Nix vergessen?
Code Completion mit PL/SQL-Unit Tests

Stephan La Rocca
PITSS GmbH
Bielefeld

Schlüsselworte
SQL Developer, Datenbank-Entwicklung, Unit-Test, Code Completion

Einleitung
Unit Tests haben ihren festen Platz in der Java-Community - bei der PL/SQL-Datenbankentwicklung führen sie allerdings noch ein ungerechtfertigtes Nischendasein. Dabei hat der Oracle hauseigene und kostenfreie SQL Developer doch ein sehr gutes Plug-In für umfangreiche Komponententests.

SQL Developer
Der SQL Developer hat in den letzten Jahren mehr und mehr an Bedeutung gewonnen und als Nutzer der ersten Stunde ist es besonders erfreulich zu beobachten, dass von Release zu Release mehr Komponenten Einzug in diese kostenfreie und ohne jegliche Oracle Client-Installation notwendige Java Applikation

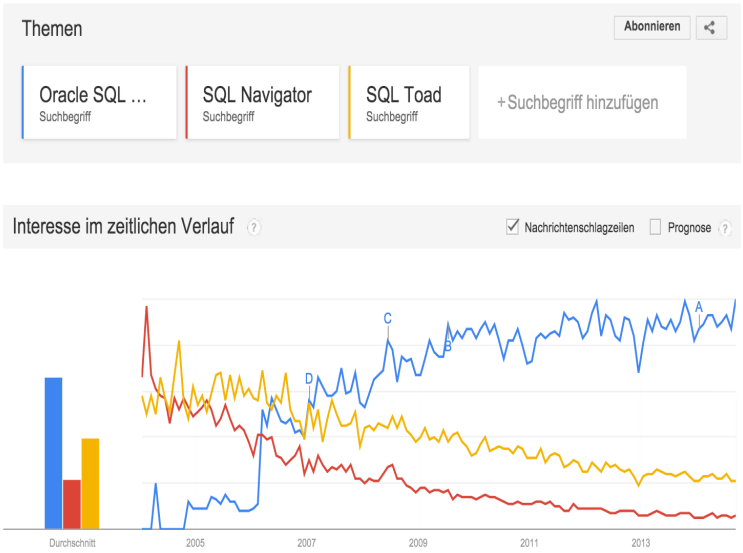

Abb. 1: Google Trend verschiedener PL/SQL Entwicklungswerkzeuge

Neben einer Vielzahl von vordefinierten Reports sind Module für den Datenbank-Administrator aber auch für den Software-Entwickler ergänzt worden. Der Vortrag setzt den Fokus auf die Komponente PL/SQL-Unit-Test, die ebenfalls kostenfrei genutzt werden kann.

Unit Tests
Unit Tests, teilweise in der Literatur auch als Komponententests oder Modultests bezeichnet, befinden sich in einer Hierarchie möglicher Testarten recht nahe am Source-Code. In der Regel wird evaluiert in den Stufen Komponenten-, Integrations-, Validierungs-, Akzeptanztest.
Unit-Tests decken in der Regel einzelne Proceduren und Funktionen auf atomare Ebene ab. Somit sind die verschiedenen Testszenarien leicht zu greifen und durch eine begrenzte Anzahl von Testfällen bereits eine nahezu vollständige Prüfung des Quellcode möglich.
Der Aufbau der Unit-Tests besteht dabei immer wieder aus den vier Phasen SETUP – RUN – ASSERT – TEARDOWN.
Im Setup werden die für den Test notwendigen Randbedingungen (z.B. Befüllen der Tabellen mit Testdaten) hergestellt. Anschließend wird in der RUN-Phase die zu prüfende Prozedure oder Funktion durchlaufen, um das Ergebnis im ASSERT mit den zu erwartenden Werten zu vergleichen. Die letzte Phase (TEARDOWN) sorgt dann dafür, dass das Environment identisch zur Ausgangssituation vor dem Test ist.
Gerade im Bereich der agilen Software-Entwicklung haben die Unit-Tests immer mehr an Bedeutung gewonnen, da es notwendig wird, geänderte Sourcen und Modelle immer wieder automatisiert testen zu können.

Komponententests in der Oracle-Datenbank
Der SQL Developer stellt mit senem Modul "Komponententest" alle Anforderungen für die Durchführung von Unit-Tests für Proceduren, Funktionen und Scripte sicher. Mit einiger "Phantasie" können darüber auch Shell/Batch-Scripte getestet werden.
Für die Administration der Unit-Tests sieht der SQL Developer ein eigenes Repository vor. Mit Hilfe eines Assistenten kann unter einem neuen User in der Datenbank ein solches Repository für die Komponententests angelegt werden. Auf die Struktur der damit angelegten Tabellen kommen wir später noch einmal zu sprechen.
Generell sind die Unit-Tests im SQL-Developer hierarchisch gegliedert, d.h. ein Unit-Test besteht aus mehreren möglichen Implementierungen (in der Regel unterschiedliche Kontexte, in denen die zu testende Prozedure oder Funktion aufgerufen wird) und kann selber wieder mit mehreren Unit-Tests zu einer Test-Suite zusammengefasst werden.
Dabei können auf unterschiedlicher Ebene getrennte Setup- und Tear-Down-Aktivitäten bestimmt werden.
Ein Anspruch des SQL-Developers ist es, dabei alle Informationen so pflegen zu lassen, dass keine zusätzliche Programmierung für das Erstellen, Durchführen und Analysieren der Tests notwendig ist.

Im folgenden sei kurz beschrieben, wie diese Schritte mit dem SQL-Developer erstellt werden können:

1.) Erstellen eines Tests
Der Wizard zum Erstellen eines Tests umfasst 6 Schritte:
1. Auswahl der Prozedur/Funktion
   Aus den implementierten Funktionen und Prozeduren innerhalb der Datenbank wird diejenige ausgewählt, für die ein Unit-Test erstellt werden soll
2. Testnamen vergeben
   Für eine spätere Identifikation innerhalb der Berichte und Analysen ist eine Benennung der Tests sinnvoll. Eine Namenskonvention mit Prefix oder Suffix bietet sich an.
3. Setup-Prozess auswählen
   Der Setup unterscheidet zwischen einem Tableset-setup, d.h. sie kopieren aus einer Vorlage eine neue Tabelle oder einzelnde Datensätze, oder einer eigenständigen PL/SQL-Prozedure für das Setup.
4. Parameter erfassen
   Wird die Prozedur mit Parametern aufgerufen, geben Sie in diesem Schritt alle notwendigen Parameterwerte ein. In der Regel wird eine Prozedur später mit mehreren

unterschiedlichen Werten aufgerufen, die jeweils eine eigene Testimplementierung datstellen.
Sie können auch Parameter erfassen, die zu einer Exception führen sollen und definieren die Art des zu erwartenden Fehlers.

5. Validierung angeben
   Eine Prozessvalidierung kann auf unterschiedlicher Ebene stattfinden. Vom einfachen Wertvergleich, über die Bestimmung des Ergebnisses eines Select-Statements bis hin zu einer eigenen Validierung in Form einer Prozedur sind alle Varianten möglich.
6. Teardown
   Der Teardown als Pendant zum Setup unterstützt gleichfalls ein setbasiertes Aufräumen durch Löschen einzelner Datensätze oder ganzer Tabellen oder auch die Verwendung eigener Prozeduren.

2.) Zusammenfassen von Tests zu einer Test-Suite
Um z.B. alle Funktionen einer Package zu testen, bietet es sich an, alle einzelnen Tests zu einer Test-Suite zusammenzufassen. Suites bekommen dabei einen sprechenden Namen und können hierarchisch zusammengefasst werden, d.h. Suites können selbst wiederum Bestandteil einer anderen Suite sein.

3.) Starten einer Test-Suite
Der Aufruf einer Suite (im Einzelfall natürlich auch eines einzelnen Tests) kann interaktiv oder programmatisch erfolgen.

1. Innerhalb des SQL Developers
   Im SQL-Developer gibt es die Möglichkeit, die erstellten Tests und Suiten direkt zu starten. Alle Log-Ausgaben und das Ergebnis sind dann sofort sichtbar.
2. Im Hintergrund zeitgesteuert oder innerhalb eines Build-Prozesses
   Für die regelmäßige Verwendung z.B. innerhalb eines nightly Builds kann der Aufruf auch über die Kommandozeile (Windows/Linux) erfolgen. Die Parameter der Kommandozeile erlauben dabei die rudimentären Konfigurationseinstellungen.

4.) Analyse der Test-Suite

1. Durch die vordefinierten Berichte
   Innerhalb des Moduls existieren bereits 9 Berichte fest verankert, die einen umfassenden Bericht über die durchgeführten Testläufe geben.
2. Durch eigene Analysen
   Sollte das allerdings einmal nicht reichen, kann auf dem View-Layer des Moduls natürlich auch direkt zugegriffen werden. Das Datenmodell ist überschaubar und sehr schnell zugänglich.

Code Completion
Von Hause bietet der SQL Developer auch einen Blick auf das Code Completion, d.h. welche Zeilen sind in meinen zu testenden Source Code auch tatsächlich durchlaufen worden. Bei einer sehr großen TestSuite fällt es aber schwer, hier im Umkehrschluss herauszufinden, welche Codezeilen habe ich vergessen zu testen. Dazu müsste der Entwickler tatsächlich jede einzelnen Test aufrufen und scannen.

Um diesen Schwachpunkt aber zu umgehen, können Sie sich einen Blick auf das Repository und die darin enthaltenen Views gönnen. Über die Modellierung von Testimplementierungen, Test-Ergebnissen und Test-Coverage finden Sie sehr schnell heraus, dass in diesem Repository jede getestente Zeile in der Datenbank referenziert wird. Mit ein paar selbst erstellten Funktionen und einer kompletten View:

```
CREATE OR REPLACE FORCE VIEW "UNITTEST"."UT_INCOMPLETE_TEST_RUNS"
("Unit_Owner", "Unit_Name", "Test_Name", "Total_Lines", "Uncovered Lines",
"Lines_Covered", "Run_Date", "Run_Id", "STATUS") AS
  select
                        s.unit_owner as "Unit_Owner",
```

```
                        s.unit_name as "Unit_Name",
                        tr.name as "Test_Name",
                        count(*) as "Total_Lines",
                        (select count(1) from
table(get_codeuncomplete(s.unit_name, tr.utr_id))) as "Uncovered Lines",
                        count(all decode(s.total_occur + s.total_time, 0,
null, 1)) as "Lines_Covered",
                        tr.run_date as "Run_Date",
                        tr.utr_id as "Run_Id",
                        tr.status as status
                    from
                        ut_test_results tr,
                        ut_test_impl_results tir,
                        ut_test_coverage_stats s
                    where
                        tr.utr_id = tir.utr_id and
                        tir.utir_id = s.utir_id and
                        s.unit_name is not null
                        and (select count(1) from
table(get_codeuncomplete(s.unit_name, tr.utr_id))) >0
                    group by
                        tr.utr_id,
                        tr.run_date,
                        tr.name,
                        s.unit_owner,
                        s.unit_name,
                        tr.status
                    order by
                        1, 2, 3, 6 desc;
```

können Sie somit schnell im Umkehrschluss alle Zeilen identifizieren, die durch ihre Test-Läufe nicht abgedeckt sind.

Ganz nebenbei protokolliert die Test-Coverage auch die benötigte Zeit pro Zeile, so dass Sie auch noch die veränderte Performance innerhalb ihrer Test-Läufe ermitteln können.

Kontaktadresse:
Stephan La Rocca
PITSS GmbH
Otto-Brenner-Str. 209
D-33604 Bielefeld

Telefon: +49 (0) 521-546 795-07
Fax: +49 (0) 521-546 795-01
E-Mail slarocca@pitss.de
Internet: www.pitss.de