Getting Started With Liquibase

Creating the database for your application seems to be easy. But it quickly gets complicated when you need to support multiple versions or work in huge teams.

A version based database migration process allows you to evolve your database together with your code and to automatically apply database updates when you deploy a new release. Liquibase is one of the available tools that help you to define and execute the required update steps.

Create a ChangeLog

The database change log is an XML, JSON, YAML or SQL file which describes all changes that need to be performed to update the database.

In most cases, you want to create 1 file for each release. Each file consists of one or more change sets.

Create a ChangeSet

A *changeSet* describes a set of changes that Liquibase executes within one transaction. You can add as many changes to a set as you like. But to avoid any problems during a rollback, you shouldn't define more than one logical change per set.

Each *changeSet* gets identified by the name of the *author* and an *id*. Liquibase stores this information together with the name of the change log file in the *databasechangelog* table to keep track on the executed change sets.

Create a Database Table

The following code snippet shows you a *changeSet* that creates the table *author* with the columns *id*, *firstname*, *lastname* and *version*. You just need a *createTable* element which has to define the name of the table you want to create. On top of that, you can specify other attributes, like the name of the database schema or the tablespace.

You also need to provide at least one *column* tag as a nested property. In the example, I use 4 of these tags, to define the 4 database columns of the *author* table.

You can also use a constraints tag to define a primary key, not null, unique, foreign key or cascade constraint.

```
<changeSet author="Thorben" id="2">
    <createTable tableName="author">
        <column name="id" type="BIGINT">
            <constraints nullable="false"/>
        </column>
        <column name="firstname" type="VARCHAR(255)"/>
        <column name="lastname" type="VARCHAR(255)"/>
        <column name="version" type="INT">
            <constraints nullable="false"/>
        </column>
    </createTable>
</changeSet>
```

Define a Primary Key

If you didn’t define the primary key when you created the table, you can add the constraint with an *addPrimaryKey* tag.

```
<changeSet author="Thorben" id="5">
    <addPrimaryKey columnNames="id"
        constraintName="pk_book"
        tableName="book"/>
</changeSet>
```

Define a Foreign Key Constraint

You can add a foreign key constraint with an *addForeignKeyConstraint* tag. You then need to provide the name of the constraint, the *baseColumnNames* and *baseTableName*, that define the column and table to which you want to add the constraint, and the *referencedColumnNames* and *referenceTableName*, that define the column and table to which the foreign key points to.

```
<changeSet author="Thorben" id="8">
    <addForeignKeyConstraint baseColumnNames="authorid"
        baseTableName="bookauthor"
        constraintName="fk_bookauthor_author"
        referencedColumnNames="id"
        referencedTableName="author"/>
</changeSet>
```

Generate a ChangeLog

You don't need to write the changeLog file yourself if you already have an existing database. You can then use the command line client to generate the file.

```
liquibase --driver=org.postgresql.Driver \
      --classpath=myFiles\postgresql-9.4.1212.jre7.jar \
      --changeLogFile=myFiles/db.changelog-1.0.xml \
      --url="jdbc:postgresql://localhost:5432/recipes" \
      --username=postgres \
      --password=postgres \
      generateChangeLog
```

Export the SQL Statements

Before you execute the *changeLog*, you should always export and review the generated SQL statements. Some database administrators also need the script for their internal documentation or plainly reject to perform any changes they didn't review themselves.

In all these situations, you can use Liquibase's command line client to generate the required SQL statements and write them to a file.

```
liquibase --driver=org.postgresql.Driver \
      --classpath=myFiles\postgresql-9.4.1212.jre7.jar \
      --changeLogFile=myFiles/db.changelog-1.0.xml \
      --url="jdbc:postgresql://localhost:5432/test_liquibase" \
      --username=postgres \
      --password=postgres \
      updateSQL
```

Execute a ChangeLog

After you've created and checked the *changeLog* yourself or used the command line client to create it, you can choose between multiple options to execute it. I use the command line client in the following example but you can also use a maven plugin to create the database as part of your build or deployment process or you can use a Servlet, Spring or CDI Listener to automatically create or update the database at application startup.

```
liquibase --driver=org.postgresql.Driver \
      --classpath=myFiles\postgresql-9.4.1212.jre7.jar \
      --changeLogFile=myFiles/db.changelog-1.0.xml \
      --url="jdbc:postgresql://localhost:5432/test_liquibase" \
      --username=postgres \
      --password=postgres \
      update
```

Liquibase documents the execution of all *changeSet*s in the *databasechangelog* table. It will uses this information for future runs of the update process to determine which changeSets need to be executed.