Nah am Livesystem und doch individuell – Aufbau von Testumgebungen

Alexander Pilfusek
Josef Witt GmbH
Weiden i. d. OPf.

Schlüsselworte
Test, Entwicklung, Testdatenbank, Entwicklungsdatenbank, CI, continuous integration, TTS, transportable tablespace

Einleitung
Ohne eigene Umgebungen für Entwicklung, Test oder CI-Läufe kommt kein Softwareentwicklungsprozess mehr aus. War vor einigen Jahren eine einzelne, persistente Datenbank als „Softwareentwicklungs-DB“ noch ausreichend, kam bei der Josef Witt GmbH eine Cluster-Integrationsdatenbank hinzu. In den folgenden Jahren wurden aus einer Entwicklungs- bzw. Testdatenbank sieben.
Die Herausforderung besteht darin, den Aufbau der verschiedenen Test- u. Integrationsdatenbanken zu standardisieren. Des Weiteren soll die Datenbank zwar sehr nah am Livesystem sein, allerdings werden verschiedene Modifikationen benötigt. So sollen manche Daten persistent sein, Änderungen am Datenmodell durchgeführt werden oder PL/SQL-Programme in einer neueren Version übergeben werden. Auch Patches will man in den Testumgebungen vor dem Praxiseinsatz testen. Prinzipiell passiert ein solcher Aufbau in drei Schritten:

1. Vorbereitung: Export bestimmter Objekte
2. Klonen: Erzeugung des Snaps, Starten der Datenbank
3. Nachbereitung: Import von Objekten, Frameworkinstallation, Datenmodell- u. PL/SQL-Übergaben

Der Vortrag geht auf die einzelnen Tätigkeiten innerhalb dieser Schritte ein und zeigt Probleme, Lösungen und Möglichkeiten auf.
Der Systemaufbau besteht aus zwei VMAXe als Storagesystem, die Datenbanken (Live-RAC, Integrations-RAC sowie verschiedene Test-Instanzen) laufen auf zwei IBM-p770 in verschiedenen LPars (logische Partitionen). Die Testinstanzen befinden sich auf einem Server und benutzen ein gemeinsames Oracle-Home. Alle Instanzen verwenden ASM.

Überblick
Der Aufbau einer Testdatenbank ist in mehrere Schritte aufgeteilt, diese werden bei der Josef Witt GmbH in einem eigenentwickelten Jobsystem ausgeführt. Jeder Job kann harte Abhängigkeiten, welche zum Stop der gesamten Kette führen, oder weiche haben. Hierbei läuft der Nachfolger läuft trotzdem. Auch eine Deaktivierung einzelner Schritte ist möglich. Die Jobs starten Shellskripte in der vorhandenen Umgebung, im Fall der Testdatenbanken auf AIX (Unix).
Die Metainformationen für die einzelnen Datenbanken werden im Livesystem in einem eigenen Datenmodell gehalten. Hier sind die zu behaltenden Daten, die auszuführenden DDLs und die zugehörigen Rückmeldungen sowie die durchzuführenden PL/SQL-Übergaben gespeichert. Durch den Klon der Livedatenbank sind die aktuellen Metadaten auch immer auf der Testdatenbank vorhanden.

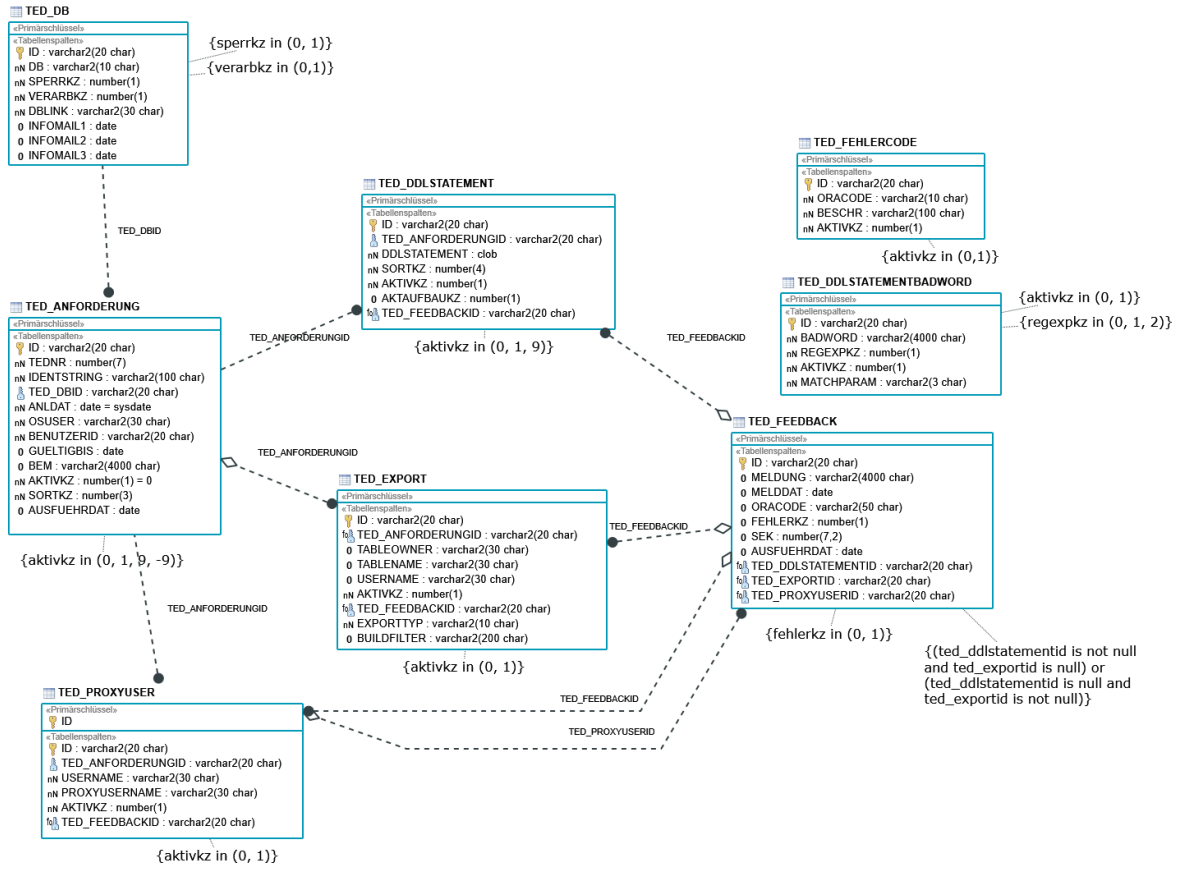

Abb. 1: Datenmodell für einen Testdatenbank-Aufbau

Mit Ausnahme des Klonens der Livedatenbank wird alles mit Oracle-Bordmitteln (Datapump, SQL, PL/SQL) realisiert. TED ist übrigens der interne Produktname. Er bedeutet „**Te**st-**D**atenbanken(-Aufbau)“.

Vorbereitung

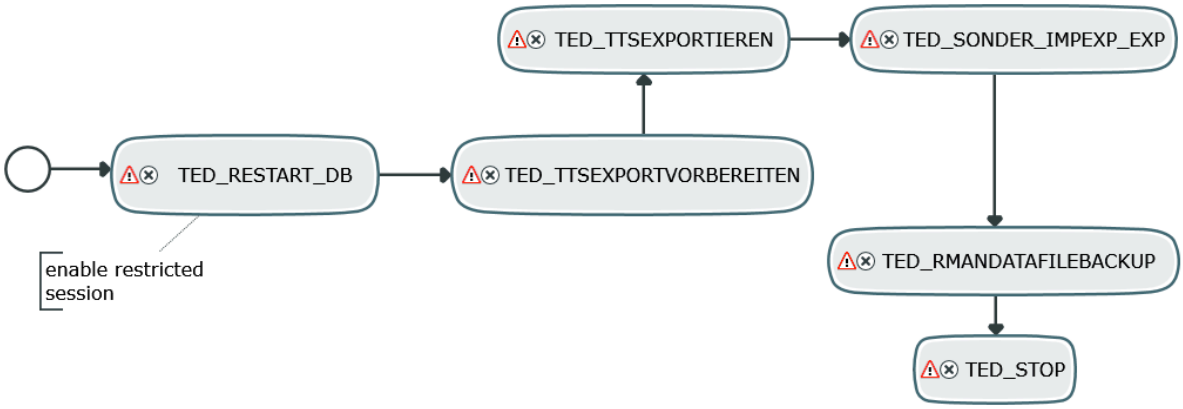

Abb. 2: Jobs bei der Vorbereitung

Aller Anfang ist schwer. Nicht beim Aufbau der Testdatenbanken. Der erste Job kümmert sich um das „Sperren“ der Datenbank. Hierbei wird die Datenbank „immediate“ heruntergefahren (per srvctl) und danach im „restricted mode“ wieder gestartet. Zum einen sind dadurch keine Benutzer mehr mit der

Datenbank verbunden, zum anderen werden keine neuen Verbindungen mehr zugelassen. Auf diese Weise vermeidet man Lock-Situationen (z. B. durch langlaufende Transaktionen).

```
srvctl stop database –d ${dbid_lower} >> $protokoll_datei
sleep 20
srvctl start database –d ${dbid_lower} –o restrict >> $protokoll_datei
# prüfen, ob die DB läuft und damit dieser Job erfolgreich ist..
```

Anschließend geht es daran, die zu erhaltenden Daten zu sichern. Vom Aufbau können folgende Objekte ausgenommen sein:

- Tabellen
- Tabellenstatistiken
- Schemata

Dazu werden die entsprechenden Objekte auf einen transportablen Tablespace verschoben. Hierbei sind einige Punkte zu beachten. Der wichtigste ist, dass Objekte, die per TTS verschoben werden, „self-contained“ sein müssen. Dies bedeutet, dass keine Abhängigkeiten wie z. B. Constraints aus dem bzw. in den Tablespace zeigen dürfen. Solche Contraints werden vor dem Verschieben der Objekte auf den TTS gedroppt. Nach dem Verschieben auf den TTS werden alle diejenigen Objekte, die nicht mehr behalten werden sollen, wieder vom TTS weg auf einen anderen, festgelegten Tablespace verschoben. Zu sichernde Statistiken werden mittels `DBMS_STATS.export_table_stats` exportiert, wobei die „Stats-Table“ bereits im TTS liegt. Per `DBMS_TTS.transport_set_check` wird letztlich geprüft, ob der TTS alle Anforderungen zum Transport erfüllt. Danach wird er auf READ ONLY gesetzt.
Der eigentliche Export ist recht einfach. Es ist ein simpler Datapump-Befehl:

```
expdp dumpfile=${dumpdatei} \
directory=DATEN_DIR \
transport_tablespaces=tts \
logfile=${se_log} << EOT
$WWDBA_PASS
EOT
```

Der Parameter dumpfile gibt die Datei an, in dem die Metainformationen abgelegt werden. Das Directory, das übergeben wird, muss ein Datenbank-Directory-Objekt sein (z. B.: `create directory DATEN_DIR as '/shared/dpump'`), der Parameter transport_tablespaces gibt den oder die transportablen Tablespace(s) an (mehrere werden mit Komma getrennt angegeben). Außerdem wird noch der Name der Logdatei spezifiziert.
Beim „Sonder-Export“ werden die Schemata und spezielle Objekte gesichert. Dazu gehören unter anderem Sequences, die nicht über Datapump gesichert werden können. Dies erfolgt über das Auslesen der Eigenschaften aus dem Data Dictionary in eine Spooldatei, die später ausgeführt wird. Auch User, die zwar Zugriff auf das Test-, nicht aber das Livesystem haben, werden hier mit ihrem Passworthash und Status abgelegt. Die auszunehmenden Schemata werden mittels Datapump exportiert. Dazu werden ebenfalls SQL-Dateien zum Droppen der User und zur Wiederherstellung der Grants erzeugt. Nach allen Exports wird ein RMAN-Backup der transportablen Tablespaces und des Controlfiles durchgeführt. Sie werden mit einem `backup as copy` bzw. `backup current controlfile` ausgelagert. Den Abschluss der Vorbereitung bildet der Stop der Testdatenbank via `srvctl stop database`. Anschließend werden alle ASM-Diskgruppen ausgehängt. Im Fall eines RACs muss dies auf allen Knoten geschehen.

Klonen

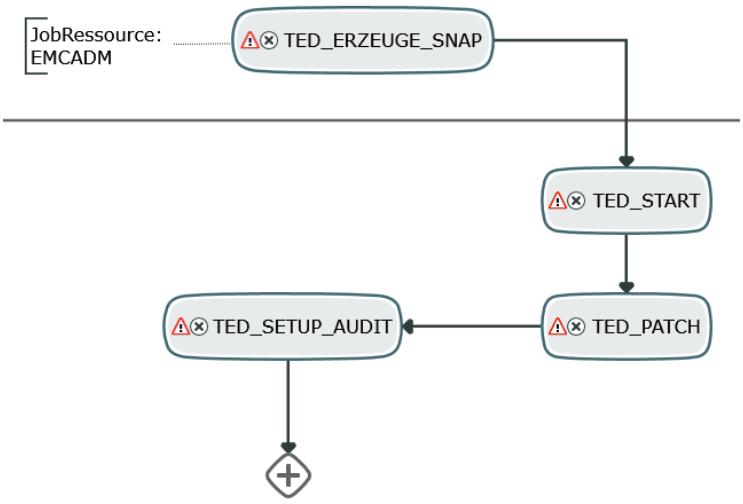

Abb. 3: Jobkette während des Klonens

Es folgt der Klon bzw. Snap des Livesystems. Welche Technik hier zum Einsatz kommt, ist unterschiedlich und für den ganzheitlichen Ablauf irrelevant. Wichtig ist nur, dass nach dem Job „TED_START“ eine Datenbank (im restricted-mode) gestartet ist, die ein Abbild des Livesystems ist.
Bei der Josef Witt GmbH wird als Storage eine VMAXe von emc verwendet. Hier kommt zum Klonen der Datenbank die Technologie „storage-space-efficent-clone“ zum Einsatz. Beim Starten der geklonten Datenbank wird zunächst ASM gestartet. Da die Datenbank den Namen der Livedatenbank besitzt, wird sie umbenannt:

```
nid target=$SYS_PASS dbname=${dbid_lower} << XXX >> ${protokoll_datei}
Y
XXX
```

Die Datenbank wird nun unter dem Namen der Testdatenbank gestartet. Dabei werden auch vorhandene Restorepoint gelöscht, ArchiveLog-Modus und Flashback ausgeschaltet.
Wird beispielsweise ein neuer PSU getestet, kommt der Job „TED_PATCH“ zum Einsatz. Die Testdatenbanken befinden sich auf einem dedizierten Server. Dies schafft die Möglichkeit, das Oracle-Home persistent zu patchen. Offene Tasks wie z. B. Post-Installation-Skripte werden dann im Patch-Job ausgeführt werden.
Wenn z. B. ein neuer PSU getestet werden soll, muss in der Regel einen Post-Installationstask durchführt werden. Das Oracle-Home bleibt in diesem Falle gepatcht (es befindet sich auf einem anderen Server und ist persistent). Die Durchführung des Post-Installationstask führt der Patch-Job durch:

```
# Bugfix-Postinatallation durchführen (im RAC genügt ein Knoten)
sqlplus / as sysdba << EOT >> $protokoll_datei_sqlplus
  @?/rdbms/admin/catbundle.sql psu apply
  exit;
EOT
```

Der Job „TED_SETUP_AUDIT“ aktiviert eine Witt-spezifische Auditumgebung. Er kann zum Beispiel genutzt werden, um Änderungen am implementierten Auditing zu testen. Weitergehend wird durch diesen Job die Überwachung der essentiellen Daten aktiviert. An dieser Stelle kann z. B. auch Data-Masking realisiert werden.

Nachbereitung

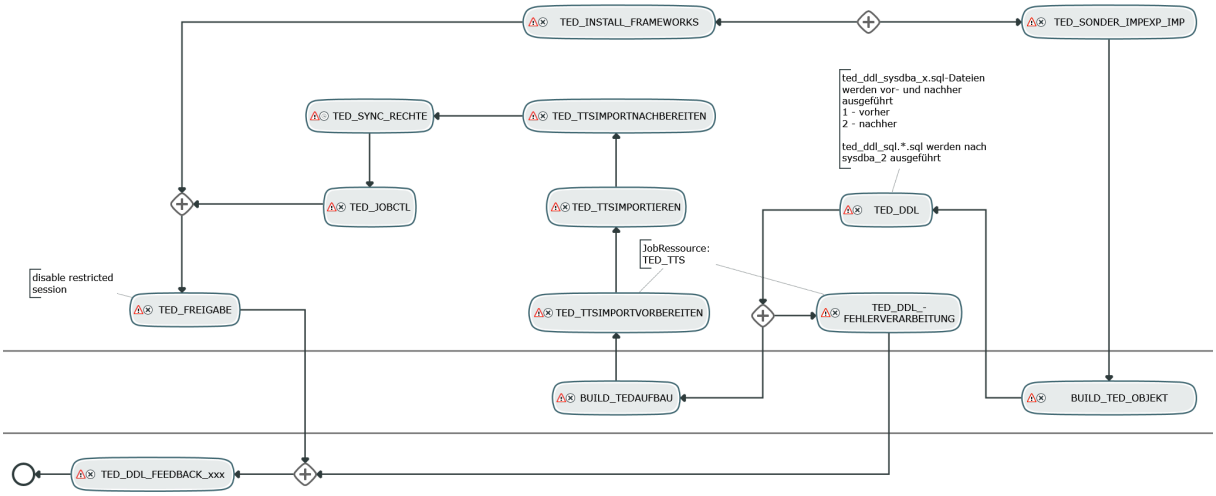

Abb. 4: Die Tätigkeiten der Nachbereitung

Nach dem Klonen der Livedatenbank steht die Individualisierung der Testdatenbank auf dem Programm. Diese kann mehr oder weniger stark betrieben werden. Parallel starten die Installation von Frameworks und der Sonder-Import. Beispiele für die Frameworkinstallation sind PL/JSON oder APEX. In der Livedatenbank sind diese aktuell noch nicht verfügbar. Aufgerufen werden eigene oder vom Hersteller zur Verfügung gestellte Installationsroutinen:

```
# PL/JSON installieren
echo `date +%H:%M:%S`" Start Install PLJSON" >> $protokoll_datei
/pljson/install.sh "/pljson/json_v1_0_4" pljson Js0nPwd >> $protokoll_datei
```

Der Sonder-Import fügt die mittels Datapump erzeugten Dumps vom Sonder-Export in die Testdatenbank ein und führt auch die erstellten Spooldateien mit SQL-Anweisungen aus (User löschen bzw. ändern, Objektgrants, etc.). Die Objekte, die von der „alten“ Testdatenbank übernommen werden sollten, sind nun also wieder vorhanden.
Im nächsten Schritt werden PL/SQL-Objekte übergeben, die den Aufbau der Testdatenbank betreffen. Da beim Job „TED_DDL“ bereits wieder PL/SQL-Packages verwendet werden und die PL/SQL-Übergaben erst später durch „BUILD_TEDAUFBAU“ auf die neue Datenbank kommen, könnten Neuerungen in den TED-Packages ohne diesen Schritt nicht bei einzelnen Aufbauten getestet werden. Alternative wäre eine Übergabe ins Live-System, wodurch diese Änderung allerdings sofort für alle Aufbauten gelten würde.
Nun werden Datenmodelländerungen für die jeweilige Testdatenbank ausgeführt. Diese sind in den TED-Tabellen gespeichert und stellen für die Softwareentwicklung mit den wichtigsten Punkt dar. Bei der Josef Witt GmbH findet die Software Innovator bei der ER-Modellierung Verwendung. Hieraus werden die SQL-Befehle mittels eines PL/SQL-Packages direkt in die TED-Tabellen geschrieben. Die Anweisungen werden mit `execute immediate` ausgeführt, mögliche Fehler werden auch in Tabellen zurückgeschrieben. Die nachgelagerte Fehlerverarbeitung analysiert diese und klassifiziert sie in „gute“ und „wirkliche“ Fehler. Bei „guten“ Fehlern archiviert der Job die entspreche SQL-Anweisung, damit wird sie beim nächsten Aufbau nicht mehr ausgeführt. Ein Beispiel für eine Deaktivierung ist ein „`ORA-00955 name is already used by an existing object`“ bei einem `create table`. Es wird davon ausgegangen, dass sich die ehemals neue Tabelle nun auf dem Livesystem befindet und damit die Tabelle nicht mehr neu angelegt werden muss.
Im nächsten Schritt werden neue bzw. geänderte PL/SQL-Objekte (Packages, Prozeduren, etc.)

übergeben. Bei der Josef Witt GmbH gibt es eine eigens entwickelte Webanwendung zur Übergabe von PL/SQL. Damit können die Entwickler – nachdem der Code im SubVersion-Repository eingecheckt wurde – die Objekte selbst auf die Testdatenbanken bringen. Nur eine Übergabe in die Livedatenbank bedingt eine Aktion des DBAs. Bei einem Testdatenbank-Aufbau werden die Sourcen direkt aus dem SVN-Repository gelesen und auf die Testdatenbank gebracht.
Für das Importieren des TTS sind wieder Vorbereitungen nötig. Da auch im Livesystem vorhandene Tabellen ausgenommen werden können, kann es beim Import zu Namenskonflikten kommen. Diese werden durch Umbenennen der „Live"-Tabellen inkl. deren Indexes, Constraints und Trigger verhindert. Der neue Name hat im Gegensatz zum Löschen den Vorteil, dass die Tabelle vom Livesystem inhaltlich noch vorhanden ist. Eventuelle Klärungen oder Vergleiche fallen somit einfacher aus. Per Datapump-Befehl `impdp` kann nun der vorher exportierte transportable Tablespace wieder an die Testdatenbank gehängt werden. Durch das Nachbereiten des Imports wird der Tablespace READ WRITE gesetzt. Außerdem werden die vorher exportierten Tabellenstatistiken ins Data Dictionary geschrieben sowie die Constraints, die aus dem TTS hinaus zeigen sollen, wieder angelegt.
Die nächsten beiden Jobs sind sehr Witt-spezifisch. Rechte werden prinzipiell über Rollen vergeben. Da die PL/SQL-Entwicklung direkt zugewiesene Rechte bedingt (Rechte, die man über eine Rolle erhält, wirken hier nicht), werden diese direkt an die PL/SQL-Entwickler vergeben. Diesen ist eine bestimmte Rolle (ROLE_PLSQLENTWICKLER) zugewiesen. Anhand dieser wird über andere Rollen iteriert und deren Rechte werden dem einzelnen User direkt zugewiesen. Damit ist eine PL/SQL-Entwicklung im eigenen Schema möglich. „TED_JOBCTL" verändert das Witt-eigene Jobsystem auf den Testdatenbanken. Damit ist es möglich, dass abhängig vom Aufbauzeitpunkt bestimmte Jobs nicht oder anders laufen.
Nun kann die Datenbank für die Benutzung freigegeben, also auf „unrestricted" gesetzt, werden. Davor werden die invaliden Datenbankobjekte mehrere Male kompiliert, damit möglichst viele gültig sind. Dies ist meist durch die Datenmodelländerungen und Programmübergaben nötig. Außerdem wird noch ein AWR-Snapshot erzeugt:

```
DBMS_WORKLOAD_REPOSITORY.create_snapshot
```

Nachdem die Datenbank verfügbar ist, wird das entstandene DDL-Feedback, d. h. die Fehler der Datenmodelländerungen, auf die Livedatenbank übertragen. Da diese auf dem Testsystem gespeichert werden, wären sie ansonsten beim nächsten Aufbau der Test-DB verloren.
Ein kompletter Aufbau mit wenig Individualisierung dauert auf den genannten Systemen ca. zwei Stunden. Hierbei fallen ca. 80 Minuten auf den Job „TED_START". Die meiste Zeit wird beim Öffnen der Datenbank mit RESETLOGS benötigt.

Kontaktadresse:
Alexander Pilfusek
Josef Witt GmbH
Schillerstraße 4-12
D-92637 Weiden i. d. OPf.

Telefon:	+49 (0) 961-400 1099
Fax:	+49 (0) 961-400 1920
E-Mail	alexander.pilfusek@witt-gruppe.eu
Internet:	www.witt-gruppe.eu