Schema evolution with Liquibase

Introduction

The data that the users have created by using the Yona app needs to be retained when upgrading to a new build of Yona server. New versions of Yona server will require changes to the database schema and these changes need to retain the existing data. This is in principle possible to do that through the standard JPA/Hibernate schema evolution. The documentation however says this:

> *Do *not* use against production databases, as there are no guarantees that the proper delta can be generated, nor that the underlying database can actually execute the required operations).*

So we need a different strategy. Directly writing SQL scripts is an option, but it is cumbersome, error prone and it is not easy to make it database independent (support HSQLDB and MariaDB). For these reasons, we have opted for Liquibase. Liquibase is an open source database-independent library for tracking, managing and applying database schema changes. The database schema is created or updated through a sequence of steps that is called a change log. Such change logs can be handwritten (in XML, YML, JSON and various other formats), but they can also be generated by Liquibase and its extensions. Given a database schema, Liquibase can generate a change log file to recreate that database schema. There is also a Liquibase Hibernate extension that can generate the database schema given an JPA/Hibernate application. Next, Liquibase in combination with the Hibernate plug-in supports generating a diff change log between the existing database schema and the revised JPA/Hibernate application. That is what we need to evolve the database schema along with the application.

This page describes the required steps to make a schema change, and it also describes how the base schema files have been generated, just in case we ever need to do it again.

Making a schema change

1 - Preparation

Create an empty database with

```
mysql --user=%YONA_DB_USER_NAME% --password=%YONA_DB_PASSWORD% < scripts\recreateYonaDB.sql
```

and then load the current schema:

```
gradlew :dbinit:liquibaseUpdate
```

2 - Generate change log

Generate the change log file of your change:

```
gradlew :dbinit:liquibaseDiffChangelog
```

This produces the file `dbinit/src/main/liquibase/updates/changelog-0000-yd-000.yml`

Rename it to the next higher sequence number and update the JIRA issue key, something like this: changelog-0012-yd-476.yml

Liquibase generates some excess change sets, which need to be removed manually from the change log:

1. There are a few (4) change sets like this:

```
- changeSet:
    id: 1567840502005-1
    author: bert (generated)
    changes:
    - dropDefaultValue:
        columnDataType: varchar(255)
        columnName: activity_category_of_changed_goal_id
        tableName: messages
```

This seems to be a bug in the Liquibase Hibernate extension. Removing these default values results in runtime errors and explicitly adding the default value in the Java code is either not allowed (e.g. `@Column(length = 24 * 4 * 12, columnDefinition = "varchar(1152) default null")`) does not resolve it. Remove all of these changes sets from the change log file.

Review the remaining change set: does it reflect what you wanted to do? If not, update it. The Liquibase documentation might come in handy.

3 - Add the change set to `updates.yml`

Edit `dbinit/src/main/liquibase/updates/updates.yml` and add your change log to it.

4 - Update the database schema

Update the database schema with your change set: `gradlew :dbinit:liquibaseUpdate`

5 - Add the new change log file to Git.

Using your favorite Git tools.

This completes the schema update.

Regenerating the base schema files

The base schema consists of two files: `entity.yml` containing the schema for the entities of Yona server, and `batch.yml, containing the database schema for Spring Batch. If it is ever necessary to regenerate these files, use the steps below.`

Entity schema

The entity schema is contained in `entity.yml`. The Liquibase Hibernate extension is used to generate this file. The command is inside `liquibase.gradle`. To run it, delete `entity.yml` and type `gradlew :dbinit:liquibaseGenerateChangeLogFromCode`.

The current version of Liquibase has some issues that require manual correction of the generated change log file:

- Replace all instances of `BLOB` with `TINYBLOB`, with the exception of `url_ciphertext` in `messages` (that can grow as long as 2k, so it requires a BLOB).
- Replace all instances of `datetime` with `datetime(6)`

Spring batch schema

Given that the schema for Spring Batch is buried as SQL script inside the JARs of Spring Batch, it was easier to first load the schema in the database (by deleting `batch.yml` and running `gradlew :dbinit:bootRun`, which is configured to use Hibernate to create/update the schema) and then have Liquibase generate the change log from the database. The command is inside `liquibase.gradle`. To run it, type `gradlew :dbinit:liquibaseGenerateChangeLogFromDB`. After that, remove all Yona related change sets from the change log file and only retain what is related to Spring Batch.

The current version of Liquibase has some issues that require manual correction of the generated change log file:

- Change all table names to UPPERCASE. On Windows, MariaDB is case insensitive, but on Windows, it is case sensitive on most varieties of Linux.

Quartz schema

The change log file for the Quartz scheduler database schema can be generated the same way as Spring Batch