Flyway

dasturlash.uz

Flyway

- Flyway is a version control application to evolve your Database schema easily and reliably across all your instances.
- Version control for your database

- Flyway - bu versiyani boshqarish tizimidir.

Flyway

- Many software projects use relational databases. This requires the handling of database migrations, also often called schema migrations.
- **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. The former has a unique version and is applied exactly once. The latter doesn't have a version. Instead, they are (re-)applied every time their checksum changes.
- 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.

Flyway + Spring Boot

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

In Property file

- baseline() is a method you can call on Flyway. You can invoke it from command line, for example.
- On the other hand, baselineOnMigrate is a setting which you can set int Flyway configuration (if you're using SpringBoot it's application.properties):
  - spring.flyway.baselineOnMigrate = true
- It means "if schema is non-empty and user didn't call flyway baseline explicitly, call it implicitly now"

Recommend using only a single migration mechanism

- Using multiple migration mechanisms can cause confuse and hard to manage, so it's a good practice to disable others when you are using Flyway.
- Spring Boot, via JPA and Hibernate, can auto-export schema DDL to a database via your definition on @Entity classes. You can turn it off by setting validate or none (none is the default value for non-embedded databases) to the spring.jpa.hibernate.ddl-auto property.
- Spring Boot can autorun classpath:schema.sql and classpath:data.sql script files for your DataSource. This feature can be controlled via spring.datasource.initialization-mode property. Its value is embedded (only apply for embedded databases) by default

Flyway migration scripts

- Unlike JPA and Hibernate, database migration in Flyway is not based on the definition of @Entity classes, you have to manually write the migration scripts in either SQL or Java, SQL is the most commonly used.
- Typically, SQL scripts are in the form V<VERSION>__<DESCRIPTION>.sql
  - <VERSION> is a dot or underscore separated version, such as '1.0' or '1_1'). <VERSION> must be unique
  - <DESCRIPTION> should be informative for you able to remember what each migration does.
  - Example: V1.0__create_book.sql

Flyway migration scripts Example

- V1.0__create_book.sql

```
CREATE TABLE `book` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `description` varchar(255) DEFAULT NULL,
  `title` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
);
```

- V1.1__insert_book.sql

```
INSERT INTO `book`(`title`, `description`) VALUES('Hello Koding', 'Coding tutorials series');
```

Migration File location

- By default, Spring Boot looks for them in classpath:db/migration folder, you can modify that location by setting spring.flyway.locations

How Flyway works

- Flyway applies migration scripts to the underlying database in the order based on the version number specified in the script file naming.
- At each execution, only pending migrations are applied. Flyway manages this via creating (if not exists) and updating a metadata table. You can find more details about this table in the latter part of this tutorial.
- The migration scripts can not be changed after applied. Flyway compares the checksum of each script in every execution and throws an exception if there's a mismatch.

Config Flyway DataSource

- Spring Boot uses either annotations or external properties to connect Flyway to the underlying data source.
- @Primary DataSource or @FlywayDataSource annotation
- spring.datasource.[url, username, password], or spring.flyway.[url, user, password] properties

Run Flyway with Spring Boot

- Spring Boot auto enable and trigger Flyway at the application startup when you include the Flyway core library into the project. In case you'd like to turn it off, update this setting spring.flyway.enabled to false (true is the default value).
- The application startup may be failed if there's an exception (such as the checksum mismatch error of migration scripts mentioned in the previous section) thrown by Flyway during migration.
- Each migration script is run within a single transaction. You can configure to run all pending migrations in a single transaction with spring.flyway.group=true (the default value is false)

Query migration status and history

- You can query migration status and history in web interface with Spring Boot Actuator by enabling it in this property management.endpoints.web.exposure.include=info,health,flyway and access to {endpoints}/actuator/flyway.
- Apart from that, you can also query the table flyway_schema_history in your database.
- It is created by Flyway to manage migration status and history. The table name can be changed via setting spring.flyway.table (flyway_schema_history is the default name)

Links

- https://hellokoding.com/database-migration-evolution-with-flyway-and-jpa-hibernate/
- https://www.callicoder.com/spring-boot-flyway-database-migration-example/
- https://www.tutorialspoint.com/spring_boot/spring_boot_flyway_database.htm

Spring Boot: Hibernate and Flyway boot order

- Spring boot runs flyway migrations before hibernate.
- To change it I overrode the spring boot initializer to do nothing.
- Then I created a second initializer that runs after hibernate is done.
- All you need to do is add this configuration class:

Hibernate and Flyway boot order - 1

- 1. Enable Flyway auto run mode:
  - spring.flyway.enabled=false

dasturlash.uz

Hibernate and Flyway boot order - 2

- 2. Create DataSource Bean

```
@Configuration
public class MigrationConfiguration {
    @Value("${spring.datasource.url}")
    private String dataSourceUrl;
    @Value("${spring.datasource.username}")
    private String dataSourceUsername;
    @Value("${spring.datasource.password}")
    private String dataSourcePassword;

    @Bean
    public DataSource getDataSource() {
        DataSourceBuilder dataSourceBuilder = DataSourceBuilder.create();
        dataSourceBuilder.url(dataSourceUrl);
        dataSourceBuilder.username(dataSourceUsername);
        dataSourceBuilder.password(dataSourcePassword);
        return dataSourceBuilder.build();
    }
}
```

Hibernate and Flyway boot order - 2

- Call Flyway migration method after application startup.

```
@SpringBootApplication
public class FlywayDemoApplication implements CommandLineRunner {
    @Autowired
    private DataSource dataSource;

    public static void main(String[] args) {
        SpringApplication.run(FlywayDemoApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        Flyway.configure().baselineOnMigrate(true).dataSource(dataSource).load().migrate();

    }
}
```

Differences Between Liquibase and Flyway

- Flyway uses SQL for defining a change.
- On the other hand, Liquibase provides flexibility to specify a change in different formats including SQL such as XML, YAML, and JSON.
- With Liquibase we can work with database-agnostic languages and easily apply schema changes to different database types.
- **Flyway is built around a linear database versioning system** that increments on each versioned change. This sometimes can create conflicts with parallel development.
- Liquibase migration doesn't need to follow any of the file name conventions. **In Liquibase, changes are managed by one ledger known as a master changelog** which will be defined as including all the migrations.

Links

- https://www.baeldung.com/liquibase-vs-flyway
- Spring Boot example:
- https://www.codeusingjava.com/boot/liq