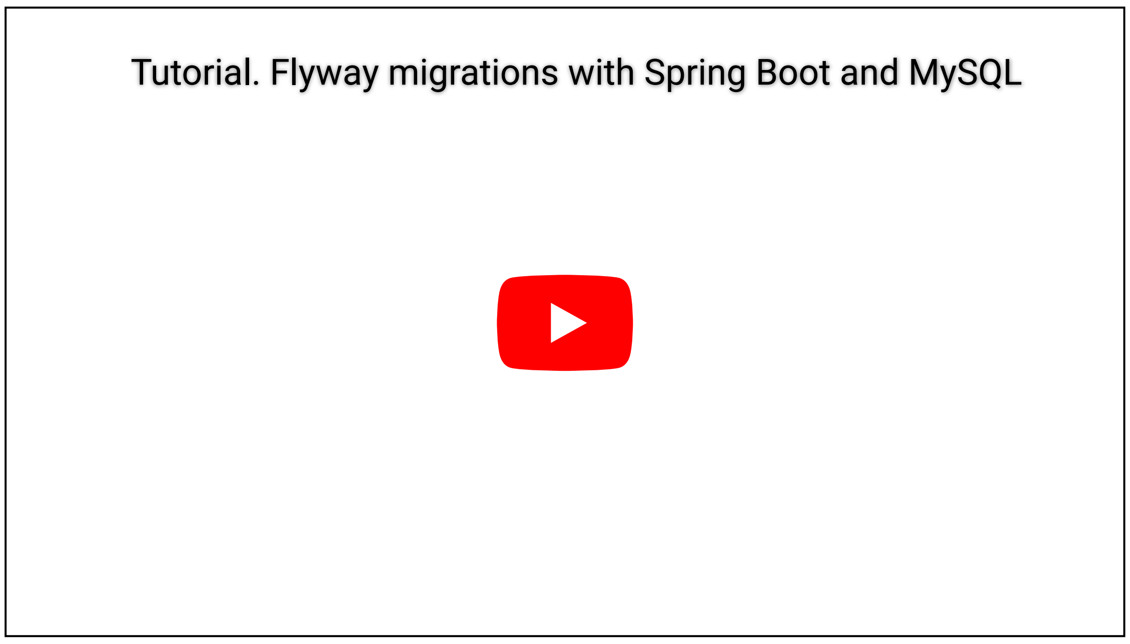
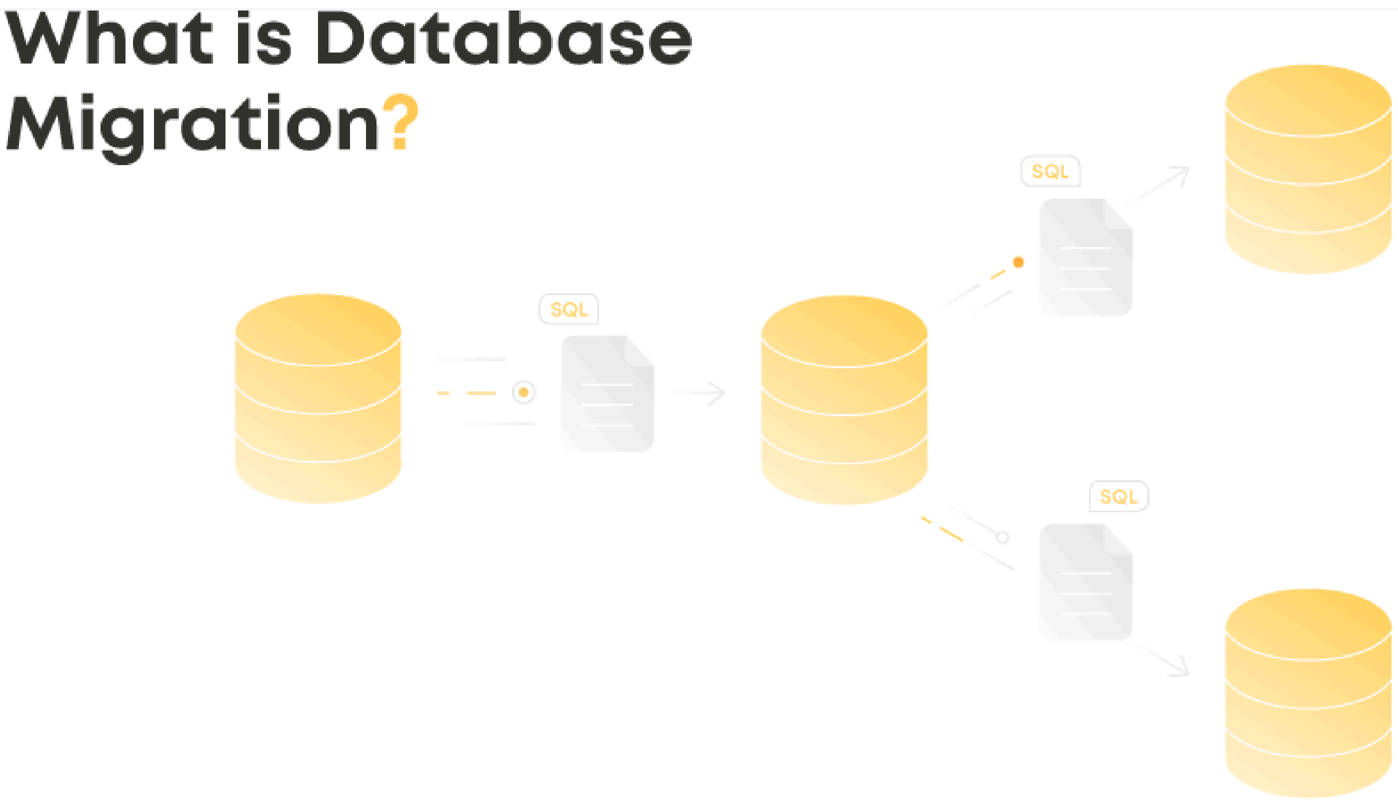
Flyway With Spring Boot And MySQL

As many development teams now use automated database migration, you should know what tools can help you configure and maintain database schemas. Using Flyway is just the right solution if you have a basic understanding of SQL. Database migration using Flyway is quickly gaining popularity in software development because it's easy to use and can be integrated with your existing application system. Furthermore, it supports many databases, including PostgreSQL, MySQL, Oracle DB, H2 db, and many others.

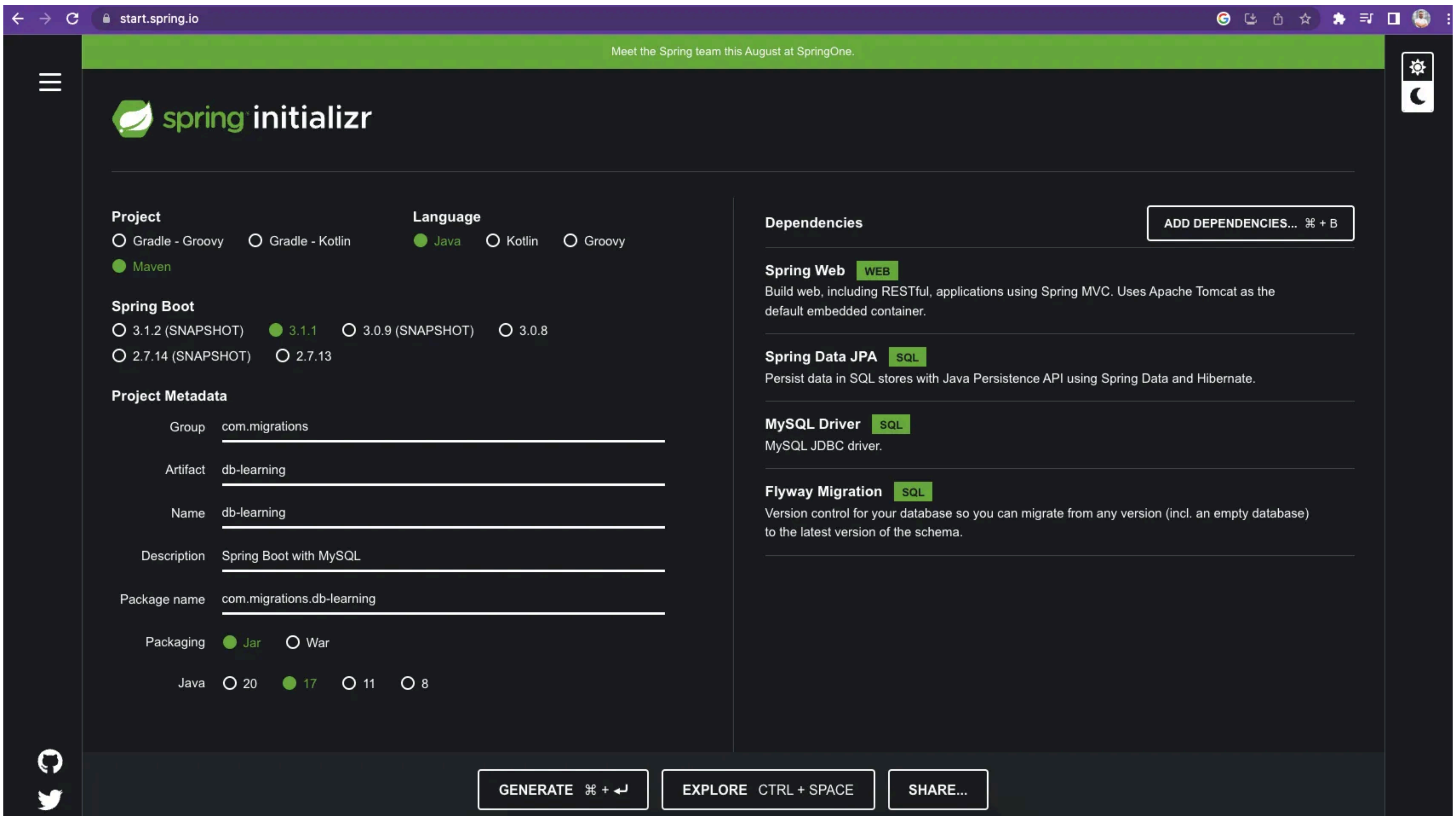


In this article, we'll review the Flyway database migration tool — one of the most popular in the Spring Boot world.



Let’s Go Over a Use Case With MySQL

To start, we’ll go to the Spring Initializr website and create a simple project.



Project
○ Gradle - Groovy ○ Gradle - Kotlin
● Maven

Language
● Java ○ Kotlin ○ Groovy

Spring Boot
○ 3.1.2 (SNAPSHOT) ● 3.1.1 ○ 3.0.9 (SNAPSHOT) ○ 3.0.8
○ 2.7.14 (SNAPSHOT) ○ 2.7.13

Project Metadata

Group: com.migrations
Artifact: db-learning
Name: db-learning
Description: Spring Boot with MySQL
Package name: com.migrations.db-learning
Packaging: ● Jar ○ War
Java: ○ 20 ● 17 ○ 11 ○ 8

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

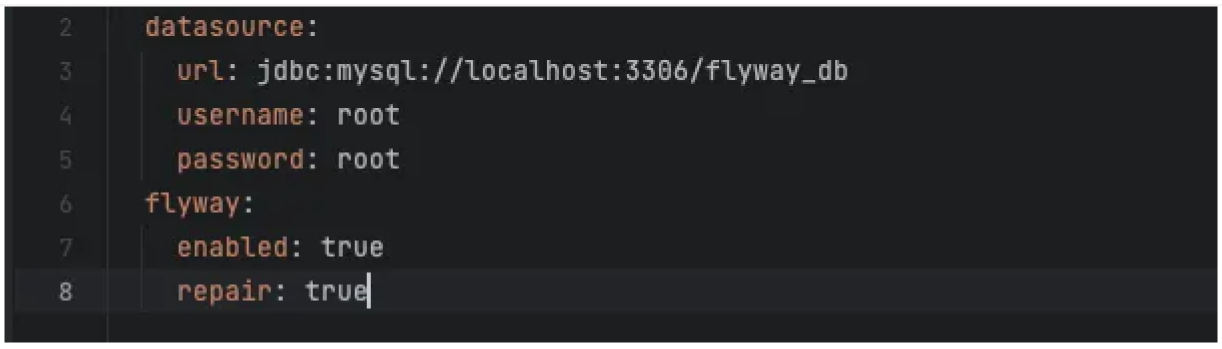
Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL
MySQL JDBC driver.

Flyway Migration SQL
Version control for your database so you can migrate from any version (incl. an empty database) to the latest version of the schema.

Suppose you have the MySQL database up and running. Ours is running at the local host and 3306 port. First, we create a simple User class.

```
@Entity
@Table(name = "users")
public class User {
```



```
  datasource:
    url: jdbc:mysql://localhost:3306/flyway_db
    username: root
    password: root
  flyway:
    enabled: true
    repair: true
```

We also need to configure our app to connect to local MySQL. Let's do this in application.yml

Creation of SQL Table Users in MySQL

It's also necessary to create SQL table users in MySQL.

First, create a schema called flyway_db in your MySQL instance.

Then you can create your migration scripts; the first step is to create a users table.
By default, Flyway will monitor the db/migration folder under the resources directory and check scripts there. The naming convention for the script is the following:

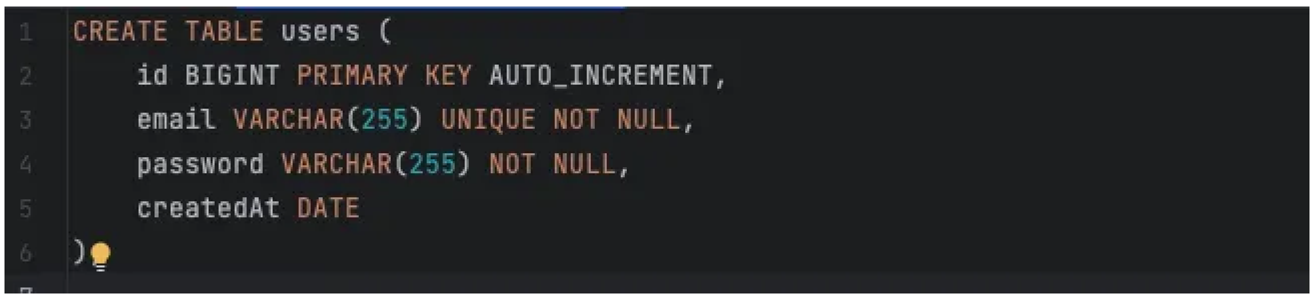
V{incremental_version_number}__{your_custom_script_name}.sql

So our script for creating a users table has the following name:

V1__create_users_table.sql

SQL Statement Creation

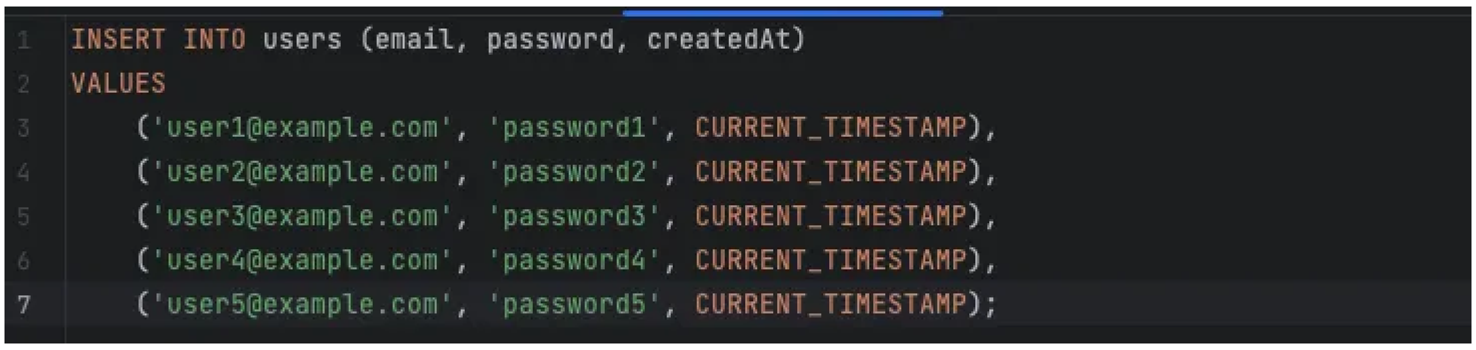
We also need to create an SQL statement to make a MySQL table:



```
CREATE TABLE users (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    createdAt DATE
)
```

This will create a 'users' table in our flyway_db database.
With Flyway, you can create a migration script to populate db with some data. So the following script will add some users to the 'users' table.

It's named: V2__insert_users.sql



```
INSERT INTO users (email, password, createdAt)
VALUES
    ('user1@example.com', 'password1', CURRENT_TIMESTAMP),
    ('user2@example.com', 'password2', CURRENT_TIMESTAMP),
    ('user3@example.com', 'password3', CURRENT_TIMESTAMP),
    ('user4@example.com', 'password4', CURRENT_TIMESTAMP),
    ('user5@example.com', 'password5', CURRENT_TIMESTAMP);
```

When we run our application, the users table will be populated with these five users. And when you review them, you'll notice that the

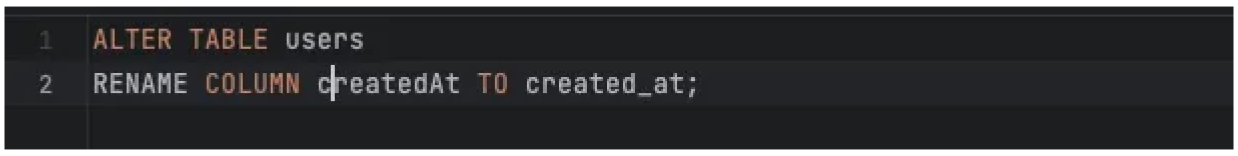
After this script, the users table has the createdAt column that is up to date. It holds the TIMESTAMP of user creation, including hours, minutes, and seconds

You probably want to make your app work as fast as you expect. Spring Framework has something special for you. Take a closer look at this guide.

Let's move on

Suppose you change an existing column name for any reason you may have during app development. Of course, you want all entities in your database to have these changes applied. To do this, we create another script.

V4__change_column_name.sql



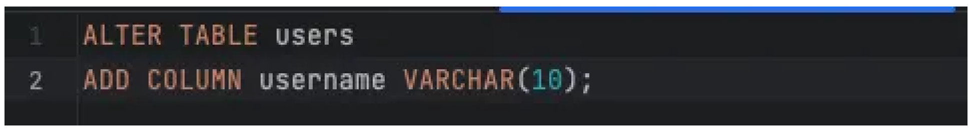
```
ALTER TABLE users
RENAME COLUMN createdAt TO created_at;
```

When the application is launched, it will automatically pick up the newly created script and apply changes to the users table. So now the createdAt column is renamed to created_at.

Adding a new column to the user entity

The app is growing and includes a lot of developed features. At some point, you may decide to add a new column to the user entity. For this, we need to create another script.

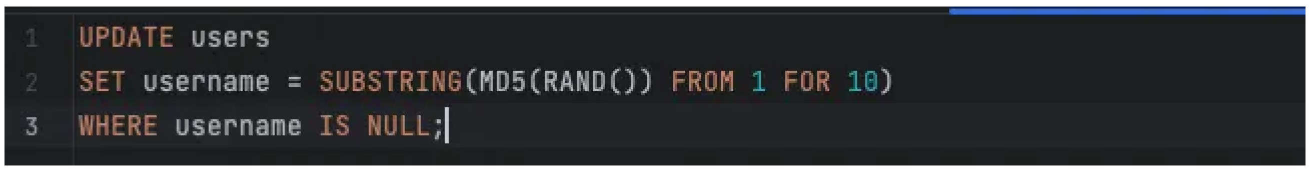
V5__add_username_to_users.sql



```
ALTER TABLE users
ADD COLUMN username VARCHAR(10);
```

This script will add a 'username' column to all user entities. So now you can create users with a username. However, old users do not have a username, and that's not a good practice. You may want to fix that and apply some random string using your own pattern. In our case, we just use a random 10 chars length string for username for old users. Flyway will assist us again.

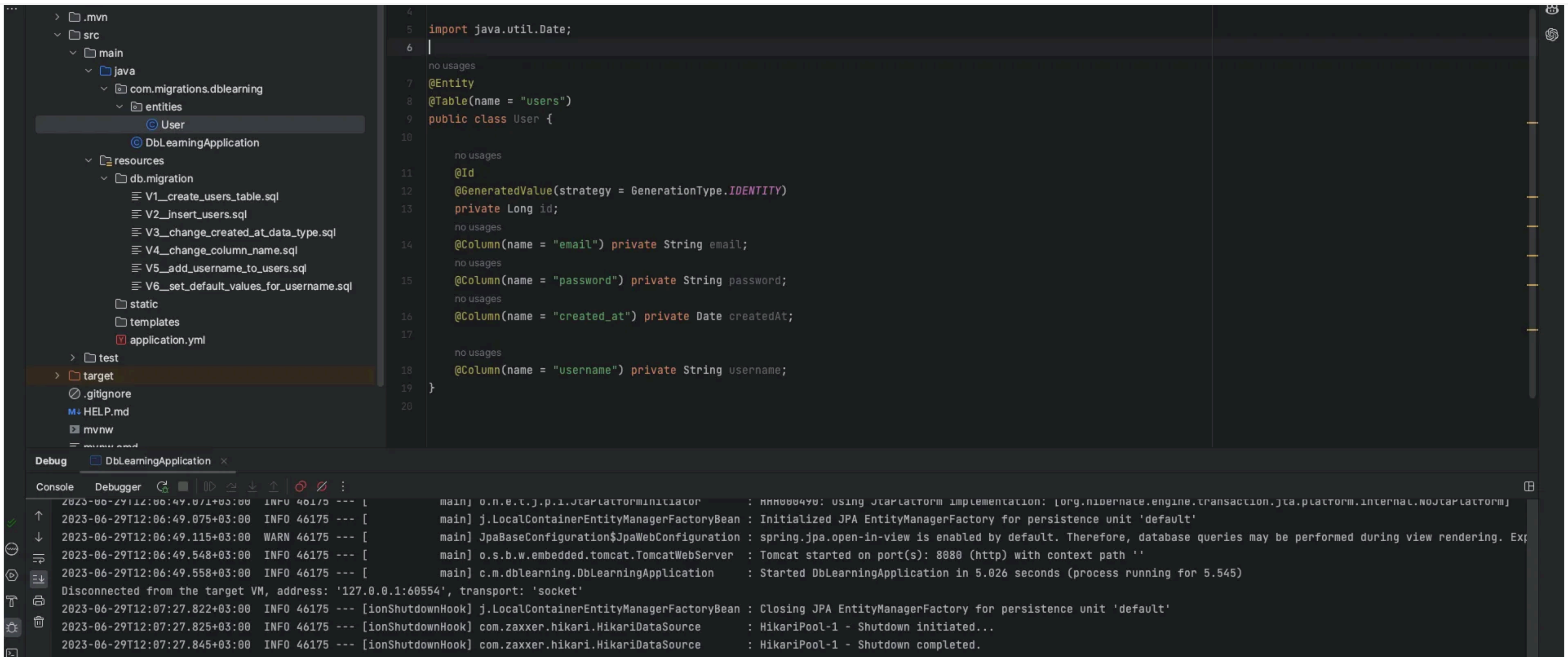
V6__set_default_values_for_username.sql



```
UPDATE users
SET username = SUBSTRING(MD5(RAND()) FROM 1 FOR 10)
WHERE username IS NULL;
```

As you see, it works. All the old users registered before the implementation of our super feature now also receive a random username. Of course, they will be able to change it any time, but the point is that now your app won't crash because of some NullPointerExceptions due to missing value for the username.

Here is what our project structure looks like:



```java
import java.util.Date;

no usages
@Entity
@Table(name = "users")
public class User {

    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    @Column(name = "email") private String email;
    no usages
    @Column(name = "password") private String password;
    no usages
    @Column(name = "created_at") private Date createdAt;

    no usages
    @Column(name = "username") private String username;
}
```

```
2023-06-29T12:06:49.071+03:00  INFO 46175 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2023-06-29T12:06:49.075+03:00  INFO 46175 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-06-29T12:06:49.115+03:00  WARN 46175 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Exp
2023-06-29T12:06:49.548+03:00  INFO 46175 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2023-06-29T12:06:49.558+03:00  INFO 46175 --- [           main] c.m.dblearning.DbLearningApplication     : Started DbLearningApplication in 5.026 seconds (process running for 5.545)
Disconnected from the target VM, address: '127.0.0.1:60554', transport: 'socket'
2023-06-29T12:07:27.822+03:00  INFO 46175 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2023-06-29T12:07:27.825+03:00  INFO 46175 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
2023-06-29T12:07:27.845+03:00  INFO 46175 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

To sum up

To sum up
Now all the scripts are located in the db/migrations folder and have an incrementing version number at the beginning of their names. So this is how Flyway works — it allows you to manage any changes you add to your MySQL database over time during app development.
Thank you for reading the guide! I hope it let you better understand the Spring Boot world and the necessity of the Flyway tool for the successful migration of your database.

Ihor Kosandyak
Lead Software Engineer
15 Aug, 2023 · 4 min read

Secure your Spring Boot API with JSON Web Tokens

If you are reading this article I assume you are a bit familiar with Spring Boot and building API using it. Because the main purpose of this article is to show you a simple way how to make your API more secured.

Ihor Sokolyk
7 Jun, 2021 · 6 min read

Liquibase is a DB migration and change operation tool developed specifically to help software developers change their databases and move seamlessly through the development and testing stages. The role of this tool is to hold all the changes made to a particular database in a single file named changelog. It also allows for loading information […]

Ihor Kosandyak
31 Jul, 2023 · 5 min read

Subscribe to our blog

Uploading files to AWS S3 Bucket using Spring Boot

Share

Intro Hi guys! Today we are going to talk about uploading files to Amazon S3 Bucket from your Spring Boot application. As you may notice almost each application, mobile or web, gives users an ability to upload their images, photos, avatars etc. So you, as a developer, should choose the best way how to save and where to store […]

Ihor Sokolyk
3 Dec, 2017 · 6 min read

Contact Us

Privacy Policy Cookie Policy Sitemap