Liquibase – Database Deployment in agilen Umgebungen

Frank Winter
ORBIT Gesellschaft für Applikations- und Informationssysteme mbH
Bonn

Schlüsselworte
Liquibase, SQL-Skripte, Deployment, Agile Methoden, DDL, DML, Continuous Integration, Continuous Deployment

Einleitung

Moderne und agile Softwareentwicklungsprojekte zeichnen sich u.a. durch ein hochfrequentes Deployment aktualisierter Softwarestände auf Entwicklungs-, Test- und Produktivsysteme aus. Während es vergleichsweise unproblematisch ist, neue Versionen einer Software zu paketieren und auszuliefern, ist dies bei Datenbankänderungen deutlich schwieriger. Die sich hierbei stellenden Herausforderungen und die zugehörigen Lösungsansätze, die das Open Source Tool „Liquibase“ hierbei liefert, werden im Folgenden dargestellt.

Die Herausforderungen

Bei dem Deployment von Datenbankänderungen stellen sich mehrere Herausforderungen:

1. Es sollen mit jeder Auslieferung nur Datenbank-Änderungen (DDL und DML) zu dem jeweils letzten Release eingespielt werden. Klassischerweise musste hierfür bislang jeweils ein separates Installationsskript geschrieben werden. Bei Auslieferungszyklen von oft nur wenigen Tagen ist das zu umständlich und damit fehleranfällig.

2. DDL-Statements und festgeschriebene DML-Statements lassen sich nicht ohne weiteres rückgängig machen. Wie geht man z.B. mit einem umfangreichen Installationsskript um, das mittendrin abbricht und bereits festgeschriebene Änderungen hinterlässt? Jede logisch in sich abgeschlossene Änderung sollte rückgängig gemacht werden können.

3. Datenbankänderungen werden oft unabhängig von anwendungsseitigen Code-Änderungen deployed. Das ist jedoch gefährlich, insbesondere, wenn auf verschiedenen Entwicklungsbranches gearbeitet wird. Welche Änderung gehört in welchen Branch? Hier hat man mit handgeschriebenen Skripten schnell den Überblick verloren. Code- und Datenbankänderungen gehören zusammen in ein Deployment. Insbesondere bei einer agilen Entwicklung und einem Continuous Deployment.

4. Entwicklern steht nicht immer eine Oracle-Datenbank zur Verfügung. Gerade für Applikationen, die auf verschiedenen Datenbanken laufen sollen, möchte man nicht für jede unterstützte Datenbank separate DDL- und DML-Skripte schreiben.

Diese Aufgabenstellungen lassen sich weitgehend mit der Open Source Library „Liquibase“ abdecken.

Was ist Liquibase?

Liquibase ist ein datenbankunabhängiges Database Change Management Tool und dient der Durchführung und Verwaltung aller Arten von Datenbankänderungen (DML und DDL). Änderungen, die über Liquibase-Skripte durchgeführt werden, lassen sich einfach verwalten und bei Bedarf weitgehend rückgängig machen.

Liquibase benötigt in der aktuellen Version nur eine Java-Laufzeitumgebung (JDK/JRE) ab Version 1.6. Es eignet sich hervorragend für den Einsatz in agilen Projekten, da es sehr gut in einer Umgebung mit Continuous Integration und Continuous Deployment integrierbar ist.

Wie arbeitet Liquibase?
Liquibase baut über JDBC eine Verbindung zu einem bestimmten Datenbankschema auf und führt dort eine beliebige Anzahl von Liquibase-Skripten (sog. **ChangeLogs**) aus. Diese ChangeLogs rufen entweder weitere ChangeLogs auf oder enthalten ein oder mehrere logisch in sich abgeschlossene Datenbankänderungen (sog. **ChangeSets**). Solche ChangeSets enthalten zumeist Daten- oder Strukturänderungen, die entweder in einer Liquibase-spezifischen, datenbankunabhängigen Syntax (siehe unten) oder in datenbankspezifischem Code (wie z.B. PL/SQL-Blöcke) formuliert sind.

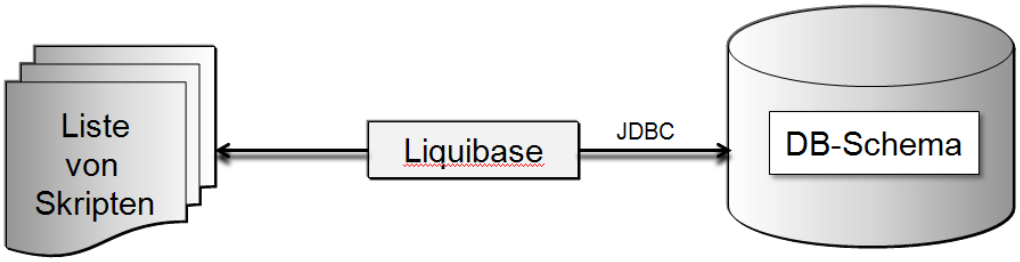

Abb. 1: Zugriff von Liquibase auf Skripte und ein Datenbankschema

Sämtliche bereits durchgeführten Änderungen werden in dem betroffenen Datenbankschema in der Tabelle DATABASECHANGELOG festgehalten, die zu jedem ChangeSet einen separaten Datensatz enthält. Damit ist Liquibase bekannt, dass die Änderung erfolgreich durchgeführt wurde.

Die Tabelle DATABASECHANGELOG enthält folgende Informationen:

Spalte	Typ	Nullable	Beschreibung
ID	VARCHAR2(63)	nein	Frei definierbare ID des ChangeSets. Muss gemeinsam mit AUTHOR und FILENAME eindeutig sein.
AUTHOR	VARCHAR2(63)	nein	Autor des Skriptes
FILENAME	VARCHAR2(200)	nein	Pfad und Name des Skriptes
DATEEXECUTED	TIMESTAMP(6)	nein	Zeitpunkt der Ausführung
ORDEREXECUTED	INTEGER	nein	Zähler für die Ausführungen
EXECTYPE	VARCHAR2(10)	nein	Ausführungstyp; meist EXECUTED oder RERAN
MD5SUM	VARCHAR2(35)		MD5-Hash je ChangeSet
DESCRIPTION	VARCHAR2(255)		automatisch generierte Beschreibung des ChangeSet, z.B. „Custom SQL“, „Create View“ oder „Add Column“
COMMENTS	VARCHAR2(255)		Kommentar des Entwicklers zu dem ChangeSet
TAG	VARCHAR2(255)		optionale Markierung, z.B. für ein bestimmtes Release; kann im Rollback verwendet werden
LIQUIBASE	VARCHAR2(20)		verwendete Liquibase-Version

Tabelle 1: Spalten der Tabelle DATABASECHANGELOG

Im Normalfall wird ein ChangeSet nur einmal ausgeführt. Es kann jedoch auch so definiert werden, dass es bei jeder Ausführung von Liquibase neu gestartet wird (z.B. für die Durchführung von Grants

zu neuen Datenbank-Objekten) oder bei einer Änderung des Skriptes erneut ausgeführt werden soll (sinnvoll z.B. bei Views oder Stored Procedures).

Wie sehen Liquibase-Skripte aus?
Liquibase-ChangeLogs lassen sich in verschiedenen Formaten definieren. Neben dem meist verwendeten XML-Formt werden YAML, JSON und SQL unterstützt. Ein Liquibase-Skript besteht üblicherweise aus folgenden Bestandteilen:

- **ChangeLog**: der gesamte Inhalt des Liquibase-Skriptes
- **ChangeSet**: Ein Satz logisch zusammenhängender Statements. ChangeSets werden in die Tabelle DATABASECHANGELOG eingetragen (Kombinierter Primärschlüssel aus ID, AUTHOR und FILENAME). Ein Change Sets kann seinerseits aus mehreren Changes bestehen.
- **Precondition**: Vorbedingung, die entweder für den ChangeLog oder ein ChangeSet gilt.

Hier ein einfaches Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog .......
   logicalFilePath="2.5.5/01_alter_table_dummy1.xml">
   <changeSet id="1" author="fwinter" dbms="Oracle" runOnChange="false" failOnError="true">
     <comment>
       Hinzufuegen der Spalte EKP in Tabelle DUMMY1
     </comment>
     <addColumn tableName="DUMMY1">
        <column name="EKP" type="VARCHAR2(50)" value="meine neue EKP">
          <constraints nullable="true" />
        </column>
     </addColumn>
    </changeSet>
</databaseChangeLog>
```

Das Beispiel verwendet die Liquibase-spezifische Syntax für die Anlage einer neuen Spalte in einer Tabelle. Bei der Verwendung dieser Syntax kann Liquibase in vielen Fällen bei Bedarf automatisch das passende Rollback-Statement generieren (in diesem Fall also ein DROP COLUMN), um die Änderung wieder rückgängig zu machen.

Liquibase bietet eine Vielzahl von Tags für die Durchführung von Datenbankänderungen. Die Dokumentation auf www.liquibase.org ist gut verständlich und bietet zahlreiche Syntaxbeispiele. Nachfolgend sei ein aus nur einem ChangeSet bestehendes Beispiel gezeigt, in dem Oracle-spezifischer Code ausgeführt wird. Das Beispiel enthält zudem die Verwendung einer Precondition.

```
<changeSet id="3" author="fwinter" dbms="Oracle" failOnError="true" runAlways="false" >
   <preConditions onFail="MARK_RAN" onFailMessage="Datensatz gibt es schon!">
      <sqlCheck expectedResult="0">
         select count(*) from DUMMY2 where UUID = 123
      </sqlCheck>
   </preConditions>
   <comment> neuer Eintrag in Tabelle Dummy2 </comment>
   <sql>
     INSERT INTO DUMMY2(UUID) VALUES(123)
   </sql>
   <rollback>
     delete from DUMMY2 where UUID = 123
   </rollback>
</changeSet>
```

Die Precondition prüft in diesem sehr einfachen Beispiel, ob es einen bestimmten Datensatz schon gibt. Ist dies der Fall, wird der ChangeSet nicht ausgeführt aber als erfolgreich gelaufen markiert (onFail = "MARK_RAN"). Es wäre natürlich ebenso leicht möglich gewesen, das Skript kontrolliert abzubrechen (onFail = "HALT").

Bei der Verwendung von SQL-Tags (<sql>) reicht Liquibase die enthaltenen SQL-Statements an die Datenbank durch und kann daher kein Rollback der Änderung automatisch generieren. Da die Verfügbarkeit eines Rollbacks dringend zu empfehlen ist, muss ein solches bei der Verwendung von SQL-Tags, wie in dem obigen Beispiel, explizit angegeben werden.

Nur zur Klarstellung: Das Rollback (ob nun automatisch generiert oder manuell programmiert) ist nicht Bestandteil einer Fehlerbehandlung, und der Abbruch eines Skriptes bewirkt nicht das Ausführen des Rollback-Teils. Ein Rollback dient dem nachträglichen Zurückrollen aller Änderungen eines ChangeSets, um das Datenbankschema wieder auf einen älteren Stand zurückzusetzen.

Wie werden Liquibase-Skripte aufgrufen?
Liquibase-Skripte lassen sich beliebig kaskadierend aufrufen. Wird Liquibase über die Kommandozeile aufgerufen, muss ein ChangeLogFile angegeben werden (z.B. update.xml). Es handelt sich hierbei um einen gewöhnlichen ChangeLog, nur dass dieses meist keine ChangeSets enthält, sondern den Aufruf weiterer ChangeLogFiles.

Hier ein einfaches Beispiel für den Aufruf weiterer Liquibase-Skripte aus dem zentralen ChangeLogFile:

```
…
  <include file="my_variables.xml" relativeToChangelogFile="true" />
  <include file="pfad1/skript1.xml" relativeToChangelogFile="true" />
  <include file="pfad2/skript2.xml" relativeToChangelogFile="true" />
…
```

Man kann sich die Arbeit zusätzlich erleichtern, indem man eine Datei namens liquibase.properties (keine XML-Datei) anlegt, die alle für die JDBC-Connection notwendigen Attribute enthält. Auch hierzu ein Beispiel:

```
#liquibase.properties
driver: oracle.jdbc.OracleDriver
classpath: /u01/app/oracle/product/11.2.0.3/dbhome_1/jdbc/lib/ojdbc6.jar
url: jdbc:oracle:thin:@myhost.acme.de:1521:oradb
username: MYUSER
password: geheimesPasswort
```

Im einfachsten Fall kann Liquibase nun über die Kommandozeile wie folgt aufgerufen werden:

```
liquibase --changeLogFile=update.xml update
```

Es wird genau ein ChangeLogFile angegeben, das seinerseits kaskadierend beliebig weitere ChangeLogs aufruft. Liquibase überprüft nun anhand der Einträge in der Tabelle DATABASECHANGELOG, welche Änderungen aus den ggf. zahlreichen ChangeLogs noch anstehen und führt nur diese aus.

Über die Kommandozeile lassen sich zahlreiche verschiedene Kommandos übergeben. Hier ein paar der wichtigsten dieser Kommandos:

Kommando	Beschreibung
Update	Führt eine Aktualisierung des Datenbankschemas durch.
updateSQL	Schreibt die SQL-Statements zur Aktualisierung des Datenbankschemas in die Tabelle STDOUT. Diese SQL-Statements werden nicht ausgeführt, sollten aber in eine Datei umgeleitet und später angewandt werden.
updateTestingRollback	Führt eine Aktualisierung des Datenbankschemas durch, rollt diese Änderungen zurück, um sie dann wieder erneut auszuführen. Gut geeignet für den Test von Update und Rollback im Rahmen der Entwicklung.
rollback <tag>	Führt ein Rollback aller Änderungen durch, die neuer sind als das ChangeSet mit einem bestimmten Tag (siehe Attribut Tag in Tabelle DATABASECHANGELOG)
rollbackToDate <date/time>	Führt ein Rollback aller nach einem bestimmten Zeitpunkt durchgeführten Änderungen durch.
rollbackCount <x>	Führt ein Rollback der letzten ChangeSets durch, wobei x eine natürliche Zahl ist.
clearCheckSums	Entfernt die Checksummen aus dem Feld MD5SUM in Tabelle DATABASECHANGE-LOG.
diff \[diff parameters\]	Erzeugt einen Report über die Differenzen zwischen zwei Datenbankschemen.
diffChangeLog \[diff parameters\]	Erzeugt ein ChangeLogFile, dessen Ausführung die Differenzen zwischen zwei Datenbankschemen ausgleicht.
generateChangeLog	Generierung eines initialen ChangeLogs aus einem bereits mit Datenbankobjekten gefüllten Datenbankschema.

Tabelle 2: Auswahl einiger wichtiger Liquibase-Kommandos

Siehe hierzu: http://www.liquibase.org/documentation/command_line

Verwendung von Variablen
Interessant und in der Praxis meist auch notwendig ist die Verwendung von Variablen, die umgebungsspezifisch gesetzt werden. Solche Variablen werden meist in einer zentralen Datei definiert (z.B. einem ChanegLog-File namens my_variables.xml). Die Definition von Variablen sieht in einer solchen Datei etwa wie folgt aus.

```
<property name="db_user" value="SCOTT"/>
<property name="db_pw" value="tiger"/>
```

Einmal (möglichst in einer zentralen Datei) bekannt gegeben, werden solche Variablen nach dem Aufruf dieser Datei in allen weiteren Liquibase-Skripten mit der Schreibweise `${variablenname}` referenziert.

Dieses Feature ist besonders wichtig im Rahmen eines Continuous Deployment, da diese Variablen über Tools wie z.B. Puppet (http://puppetlabs.com) je nach Umgebung automatisch unterschiedlich gesetzt werden, ohne dass hier jegliches manuelles Eingreifen notwendig ist.

Was kann Liquibase sonst noch?
Liquibase kann noch mehr. Für die Generierung eines initialen ChangeLogs aus einem bereits mit Datenbankobjekten gefüllten Datenbankschema bietet Liquibase z.B. die Option „generateChangeLog“ an. Das generierte ChangeLog enthält die Definitionen eines Großteils der bestehenden Datenbankobjekte (leider werden nicht alle Typen unterstützt, z.B. keine Stored Procedures). Interessant ist u.a. auch die Generierung von Diff-Skripten oder Diff-Reports, die den Unterschied zwischen zwei Datenbankschemen beheben bzw. dokumentieren.

Fazit

Durch die eigenständige Verwaltung, welche Datenbankänderungen im Rahmen eines Deployments bereits gelaufen sind und welche noch laufen müssen, ist Liquibase ein sehr nützliches Tool in agilen Umgebungen, in denen sehr häufig Deployments stattfinden. In Entwicklungs- und Testumgebungen spielen darüber hinaus die Rollback-Möglichkeiten eine wichtige Rolle, da auf diese Weise auch der datenbankseitige Teil einer Applikation wieder in ein älteres Release zurückversetzt werden kann. Hierbei gibt es naturgemäß gewisse Einschränkungen (z.B. ist ein DROP COLUMN nicht einfach durch ein ADD COLUMN rückgängig zu machen). In einem Oracle-Umfeld sollte man daher zusätzlich über die Verwendung von sog. „Restore Points“ nachdenken.

Kontaktadresse:

Frank Winter
ORBIT Gesellschaft für Applikations- und Informationssysteme mbH
Mildred-Scheel-Str. 1
D- 53175 Bonn

Telefon: +49 228 95693-748
Fax: +49 228 95693-99
E-Mail frank.winter@orbit.de
Internet: www.orbit.de