DATABASE MIGRATION: FLYWAY (JAVA EE Version)

INSTALLMENT:

-Dependency

Add this dependency into your pom.xml file

```xml
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
    <version>9.3.0</version>
</dependency>
```

-Configuration

Write a flyway executor bean to configure and run flyway on start up

Note: Flyway needs an empty schema to run the first time. We have 2 option:

1/ Make hibernate depends on flyway bean

2/ Use clean on the first run and remove it afterward.

Here I'll use the 2nd option.

1st run bean:

```java
@Startup
@Singleton
@TransactionManagement(TransactionManagementType.BEAN)

public class FlywayMigrationExecutor {
    1 usage
    @Resource(lookup = "java:/agiletermDS")
    DataSource dataSource;

    Nhut Tran Minh *
    @PostConstruct
    public void migrate() {
        Flyway flyway = Flyway.configure()
                .dataSource(dataSource)
                .schemas("public")
                .cleanDisabled(false)
                .load();
        flyway.clean();
        flyway.migrate();
    }
}
```

Change this base on your JNDI Name

Change this base on your schema name.
It's "public" by default.

Remove after 1st run

2nd and afterward run bean (remove clean):

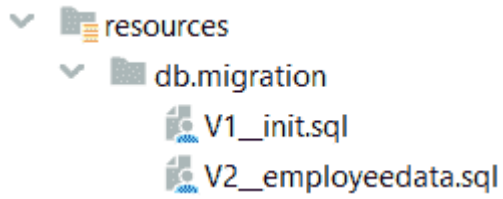
```java
@Startup
@Singleton
@TransactionManagement(TransactionManagementType.BEAN)

public class FlywayMigrationExecutor {
    1 usage
    @Resource(lookup = "java:/agiletermDS")
    DataSource dataSource;

    Nhut Tran Minh
    @PostConstruct
    public void migrate() {
        Flyway flyway = Flyway.configure()
                .dataSource(dataSource)
                .schemas("public")
                .load();
        flyway.migrate();
    }
}
```

-Folder Structure

After that inside the resources folder you will have to create a “db” folder. And inside of “db” folder you’ll have to create a “migration” folder. This is an example of structure.



FLYWAY MIGRATION SCRIPT:

By default, Flyway attempts to read database migration scripts from db/migration folder. We just write normal SQL script for database in these script files.

Please note that you can’t deploy the app without providing these script. So please provide them before deploy the app.

Script naming standard:

`V<VERSION_NUMBER>__<NAME>.sql`

(Between `<VERSION_NUMBER>` and `<NAME>` there’ll be 2 underscores)

Example:

V1__create_table.sql

V2__add_student_data.sql

You can check version update by viewing “flyway_schema_history” in your database.
This is an example:

```
select * from flyway_schema_history;
```

	installed_rank [PK] integer	version character varying (50)	description character varying (200)	type character varying (20)	script character varying (1000)	checksum integer	installed_by character varying (100)	installed_on timestamp without time zone	execution_time integer	success boolean
1	1	1	Create table	SQL	V1__Create_table.sql	1694689770	admin	2022-07-21 14:01:54.578772	29	true
2	2	2	Insert data	SQL	V2__Insert_data.sql	-257996884	admin	2022-07-21 14:40:52.997933	28	true

ROLLBACK
Because the Flyway undo is a commercial feature of Flyway. We need to simulate a rollback via a migration.
Reference : https://www.baeldung.com/flyway-roll-back
This is an example:
We created a migration file called *V1.0__create_book_table.sql*

```
create table book (
id numeric,
title varchar(128),
author varchar(256),
constraint pk_book primary key (id)
);
```

Then, at some point, say we need to reverse the last migration.

In order to restore the database to before the *book* table was created, let's create migration called *V2.0__drop_table_book.sql*

```
drop table book;
```

As far as Flyway is concerned, the second migration file is just another standard migration. The actual restoring of the database to the previous version is done entirely through SQL. For example, in our case, the SQL of dropping the table is the opposite of the first migration, which creates the table.