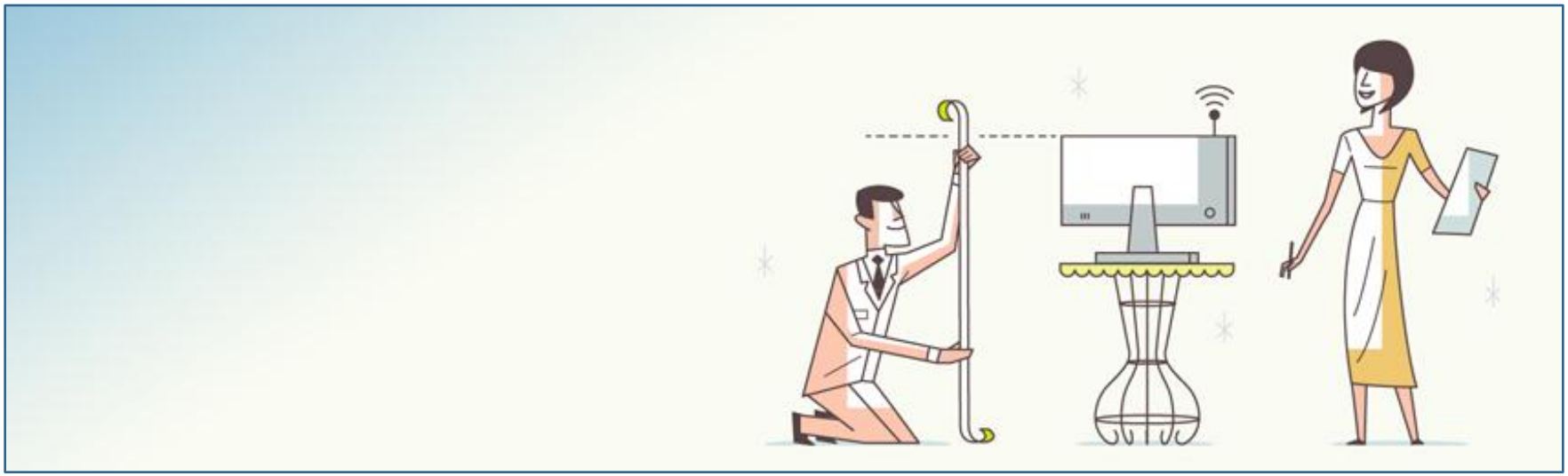
LIQUIBASE



- Database Deployment in agilen Umgebungen

DAS ORBIT ORACLE-PORTFOLIO

ORACLE BUSINESS INTELLIGENCE

ORACLE BI Enterprise Edition

ORACLE Application Express

ORACLE APEX

DWH/OLAP

ORACLE WAREHOUSE BUILDER

ORACLE DATA INTEGRATOR

PL/SQL

Oracle 11g Administration

OLTP

11g ORACLE DATABASE

ORACLE Datenbank

LICENSING

ORACLE

ORACLE Gold Partner

Specialized
Oracle Business Intelligence Foundation

ORACLE

SOLUTION PARTNER COMMUNITY

BUSINESS INTELLIGENCE

EPM – BI – DWH

Database Appliance

T-Series

M-Series

Sun ORACLE

ORACLE Systems Familie

Storage-, Backup- und Virtualisierungs-lösungen

PROBLEME BEIM DEPLOYMENT VON DB-ÄNDERUNGEN

- Es sollen mit jeder Auslieferung nur Änderungen zu dem jeweils letzten Release eingespielt werden. Separate Installationsskripte sind bei kurzen Auslieferungszyklen umständlich und fehleranfällig.
- DDL-Statements und committete DML-Statements lassen sich nicht ohne weiteres rückgängig machen. Wie geht man z.B. mit einem umfangreichen Installationsskript um, das mittendrin abbricht?
- Datenbankänderungen werden oft gänzlich unabhängig von Code-Änderungen eingespielt. Das ist aber gefährlich. Code und Datenbankänderungen gehören zusammen in ein Deployment.
- Entwicklern steht nicht immer eine Oracle-Datenbank zur Verfügung.

WAS IST LIQUIBASE?

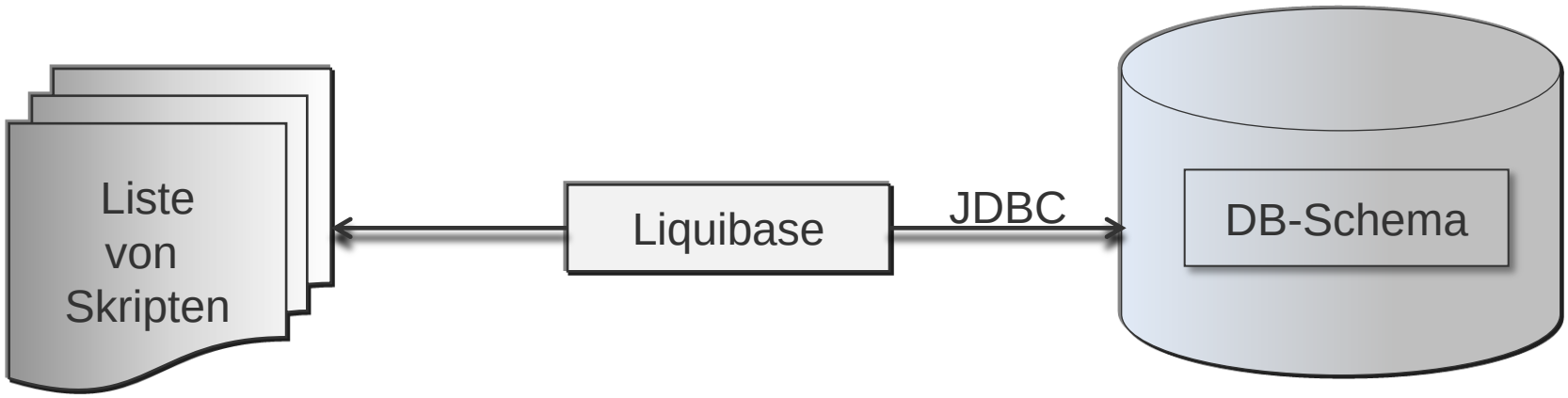
- Liquibase ist ein Database Change Management Tool und dient primär der Durchführung und Verwaltung aller Arten von Datenbankänderungen (DML und DDL).
- Änderungen lassen sich einfach verwalten und bei Bedarf wieder rückgängig machen.
- Liquibase ist Open Source und benötigt nur eine Java-Laufzeitumgebung (JDK/JRE) ab Version 1.6.
- Liquibase unterstützt alle gängigen Datenbanken.
- Liquibase eignet sich hervorragend für den Einsatz in agilen Projekten, da es sehr gut in einer Umgebung mit Continuous Integration und Continuous Deployment einsetzbar ist.

INSTALLATION VON LIQUIBASE

- Die Installation ist denkbar simpel. Download von http://www.liquibase.org/download/ und einfach entpacken.
- RPMs sind ebenfalls verfügbar.
- Liquibase ist über zahlreiche Plugins erweiterbar (z.B. für die Arbeit mit Maven oder Ant).
- Für einige spezifische Oracle-Statements verwenden wir liquibase-oracle-1.2.0.jar. Ist aber nicht zwingend notwendig.

FUNKTIONSWEISE VON LIQUIBASE

- Liquibase baut über JDBC eine Verbindung zu einem bestimmten DB-Schema auf und führt (im Normalfall) alle Skripte, die bislang nicht gelaufen sind, gegen dieses Schema aus.



- In dem DB-Schema liegt die Tabelle DATABASECHANGELOG, die vermerkt, welche Änderungen bislang erfolgt sind.

FUNKTIONSWEISE VON LIQUIBASE

- Beispielinhalte der Tabelle DATABASECHANGELOG (Auszug):

ID	AUTHOR	FILENAME	DATEEXECUTED	EXECTYPE	MD5SUM	TAG
1	fwinter	pfad/datei1.xml	22.07.2013	RERAN	3:3fc40722eca263fcc651d4917d981bb7	
2	fwinter	pfad/datei2.xml	23.07.2013	EXECUTED	3:811a419bffccd6f0051e48182efb18bc	
3	fwinter	pfad/datei3.xml	24.07.2013	EXECUTED	3:4f594b0b41217d9ade86c0e1e7f7cfc0	
4	fwinter	pfad/datei4.xml	25.07.2013	EXECUTED	3:ee32ff5292b3233cfb1d2c754149c0fb	
5	fwinter	pfad/datei5.xml	26.07.2013	EXECUTED	3:eb2677186482441238810fdd4eac4d6	
6	fwinter	pfad/datei6.xml	27.07.2013	EXECUTED	3:d0c666c1a80745eb35b1e19e67a197b1	
7	fwinter	pfad/datei7.xml	28.07.2013	EXECUTED	3:7909a892599b5ff6060482247d8c64c4	

- Liquibase merkt sich auf diese Weise, wann welches Skript gelaufen ist und legt zu dem Skript einen MD5-Hash (nach Variablenersetzung!) an.

- Ändert sich ein Skript, nachdem es gelaufen ist, führt dies zu einem Fehler (außer im Skript ist runOnChange="true" definiert).

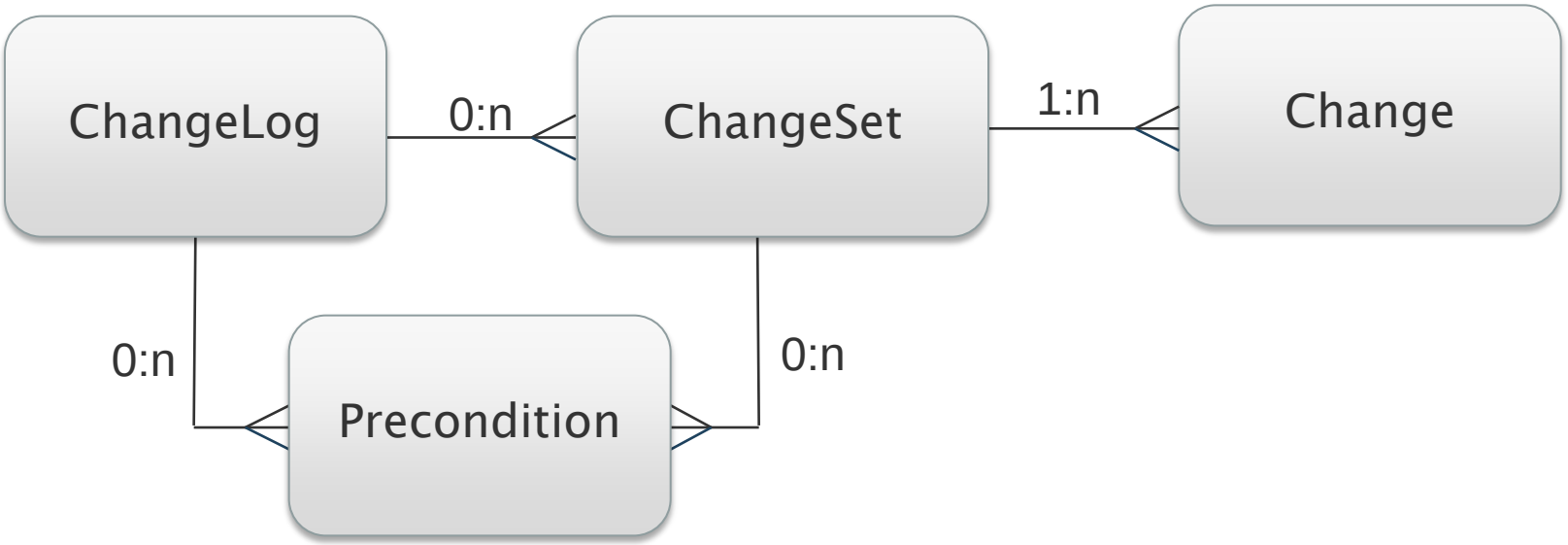
SPALTEN DER TABELLE DATABASECHANGELOG

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
ORDER-EXECUTED	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)		Beschreibung des Entwicklers zu dem ChangeSet
TAG	VARCHAR2(255)		optionaler Tag z.B. für ein bestimmtes Release; kann im Rollback verwendet werden
LIQUIBASE	VARCHAR2(20)		verwendete Liquibase-Version

AUFBAU EINES LIQUIBASE-SKRIPTES

- Es gibt verschiedene Syntaxvarianten für Liquibase-Skripte: XML, YAML, JSON und sogar SQL. Meist werden Skripte im XML-Format verwendet.
- Ein Liquibase-Skript besteht üblicherweise aus folgenden Bestandteilen:
  - **ChangeLog**: Die Skriptdatei, die Änderungen enthält.
  - **ChangeSet**: Ein Satz logisch zusammenhängender Statements. Change Sets werden in die Tabelle DATABASECHANGELOG eingetragen (eindeutig über ID, AUTHOR, FILENAME). Ein Changelog File kann mehrere Change Sets beinhalten. Ein Change Sets kann seinerseits aus mehreren Changes bestehen.
  - **Preconditions**: optionale Vorbedingungen. Gelten für das gesamte Changelog File oder einzelne Change Sets.

AUFBAU EINES LIQUIBASE-SKRIPTES



AUFBAU EINES LIQUIBASE-SKRIPTES

➔ **Beispiel 1** (ChangeLogFile, Liquibase-spezifische Syntax)

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
xmlns="
xmlns:xsi="
xmlns:ora="
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-2.0.xsd
http://www.liquibase.org/xml/ns/dbchangelog-ext
   logicalFilePath="3.15.0/01_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>
```

AUFBAU EINES LIQUIBASE-SKRIPTES

➔ **Beispiel 2** (hier nur ChangeSet, Verwendung des SQL-Tags)

```
…
<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>
…
```

AUFBAU EINES LIQUIBASE-SKRIPTES

- Bei der Verwendung von SQL-Tags (<sql>) reicht Liquibase die enthaltenen SQL-Statements durch und kann daher automatisch kein Rollback der Änderung generieren.
- Das Rollback ist nicht Bestandteil einer Fehlerbehandlung und der Abbruch eines Skriptes bewirkt nicht das Ausführen des Rollbackteils! Ein Rollback dient dem nachträglichen Zurückrollen aller Änderungen eines ChangeSets, um das Datenbankschema wieder auf einen älteren Stand zurückzusetzen.
- Die Precondition prüft in dem obigen Beispiel, ob es einen bestimmten Datensatz schon gibt.

BEISPIELE PRECONDITIONS

➔ Siehe auch:
http://www.liquibase.org/documentation/preconditions.html

```
<preConditions onFail="MARK_RAN" onFailMessage="tablespace MY_TBS already exists">
    <sqlCheck expectedResult="0">
        SELECT  count(*)
        FROM    DBA_TABLESPACES v
        where   v.tablespace_name = 'MY_TBS'
    </sqlCheck>
</preConditions>


<preConditions onFailMessage="user ${db_user} already exists">
    <sqlCheck expectedResult="0">
        SELECT COUNT(*)
        FROM all_users
        WHERE username = UPPER('${db_user}');
    </sqlCheck>
</preConditions>

<preConditions>
   <sqlCheck expectedResult="1">
       SELECT COUNT (*) FROM schema_version WHERE version = 12345
   </sqlCheck>
</preConditions>
```

BEISPIELE PRECONDITIONS

```
<preConditions>
   <columnExists tableName="MY_TABLE" columnName="type" />
   <columnExists tableName="MY_TABLE" columnName="cr_date" />
</preConditions>

<preConditions onFail="HALT">
    <changeSetExecuted id="4" author="fwinter" changeLogFile="pfad1/bla.xml"/>
    <changeSetExecuted id="6" author="fwinter" changeLogFile="pfad1/bla2.xml"/>
</preConditions>

<preConditions onFail="HALT" onFailMessage=„my message">
     <viewExists schemaName="${db_user}" viewName="VW_MY_VIEW" />
     <and>
        <sqlCheck expectedResult="1">
           SELECT COUNT(*) FROM DBA_USERS WHERE USERNAME = UPPER('${db_user}')
        </sqlCheck>
     </and>
</preConditions>

<preConditions onFail="MARK_RAN">
    <not>
       <tableExists tableName="MY_TABLE" />
    </not>
</preConditions>
```

BEISPIELE LIQUIBASE-SYNTAX FÜR CHANGES

- http://www.liquibase.org/documentation/changes/index.html
  (linke Spalte unter „Bundled Changes“)
- Es lohnt sich diese Seite mal in Ruhe anzuschauen.

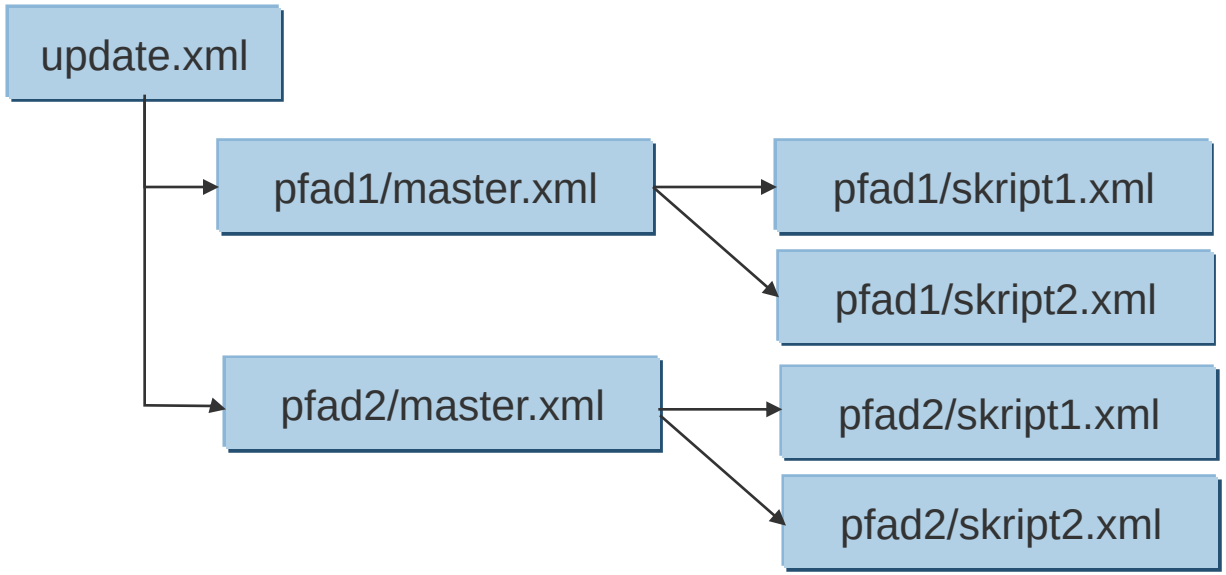
WIE WERDEN LIQUIBASE-SKRIPTE AUFGERUFEN?

- Liquibase-Skripte lassen sich beliebig kaskadierend aufrufen.

- Aufruf von Skripten aus einem anderen Skript:

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

- Mögliche Struktur für einen kaskadierenden Aufruf:



WIE WERDEN LIQUIBASE-SKRIPTE AUFGERUFEN?

- 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.

```
#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
```

- Diese Informationen lassen sich auch als Parameter an Liquibase übergeben.
- Im einfachsten Fall kann Liquibase nun über die Kommandozeile wie folgt aufgerufen werden:

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

LIQUIBASE KOMMANDOS

Kommando	Beschreibung
update	Führt eine Aktualisierung des Datenbankschemas durch.
updateSQL	Schreibt die SQL-Statements zur Aktualisierung des Datenbankschemas nach 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 + Rollback im Rahmen der Entwicklung.
rollback <tag>	Führt ein Rollback aller Changes durch, die neuer sind als das ChangeSet mit einem bestimmten Tag (siehe Attribut Tag in Tabelle DATABASECHANGELOG)
rollbackToDate <date/time>	Führt ein Rollback aller Changes nach einem bestimmten Zeitpunkt durch.
rollbackCount <x>	Führt ein Rollback der letzten x ChangeSets durch.
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, das die Differenzen zwischen zwei Datenbankschemen ausgleicht.
generateChangeLog	Generierung eines initialen ChangeLogs aus einem bereits mit Datenbankobjekten gefüllten Datenbankschema.

LIQUIBASE VARIABLEN

- Solche Variablen werden meist in einer zentralen Datei definiert (z.B. properties.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.

```
…
<sql splitStatements="true">
     CREATE USER ${db_user} IDENTIFIED BY "${db_pw}";
     GRANT CONNECT TO ${db_user};
     GRANT RESOURCE TO ${db_user};
</sql>
…
```

LIQUIBASE ALS DBA-USER AUSFÜHREN

- Liquibase-Skripte sind immer an ein bestimmtes Schema gebunden; meist das zentrale Schema für eine bestimmte Komponente.
- Es gibt aber gelegentlich den Bedarf, Skripte als User mit DBA-Rechten auszuführen (neue User anlegen, Rechte an weitere Schemen (z.B. APP-User) vergeben, Synonyme in anderen Schemen anlegen, Directories erzeugen etc.).
- Liquibase-Skripte, die als DBA auszuführen sind, müssen teilweise vor und teilweise nach dem Hauptskript (--> update.xml) für die jeweilige Komponente ausgeführt werden.
- Aus diesem Grunde sollte es bei Bedarf für den DBA eine update_dba_pre.xml und update_dba_post.xml geben.

Ein praktisches Beispiel ....

VIELEN DANK

- für Ihre Aufmerksamkeit

➔ Falls Sie Fragen zu dieser Präsentation haben, sprechen Sie uns einfach an.

➔ Ihr Ansprechpartner

Frank Winter

+49 228-95693-748

Frank.winter@orbit.de

www.orbit.de