PL/SQL: Drum test-automatisiere, wer sich sich ewig bindet!

Torsten Kleiber
IKB Deutsche Industriebank AG
Düsseldorf

Schlüsselworte

PL/SQL, Development, Build, Unit Test, Regression Test, Continuous Integration, Jenkins, Erfahrungsbericht, Nutzwertanalyse

Einleitung

Auf den zuletzt von mir besuchten Konferenzen fiel oft die Frage, ob man PL/SQL überhaupt automatisiert testen kann oder muss. Die meisten der teilnehmenden kannten die heute vorgestellten Frameworks nicht oder sahen nicht den Vorteil von automatischen Test. Der Vortrag soll Ihnen eine Entscheidungsgrundlage liefern, ob Sie demnächst auch automatisch testen wollen und können!

Welche Fehler kann ich mit Unit-Tests überhaupt finden?

Was sind die Ursachen, die zu unbeabsichtigten Fehlern in meinen Programmen führen? Sind das die, die Ihnen auch einfallen?

Änderung bestehender Programme

- Aufrufparameter: Typ, Reihenfolge, Optionalität, Defaults
- Ausführungspfade: Verschiebung von Code Blöcken
- Business-Logik: Erweiterung, Entfernen, Bedingungen
- Wrapper
- Refactoring

Datenbank oder Infrastruktur Patching/-Upgrade

- Betriebssysteme
- Dateisystem
- Berechtigungen
- Optimizer
- Indizes
- Deprecated / Deleted Features

Änderung des Datenmodells

- Umbenennung von Objekten
- Neue Spalten
- Optionalität: NULL / NOT NULL Spalten
- Defaults
- Constraints
- Referentielle Integrität
- %TYPE und %ROWTYPE-Referenzen
- DML Ausführungen

Daten

- Änderungen
- Mengen

Die meisten der Probleme dürften auch in Ihrem Umfeld auftreten, damit sind Unit Tests eigentlich unverzichtbar, um den Code reproduzierbar zu testen.

Unit Tests sollen regelmäßig wiederholt werden, damit Modifikationen in bereits getesteten Teilen der Software keine neuen Fehler („Regressionen“) verursachen, damit werden sie zu Regressionstests.

Wie sieht eigentlich mein Entwicklungsprozess aus?

Wie viele Entwickler habe ich?

- 1-2
- 10
- Hunderte?

Umso höher die Zahl meiner Entwickler für eine Source Basis ist, umso wahrscheinlicher sind Abhängigkeiten und Schnittstellen. Diese klar zu definieren und mit Tests zu versehen ist essentiell. Außerdem steigt ebenfalls die Wahrscheinlichkeit von konkurrierenden Zugriffen sowie die Größe der Code Basis. Gut geschriebene Tests mit einer hohen Code Abdeckung können hier bei der Beschreibung Funktionalität helfen.

Muss ich branchen?

- Wie schnell werden Änderungen abgenommen? Kann jede Änderung immer mit dem nächsten Release eingesetzt werden?
- Werden Projekteinsätze häufig verschoben?

- Konkurrieren Source-Änderungen verschiedener Projekte oder Entwickler. Das tritt häufig bei zentralen Bibliotheken auf.
- Müssen Sourcen aufgrund z.B. gesetzlicher Bestimmungen erst zu einem fixen Termin eingesetzt werden, sollen aufgrund des Aufwands aber über lange Zeit eingesetzt werden.
- Das Risiko des Verlusts von Änderungen soll minimiert werden, auch nicht fertige Arbeit soll gesichert werden.
- Dennoch sollen die Änderungen früh integriert werden, das Zusammenspiel der Komponenten soll getestet werden.

Diese Konflikte kann ich nur mit einem der vielen Branching Modelle, Feature Toggles oder Branch by Abstraction nutzen. Feature Toggles sind meist nur bei Framework-loser möglich. Framework-Wizards unterstützen meist keine Feature Toggles und zerstören potenziell manuell eingefügten Toggle Code. Branch by Abstraction bedeutet das Einziehen einer Zwischenschicht und wird meist nur beim Austausch großer Schnittstellen, Schichten und Technologien eingesetzt.

Welches Wissen haben meine Mitarbeiter?

- Sprachen: PL/SQL, SQL, Java, Junit, Ruby, RSpec, …
- Buildtools: Maven, ANT, Gradle, …
- CI, CD Server: Jenkins, Hudson, Bamboo, Travis, …
- DevOps: Build, Deployment, Administration

Habe ich eine Entwickler-Basis, die nur im Oracle Datenbank Umfeld unterwegs ist, sind diese mit Java oder Ruby basierten Testframeworks wahrscheinlich überfordert. Habe ich Fullstack-Entwickler die sowohl in der Datenbank- als auch in der Java Frontend Entwicklung tätig sind, ist die Hemmschwelle vielleicht geringer. Je nach vorhanden Automatisierungs-Server kann die Unterstützung deren Reporting-Mechanismen ein ausschlaggebender Punkt sein.

Muss ich oft meinen Code umstrukturieren?

- Tausche ich regelmäßig Frameworks aus oder muss diese aktualisieren, z.B. technische für das Logging oder Dateizugriffe, aber auch Business Logik wie Rechenkerne etc.?
- Will ich die technische Schulden meiner Code-Basis gering halten und z.B. nicht mehr verwendeten Legacy Code regelmäßig entfernen?
- Will ich agil bei Bedarf modularisieren, das heißt Module erst bei der 2 Verwendung aus dem Code herauslösen.
- Ergeben sich im Zeitverlauf regelmäßige Architekturänderungen, die auf die gesamte Code Basis ausgerollt werden müssen.

Regelmäßiges Refactoring ohne automatische Tests hat ein hohes Risiko und wird deshalb meist vermieden. Die Einführung von automatischen Tests erlaubt also oft erst dies.

Welche Frameworks gibt es für die Testautomatisierung mit PL/SQL?

Tool	Kosten	Source	Architektur	Status
Quest Code Tester (Option für TOAD/SQL Navigator)	Kosten-pflichtig	Closed	Deklarativ, Repository	Aktiv
Allroundaoutomations PL/SQL Developer Test Manager	Kosten-pflichtig	Closed	Deklarativ, Text-Dateien	Aktiv
Oracle SQL Developer	Frei	Closed	Deklarativ, Repository	Aktiv
utPLSQL	Frei	Open	PL/SQL Packages	Aktiv
ruby-plsql-spec	Frei	Open	Ruby DSL Dateien	Wenig aktiv
DbUnit	Frei	Open	Junit Extension, Java & XML Dateien	Aktiv
DbFit	Frei	Open	Fitnesse Wiki, HTML Dateien	Aktiv
PLUTO	Frei	Open	PL/SQL Object Types & Members	Inaktiv
PL/Unit	Frei	Closed	PL/SQL Packages	Inaktiv

Wir schauen uns im Folgenden näher die kostenlosen aktiven Frameworks SQL Developer, utPLSQL und ruby-plsql-spec an.

Testszenario

Bei allen 3 Sprachen versuchen wir soweit wie möglich folgendes Szenario zu verwenden:

- Anlegen eines Tests für eine noch anzulegende neue Package Prozedur (TDD)
- Implementieren der Prozedur und für den Test ausprogrammieren
- Weitere Tests um die Testabdeckung zu erhöhen
- Datenmodelländerung durchführen, die einen Test bricht
- Exception provozieren und Test dafür erstellen

- Life refactoring unter Beobachtung der Tests

Im Vortrag betrachten wir dann die Ergebnisse und Laufzeiten in einer Nutzwertanalyse.

SQL Developer

Architektur

- Ist Bestandteil der kostenlosen IDE Oracle SQL Developer
- Separates Datenbank Schema empfohlen
- Erzeugung von Public Synonymen
- Administrator und User Rollen
- Nicht-Invasiv (Test-Repository kann in anderer Datenbank liegen)

Know How

- PL/SQL
- SQL

Leistungsumfang

- Deklarative Anlage der Test-Suiten, Tests und Test-Implementierung
- Wiederverwendung von Routinen für Startup / Teardown / Validations über Libraries
- Dynamische Parametervergabe möglich
- Automatische Generierung von Test-Implementierungen per definierten Lookups
- Code Coverage
- Kommandozeilentool für die Testausführung: sdcli

ruby-plsql-spec

Allgemeines

- Basiert auf ruby-plsql und RSpec
- ruby-plssql 2008, ruby-plsql-spec 2009 erstellt von Raimonds Simanovskis
- Gepflegt bis 2015/2016

Architektur

- Ruby Runtime mit Gems (Installationspakete) für die Frameworks
- Abhängigkeiten von weiteren Frameworks und Betriebssystem Packages

- Testdefinition als Ruby-Textdatei
- Nicht-Invasiv (kein Code in der Datenbank)

Know How

- Ruby
- RSpec
- PL/SQL
- SQL

Leistungsumfang

- Konfiguration von 1..n Datenbankverbindungen
- Session-, User- und Datenbank-übergreifende Tests möglich
- Code Coverage
- Diverse Reporter Formate

utPLSQL

Allgemeines

- 2000 erstellt von Steven Feuerstein
- Gepflegt bis 2005
- 2014 wiederbelebt, einer Entwickler von ruby-plsql/ruby-plsql-spec an Bord
- Version 3 komplette Neuerstellung, viele Konzepte von ruby-plsql/ruby-plsql-spec übernommen

Architektur

- Separates Datenbank Schema empfohlen
- Erzeugung von Public oder Private Synonymen
- Test-Packages im Schema des zu testenden Objekts, damit Invasiv

Know How

- PL/SQL

Leistungsumfang

- Headless oder konfigurierbare Installation
- Skripte für Download / Entpacken, Infrastructure as Code möglich

- Upgrade = Deinstallation + Installation
- V2 und V3 sind parallel betreibbar
- Migration soll möglich sein, fkt. im Test nicht
- Test-Gruppierung über Annotations
- Kommandozeilentools über JAVA-CLI, SQL-CLI
- Code Coverage
- Diverse Reporter Formate

Nutzwertanalyse

Anhand einer Nutzwertanalyse begründen wir nach der Demonstration unsere Entscheidung für eines der genannten Frameworks anhand folgender Kriterien:

- Architektur
  - deklarativ im Repository kontra Code im VCS
  - Invasiv kontra non-invasiv
    - Datenbanken mit Testcode unterschiedlich von Produktion
    - Import von Produktionsdaten zerstört u.U. den Testcode
  - Wiederverwendbarkeit bei anderen Datenbanksprachen
- Knowledge
  - Sprachen: PL/SQL, SQL, Java, Junit, Ruby, RSpec, …
  - Buildtools: Maven, ANT, Gradle, …
  - CI, CD Server: Jenkins, Hudson, Bamboo, Travis, …
  - DevOps: Build, Deployment, Administration
  - Laufzeitumgebung / Tools
- Leistungsumfang
  - Handling Definition von Suiten, Tests und Testfällen
  - TDD
  - Exception Testing
  - Code Abdeckung
  - Branching

- Build Systeme
- CI/CD: Jenkins
- Testreporting

- Leistungsfähigkeit
  - Korrektheit der Test
  - Performance
  - Anzahl Connections
  - Laufzeiten
  - Testaufwand (Zeilen)

Kontaktadresse:

Torsten Kleiber

IKB Deutsche Industriebank AG

Wilhelm-Bötzkes-Straße 1

D-40474 Düsseldorf

Telefon: +49 (0) 12-345 6789

Fax: +49 (0) 12-345 6788

E-Mail torsten.kleiber@ikb.de

Internet: www.ikb.de