Aufbau verschiedener Testsysteme von einem Livesystem

DOAG Datenbank Day 2016
16.03.16

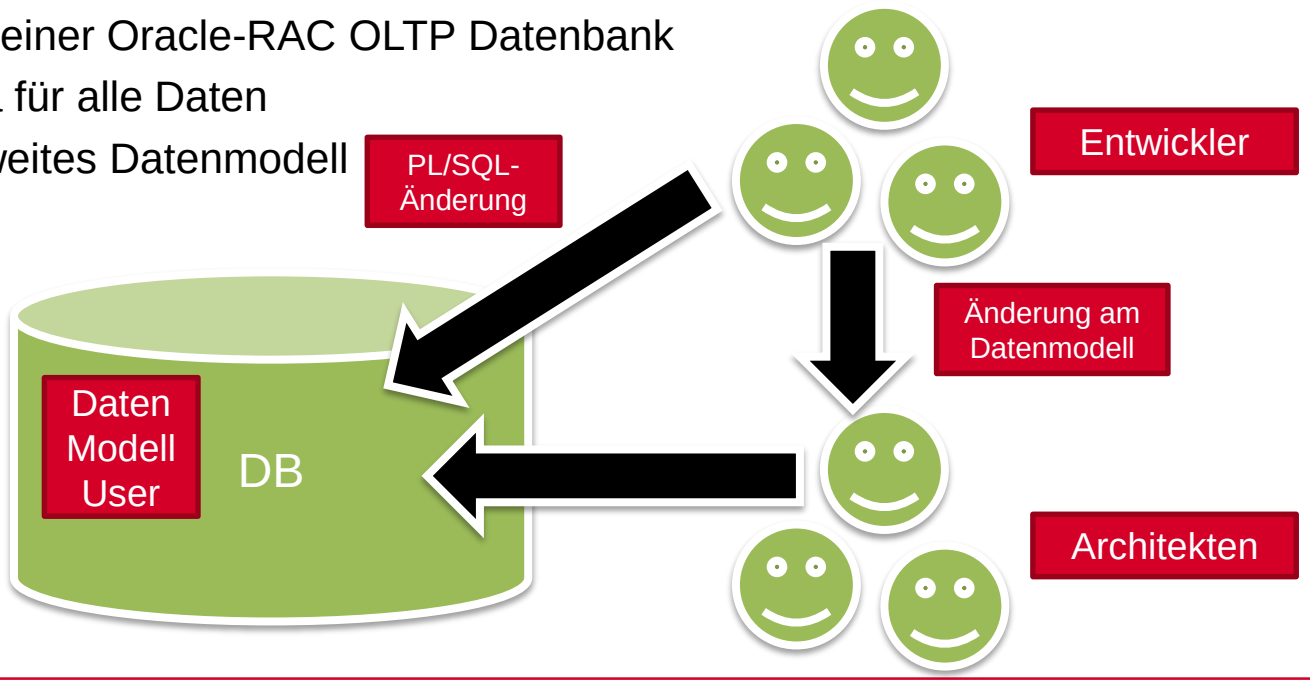
Alexander Pilfusek
IB-BS

Software-Entwicklung @ Witt

Josef Witt GmbH

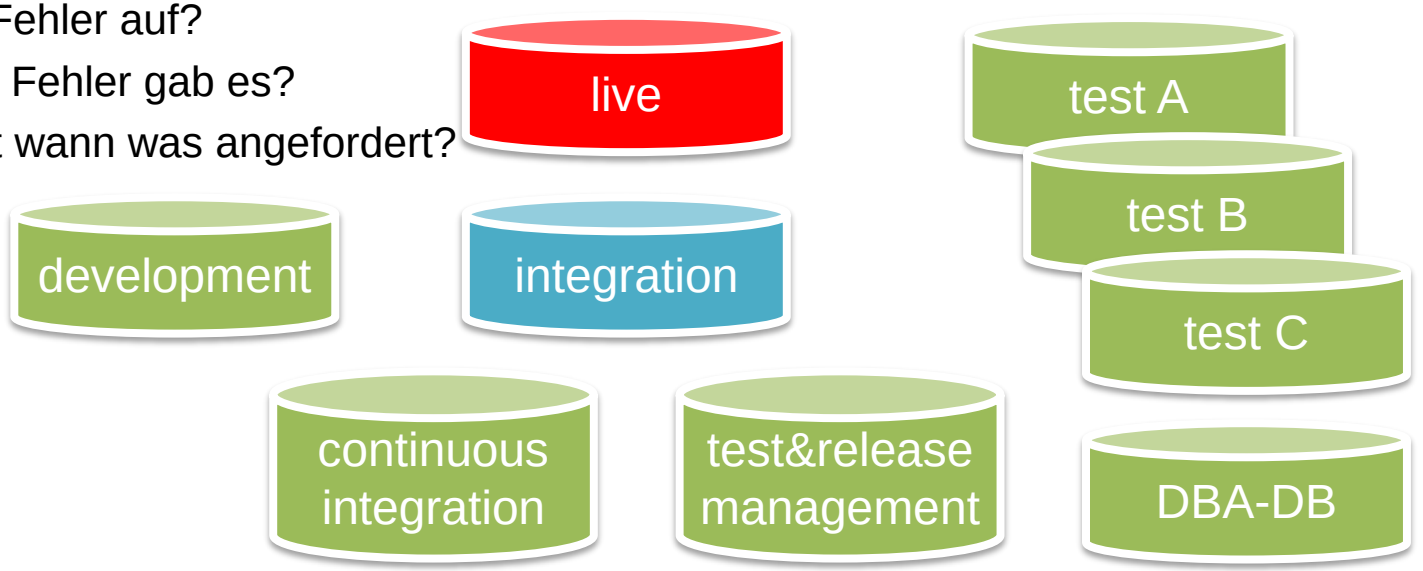
- Individual-Software wird durch unsere Softwareentwicklung entwickelt
- über 100 Software Produkte
- über 50 Entwickler (PL/SQL, Java, PowerBuilder, Web, ..)

- Daten liegen in einer Oracle-RAC OLTP Datenbank
- ein DB-Schema für alle Daten
- unternehmensweites Datenmodell



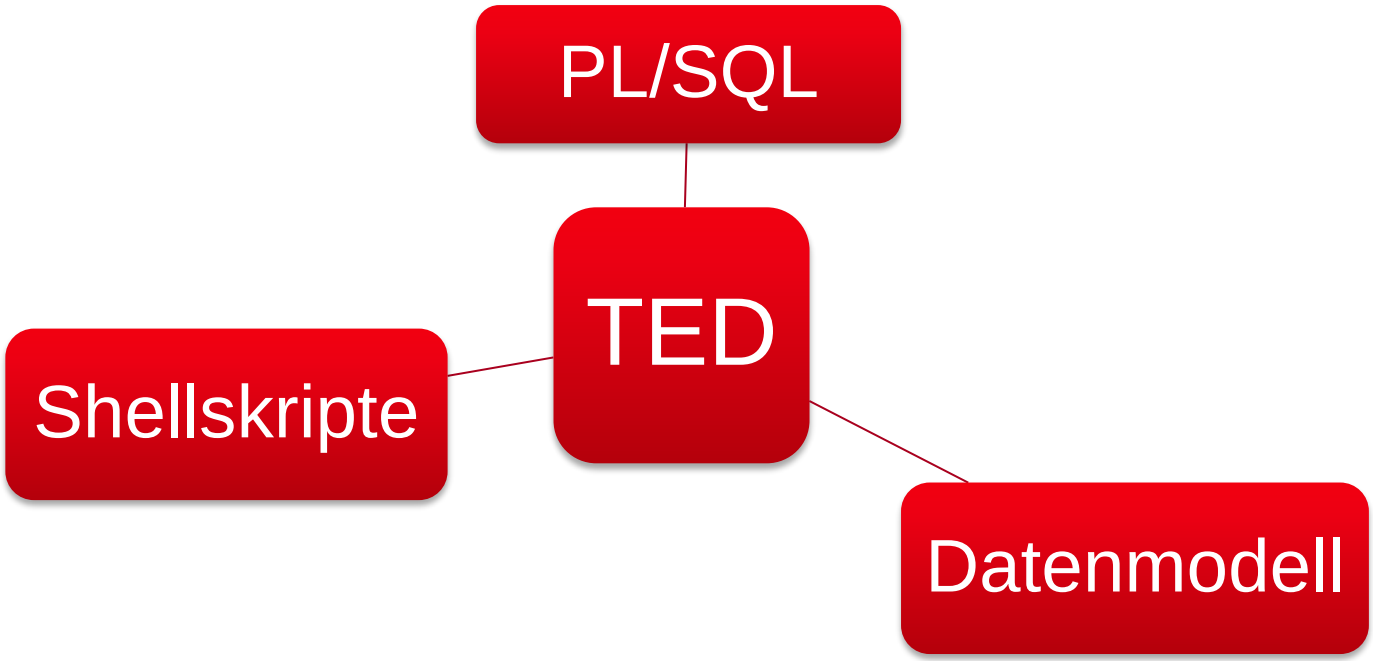
Software-Entwicklung @ Witt

- sieben Datenbanken für Entwicklung & Test
- eine RAC-Datenbank für Integration
- alle Snaps/Clones sind vom Live-System
- anfangs gab es nur eine persistente Datenbank und ein SQL-File für die DDLs
- Probleme:
  - Traten Fehler auf?
  - Was für Fehler gab es?
  - Wer hat wann was angefordert?

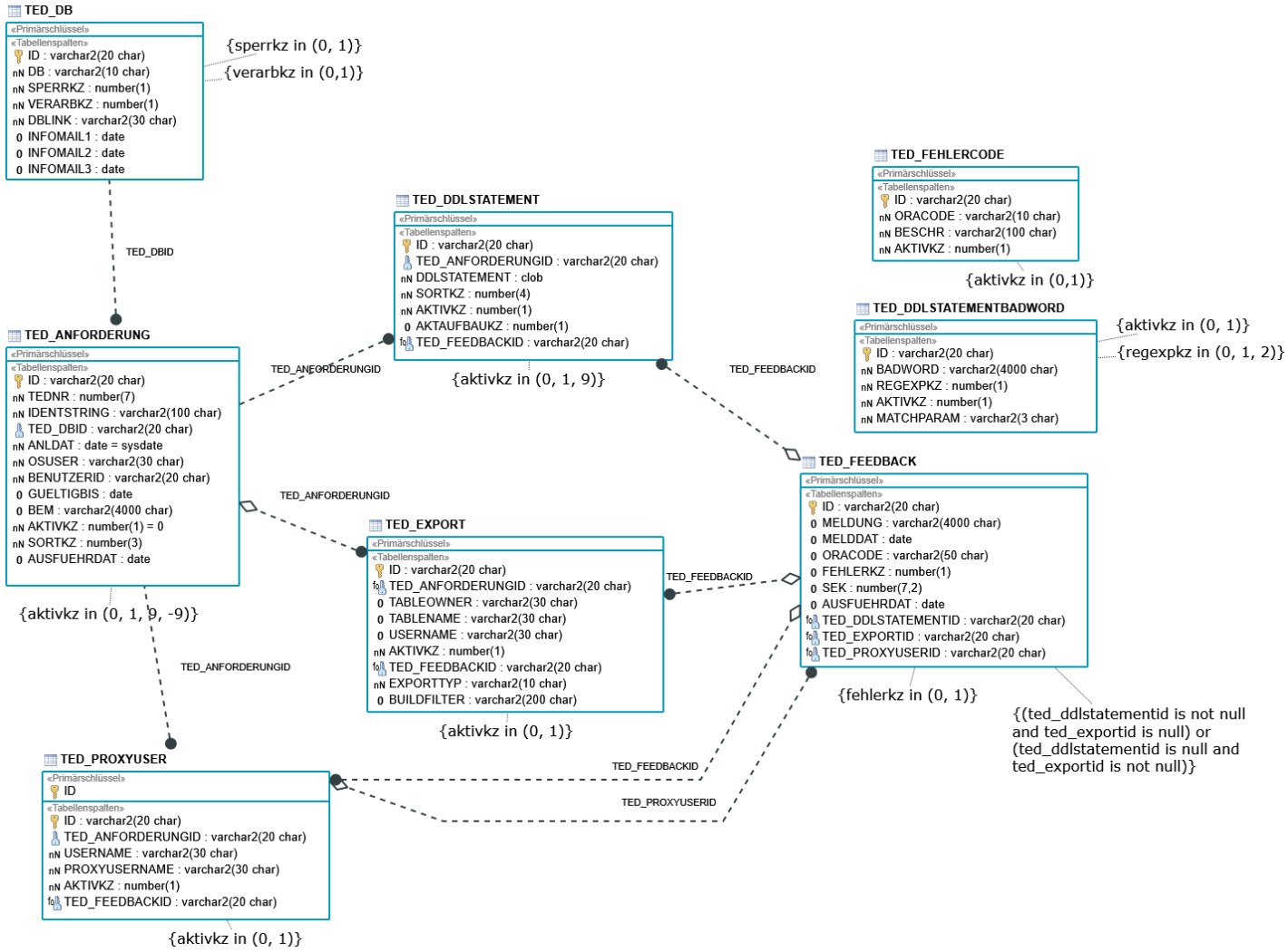


TED

TEstDatenbanken - Aufbau

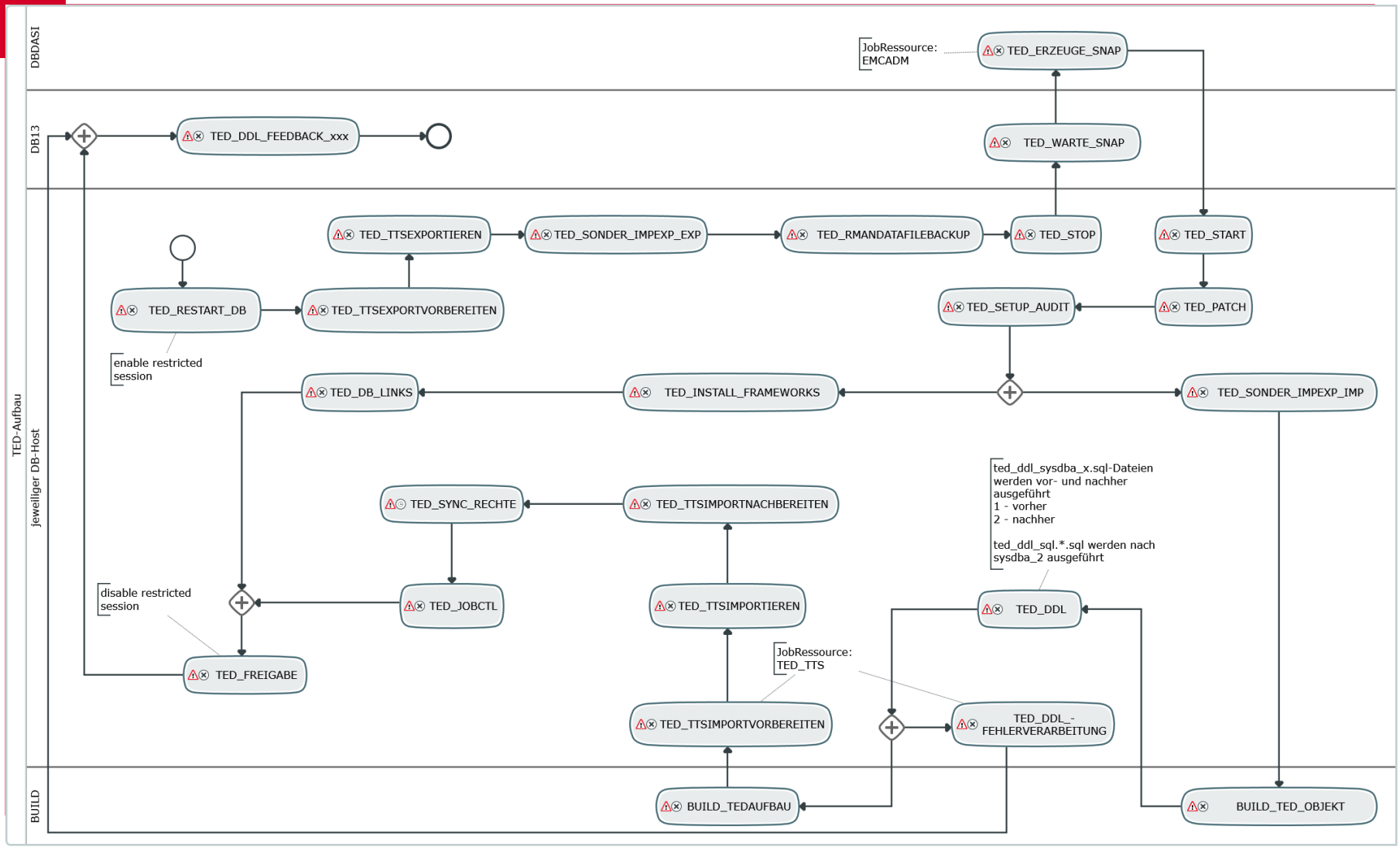


TED – Datenmodell

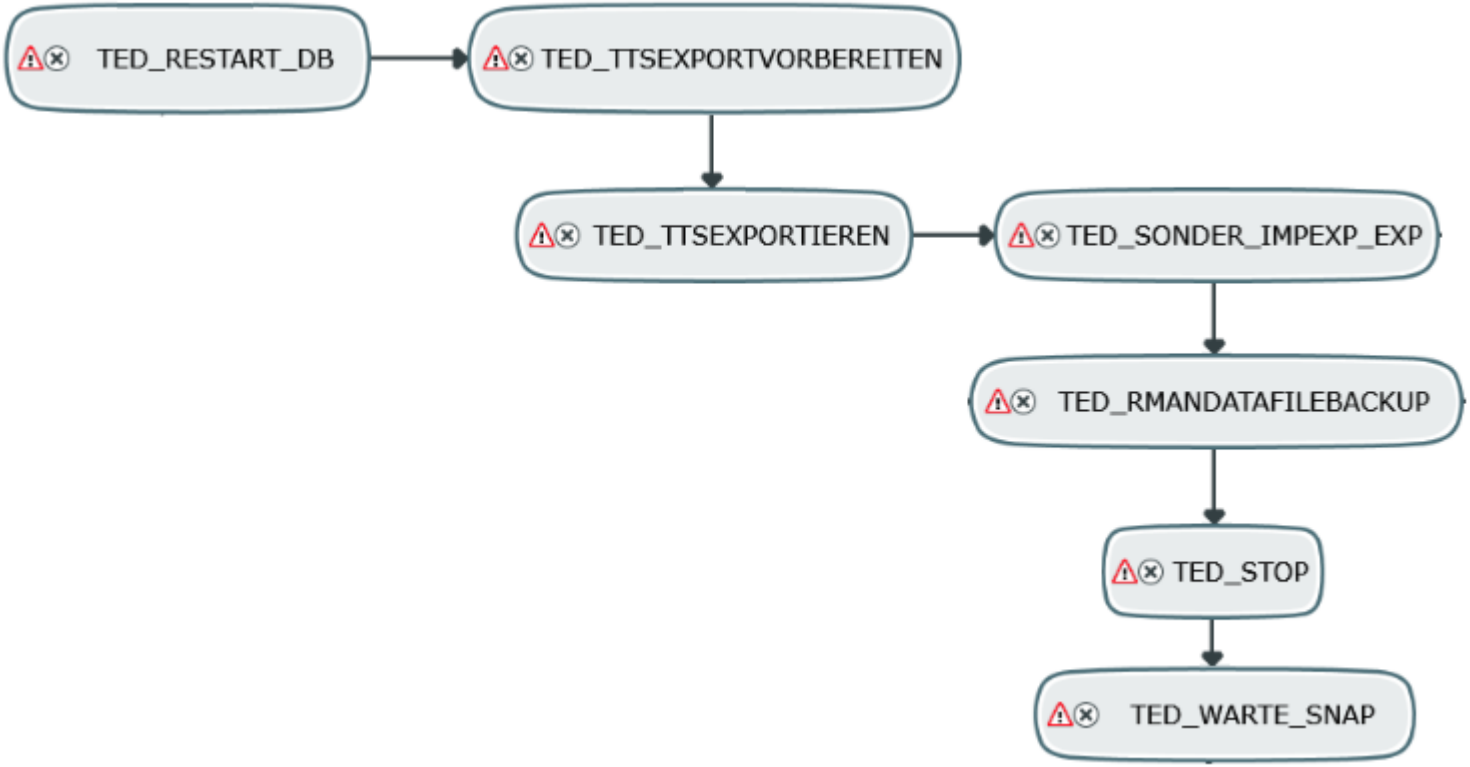


Witt-Gruppe

Ablauf eines TED-Aufbaus



Vorbereitung



Vorbereitung

TED_RESTART_DB

- Stoppen der Datenbank (immediate)
- Starten im restricted-Modus
  → Vermeidung von Locksituationen durch Langläufer-SQL der Anwender

```
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..
```

- Abfrage des Status ebenfalls per srvctl:

```
erg=$(srvctl status database -d ${dbid_lower})
if (! test "$erg" = "Database is running.") then
```

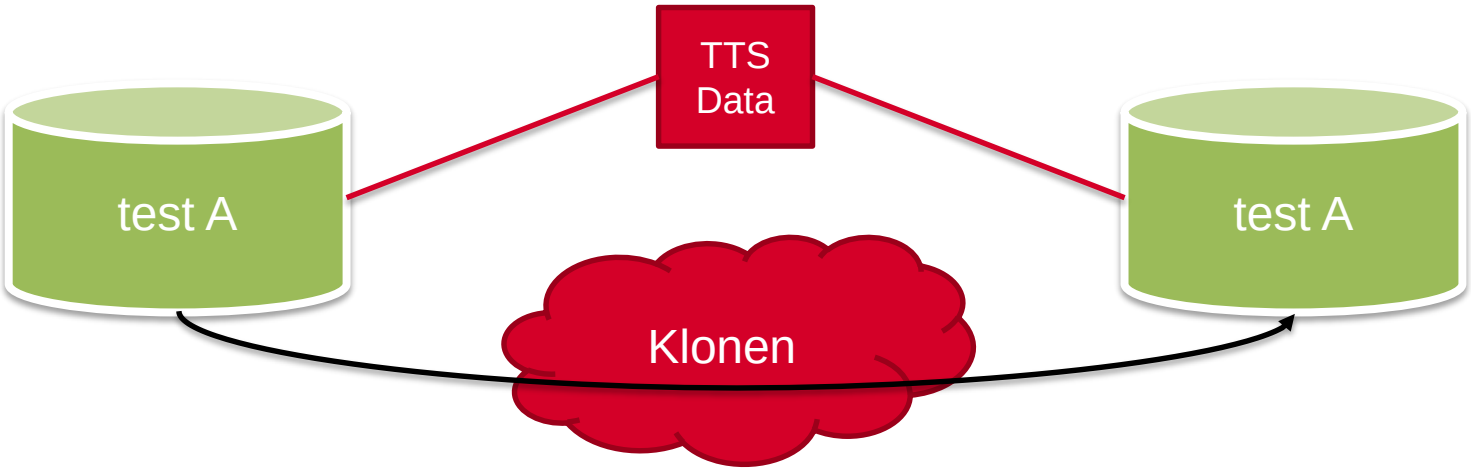
- bei einem RAC wird überprüft, ob alle Knoten laufen

Vorbereitung

TED_TTSEXPORTVORBEREITEN

Individualisierung..

- ausgenommen vom Aufbau können sein:
  - Tabellen
  - Tabellenstatistiken
  - Schemata
- Realisierung von „Tabellen-Ausnahmen“ durch Transportable Tablespace
- Objekte in TTS müssen „self contained“ sein, d. h. es darf keine Abhängigkeiten in und aus dem TTS geben (z. B. Constraints)



Vorbereitung

TED_TTSEXPORTVORBEREITEN

Individualisierung..

- Erstellung der „TED-Snapshot-Tabellen“, um Datenintegrität zu wahren
- Sichern der Statistiken, die „STATTAB“ liegt im TTS

```
dbms_stats.export_table_stats( ownname  => rec.tableowner
                             , tabname  => rec.tablename
                             , partname => null
                             , stattab  => TED_STATSTABLE
                             , cascade  => true
                             , statown  => TED_STATSTABLEOWNER );
```

- Constraint von den zu sichernden Tabellen droppen
- Self-Contained-Check durchführen:

```
sys.DBMS_TTS.TRANSPORT_SET_CHECK( TTS_NAME, TRUE );
```

- Tabellen, die erhalten bleiben sollen, in Transportable Tablespace verschieben
- Tabellen, die nicht mehr erhalten bleiben sollen, vom TTS in einen anderen Tablespace verschieben

Vorbereitung

Individualisierung..

- TTS mittels expdp exportieren:

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

- Es werden nur die Metadaten des Tablespaces exportiert. Die Daten liegen im Datafile und werden nicht angefasst.

- Das TTS-Datafile bleibt im ASM der Testdatenbank liegen.

Vorbereitung

Individualisierung..

- Sicherung von Objekten, die per TTS nicht möglich sind:
  - Schemata
  - Sequences
  - Spezielle Tabelle (z. B. ID-Tabelle, „TED-Snapshot-Tabellen“)
- Schemata werden über DBMS_DATAPUMP gesichert, die zu sichernden Schemata werden im TED-Datenmodell gehalten
- Sequences werden aus dem Data Dictionary gelesen und in eine SQL-Datei geschrieben, die später wieder ausgeführt wird; die Namen der Sequenes werden ebenfalls im TED-Datenmodell gehalten
- Spezielle Tabellen per expdp mit Parameterfile

Vorbereitung

- Sicherung des TTS-Datafiles per RMAN

```
run {
  backup as copy datafile '$transdatafile'
   TAG TTSDFSicherung
   FORMAT '$db/$(basename $transdatafile).$(date +%y%m%d_%H%M%S).bak';
  backup current controlfile
   TAG TTSDFSicherungCTL
   FORMAT '$db/rman_d%d_ctlfile_t%t_s%s_p%p';
}
```

- Löschen von alten Backups, nur drei werden behalten

```
find . \! -newer $oldest_ctlfile_piece \
          -mtime +$keep_backups_daysmin \
          -exec rm {} \;
```

Vorbereitung

TED_STOP

- Stoppen der Datenbank

```
srvctl stop database -d $dbid_lower -o immediate >> $protokoll_datei
2>&1
```

- Aushängen der ASM-Diskgruppen

```
diskgroups="data redogb"
for diskgroup in $diskgroups; do
  asmcmd umount -f ${diskgroup}${dbid}
done
```

- Ist die Testdatenbank ein RAC, müssen die Diskgruppen auf allen Knoten ausgehänt werden!

```
knotenliste=$(olsnodes)
```

Vorbereitung

- Realisierung von Abhängigkeiten zu Praxisjobs
- geklont wird erst, nachdem der Vorgängerjob (evtl. erfolgreich) gelaufen ist
- Testdatenbank hat dann den Stand nach dem Job

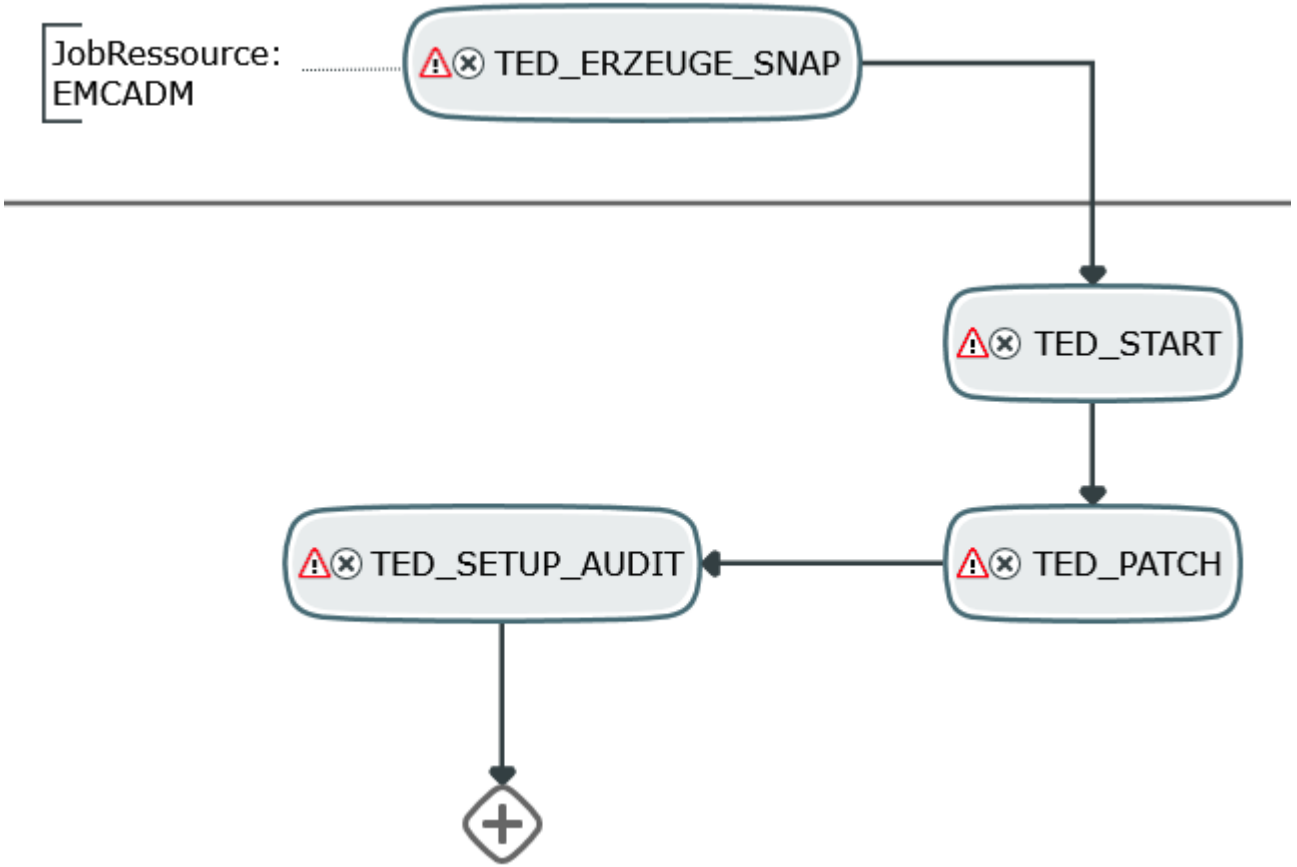
- realisiert über “Go-Datei“
- Vorgängerjob schreibt eine Datei:

```
echo `date +%H:%M:%S`“ Go-Dateo fuer TDBA-Aufbau erzeugen“ >>
$protokoll_datei
touch $DATEN/tdba.go
```

- Job wartet auf die Datei

```
dauer=0
until (test -f $DATEN/$daten_datei) || (test $dauer -gt $zeit_warn) do
 sleep $sleep_wert
 dauer=`expr $dauer + $sleep_wert`
done
```

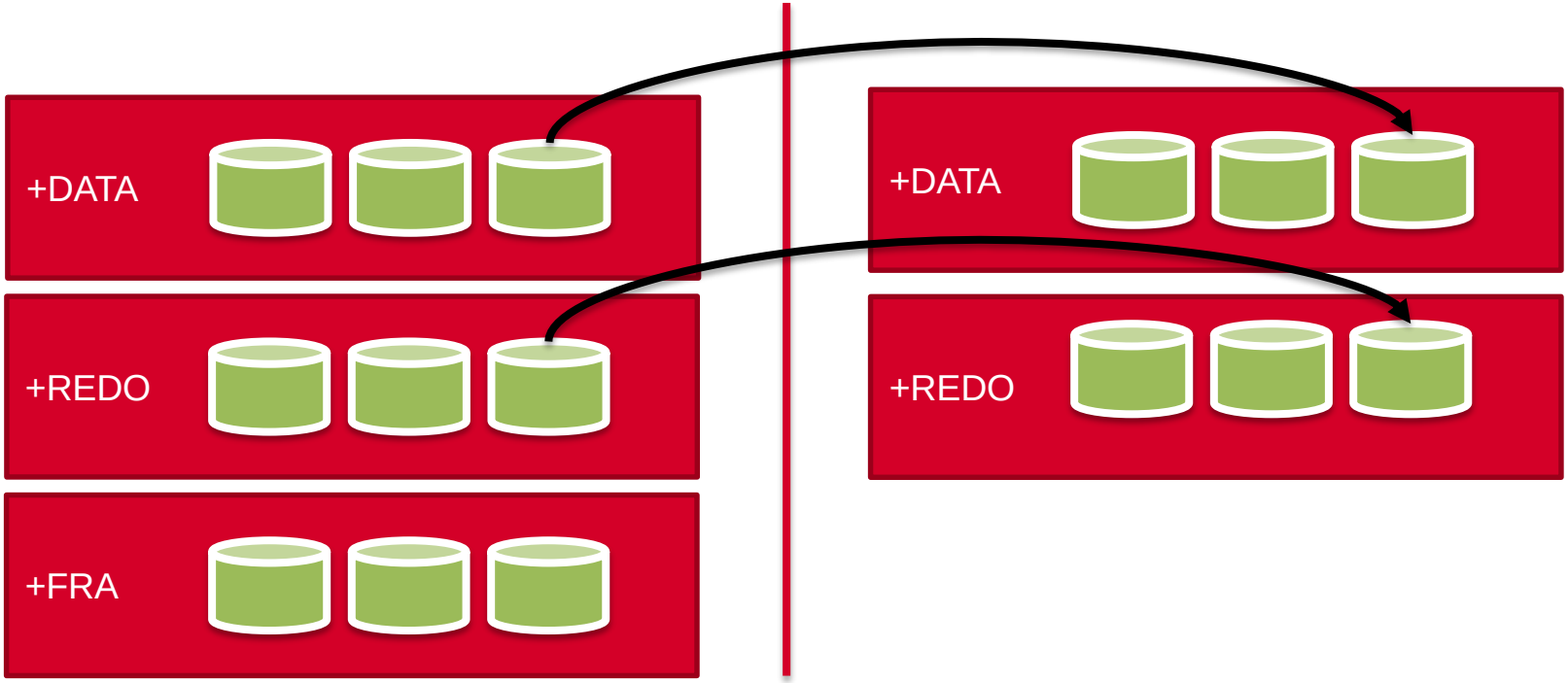
Klonen


JobRessource:
EMCADM
TED_ERZEUGE_SNAP
TED_START
TED_PATCH
TED_SETUP_AUDIT

Klonen

TED_ERZEUGE_SNAP

- Data- und Redo-Diskgruppen sind LUNs in unserer SAN-Umgebung
- Erzeugt Binärkopien von allen Disks und behält die Data- und Redo-Gruppen bei
- Witt benutzt Storage Möglichkeiten und das Skript-Framework emcadm



Witt-Gruppe

Klonen

- Mountet die Diskgruppen
- im Falle mehrerer Testdatenbanken auf einem einzelnem Host müssen die Diskgruppen umbenannt werden
  - Disk-Header (Diskgruppe und –name) muss im ASM eindeutig sein
  - wenn die Disk gemountet wird, muss sichergestellt sein, dass es nur eine Kopie jeder Disk gibt
  - Klone von anderen Testsystemen (auf dem gleichen Host) müssen solange warten, bis die Umbennenung fertig ist
- Umbenennen der Datafiles, weil neue Diskgruppennamen vergeben wurden

Klonen

- Datenbank mounten und Archiving und Flashback Logs ausschalten
- neue Datenbank ist “Crash-konsistent“, also muss ein Instance-Recovery durchgeführt werden (Öffnen und geordnetes Herunterfahren der Instanz)
- Umbenennen der Datenbank via nid (DB muss im „Mount“ sein) und Öffnen der Datenbank mit „resetlogs“

```
nid target=$SYS_PASS DBNAME=$dbid_lower << XXX >> $protokoll_datei
Y
XXX

sqlplus '/ as sysdba' << XXX >> $protokoll_datei 2>&1
startup mount;
alter system enable restricted session;
alter database open resetlogs;
shutdown immediate;
exit;
XXX
```

Klonen

TED_PATCH

Individualisierung..

- Testen neuer PSUs
- meistens deaktiviert
- Oracle-Home wird einmaliger gepacht, da dedizierter Server und eigenes Oracle-Home
- Durchführung des Post-Installation-Tasks

```
sqlplus / as sysdba << EOT >> $protokoll_datei_sqlplus
  @?/rdbms/admin/catbundle.sql psu apply
  exit;
EOT
```

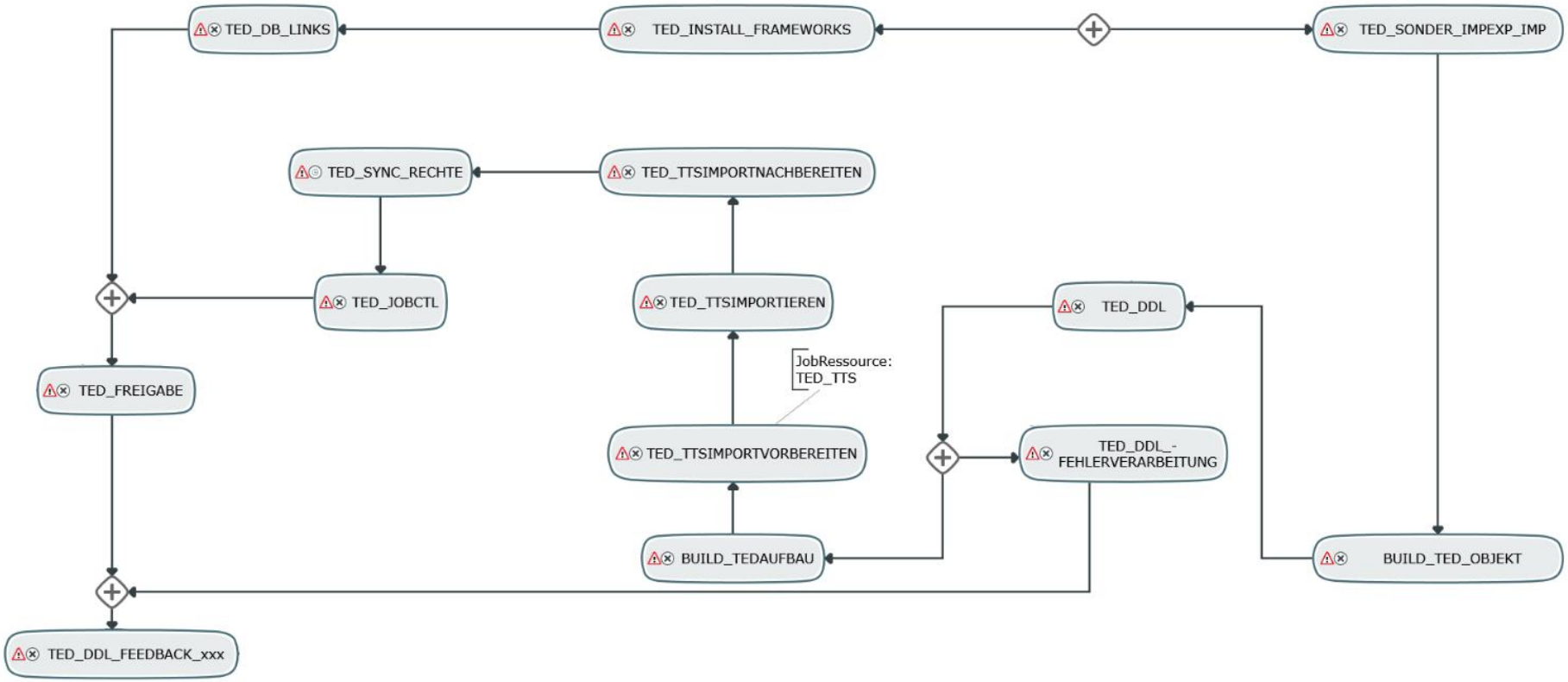
Klonen

Individualisierung..

- Aktivierung / Änderung der Witt-spezifischen Auditumgebung

- z. B. einsetzbar für Data-Masking

Nachbereitung



Nachbereitung

Individualisierung..

- Installation von Frameworks, die auf der Livedatenbank (noch) nicht zur Verfügung stehen

- Aufruf von Install-Skripts, entweder selbst geschriebene oder vom Hersteller zur Verfügung gestellte

```
# PL/JSON installieren
echo `date +%H:%M:%S`" Start Install PLJSON" >> $protokoll_datei
/pljson/install.sh „pljson/json_v1_0_4" pljson Js0n >> $protokoll_datei
```

Nachbereitung

Individualisierung..

- Entwicklern wird ein DB-Link zur Praxisdatenbank erstellt
  → Vergleiche zwischen Test und Praxis möglich
- Benutzung eines eigenen Read-Only-Accounts (NICHT select any table)
- User stehen in einer Konfigurationsdatei (Umstellung auf DB steht noch aus)

- DB-Links müssen immer vom User selbst erstellt werden
- Lösung: Proxy-Authentifizierung

```
execute immediate 'alter user ' || rec.result || ' grant connect
through WWDBA';
execute immediate 'grant create database link to ' || rec.result;
sqlplus -s wwdba[${USER4DBLINK[i]}]/$(echo $WWDBA_PASS | cut -d/ -f2)
```

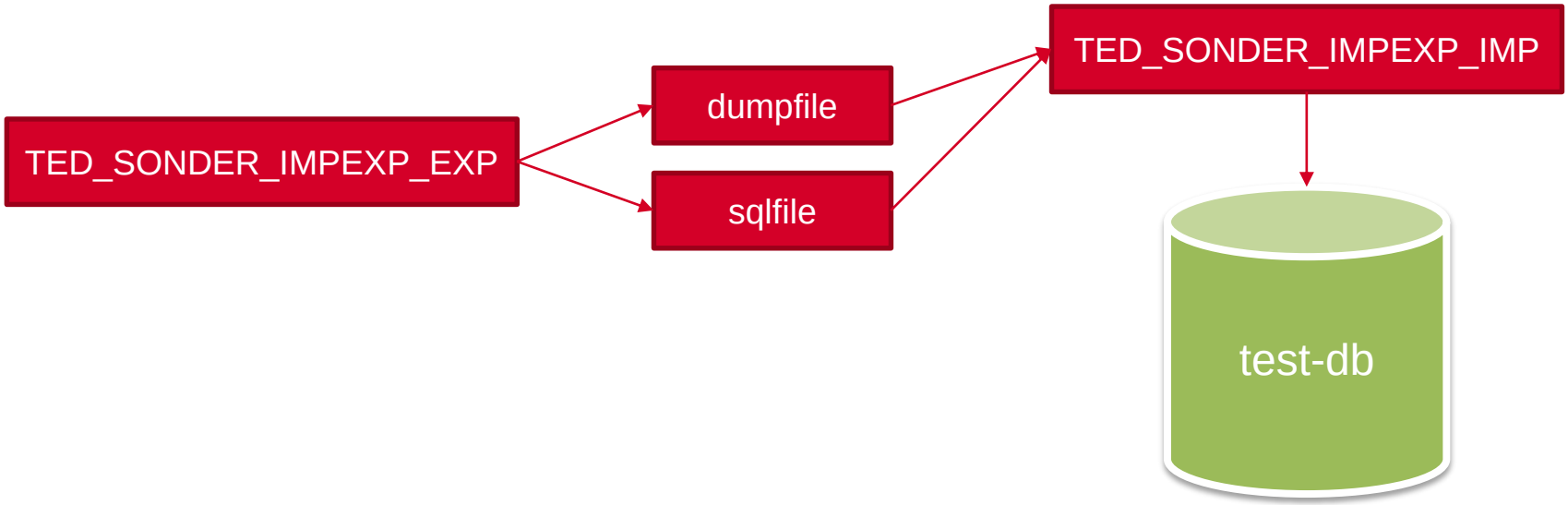
- nach der Anlage der DB-Links werden die Rechte wieder entfernt

Nachbereitung

Individualisierung..

- Ausgaben des Jobs „TED_SONDER_IMPEXP_EXP“ werden eingespielt
  - Dumps (impdp)
  - SQL-Spool-Files (z. B. Sequences, User)

- Objekte der „alten“ Testdatenbank sind nun (wieder) vorhanden



Nachbereitung

- Übergabe von PL/SQL-Objekten, die den TED-Aufbau betreffen
- Die nächsten Schritte, speziell der Job TED_DDL, laufen über PL/SQL-Packages.
- Durch diese „Extra-Übergabe“ dieser Packages sind Tests in einzelnen TED-Aufbauten möglich.
- Alternative: Übergabe ins Live-System, wäre aber gültig für alle Test-Aufbauten

Nachbereitung

TED_DDL

Individualisierung..

- Anwenden von Datenmodelländerungen in der Testdatenbank
- „Herzstück“ von TED

- DDL-Befehle werden aus dem Datenmodell gelesen und per „execute immediate“ ausgeführt
- Rückmeldung (ORA-Fehler) wird zurück in die Tabellen geschrieben
- Einstellen der DDL-Befehle über PL/SQL-Schnittstelle durch Modellierungstool
- Ausführung und Aktivierung der DDL-Änderungen durch DBA
  - erste Ausführung manuell
  - Aktivierung über PL/SQL- bzw. Shell-Schnittstelle

- Grants werden vor der Ausführung geprüft, ob diese noch nötig sind

Nachbereitung

- Analysieren der Fehler der DDL-Befehle vom Job TED_DDL
- Klassifizierung der Fehler:
  - „gut“ bzw. erwartbar, diese Statements werden deaktiviert, z. B.
    - ORA-00904 und DROP bzw. SET UNUSED

```
rec.oracode = 'ORA-00904' and
regexp_like( rec.stmtbeginn, 'drop', 'i' ) or
regexp_like( rec.stmtbeginn, '(.)*set([[:space:]])+unused',
'i' )
```

    - ORA-00942 und ein DROP

```
rec.oracode = 'ORA-00942' and
instr( upper( rec.statementbeginn ), 'DROP' ) > 0)
```

  - „schlecht“ bzw. nicht erwartbar, diese Statements werden wieder ausgeführt
    - Jeder andere Fehler ☺
- Löschungen ganzer Anforderungen eruieren
  - z. B. noch auszuführende DDLs Anforderungen sind nur Kommentare

Nachbereitung

TED_DDL_FEHLERVERARBEITUNG

```
regexp_like( stmt, '(.)*comment([[:space:]])+on(.)*', 'i' )
or
regexp_like( stmt,
'(.)*create([[:space:]])+or([[:space:]])+replace([[:space:]])+public([[:spac
e:]])+synonym(.)*', 'i' )
or
regexp_like( stmt, '(.)*pkg_userrechte(.)*', 'i' )
or
regexp_like( stmt,
'(.)*enable([[:space:]])+novalidate([[:space:]])+constraint(.)*', 'i' )
or
regexp_like( stmt, '(.)*modify([[:space:]])+constraint(.)*', 'i' )
or
regexp_like( stmt, '(.)*alter([[:space:]])+index(.)*noparallel(.)*', 'i' )
or
regexp_like( lower( stmt ), '(.)*revoke([[:space:]])+all(.)*', 'i' ) )
```

Nachbereitung

Individualisierung..

SVN

- Einchecken der Sourcen
- PL/SQL, Views (u. a.)

BEEPER

- Webanwendung
- Reifegrad der Version festlegen
- Test, Integration, Praxis

BUILD_ TEDAUFBAU

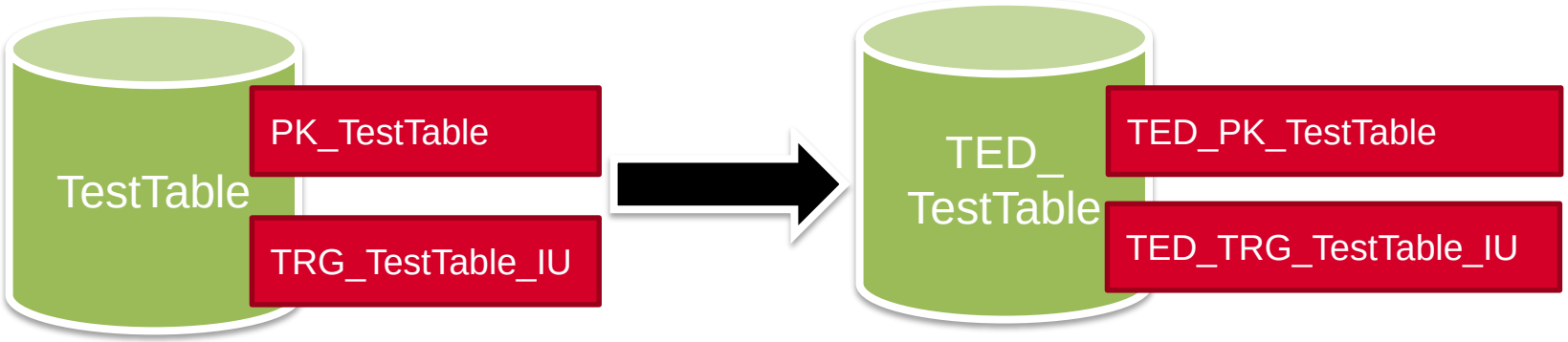
- Ermittlung der zu übergebenden Versionen
- Übergabe auf die Datenbank

Nachbereitung

TED_TTSIMPORTVORBEREITEN

Individualisierung..

- Tabellen, die wieder importiert werden sollen, sind schon vorhanden
- existierende Tabellen werden umbenannt:
  - Tabellen
  - Constraints
  - Trigger
- restliche Constraints, die in/aus den TTS zeigen, werden gepeichert und gelöscht
- Vorteile des Umbenennens gegenüber dem Löschen:
  - Praxis-Tabelle noch vorhanden → Klärungen, Vergleiche möglich
  - Schneller ☺



Nachbereitung

TED_TTSIMPORTIEREN

Individualisierung..

- Importieren der Metadaten aus der erzeugten Dump-Datei

```
impdp \
dumpfile=$dumpdatei \
directory=DATEN_DIR \
TRANSPORT_DATAFILES="'$transdatafile'" \
logfile=$se_log << EOT
  $WWDBA_PASS
EOT
```

- Exportierte Tabellen inkl. Daten stehen wieder zur Verfügung

Nachbereitung

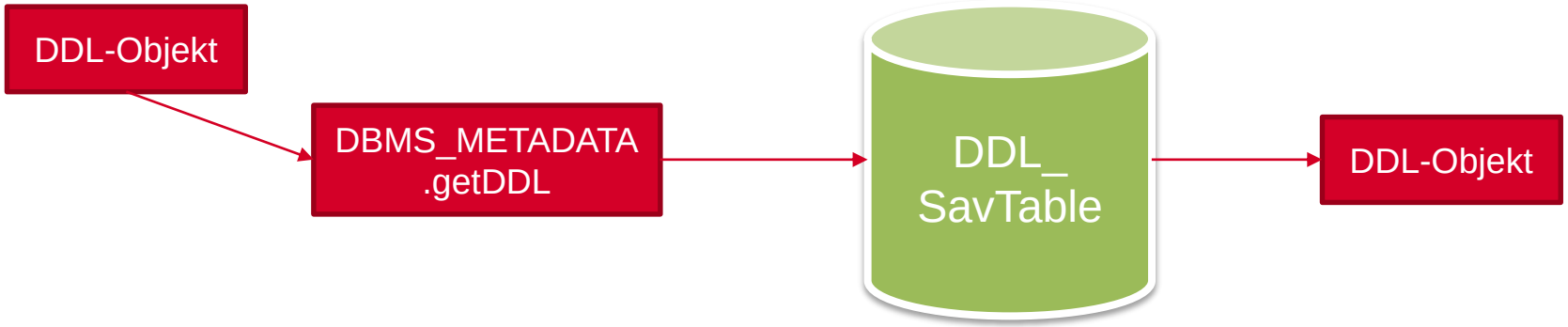
TED_TTSIMPORTNACHBEREITEN

Individualisierung..

- TTS-Tablespace wieder auf READ WRITE setzen
- Tabellenstatistiken importieren

```
dbms_stats.import_table_stats( ownname => rec.tableowner
                             , tabname => rec.tablename
                             , partname => null
                             , stattab => TED_STATSTABLE
                             , cascade => true
                             , statown => TED_STATSTABLEOWNER
                             , force => true );
```

- Constraints, die vorher gepspeichert wurden, werden wieder angelegt

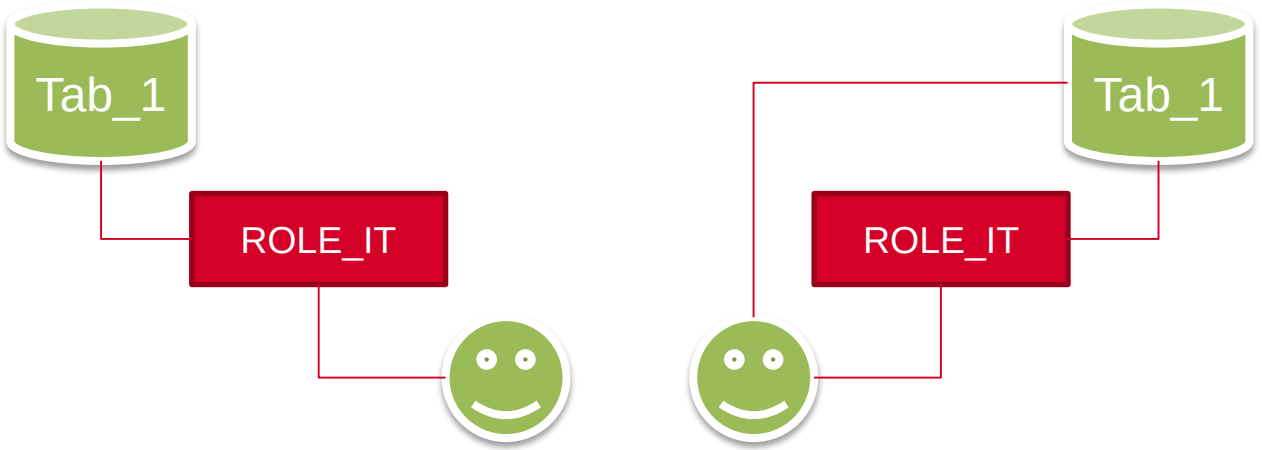


Nachbereitung

TED_SYNC_RECHTE

- PL/SQL-Entwicklung benötigt direkte Rechte, Rechte über Rollen wirken nicht
- Rechte werden über Rollen vergeben

- durch Rechte-Sync werden Rechte, die über Rollen erhalten werden, direkt dem User zugewiesen



Nachbereitung

Individualisierung..

- Veränderung des Witt-Job-Systems auf den Testdatenbanken
- Jobsystem läuft auch auf den Testdatenbanken
- Möglichkeit der Einflussnahme auf einzelne Jobs
- SQL-Befehle auf das Datenmodell des Jobsystems

Nachbereitung

TED_FREIGABE

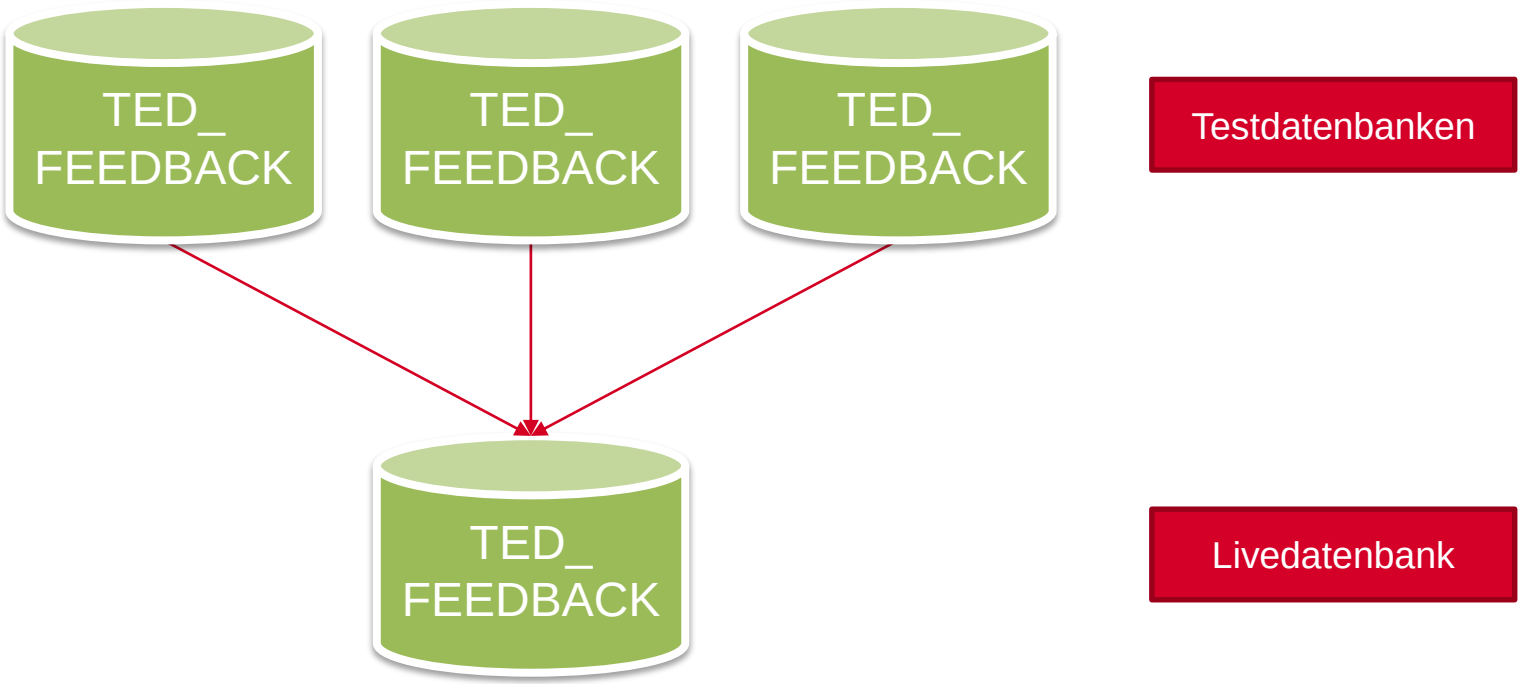
- Möglichkeit, eine SQL-Datei auszuführen (ted_freigabe.sql)
- Objekte nachkompilieren (Views, Pacakges, etc.)
- Restore-Point erzeugen (bei no-archivelog-DBs Fehler ☺)
  ```
  create restore point TEDAUFBAU guarantee flashback database;
  ```
- Freigabe der (restricted) Datenbank für die User
  ```
  alter system disable restricted session;
  ```
- AWR-Snapshot erzeugen
  ```
  EXEC DBMS_WORKLOAD_REPOSITORY.create_snapshot;
  ```

- Vorteile von „Restricted Session“:
  - Keine Sperren durch User möglich
  - Aufbau für sich alleine
- Nachteile:
  - Keine Parallelisierung mit dbms_parallel_execute möglich!

Nachbereitung

TED_DDL_FEEDBACK_xxx

- Übertragung des TED-Feedbacks in die Livedatenbank
- Feedback ist dadurch persistent vorhanden



Fazit

- Stabiler Aufbau ist notwendig: Besser ein Testsystem mit kleinen Fehlern als gar keins
- eigener Klon (bzw. Datenbank) für das Testen des TED-Ablaufs ist nützlich (DBADB)
- Beschleunigung des Öffnen mit “resetlogs“ durch Droppen einiger Redo-Log-Gruppen (first writes auf Snaps können massiv langsamer sein)
- Möglichkeit, Skript als sys (oder anderer User) auszuführen, wird benötigt
- TED wächst und wächst..