Update A Database With Liquibase

A version-based database migration tool and process allow you to evolve your database together with the code. You can then perform the required update operations when you install a new version of your application on your development, test or production system.

The Update Process

The update process consists of 3 parts:

1. You should create a backup and tag the current version of the database so that you can roll back all your changes if necessary.
2. While you implement your code changes, you should define a *changeLog* that describes the required changes of your database.
3. And when you install your update, you need to execute the database migration and roll it back if any error occurs.

Tag the current database

The tag is not required to roll back your changes. Liquibase can also roll back the executed *changeSet*s one by. But I prefer to tag my database before I perform any updates. That gives me a defined state to which I can go back easily if anything goes wrong.

You can create a tag with Liquibase's command line client by calling the tag command with the name of the tag and the connection information.

```
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 \
    tag v1.00
```

Liquibase can't restore any data that got deleted during the database migration. You should, therefore, create a backup before you perform any update.

Define the Update ChangeLog

You should have 1 changelog file for each software update. So, you have 1 file for version 1.0 and another one for version 1.1.

As soon as you have more than 1 changelog file, you should add a master *changeLog* that includes all other files. So, for this series of posts, I have a *db.changelog.xml* file which includes the files *db.changelog-1.0.xml* and *db.changelog-1.1.xml*.

```
<databaseChangeLog>
    <include file""myFiles/db.changelog-1.0.xml"/>
    <include file""myFiles/db.changelog-1.1.xml"/>
</databaseChangeLog>
```

You can then provide the master changelog to the Liquibase client. It will iterate through the included files and check which *changeSet*s need to be executed to update the database to the latest version.

Add a Table

You can use a *createTable* tag to tell Liquibase to create a new database table. The follwing XML snippet creates the *author* table with the columns *id*, *firstname*, *lastname* and *version*.

This *changeSet* also contains a rollback tag. That's because Liquibase doesn't generate a rollback operation when you create a new database table. If you want to remove the table when you perform a rollback, you need to use the rollback tag to provide your own rollback operation. You can use it with all other Liquibase tags, or you can provide SQL statements that shall be executed.

```
<changeSet author="Thorben" id="1">
    <createTable tableName="publisher">
        <column name="id" type="BIGINT">
            <constraints nullable="false"/>
        </column>
        <column name="name" type="VARCHAR(255)"/>
        <column name="version" type="INT">
            <constraints nullable="false"/>
        </column>
    </createTable>
    <rollback>
        <dropTable tableName="publisher" />
    </rollback>
</changeSet>
```

Rename a Table

You can rename a database table with a *renameTable* tag. It requires 2 attributes: the *oldTableName* and the *newTableName*. You can also define the *catalogName* and *schemaName*.

You don't need to provide the rollback operation when you rename a table. Liquibase can generate the required statement. But you can use the rollback tag to override the generated statement.

```
<changeSet author="Thorben" id="2">
    <renameTable  oldTableName="author"
        newTableName="book_author"/>
</changeSet>
```

Drop a Table

You can drop a database table with the *dropTable* tag. As you can see in the following code snippet, you just need to provide the *tableName* as an attribute.

Be careful and create a database backup before you drop a table. Otherwise, you will not be able to restore any data that's stored in the table.

```
<changeSet author="Thorben" id="1">
    <dropTable tableName="publisher" />
</changeSet>
```

Add a Column

You can add a database column with an *addColumn* tag with a *tableName* attribute and one or more *column* tags.

Liquibase can generate the rollback operation, so you only need to specify it if you want to override the generated statement.

```
<changeSet author="Thorben" id="3">
    <addColumn tableName="book_author">
        <column name="dateofbirth" type="DATE"/>
        <column name="middlename" type="VARCHAR(255)"/>
    </addColumn>
</changeSet>
```

Rename a Column

You can rename a database column with the *renameColumn* tag. It requires the attributes *tableName*, *oldColumnName* and *newColumnName*.

You don't need to provide a *rollback* tag for this operation. Liquibase generates the required statements.

```
<changeSet author="Thorben" id="4">
    <renameColumn tableName="book_author"
        oldColumnName="firstname"
        newColumnName="first_name" />
</changeSet>
```

Drop a Column

You can drop a database column with the *dropColumn* tag and the attributes *tableName* and *columnName*.

Before you drop a database column, you should create a backup of your database. Liquibase can't generate the rollback operation and it's most often impossible to recreate the deleted data without a backup.

```
<changeSet author="Thorben" id="5">
    <dropColumn tableName="book_author"
        columnName="middle_name" />
</changeSet>
```

Merge 2 Columns

This operation creates a new table column, sets the concatenated value of the 2 old columns as the value of the new one and drops the 2 old table columns.

You can define this operation with a *mergeColumn* tag and attributes to define the *tableName,* the *finalColumnName* and *finalColumnType,*

the names of the 2 old columns as *column1Name* and *column2Name* and an optional *joinString*.

```
<changeSet author="Thorben" id="6">
    <mergeColumns tableName="book_author"
        finalColumnName="name"
        finalColumnType="VARCHAR(255)"
        column1Name="first_name"
        column2Name="last_name"
        joinString=" ">
</changeSet>
```

A merge of 2 database columns is another operation that Liquibase can't rollback automatically. I always recommend to create a database backup before you perform this operation and to use that instead of a set of rollback operations.

Execute the Update

You just need to call the Liquibase client with the master *changeLog* file. It will check all included files and determine which *changeSet*s it needs to execute. The client will then generated the required SQL statements and either export or execute them.

You can generate the SQL statements with the following call of the Liquibase client application.

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

After you reviewed the generated statements, you can call the update command with the same parameters. Liquibase will then find and execute the required *changeSet*s to migrate the database to the latest version.

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