Database migration using Flyway

First problem is we should not give hibernate full control over table management (ddl_auto: update).
So we should never delegate database management to our application as any developer changes the property of ddl_auto then it can destroy whole database.
Second problem is when we have to upload/edit static data using script then we have to redeploy the application from basic configuration and further manage it on the basis of environment (dev or prod)

Add the dependency

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

Flyway by default looks for db.migration folder inside the resources folder.
This folder contains some sql files.
Depending on the flyway history it will execute the sql files one by one.

We can change the ddl-auto to validate to verify the models and tables formed in the database to have exact schema.

We have to add configuration:
flyway:
    baseline-on-migration: true
    enabled: true
    user:
    password:
User and password are the credentials for database we are working with (like Postgres)

To tag existing schema with some version and description that also we can do:
flyway:
    baseline-description: “init”
    baseline-version: 0 (by default it is 1)

Flyway has some pattern for writing the name of sql file...
Like V1__ should be prefixed with name of file, here version number can be anything.

Flyway tries to hash the contents of the sql and give some checksum for it. So if we try to change the file name or file content we will get exception.

The later versions will override the lower versions.
We can create new sql file like V2__create_hibernate.sequence.sql

So any new things come in later version of the sql file will be included over the previous changes.

Flyway manages all the history using flyway_schema_history table where we get tracking of all versions, time, checksum, etc.

//about baseline migration

In Flyway, a database migration tool, the concept of "baseline migration" is used to establish a starting point for versioned database migrations. When you set `flyway.baselineOnMigrate` to `true`, it instructs Flyway to automatically apply a baseline migration to an existing database schema if no migrations have been applied yet.

Here's what happens when you set `flyway.baselineOnMigrate` to `true`:

1. **Initial Setup**: When Flyway is first applied to an existing database schema with no prior migrations, it detects that there are no migration history records.

2. **Baseline Migration**: Flyway automatically applies a baseline migration to the database. This baseline migration essentially establishes the current state of the database schema as the starting point for subsequent migrations. It typically involves creating a special table (`schema_version` by default) to track migration history.

3. **Migration Execution**: After the baseline migration is applied, Flyway continues to execute subsequent migrations according to their version numbers. Migrations that have a higher version number than the baseline are applied in order.

By setting `flyway.baselineOnMigrate` to `true`, you ensure that Flyway handles the initial setup of the database schema automatically, even if no migrations have been explicitly defined. This is particularly useful when you're introducing Flyway to an existing project with an already established database schema.

Here's an example of how you can configure Flyway in a Java application properties file (e.g., `application.properties`):

```properties
# Enable baseline migration
flyway.baselineOnMigrate=true
```

This setting instructs Flyway to perform a baseline migration if necessary when

migrations are executed. It's a convenient way to ensure that your database schema is managed consistently, regardless of its initial state.

https://www.baeldung.com/database-migrations-with-flyway

https://docs.spring.io/spring-boot/docs/3.0.0-M5/reference/htmlsingle/#howto.data-initialization.migration-tool.flyway