Testen leicht gemacht für PLSQL-Module

Dipl.-Ing. Michael Noll
DB Systel GmbH
Frankfurt am Main

Schlüsselworte
Modultest, PLSQL, Automatisierung, Development, Tools, Best Practices, Tips and Tricks, Praxisbericht.

Einleitung

In Unternehmen die eine Vielzahl Oracle-Datenbanken mit unterschiedlicher Größe und Aufgabenstellungen betreiben, in welchen PLSQL zur Realisierung komplexer, performance-kritischer Implementierungen eingesetzt wird, besitzen Modultests zur Qualitätssicherung eine besondere Bedeutung.
Automatisierung der Vorgehensweise bei der Durchführung von Modultests unabhängig von der zu testenden Fach-Logik, möglichst wenige einzuhaltende Rahmenbedingungen sowohl bzgl. benötigter Zusatz-Software/Lizenzen als auch bzgl. Einarbeitung der Entwickler/Tester können erhebliche Kosten/Aufwände innerhalb der Entwicklung/Qualitätssicherung einsparen.
Der folgende Vortrag erörtert eine effiziente Möglichkeit zur Erstellung eines generell einsetzbaren PLSQL-Modul-Test-Automaten (PMTA) bzw. dessen Aufbau und Vorgehensweise mit Oracle-"Bord-Mitteln".

Ziele

Ein automatisierter Modultest für PLSQL-Funktionalität sollte folgende Ziele verfolgen :

- Reduktion personeller Resourcen-Bindung durch manuelle Aktivitäten beim Test
- Verbesserung der Termineinhaltung durch Testdurchführung außerhalb der Bürozeiten
- Vollständigkeit der Testabdeckung
- Sicherstellung inhaltlicher Sinnhaftigkeit der Tests durch Design der Testmodule als Bestandteil der Realisierung
- Sicherstellung korrekter Funktionsweise von Modulen deren Beurteilung bisher Experten-Knowhow erforderte
- Reduktion der Ergebnis-Dokumentationsaufwände durch automatische Report-Erzeugung
- Eindeutige Reproduzierbarkeit von Testfällen
- Reduktion von Entwicklungsaufwänden bzgl. Entwicklungs-Tests durch entwicklungsbegleitenden Einsatz
- Einsatzmöglichkeit auf Entwicklungs-/Abnahme-/Produktions-Umgebung
- Frühzeitige Erkennung von potentiellen Performance-Problemen bei Oracle-Versions-Wechseln durch Laufzeitmessungen

Anforderungen

Im dem Vortrag zu Grunde liegenden Fall wurden als Anforderungen an einen PLSQL-Modul-Test-Automaten (PMTA) folgende Anforderungen gestellt :

- automatisch ausführbar
- automatische Laufzeitauswertungen
- automatische ausführliche und akkumulierte Report-Generierung (Excel)
- automatische Erfassung der Log-Informationen pro Testfall
- konform zu Firmen-internen Oracle-BF-Vorgaben

- unabhängig von Zusatz-Tools (Umsetzung mit Oracle-“Bord-Mitteln“)
- eigenständiges Modul unabhängig von anderen technisch / fachlichen Libraries
- vollständig als “ADDON“ installierbar/deinstallierbar
- anwendbar in Entwicklungs-/Abnahme-/(Produktions-) Umgebung
- besitzt zentrale Konfiguration (pro Testfall ein Eintrag)
- ermöglicht die Übergabe beliebiger Input-Parameter bzgl. Name, Typ (inkl. User-Defined Typen !), Anzahl
- auszuführen unter Produktions-Bedingungen (Rechte/gesperrte Owner etc.)
- Anlehnung an klassische Test-Vorgehensweisen in anderen Programmiersprachen
  (zu jedem Modul stellt der Entwickler ein entsprechendes Test-Modul zur Verfügung)
- Testfälle
  - können optional vorgelagerte Pre-Prozesse einschließen (z.B. Test-Daten einspielen)
  - können optional nachgelagerte Post-Prozesse einschließen (z.B. Test-Daten entfernen)
  - ermöglichen die Ausführung von Pre- und Post-Prozessen unter dem gleichen oder andern User als der Haupt-Test
  - sind als Definer- oder Invoker-Right-Aktionen ausführbar
  - werden konfigurativ gebündelt nach Owner und Modul
  - können Owner-, Modul-, Sub-Modul, Test-Fall-weise über Konfiguration aktiviert/deaktiviert werden
  - sind einzeln manuell wiederholbar

Make-/Buy-/Reuse-Entscheidung

Ausgehend von den definierten Anforderungen wurden verschiedene komerzielle und Open-Source Produkte mit gleicher oder ähnlicher Zielrichtung evaluiert, u.a.

- utPLSQL
- Quest code tester
- PL/Unit
- PLUTO – PL/SQL Unit Testing for Oracle
- ruby-plsql-spec
- DBFit
- DBUnit
- SQLDeveloper

Da bei den untersuchten Produkten Aktualität/Support, anfallende Kosten, Bereitstellungs- oder Einarbeitungs-Aufwand, bzw. ggf. nicht unterstützte Datentypen den festgelegten Anforderungen widersprachen, wurde ausgehend von zwei Projekten als Bedarfsträger eine “Make“-Entscheidung getroffen

Festlegung des prinzipiellen Aufbaus des PMTA

Der Entwurf der inneren Abläufe des PMTA sah folgende Schritte vor :

- Voraussetzungen
  - Zu testende Objekte liegen vor
  - Test-Fälle sind bekannt und als PMTA-Konfigurations-Skripte hinterlegt
  - Test-Objekte (Logische Entscheidung über Erfolg/Misserfolg eines Tests) sind gemäß PMTA-Design-Pattern in Skript-Form hinterlegt
- PMTA-Objekte (inkl. Test-Konfiguration) anlegen
  - Berechtigungen auf PMTA-Objekte an nutzende Schemata vergeben
    [automatisch über PMTA-“Core“]

- Test-Objekte anlegen, welche die Test-Beurteilungs-Logik enthalten
  (erstellt durch Entwickler)
  [über Installations-Skripte]
- Berechtigungen auf Test-Objekte analog Original-Objekte an Test-User (gemäß Test-Konfiguration) vergeben
  [automatisch über PMTA-“Core“]
- Private Synonyme auf Test-Objekte analog Original-Objekte an Test-User (gemäß Test-Konfiguration) vergeben
  [automatisch über PMTA-“Core“]
- Pre-/Post-/Core-Test-Skripte jeweils pro Test-Fall generieren
  [automatisch über PMTA-“Core“]
- Steuerungs-Skript zum Aufruf der Test-Fall-Skripte generieren
  [automatisch über PMTA-“Core“]
- Ausführung des Steuerungs-Skripts (sequentieller Durchlauf der jeweiligen Pre-/Post-/Core-Test-Skripte)
  - Technische Schicht ruft Test-Objekte mit Parametern gemäß Konfiguration auf
  - Test-Objekt ruft Original-Objekt mit weitergereichten Parametern auf
  - Original-Objekt produziert Ergebnis
  - Test-Objekt beurteilt Ergebnis des Original-Objekts bzgl. Test-Erfolg
  - Test-Objekt meldet Test-Ergebnis an Technische Schicht

  [über generierte Skripte (PMTA-“Core“)]
- Ergebnis-Protokollierung und Laufzeit-Messung der Skript-Ausführungen
  [automatisch über PMTA-“Core“]
- Aufbereitung des Gesamt-Ergebnisses in Kurz-Form (für Testmanager) und Lang-Form (für Entwickler)
  [automatisch über PMTA-“Core“]
- Optional : PMTA-Objekte entfernen
  [über generierte Skripte (PMTA-“Core“)]

Wie die markierten Anmerkungen bereits nahelegen besteht die Umsetzung des hier vorgestellten PMTA im Kern aus einem kleinen Framework dessen Hauptbestandteil ein Skriptgenerator ist.

Festlegung der internen Architektur

Bei der Realisierung des inneren Abläufe wurde eine Architektur in drei Schichten gewählt, um eine optimale Entkopplung von Test-Steuerung, Prüfungs-Logik und zu testenden Objekten zu erzielen :

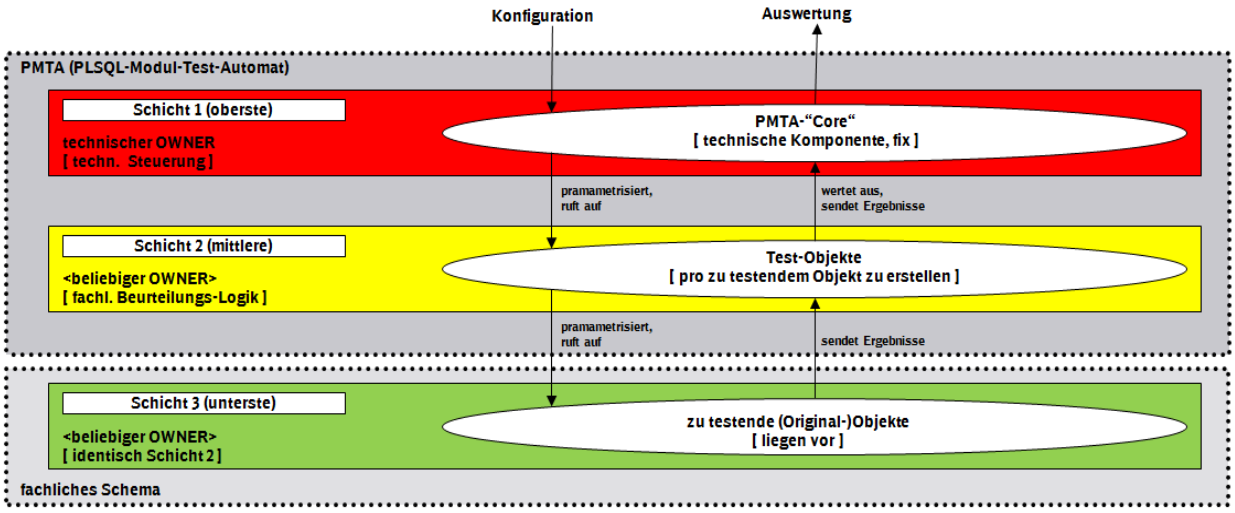

Abb. 1: Interne Architektur des PMTA

Überblick des Zusammenspiels der PMTA-Objekte

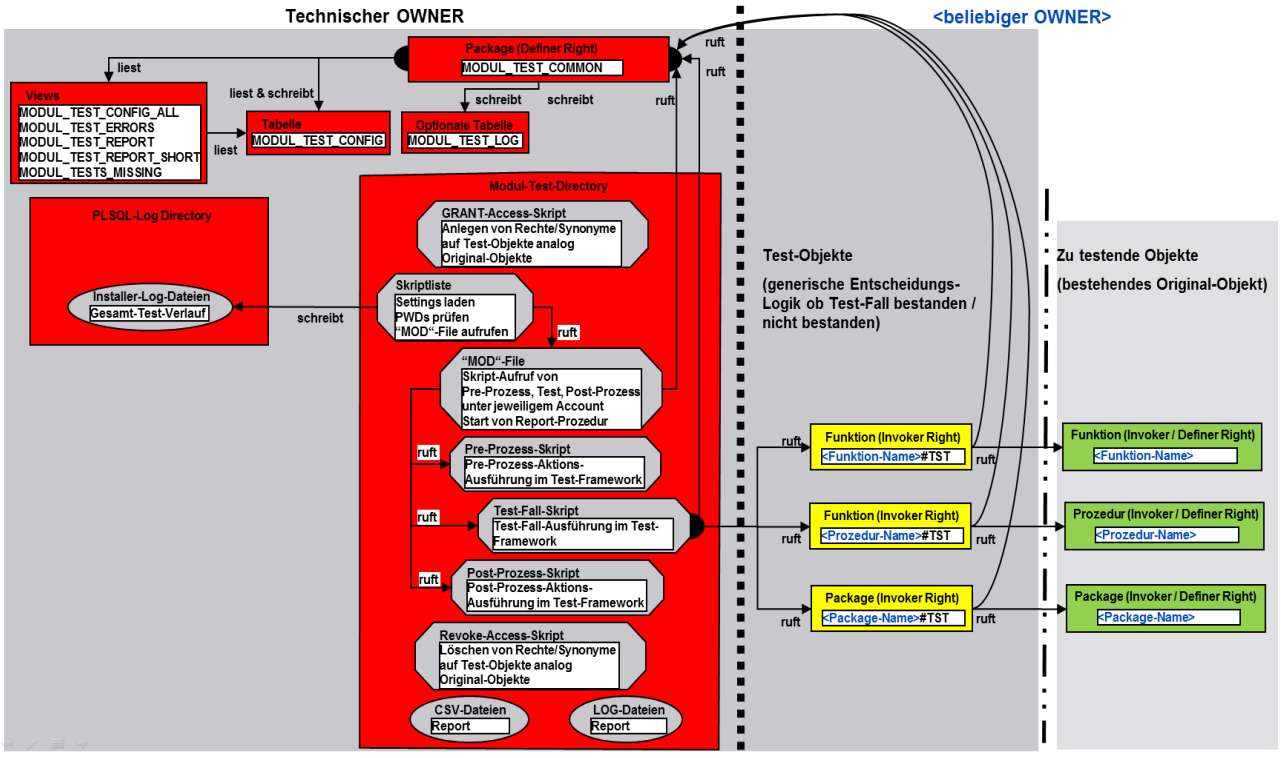

Abb. 2: Überblick des Zusammenspiels der PMTA-Objekte

Farb-Legende :

- Rot - Bestandteil des PMTA-Core
- Gelb - Test-Objekte
- Grün - Original, zu testende Objekte

Details der Implementierung

1. Datentypen für Konfigurations-Tabelle :

```
CREATE TYPE TY_ADR_CLOB_BLOB_TAB_ROW AS OBJECT(
  ITEM_NAME     VARCHAR2(4000),
  ITEM_TYPE     VARCHAR2(128),
  C_VALUE       CLOB,
  B_VALUE       BLOB
);
/

CREATE TYPE TY_ADR_CLOB_BLOB_TAB AS TABLE OF TY_ADR_CLOB_BLOB_TAB_ROW;
/
```

2. Konfigurations-Tabelle :

```
CREATE TABLE MODUL_TEST_CONFIG
(
  MODUL_OWNER             VARCHAR2(30),
  MODUL_NAME              VARCHAR2(30),
  SUB_MODUL_NAME          VARCHAR2(30),
  MODUL_USER              VARCHAR2(30),
  TEST_ID                 INTEGER,
  PRE_PROCESS_SUCCESSFUL  VARCHAR2(7)    DEFAULT NULL, /* TRUE,FALSE,PENDING */
  TEST_SUCCESSFUL         VARCHAR2(7)    DEFAULT NULL, /* TRUE,FALSE,PENDING */
  POST_PROCESS_SUCCESSFUL VARCHAR2(7)    DEFAULT NULL, /* TRUE,FALSE,PENDING */
  TEST_START              TIMESTAMP      DEFAULT NULL,
  TEST_END                TIMESTAMP      DEFAULT NULL,
  MODUL_TYP               VARCHAR2(30),                /* Package,Procedure,Function */
  SUB_MODUL_TYP           VARCHAR2(30)   DEFAULT ' ',  /* Procedure,Function */
  INPUT_PARAMETERS        TY_ADR_CLOB_BLOB_TAB,
  EXPECTED_FUNCT_RESULT   CLOB           DEFAULT NULL,
  PRE_PROCESS_USER        VARCHAR2(30)   DEFAULT NULL,
  PRE_PROCESS_STATEMENTS  CLOB           DEFAULT NULL,
  POST_PROCESS_USER       VARCHAR2(30)   DEFAULT NULL,
  POST_PROCESS_STATEMENTS CLOB           DEFAULT NULL,
  EXECUTION               VARCHAR2(8)        NOT NULL, /* ENABLE,DISABLE */
  DESCRIPTION             VARCHAR2(4000),
  CONSTRAINT PK_TEST_RESULT PRIMARY KEY
            (MODUL_OWNER,MODUL_NAME,SUB_MODUL_NAME,TEST_ID)
)
NESTED TABLE INPUT_PARAMETERS STORE AS TEST_INPUT_PARA
MONITORING;
```

Weitere Details der Implementierung bzw. besondere Herausforderungen bei der Unterstützung bzw. Konfiguration beliebiger Parameter-Typen und Parameter-Anzahl werden in der beigefügten Powerpoint-Präsentation erläutert.
Diese stellt die erforderlichen Funktionalitäten innerhalb der Skriptgenerator Frameworks sowie Templates generierter Skripte für verschiedene Test-Objekt-Typen dar.

Use-Case abhängige Aussteuerungs-Varianten (bei Test-Durchführung)

1. Komplett-Installation / Reinstallaton (SKRIPTLISTE_MODUL_TEST_INSTALL.SQL)
   - Anwenundgsfall für
     - initiale Installation oder

    - neuer PMTA-Lauf nach dem neue Test-Objekte und entsprechende Test-Fall-Konfigurationen hinzugefügt wurden
- Voraussetzungen
    - keine

2. Tests durchführen (SKRIPTLISTE_MODUL_TEST_START_EXECUTION.SQL)
   Wiederholung / Neuauswertung aller Test-Fälle gemäß Konfiguration
   - Anwenundgsfall für
     - PMTA-Installation ist gelaufen, Auswertung ist noch nicht erfolgt oder
     - Test sollen ohne Änderung an den Test-Objekten / Test-Fall-Konfigurationen wiederholt werden oder
     - Test sollen nach Bug-Fix an zu testenden Objekten ohne Änderung an den Test-Objekten / -Fall-Konfigurationen wiederholt werden
   - Voraussetzungen
     - PMTA ist installiert
     - Test-Fall-Generator oder Auswertung ist mindestens einmal vollständig gelaufen

3. Tests erweitern (SKRIPTLISTE_MODUL_TEST_START_GENENERATOR.SQL)
   Neu-Aufbau der Test-Fall-Skripte
   - Anwenundgsfall für
     - Test-Konfigurationen ohne Änderungen der Test-Objekte wurden geändert oder hinzugefügt
   - Voraussetzungen
     - PMTA ist installiert
     - Test-Fall-Generator oder Auswertung ist mindestens einmal vollständig gelaufen
     - Test-Objekte / Test-Fall-Konfigurationen sind seit letzten Lauf / Installation unverändert

Besonderheiten der Implementierung

1. Konfiguration der Datentypen bei Input-Parametern bzw. Function-Return SOLL-Werten
   Lösung :
   - Konfiguration nimmt numerische oder Charakter-basierten Werten als CLOB entgegen
   - Konfiguration nimmt binäre Werten als BLOB entgegen
   - Skript-Generator sorgt für korrekte Typen-Übergabe
   - Ausnahme : Verarbeitung von Function-Return SOLL-Werten. Diese benötigen in der Beurteilungslogik ggf eine Typen-Konvertierung

2. Konfiguration von Input-Parameter Name, Typ, Anzahl
   Lösung :
   - Input-Parameter-Spalte besteht aus Objekt-Typ (nested table)
   - Aufbau der Nested table
     - Spalte 1 – Name des Parameters
     - Spalte 2 – Typ des Parameters inkl. Länge
     - Spalte 3 – Wert des Parameters sofern numerisch, Character-basiert oder User-defined
     - Spalte 4 – Wert des Parameters sofern binär

3. Konfiguration von Input-Parametern die “echte“ Clob oder Blob-Werte (> 32K) aufnehmen
   Lösung :
   - Lade-Prozeduren, die Dateien in das jeweilige Feld der nested table schreiben

4. Konfiguration von dynamischen Function-Return SOLL-Werten (Wert muss zur Laufzeit ermittelt werden)
   Lösung
   - Hinterlegung eines SELECT-Statements in der EXPECTED_RESULT-Spalte der Konfiguration
   - Voranstellung einer Kommentar-Markierung /* USER_DEFINED_STMT*/ vor dem SELECT- Statement
   - Dynamische Auswertung des Statements zu Laufzeit innerhalb der Funktion GET_FUNC_TOBE

5. Konfiguration von dynamischen Input-Parameter-Werten
   Lösung
   - Hinterlegung eines SELECT-Statements in der C_VALUE der Konfiguration
   - Voranstellung einer Kommentar-Markierung /* USER_DEFINED_STMT*/ vor dem SELECT- Statement
   - Dynamische Auswertung des Statements zu Laufzeit innerhalb des Test-Scripts im Werte-Zuweisungsabschnitt
     (SELECT-Statement muss dann ein einen Aufbau SELECT … INTO <Variablen-Name> FROM … WHERE … besitzen)

6. Konfiguration von User-Defined Input-Parameter-Werten
   Lösung
   - Hinterlegung des User-Defined-Datentyp-Namens unter ITEM_TYPE
   - Voranstellung einer Kommentar-Markierung /* USER_DEFINED_TYPE */ vor dem Typ-Namen in der Konfiguration
   - Hinterlegung eines Statements mit explizitem Type-Cast als C_VALUE der Konfiguration
   - Voranstellung einer Kommentar-Markierung /* USER_DEFINED_TYPE */ vor dem expliziten Type-Cast-Statement
   - Dynamische Auswertung des Typs zu Laufzeit innerhalb des Test-Scripts im Deklarations- und Werte-Zuweisungsabschnitt

Kontaktadresse:
Dipl.-Ing. Michael Noll
DB Systel GmbH
Weilburger Straße 28
60326 Frankfurt am Main

Telefon: +49 (0) 265-50204
E-Mail michael.noll@deutschebahn.com
Internet: www.dbsystel.de