Fortgeschrittene SQL-Techniken für APEX-Formulare und -Reports

Andreas Wismann
WHEN OTHERS
D-41564 Kaarst

Schlüsselworte
APEX 4.2, SQL, Tabular Forms, Classic Report, Interactive Report

Einleitung
"Darf es noch ein wenig mehr sein?" Anhand von fünf Praxisbeispielen wird in der Präsentation gezeigt, wie Sie in Oracle Application Express mit erweiterten SQL-Techniken mehr aus Ihren Eingabemasken und (Interactive) Reports herausholen können. Abfragetechniken wie Connect By, Sliding Windows, Pivot/Unpivot, Model Clause und Regular Expressions stellen vielfältige Anwendungsmöglichkeiten zur Verfügung, mit denen sich die Ergebnisse verbessern und auch die Entwicklungszeiten verkürzen lassen. Der Vortrag richtet sich an Entwickler, die bereits mit der Erstellung von Formularen und Reports in APEX vertraut sind und ihre "Schlagfertigkeit" beim Erstellen von SQL-Abfragen verbessern möchten.

Prinzipielle Überlegung: APEX möglichst effizient nutzen
Einer der wesentlichen Vorteile von Application Express ist der schnelle Entwicklungszyklus. Innerhalb weniger Minuten kann ein interaktiver Report oder ein gebrauchsfertiges Formular erstellt sein. Dies setzt jedoch voraus, dass APEX mit „geradlinigen“ Daten gefüttert werden kann. Je mehr „Sonderwünsche“ in APEX selbst implementiert werden müssen, desto seltener können die Assistenten („Wizards“) verwendet werden, desto länger dauert die Programmierung. Es sollte darum angestrebt werden, Komplexität von den APEX-Masken und Reports fernzuhalten, stattdessen möglichst viele Teile der Geschäfts- und Präsentationslogik, möglichst sauber voneinander getrennt, in die Ressourcen der Datenbank zu verlagern.

Dieses Vorgehen steigert auch die Wartbarkeit Ihrer Anwendung. Erstens, weil sich die Entwickler um die Programmierung klar voneinander getrennter Teilaufgaben kümmern können. Sonst wird vielleicht jemand, der sich (nur) mit der HTML-Formatierung Ihres Reports auskennt, an demselben SQL-Statement „herumschrauben“, in dem Sie Ihre komplexe Geschäftslogik kodiert haben. Zweitens, weil Sie dadurch die Zentralisierung und die Wiederverwendbarkeit Ihres Quellcodes steigern.

Views mit INSTEAD-OF-Triggern
Glücklich sind APEX-Entwickler, wenn sie die Daten für eine Eingabemaske aus seiner einzigen Tabelle holen und auch wieder dort abspeichern können, weil das mit den APEX-Wizards sehr schnell von statten geht! Es dürfte bereits bekannt sein, dass ein voluminöses SQL-Statement mit diversen Join-Bedingungen und Filtern im Programmierfenster eines Interaktiven Reports nicht besonders gut aufgehoben ist. Stattdessen möchte man lieber eine entsprechende View in der Datenbank formulieren und diese wiederum in einem sehr einfachen SQL-Statement abfragen.

Davon ausgehend besteht die Möglichkeit, mehr als nur „passive“ Views zu verwenden. In Oracle können DML-Trigger nämlich nicht nur über Tabellen, sondern auch über (beinahe beliebig komplexe) Views erstellt werden. Diese so genannten „INSTEAD-OF-Trigger“ behandeln dann das Verhalten der Anwendung, wenn Insert-, Update- oder Delete-Befehle per SQL gegen diese View ausgeführt werden -- im Prinzip wie ein kleines PL/SQL-Programm, das man ansonsten vielleicht in APEX schreiben würde.

Die Vorgehensweise:

```
CREATE OR REPLACE TRIGGER ii_meine_view
INSTEAD OF INSERT ON meine_view
FOR EACH ROW
BEGIN
  -- Beispiel:
  INSERT INTO meine_tabelle_1 VALUES (…);
  UPDATE meine_tabelle_2 SET … WHERE … ;
  if … then
    DELETE FROM meine_tabelle_3 WHERE … ;
  end if;
  -- weiterer PL/SQL-Code …
END;
```

Für Views existieren die Trigger-Arten

- INSTEAD OF INSERT
- INSTEAD OF UPDATE
- INSTEAD OF DELETE

Für APEX sieht eine solche View im Prinzip aus wie eine herkömmliche Tabelle. Da eine ROWID-Pseudospalte bei Views für DML-Operationen nicht aussagekräftig ist, muss mindestens eine echte Primary-Key-Spalte (im Fall von APEX: maximal zwei PK-Spalten) n der View verwendet werden, um die volle Funktionalität zu gewähren.

Ein Nachteil dieser Methode sollte nicht verschwiegen werden: Von APEX automatisch erzeugte Validierungen in Tabular Forms können bei der Verwendung von Instead-Of-Triggern zu Laufzeitfehlern führen, da sie auf der ROWID basieren. Man sollte sie nach der Erstellung der Tabular Form nachträglich löschen.

Ein weiterer Vorzug der INSTEAD-OF-Trigger: Mit ihnen können feingranulare Datenzugriffs-Berechtigungen abgebildet werden, beispielsweise wenn Benutzer(-gruppen) nur bestimmte Wertebereiche verwenden oder ausgewählte Spalten bearbeiten dürfen. Ein Anwendungsfall wird im Vortrag präsentiert.

CONNECT-BY-Abfragen
Formulare werden oft verwendet, um Termine zu pflegen. Möchte man einen Kalender auf Basis einer Tabular Form implementieren, in dem die Benutzer auch weit in der Vergangenheit oder in der Zukunft liegende Termine eintragen können, so müsste man im Prinzip einen weitestgehend leeren „hundertjährigen Kalender“ in einer Tabelle hinterlegen. Es gäbe dann pro Tag innerhalb der nächsten +/- 50 Jahre jeweils einen Datensatz, was zu 36500 Datensätzen führt (plus ein paar für die Schaltjahre), von denen die meisten dennoch nur zur Darstellung leerer Seiten nötig wären. Benötigt man zusätzlich eine zeilenweise Unterteilung nach Stunden wie in einem Notizbuchkalender, so käme man bereits auf fast eine Million Datensätze, die nur der hübschen Formatierung dienen. Denn faktisch interessieren ja nur die tatsächlich „belegten“ Termine. Zudem wäre man auf einen bestimmten Zeitrahmen festgelegt (der durch den zeitlich ersten und letzten Datensatz repräsentiert wird).

Wie lässt sich das Gerüst für einen solchen Kalender in APEX realisieren, ohne Daten-Ballast zu generieren? Per SQL recht elegant, mit dem Connect By-Befehl sowie den im vorigen Abschnitt erläuterten INSTEAD-OF-Triggern. Führen Sie diese Abfrage am SQL-Prompt aus:

```
SELECT datum FROM (
  SELECT TRUNC(SYSDATE) + LEVEL - 1 AS datum
    FROM DUAL
 CONNECT BY LEVEL < 100 * 365
)
WHERE ROWNUM <= 7
```

Als Ergebnis erhalten Sie 7 Zeilen, und zwar beginnend mit dem heutigen Kalendertag, für die kommenden 7 Tage. Weshalb das? Durch die Angabe von CONNECT BY starten Sie quasi eine Schleifenverarbeitung in der Abfrage, welche so viele Datensätze ausliefert, wie Sie spezifizieren. Die Pseudospalte LEVEL ist sozusagen der „Schleifenzähler“ und wird pro Datensatz um 1 erhöht, beginnend mit 1. Mit der Berechnungsvorschrift in der CONNECT BY-Klausel legen Sie fest, unter welchen Bedingungen weitere Datensätze erzeugt werden. In diesem Fall also genau so viele, um 100 Jahre darzustellen. Sie holen aber nur die ersten 7 Zeilen ab, was im äußeren SELECT durch den ROWNUM-Filter erreicht wird.

In APEX (und vielen anderen SQL-Editoren) kann die äußere ROWNUM-Einschränkung wegfallen, weil der Client je nach Einstellung sowieso nur eine bestimmte Anzahl von Datensätzen aus der Datenquelle ausliest. APEX-Formulare und -Reports holen üblicherweise 15 Zeilen ab.

Nun haben Sie ein SQL-Statement, mit dem Sie im Prinzip einen Kalender aufbauen können. Als nächstes wollen Sie den Offset bei der Datumsberechnung anpassen, um jeweils 50 Jahre in die Vergangenheit und in die Zukunft schauen zu können:

```
SELECT TRUNC(SYSDATE) + LEVEL – 1 – 50 * 365 AS datum
```

Wenn Sie nun in einer zweiten Tabelle lediglich die vorhandenen Termine speichern, können Sie die das Datums-SQL mit der Termintabelle per Outer Join verknüpfen und das Ganze zu einer View verschmelzen:

```
CREATE TABLE termine (
  termin_id NUMBER NOT NULL PRIMARY KEY
 ,datum DATE NOT NULL
 ,titel VARCHAR2(1000) NOT NULL
);

CREATE SEQUENCE TERMINE_SEQ NOCACHE;

CREATE OR REPLACE VIEW TERMINE_VIEW AS
SELECT kalender.datum
     , termine.termin_id
     , termine.titel
  FROM (
        SELECT TRUNC(SYSDATE) + LEVEL - 1 - 50 * 365 AS datum
          FROM DUAL
       CONNECT BY LEVEL < 100 * 365
  )    kalender
  LEFT OUTER JOIN termine
    ON (kalender.datum = termine.datum)
;

INSERT INTO termine (termin_id, datum, titel)
VALUES (termine_seq.nextval, TRUNC(SYSDATE), 'Bereichsmeeting');

INSERT INTO termine (termin_id, datum, titel)
```

```
VALUES (termine_seq.nextval, TRUNC(SYSDATE) + 4, 'Teambesprechung');
```

Somit können Sie Ihren Kalender für die nächste Woche mitsamt den vorhandenen Terminen bequem abfragen:

```
SELECT *
  FROM TERMINE_VIEW
 WHERE DATUM BETWEEN TRUNC(SYSDATE) AND TRUNC(SYSDATE) + 6
 ORDER BY datum, termin_id
```

DATUM	TERMIN_ID	TITEL
12.09.2013	1	Bereichsmeeting
13.09.2013	-	-
14.09.2013	-	-
15.09.2013	-	-
16.09.2013	2	Teambesprechung
17.09.2013	-	-
18.09.2013	-	-

Nun schreiben Sie drei INSTEAD OF-Trigger für die View:

```
CREATE OR REPLACE TRIGGER ii_termine_view
INSTEAD OF INSERT ON termine_view
FOR EACH ROW
BEGIN
  INSERT INTO termine (termin_id, datum, titel)
  VALUES (termine_seq.nextval, :NEW.datum, :NEW.titel);
END;
/

CREATE OR REPLACE TRIGGER iu_termine_view
INSTEAD OF UPDATE ON termine_view
FOR EACH ROW
BEGIN
  UPDATE termine
     SET termin_id = :NEW.termin_id -- wird im Vortrag diskutiert
       , datum     = :NEW.datum
       , titel     = :NEW.titel
   WHERE termin_id = :OLD.termin_id;
END;
/

CREATE OR REPLACE TRIGGER id_termine_view
INSTEAD OF DELETE ON termine_view
FOR EACH ROW
BEGIN
  DELETE FROM termine
   WHERE termin_id = :OLD.termin_id;
END;
/
```

Um zu gewährleisten, dass die Trigger funktionieren, testen Sie sie:

```
INSERT INTO termine_view (datum, titel)
VALUES (TRUNC(SYSDATE) + 5, 'Audit');

SELECT * FROM termine; -- Zugriff auf die Tabelle genügt als Test

UPDATE termine_view
   SET titel = 'Audit IT'
 WHERE titel = 'Audit';

SELECT * FROM termine;

DELETE FROM termine_view
 WHERE titel = 'Audit IT';

SELECT * FROM termine;
```

Sie werden sehen, dass die View wie eine Tabelle auf Ihre DML-Befehle reagiert. Den letzten Schritt, daraus ein tabellarisches Web-Formular zu erstellen, erledigen Sie im Handumdrehen mit Hilfe der Standard APEX-Assistenten. Als Primary Key-Feld geben Sie den Primärschlüssel der Tabelle TERMINE an. Den Datumsbereich, der jeweils angezeigt werden soll, filtern Sie dabei auf die übliche Art und Weise, beispielsweise mit einem versteckten Item.

Weitere fortgeschrittene SQL-Techniken für APEX-Formulare und -Reports, die ich in meinem Vortrag am Dienstag, 19.11.2013, ab 15:00 Uhr in Raum 13 vorstellen werde:

- Pipelined Functions
- Result Cache
- SQL Model Clause

Dieses Manuskript, die Präsentationsfolien sowie alle Beispiel-Quellcodes können Sie im Anschluss an die DOAG 2013 auf meiner Webseite herunterladen:

http://when-others.com/download/doag/2013

Kontaktadresse:

Andreas Wismann
WHEN OTHERS
Hirschstraße 10
D-41564 Kaarst

Telefon: +49 (0) 2131-314 9966
E-Mail: wismann@when-others.com
Internet: www.when-others.com