Von Forms nach APEX

Michael Tietz
imining gmbh
München

Schlüsselworte:

Forms, View, Instead-Of-Trigger, Reports, OLE2, TEXT_IO, HOST, APEX, Apache FOP

Einleitung

Eine Projektstudie über eine teilautomatisierte Ablöse einer in den ersten Anfängen aus dem Designer generierten Forms Anwendung. Ziel der Studie war es, aus einer echten Forms Applikation typische Programmierweisen in Forms zu identifizieren, einen Migrationsleitfaden für die Ablöse zu erstellen und die Automatisierbarkeit bestimmter Migrationsabläufe nachzuweisen. Des weiteren sollte der Migrationsleitfaden für die spezifischen Merkmale der Applikation (Ausgabe nach Word und Excel, Speichern von Daten als Datei, Druckausgabe von Reports und Einbindung von Client Software) eine Lösungsmöglichkeiten präsentieren.

Die Logik in PL/SQL in der Datenbank

In der Theorie: Transfer der PL/SQL Logik in die Datenbank Packages. Zugriff über Views, Instead Of-Trigger.
In der Praxis: Datenbank Design für Forms Applikationen ist in der Vergangenheit häufig ohne numerische Schlüssel oder auch ohne eine bestimmte Spalte als eindeutiger Schlüssel ausgekommen. Vielfach wurden ‚Null' – able Foreign Keys in Master Detail Beziehungen von der Logik abhängig zugelassen. Hier sind Vorarbeiten im Schema notwendig. Ids in Master Tabellen werden über Trigger gefüllt, die Ids der Master Tabellen müssen in Detail Tabellen im PL/SQL Code hinzugefügt werden. Views, Instead-Of-Trigger und Datenbank Packages können mit PL/SQL Mitteln generiert werden:

Code Beispiel für Generierung von Views:

```
src                CLOB;
...
src := REPLACE (src, v_view_name, v_table_name) || CHR (13) || 'where 1=0';
v_cur_hdl := DBMS_SQL.open_cursor;
DBMS_SQL.parse (v_cur_hdl, src, DBMS_SQL.native);
v_rows_processed := DBMS_SQL.EXECUTE (v_cur_hdl);
DBMS_SQL.describe_columns (v_cur_hdl, v_col_cnt, desc_t);
c := 0;
FOR j IN 1 .. v_col_cnt
LOOP
  v_columns (j) := LOWER (desc_t (j).col_name);
  IF UPPER (v_columns (j)) = v_table_name || '_ID' THEN
     c := 1;
  END IF;
```

```
END LOOP;
DBMS_SQL.close_cursor (v_cur_hdl);
...
```

Die Logik aus Forms extrahieren

In der Theorie: Form Builder mit Form parallel nutzen oder fmt Datei erzeugen
In der Praxis: Für kleine Forms mit dem Form Builder ausreichend. Für große Forms mit Library Zugriffen haben wir mit Tool jform (auch mit C-API oder JDAPI möglich) zum Einen eine Block – Item Darstellung in Form von Master-Detail Tabellenpaar erzeugt und hier alle für die Migration wichtigen Eigenschaften aus Forms hinterlegt. Zum Anderen sollte der PL/SQL Code so aufbereitet werden, dass die Business Logik also z.B. Package Aufrufe zur Prüfung von Controller Logik also z.B. go_item separiert dargestellt werden kann. Dazu wurden Build-Ins im PL/SQL Code ausblenden und kaskadierender PL/SQL Code an Triggern aufgelöst, d.h. eine Funktion durch der Code der gerufenen Funktion ersetzt. Das Verhalten kaskadierend zündende Trigger konnte nicht berücksichtigt werden.

Beispiel für das Expandieren von PL/SQL Code:

```
10) Form Trigger PRE-UPDATE:
    -----------------------------------
    - Subclassed: Not
    - Style: Trigger based on PL/SQL blocks

    /* Checked to delete in log:
        I_NEW_FORM_ACTION;
    */

    test_commit;
      {CALL LEVEL 1:
      procedure test_commit is
      begin
        test_auktko;

        {CALL LEVEL 2:
        procedure test_auktko is
        begin
          /* Prozedur prueft qualifizierte Eingabe */
          begin
            :global.return_code := ab0022_p.ab0022_test(:auktko.arb_ord_nr
                                                        ,:auktko.arb_ord_zae
                                                        ,:auktko.knot_nr
                                                        ,'');
          exception
          when others
          then
            i_exception;
          end;
        end;
        }
      end;
      }
```

Die Validierung mit Datenbank seitigem PL/SQL

In der Theorie: Nutzung von User Defined Exceptions
In der Praxis: Die Prüf Logik in der Datenbank zieht nur bei der Validierung. Durch User Defined Exception lassen sich Fehlerausgaben realisieren, Verhalten wie Cursor in bestimmten Feldern zu positionieren oder automatisch zu navigieren ist nicht vorgesehen. In Forms Applikationen ist die Prüflogik häufig 2 Mal ausprogrammiert, einmal am Item um den Anwender ein möglichst schnelles Feedback zu geben, ein zweites Mal vor der Commit Steuerung. Natürlich lassen sich Validierungen aus LOVs oder statische Pflichtfelder wieder über die Möglichkeiten der Oberfläche abbilden, ein in Forms programmierter ‚Komfort' ist aber wieder nur durch Programmierung zu erreichen. Die Nutzung von Alerts (Ja/Nein) zur Programmverzweigung kann nur unzureichend über Code Reentrance mit Parametern in der Datenbank abgebildet werden.

Übertrag der oben gezeigten Logik in die Datenbank

```
PROCEDURE check_update (old_rec IN OUT tab_rec, new_rec IN OUT tab_rec)
IS
 ret number;
 BEGIN

  test_for_nulls(new_rec);

  if old_rec.arb_ord_nr <> new_rec.arb_ord_nr or
     old_rec.arb_ord_zae <> new_rec.arb_ord_zae or
     old_rec.knot_nr <> new_rec.knot_nr
  then
    fatrg.raise_error_nr('9001','AUKTKO_VI','ARB_ORD_NR');
  end if;

  test_values(new_rec);
  ret := ab0022_p.ab0022_test (new_rec.arb_ord_nr
                              ,new_rec.arb_ord_zae
                              ,new_rec.knot_nr
                              ,'');
  IF ret <> 0
  THEN
    fatrg.e_fcall_error ('AB0022', ret, 'AUKTKO', 'ARB_ORD_NR');
  END IF;
END;

PROCEDURE upd(old_rec in out TAB_REC, new_rec in out TAB_REC)
 IS
 BEGIN
  check_update(old_rec, new_rec);
  UPDATE AUKTKO
     SET txt_nr       = new_rec.txt_nr,
         txt_bez      = new_rec.txt_bez
   WHERE AUKTKO_ID=old_rec.auktko_id;
 EXCEPTION
 WHEN fatrg.cannot_update_null OR fatrg.check_constr_violated OR
fatrg.unique_constr_violated OR fatrg.parent_not_found THEN
   fatrg.handle (SQLCODE, SQLERRM, view_name);
 END;
```

Das Forms Migrationswerkzeug in APEX

In der Theorie: Das Migrationswerkzeug erstellt eine Übersicht über die Module und Objekte als ‚Leitfaden zur manuellen Bearbeitung', automatisch migriert wird wenig:
Ein Tabellen oder View basierter Block und seine Item, also eigentlich die über einen Block realisierte Query, wird in eine Region abgebildet.
Record Group basierende LOVs werden als Query für eine LOV abgebildet.
Ein Editor auf einem Textitem wird als HTML Editor hinterlegt.
Post-Query Trigger werden als ‚Enhanced Querys' abgelegt.
In der Praxis: Probleme mit dem Migrationswerkzeug bei kaskadierenden PLLs und referenzierten Objekten aus Objekt Gruppen aus OLBs. Zur Laufzeit in Forms eingeblendete Objekte eines ‚Work Blocks', z.B. dynamisch positionierte Buttons erscheinen ‚willkürlich' in verschiedenen Regionen und führen zu Fehlern, da es sich dabei sicher nicht um ‚Basetable' Items handelte.

Nutzung von konvertierten Masken Layouts

In der Theorie: Zugriff mit konvertierten Forms Layouts auf die Views in der Datenbank
In der Praxis: Layouts werden mit Layout Wizzard schneller erzeugt. Probleme entstehen beim Zugriff auf Views, diese werden im Wizzard nicht angeboten. Das kann umgangen werden, indem man mit dem Layout Wizzard auf die Tabelle geht und später den Zugriff auf den View korrigiert. Die Nutzung von ‚Enhanced Querys' als Abbildung von Post-Query Triggern ist in diesem Zusammenhang eher ein Forms Relikt. In Forms war das eine performantere Lösung als ein View mit einem Join, bzw. eine einfache Lösung für Anzeigefelder. Da jetzt bereits ein Datenbank View vorhanden ist, kann dieser auch gleich als ‚Function View' ausgeprägt werden.

Das Auswertung mit Reports

In der Theorie: Reporting Möglichkeiten mit APEX
In der Praxis: Reports in der Applikation mehr als Listings denn als Dokumente mit dynamischem Layout zu sehen. Soweit wie möglich wurde die ‚Druckvorbereitung' durch Bereitstellung von Tabelleninhalten in der Datenbank realisiert und basierend auf diesen Tabellen ein neuer Report erzeugt.

Die Auswertung mit Word und Excel (OLE2)

In der Theorie: Word und Excel wurden als ‚Druckvorbereiter' genutzt, um mit Makros die Daten zu positionieren und das generierte Dokument manuell vom Anwender nachzubearbeiten.
In der Praxis: Durch die Brust ins Auge Technologie: Textdatei mit Daten erzeugen und mit Word Makros weiterverarbeiten oder mit Apache Formating Objects Processor RTF und XML Dokumente erzeugen. Fehler bei den RTF Dateien führten dazu, dass doch mit der zuerst genannten Technik gearbeitet wurde, da hier auch ausreichend Know How vorhanden war. Der BI Publisher war wegen der Lizenz Frage keine Alternative.

Fazit

Forms Applikationen, wie die hier in Teilen gezeigte Adressverwaltung lassen sich mit APEX realisieren, wenn Layout und Bedienverhalten abweichen darf und nicht zu 100 Prozent 1:1 nachempfunden werden muss. Die Microsoft Office Thematik lässt sich nicht verdrängen, wobei die

Lösung mit Forms und Word auch vorher schon frei von Architektur und Design war. Für Drucke mit dynamischen Layouts wie aus Reports bekannt wird man wohl noch auf andere Reporting Produkte zurückgreifen müssen.

Kontaktadresse:

Michael Tietz
imining gmbh
Berduxstr. 22
D-81245 München

Telefon:	+49 (0) 89-9230 663-21
Fax:	+49 (0) 89-9230 663-1
E-Mail	Michael.Tietz@imining.de
Internet:	www.imining.de