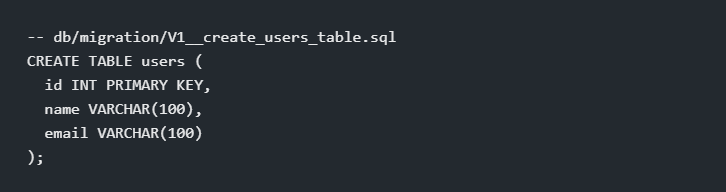
REPAIR

CHECK

SNAPSHOT

**Example of Flyway Migration:**

Let's consider a scenario where we need to add a new table called "users" to an existing database. The Flyway migration script for creating the "users" table might look like this:



Once the migration script is created, Flyway will automatically execute it during the application startup process. It ensures that the database schema is up to date with the latest migrations.

Chocolety installation:

Run Get-ExecutionPolicy. If it returns Restricted, then run Set-ExecutionPolicy AllSigned or Set-ExecutionPolicy Bypass -Scope Process.

**Cmd:** Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

Command to check: choco or choco -

Installing java using powershell:

**Cmd:** choco install javaruntime

Installing mysql:

**Cmd:** choco install mysql

**Cmd:** mysql –V

**Cmd:** mysql –u root –p – enter--- for pwd: enter

Create username and password:

-- use mysql;

-- CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin';

-- GRANT ALL PRIVILEGES ON \* . \* TO 'admin'@'localhost';

-- FLUSH PRIVILEGES;

Installing flyway:

**Cmd:** choco install flyway

Add bin path in Environment variables

**Cmd:** flyway version

**Cmd:** cd flyway/conf/

Create conf file:

Flyway.conf:

flyway.url=jdbc:mysql://localhost:3306/flywaydb

flyway.user=admin

flyway.password=admin

#flyway.sql-migration-prefix=v

flyway.locations=filesystem:C:/Users/window/santa/secretsanta/src/main/resources/db/migration/mysql/0.0

Add .sql files which database has to update.

Sample files:

**V1\_\_person.sql:**

create table PERSON (

ID int not null,

NAME varchar(100) not null

);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**V2\_\_details.sql:**

insert into PERSON (ID, NAME) values (1, 'Santa');

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Cmd:** flyway migrate

Flyway using maven:

Add flyway dependencies, plugin details in pom.xml

Place the .conf file in git and must give MySQL details:

Give the path of conf file in pom.xml:

Run maven command to migrate:

**Cmd:** mvn clean flyway:migrate

Output:

Clean the db:

**Cmd:** mvn flyway:clean

Output:

It will clean the db.

Flyway info:

**Cmd:** mvn flyway:info

Output:

It will show all versions which got updated and its state.

Flyway validation:

Flyway will compare the migrations applied to the database against the available migration scripts on the filesystem. It checks if all migrations applied successfully and if they match the migrations available in the filesystem in terms of checksums and versions.

**Cmd:** mvn flyway:validate

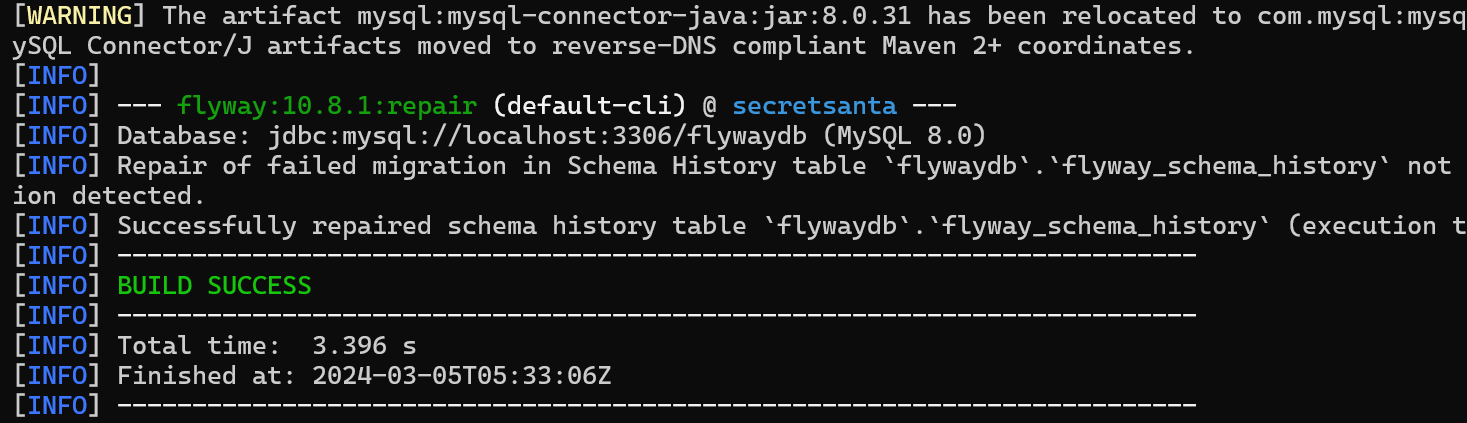
Output:

Flyway repair:

Repair is used to repair the metadata table of the database if it becomes corrupted or out of sync with the migration scripts.

**Cmd:** mvn flyway:repair

Output:



**Differences between Flyway and Liquibase:**

1. Configuration: Flyway follows a convention-over-configuration approach, while Liquibase allows more flexibility and customization through its XML, YAML, or JSON formatted change logs.
2. Syntax: Flyway uses SQL-based migration scripts, whereas Liquibase uses XML, YAML, or JSON to define the database changes.
3. Feature Set: Liquibase offers a broader range of features, including rollback support, complex change sets, preconditions, and more.
4. Learning Curve: Flyway's simplicity makes it easier to get started with, whereas Liquibase's flexibility may require more effort to learn and configure.