bulatoren wie „Startseite“, „Kommunikation“ und „Benachrichtigungen“ die gleichen Inhalte haben.

Wichtig ist, dass die OTN-Community ein öffentliches Forum ist. Um einen Kontakt mit einem Support-Mitarbeiter über die Community herzustellen, ist darauf zu achten, dass der korrekte Space oder Sub-Space in der My-Oracle-Support-Community ausgewählt ist.

Der beste Weg, um die neue My-Oracle-Support-Community-Plattform kennenzulernen, liegt in seiner Benutzung. Bei Problemen in der Community empfiehlt sich der bekannte Weg, einen nicht-technischen Service Request zu eröffnen.

Karl-Heinz Urban
karl-heinz.urban@oracle.com

Liquibase – Database Deployment in agilen Projekten

Frank Winter, ORBIT Gesellschaft für Applikations- und Informationssysteme mbH

Moderne und agile Softwareentwicklungs-Projekte zeichnen sich unter anderem 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 Datenbank-Änderungen deutlich schwieriger. Der Artikel zeigt die dabei auftretenden Herausforderungen sowie die passenden Lösungsansätze, die das Open-Source-Tool „Liquibase“ bietet.

Beim regelmäßigem Deployment von Datenbank-Änderungen (DDL und DML) stellen sich in den meisten Entwicklungsprojekten folgende Herausforderungen:

- Es sollen mit jeder Auslieferung nur Änderungen zu dem jeweils letzten Release eingespielt werden. Separate Installationsskripte sind bei kurzen Auslieferungszyklen jedoch aufwändig und fehleranfällig.
- Wenn in verschiedenen Entwicklungssträngen parallel gearbeitet wird, ist die Version eines Datenbank-Schemas nicht immer eindeutig identifizierbar und es ist schwer, parallel entstandene Schema-Änderungen zusammenzuführen.
- Datenbank-Änderungen werden oft gänzlich unabhängig von Code-Änderungen eingespielt. Das ist aber gefährlich. Code- und Datenbank-Änderungen gehören insbesondere bei einer Continuous Delivery in eine gemeinsame Auslieferung.
- Änderungen von Daten und Strukturen lassen sich nicht ohne Weiteres rückgängig machen. Wie geht man beispielsweise mit einem umfangreichen Änderungsskript um, das mittendrin abbricht? Welchen Zustand hat das Datenbank-Schema?

Diese Herausforderungen lassen sich weitgehend mit der Open-Source-Library Liquibase in den Griff bekommen. Es handelt sich um ein datenbankunabhängiges Database-Change-Management-Tool und dient der Durchführung und Verwaltung aller Arten von Datenbank-Änderungen (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 ab Version 1.6. Es eignet sich hervorragend für den Einsatz in agilen Projekten, da es sich sehr gut in eine Umgebung mit Continuous Integration beziehungsweise Continuous Delivery integriert.

Liquibase baut über JDBC eine Verbindung zu einem bestimmten Datenbank-Schema auf und führt dort eine beliebige Anzahl von Liquibase-Skripten (sogenannte „ChangeLogs“) aus *(siehe Abbildung 1)*. Diese rufen entweder weitere ChangeLogs auf oder enthalten eine oder mehrere logisch in sich abgeschlossene Datenbank-Änderungen (sogenannte „ChangeSets“). Diese wiederum enthalten zumeist Daten- oder Struktur-Änderungen, die entweder in einer Liquibase-spezifischen, datenbankunabhängigen Syntax (siehe unten) oder in datenbankspezifischem Code (wie PL/SQL-Blöcke) formuliert sind.

Sämtliche bereits durchgeführten Änderungen werden in dem betroffenen

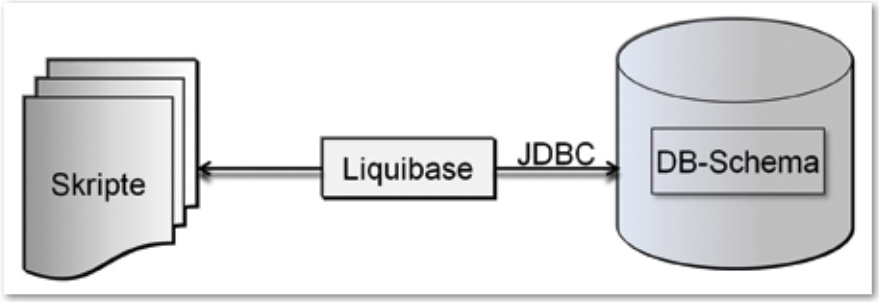

Abbildung 1: Zugriff von Liquibase auf Skripte und ein Datenbankschema

Spalte	Beschreibung
ID	Frei definierbare ID des ChangeSet. Muss gemeinsam mit den Feldern „AUTHOR“ und „FILENAME“ eindeutig sein
AUTHOR	Autor des Skripts
FILENAME	Pfad und Name des Skripts
DATEEXECUTED	Zeitpunkt der Ausführung
ORDEREXECUTED	Zähler für die Ausführungen
EXECTYPE	Ausführungstyp; meist „EXECUTED“ oder „RERAN“
MD5SUM	MD5-Hash je ChangeSet
DESCRIPTION	Automatisch generierte Beschreibung des ChangeSet, wie „Custom SQL“, „Create View“ oder „Add Column“
COMMENTS	Kommentar des Entwicklers zum ChangeSet
TAG	Optionale Markierung, etwa für ein bestimmtes Release; kann im Rollback verwendet werden
LIQUIBASE	Verwendete Liquibase-Version

Tabelle 1: Spalten der Tabelle DATABASECHANGELOG

Datenbankschema in der *Tabelle 1* „DATABASECHANGELOG“ festgehalten, die zu jedem ChangeSet einen separaten Datensatz enthält. Damit ist Liquibase bekannt, welche Änderung erfolgreich durchgeführt wurde.

Im Normalfall wird ein ChangeSet nur einmal ausgeführt. Es kann jedoch definiert werden, dass ein bestimmtes ChangeSet bei jeder Ausführung von Liquibase neu gestartet wird (beispielsweise für die Durchführung von Grants zu neuen Datenbank-Objekten) oder dass es bei einer inhaltlichen Änderung erneut auszuführen ist (sinnvoll etwa bei Views oder Stored Procedures).

Die Liquibase-Skripte

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

- *ChangeLog*
  Der gesamte Inhalt des Liquibase-Skripts
- *ChangeSet*
  Ein Satz logisch zusammenhängender Statements. ChangeSets werden nach der Ausführung in die Tabelle „DATABASECHANGELOG“ eingetragen (kombinierter Primärschlüssel aus „ID“, „AUTHOR“ und „FILENAME“). Ein ChangeSet kann seinerseits aus mehreren Changes bestehen.
- *Precondition*
  Vorbedingung, die entweder für den ChangeLog oder ein ChangeSet gilt

Listing 1 zeigt ein einfaches Beispiel dazu. Es verwendet die Liquibase-spezifische Syntax für die Anlage einer neuen Spalte in einer Tabelle. Bei deren Verwendung kann Liquibase in vielen Fällen bei Bedarf automatisch das passende Rollback-Statement generieren (in diesem Falle also ein „DROP COLUMN“), um die Änderung wieder rückgängig zu machen.

Liquibase bietet eine Vielzahl von Tags für die Durchführung von Datenbank-Änderungen. Die Dokumentation auf „www.liquibase.org“ ist gut verständlich und bietet zahlreiche Syntax-Beispiele. *Listing 2* zeigt

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog .......
   logicalFilePath="4.5.5/01_alter_table_dummy1.xml">
   <changeSet id="1" author="fwinter" dbms="Oracle"
runOnChange="false" failOnError="true">
     <comment>
       Hinzufuegen der Spalte ORT in Tabelle DUMMY1
     </comment>
     <addColumn tableName="DUMMY1">
        <column name="ORT" type="VARCHAR2(50)" value="ein neuer Ort">
          <constraints nullable="true" />
        </column>
     </addColumn>
    </changeSet>
</databaseChangeLog>
```

Listing 1

ein aus nur einem ChangeSet bestehendes Beispiel, in dem Oracle-spezifischer Code ausgeführt wird. Das Beispiel enthält zudem die Verwendung einer Precondition.

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 automatisch kein Rollback der Änderung generieren. Da die Verfügbarkeit eines Rollbacks dringend zu empfehlen ist, muss ein solches bei der Verwendung von SQL-Tags, wie im 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 Skripts bewirkt nicht das Ausführen des Rollback-Teils. Ein Rollback dient dem nachträglichen Zurückrollen aller Änderungen eines ChangeSets, um das Datenbank-Schema wieder auf einen älteren Stand zurückzusetzen.

```
<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>
```

Listing 2

Aufruf der Liquibase-Skripte

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

Listing 3 zeigt ein einfaches Beispiel für den Aufruf weiterer Liquibase-Skripte aus dem zentralen ChangeLogFile. Man kann sich die Arbeit erleichtern, indem man eine Datei namens „liquibase.properties“ (keine XML-Datei) anlegt, die alle für die JDBC-Connection notwendigen Attribute enthält *(siehe Listing 4)*.

Im einfachsten Fall kann man Liquibase nun über die Kommandozeile „liquibase --changeLogFile=update.xml update“ aufrufen. Es ist 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 gegebenenfalls zahlreichen ChangeLogs noch anstehen und führt nur diese aus. *Tabelle 2* zeigt die wichtigsten Komman-

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

Listing 3

```
#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
```

Listing 4

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

Listing 5

Kommando	Beschreibung
Update	Führt eine Aktualisierung des Datenbank-Schemas durch.
updateSQL	Schreibt die SQL-Statements zur Aktualisierung des Datenbank-Schemas nach „STDOUT“. Diese SQL-Statements werden nicht direkt ausgeführt, sollten aber in eine Datei umgeleitet und später angewandt werden.
updateTestingRollback	Führt eine Aktualisierung des Datenbank-Schemas 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 x ChangeSets durch.
clearCheckSums	Entfernt die Checksummen aus dem Feld „MD5SUM“ aus der Tabelle „DATABASECHANGELOG“.
diff \[diff parameters\]	Erzeugt einen Report über die Differenzen zwischen zwei Datenbank-Schemata.
diffChangeLog \[diff parameters\]	Erzeugt ein ChangeLogFile, dessen Ausführung die Differenzen zwischen zwei Datenbank-Schemata ausgleicht.
generateChangeLog	Generiert ein initiales ChangeLog aus einem bereits mit Datenbank-Objekten gefüllten Datenbank-Schema.

Tabelle 2: Auswahl der wichtigsten Liquibase-Kommandos

dos, die sich über die Kommandozeile absetzen lassen (siehe „www.liquibase.org/documentation/command_line“).

Verwendung von Variablen

Interessant und in der Praxis meist auch notwendig ist der Einsatz von Variablen, die umgebungsspezifisch gesetzt sind. Diese werden meist in einer zentralen Datei definiert (zum Beispiel in einem ChangeLog-File namens „my_variables.xml“). *Listing 5* zeigt die Definition von Variablen in einer solchen Datei. Einmal − möglichst in einer zentralen Datei − bekanntgegeben, 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 einer Continuous Delivery, da diese Variablen über Tools wie Puppet (siehe „http://puppetlabs.com“) je nach Umgebung automatisch unterschiedlich gesetzt werden können, ohne dass hier weiteres manuelles Eingreifen notwendig ist.

Weitere Liquibase-Features

Liquibase kann noch mehr. Für die Generierung eines initialen ChangeLogs aus einem bereits mit Datenbank-Objekten gefüllten Datenbank-Schema bietet Liquibase beispielsweise die Option „generateChangeLog“. Das damit generierte ChangeLog enthält die Definitionen eines Großteils der bestehenden Datenbank-Objekte (leider sind nicht alle Objekt-Typen unterstützt). Interessant ist unter anderem auch die Generierung von Diff-Skripten oder Diff-Reports, die den Unterschied zwischen zwei Datenbank-Schemata beheben beziehungsweise dokumentieren.

Fazit

Durch die eigenständige Protokollierung und Verwaltung, welche Datenbank-Änderungen im Rahmen einer Auslieferung bereits gelaufen sind und welche noch laufen müssen, ist Liquibase ein sehr nützliches Tool in agilen Projekten, bei denen häufige Auslieferungen stattfinden. Gerade in Entwicklungs- und Test-Umgebungen 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 allerdings gewisse Einschränkungen; zum Beispiel ist ein „DROP COLUMN“ nicht einfach durch ein „ADD COLUMN“ rückgängig zu machen. Im Oracle-Umfeld sollte man daher zusätzlich über die Verwendung sogenannter „Restore Points“ nachdenken.

Frank Winter
frank.winter@orbit.de