Interactive Report zu Excel und zurück

Henner Hucke
DATAGROUP Enterprise Services GmbH
Dahlewitz

Schlüsselworte
APEX, Interactive Report, Excel

Einleitung
Der Interaktive Report in APEX ist eines der Hauptargumente für den Einsatz von APEX im Unternehmen. Die zahllosen Möglichkeiten dieses Tools Daten zu filtern, zu sortieren, zu gruppieren und darzustellen sollten eigentlich dazu führen, dass für diesen Zweck der Einsatz anderer Tools nicht mehr in Betracht gezogen wird. Es gibt aber immer wieder Fälle, in denen Daten im Excel Format für die Weiterverarbeitung aus APEX exportiert bzw. importiert werden müssen. Der Export der Daten sollte dann nach Möglichkeit so erfolgen, dass sich die vom Anwender eingestellten Filter / Sortierungen auch in der exportierten Excel Datei widerspiegeln. Die Nutzung des CSV-Exports führt in der Regel zu guten Ergebnissen. Es gibt leider Fälle, bei denen der Excel-Import-Filter die importierten Daten verfälscht. Hier wäre ein typgerechter Export direkt in das XLSX-Format hilfreich. Der Import von Excel-Daten in eine APEX Applikation ist ebenfalls oft vom Anwender gewünscht. Es muss also ein Weg gefunden werden, die Datei in die Datenbank zu laden und zu verarbeiten. Als Nebenbedingung sollen neben der Oracle Datenbank keine weiteren Komponenten verwendet werden. Hier boten sich die PL/SQL Packages `AS_XLSX` und `AS_READ_XLSX` von Anton Scheffer an. Diese erlauben das Lesen und Schreiben von Excel Dateien in der Datenbank. Tests zeigten, dass mittels der Packages Excel Dateien erzeugt oder gelesen werden können. Somit musste nur noch ein Weg für die Umsetzung der „WYSIWYG“ Anforderung gefunden werden. Glücklicherweise bietet Oracle APEX mit dem Package `APEX_IR` ein Tool für diesen Zweck an. Jetzt waren alle Komponenten für eine Umsetzung der Anforderungen beisammen und es konnte mit der Umsetzung begonnen werden.

Lösungsansatz Export
Die notwendigen Schritte für den Export der Daten eines Interactive Report in eine Excel Datei sind:

- SQL mit Sortierung und Filterkriterien aus dem Interaktiven Report extrahieren
- Ergebnismenge des SQL in XLSX Binary Large Object umwandeln
- Download des entstandenen BLOBs als Datei aus APEX heraus

SQL des Interactive Report
Die APEX-Installation in der Datenbank enthält das Package APEX_IR enthält die benötigte Funktionalität in der Funktion `get_report`. Der Aufruf von `apex_ir.get_report` liefert in der Variable vom Typ `t_report` das verwendete SQL Statement inklusive der verwendeten Sortierungen und Filterkriterien zurück. Die Filter sind als Bindevariablen im SQL-Statement aufgeführt. Die aktuell verwendeten Werte für die Filter befinden sich in der PL/SQL Tabelle `binds` vom Type `t_bind_list`. Mit diesen Informationen kann man das SQL-Statement noch einmal ausführen und erhält das gleiche Ergebnis so wie es auch im Interactive Report angezeigt wird.

```
function get_report (
    p_page_id        in number,
    p_region_id      in number,
    p_report_id      in number default null,
    p_view           in varchar2 default c_view_report
```

```
)
return t_report;
```

Der Typ `t_report` ist folgendermaßen definiert:

```
type t_report is record (
  sql_query     varchar2(32767),
  binds    wwv_flow_plugin_util.t_bind_list
);

type t_bind_list is table of t_bind index by pls_integer;

type t_bind is record (
    name         varchar2(30),
    value        varchar2(32767)
);
```

Um diese Funktion zu verwenden, benötigt man neben der ID der Apex-Seite (page_id) noch die ID der Region und die ID des Reports. Die ID der Region kann man mit Hilfe der View `apex_application_page_regions` bestimmen:

```
SELECT region_id INTO l_region_id
FROM apex_application_page_regions
WHERE application_id = p_app_id
AND page_id = p_page_id
AND static_id = p_static_id
AND source_type = 'Interactive Report';
```

Die ID des Reports lässt sich dann mittels der Funktion `apex_ir.get_last_viewed_report_id` feststellen:

```
l_report_id := apex_ir.get_last_viewed_report_id (
     p_page_id => p_page_id,
     p_region_id => l_region_id
);
```

Umwandlung in XLSX

Für die Erstellung von Exceldateien auf Basis von Abfragen an eine Oracle Datenbank gibt es viele unterschiedliche Werkzeuge. Allerdings gab es im Projekt eine Nebenbindung, es sollte neben APEX und der Datenbank kein zusätzliches externes Tool verwendet werden. Somit konnten Produkte wie der BI-Publisher von Oracle nicht genutzt werden. Diese Nebenbedingung würde der Einsatz der Apache POI Bibliotheken als Java Stored Procedure in der Datenbank erfüllen, nur leider konnten diese Bibliotheken mit all ihren Abhängigkeiten nicht erfolgreich in die Datenbank geladen werden. Eine Recherche im Internet führte zu dem Blog von Anton Scheffer auf welchem er die von ihm entwickelten PL/SQL Packages AS_XLSX und AS_READ_XLSX vorstellt. Das Package AS_XLSX bietet die geforderte für den Export notwendige Funktionalität und ließ sich erfolgreich in die Datenbank laden. Die Prozedur `as_xlsx.query2sheet` führt ein als Parameter übergebenes SQL-Statement aus und speichert die Ergebnismenge in von PL-SQL Tabellen ab. Die Funktion `as_xlsx.finish` wandelt dann das Abfrageergebnis aus den PL-SQL Tabellen in das XLSX Format um und gibt es als BLOB zurück. Für die Verwendung mit dem Interactive Report wurde eine

Überladung der Prozedur `as_xlsx.query2sheet` implementiert, welche einen Parameter vom Typ `apex_ir.t_report` akzeptiert. Die einzige nennenswerte Änderung am bestehenden Code von Anton Scheffer besteht darin, dass SQL-Statement aus der Variable vom Typ `apex_ir.t_report` zu parsen und ein einer Schleife die Bindevariablen des SQL-Statements mit den Werten der PL/SQL Tabelle aus dem `apex_ir.t_report` zu versehen.

```
PROCEDURE query2sheet(p_report apex_ir.t_report) AS
...
  t_c       INTEGER;
...
BEGIN
  -- SQL aus IR entnehmen, Binds versorgen
  -- und dann weiter mit Code von Anton Scheffer
  as_xlsx.new_sheet;
  t_c := dbms_sql.open_cursor;
  dbms_sql.parse(t_c, p_report.sql_query, dbms_sql.NATIVE);
  -- die Binds versorgen
  FOR i IN 1 .. p_report.binds.COUNT LOOP
    dbms_sql.bind_variable(t_c, ':' || p_report.binds (i).NAME,
                           p_report.binds (i).VALUE
    );
  END LOOP;
...
```

Download der Exceldatei

Der letzte Schritt ist der Download des in der Datenbank erzeugten Binary Large Objects, welches den Inhalt einer Excel Datei entspricht, zum PC des Benutzers. Dies kann wiederum mit Datenbankmitteln unter Zuhilfenahme der PL/SQL Packages `htp, owa_util` und `wpg_docload` bewerkstelligt werden. Umgesetzt wurde dies mittels einer APEX-Seite in welcher eine On Load/ Before Header Prozedur implementiert ist. Innerhalb dieser Prozedur wird

- das BLOB erzeugt
- ein HTML-Header zum Browser geschickt, welcher diesen über einen folgenden Datenstrom im Excelformat informiert
- das eigentliche BLOB als Datenstrom zum Browser gesendet
- die weitere Generierung der APEX-Seite abgebrochen

Der wesentliche Teil des Quellcodes für den Download stellt sich folgendermaßen dar:

```
  -- xlsx als BLOB generieren
  l_blob := ...;
  -- dem Browser im Header sagen, dass jetzt XLSX kommt
  sys.htp.init;
  sys.owa_util.mime_header(
    'application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet',
    FALSE,'UTF-8');
  sys.htp.p('Content-length: ' || sys.dbms_lob.getlength( l_blob ));
  sys.htp.p('Content-Disposition: inline; filename="' || l_filename
|| '"' );
  sys.owa_util.http_header_close;
```

```
  -- BLOB als Download anbieten
  sys.wpg_docload.download_file(l_blob);
  -- und jetzt aufhoeren eine APEX Seite zu erstellen
  apex_application.stop_apex_engine;
...
```

Lösungsansatz Import
Die notwendigen Schritte für den Import einer Exceldatei in einen Interactive Report sind:

- Upload der Excel Datei in ein Binary Large Object
- Parsen der Excel Datei und Schreiben der geladenen Daten in ein Datenbankobjekt welches als Quelle eines Interactive Report dient

Umsetzung Import
Der Import einer Excel-Datei in eine Oracle Datenbank dient in der Regel dazu Daten in ein bestehendes Datenbankschema zu importieren. Das Zieldatenbankschema unterscheidet sich für jeden Anwendungsfall. Daher werden die Daten aus dem Excel-File im Beispiel nur in eine APEX-Collection geladen und der Inhalt der Collection in einem interaktiven Report angezeigt. Auf eine Implementierung einer Kopierroutine welche die Daten aus der Collection in Datenbanktabellen schreibt wurde verzichtet.
Der Upload einer Datei ist mit Hilfe der Standardkomponenten von APEX möglich. Für das Parsen der Excel-Datei wurde aufgrund der guten Erfahrungen mit dem Package `AS_XLSX` auf das Package `AS_READ_XLSX` vom gleichen Autor zurückgegriffen. Für den Anwendungsfall von Interesse ist die Funktion `as_read_xlsx.read` welche als Parameter ein BLOB mit dem Inhalt einer Exceldatei erwartet und als Resultat eine Ergebnismenge zurückgibt.

```
function read( p_xlsx blob,
  p_sheets varchar2 := null,
  p_cell varchar2 := null
) return tp_all_cells pipelined;
```

Der Aufruf der Funktion liefert dann eine Ergebnismenge, in der jede Zelle jeder Zeile eines Tabellenblattes der Exceldatei eine Ergebniszeile darstellt. Eine Zeile der Ergebnismenge ist folgendermaßen definiert:

```
type tp_one_cell is record (
    sheet_nr number(2),
    sheet_name varchar(4000),
    row_nr number(10),
    col_nr number(10),
    cell varchar2(100),
    cell_type varchar2(1),
    string_val varchar2(4000),
    number_val number,
    date_val date,
    formula varchar2(4000)
);
```

Im Beispiel wird eine Excel-Datei geparst und die Werte der Spalte A des ersten Tabellenblattes beginnend mit der Zeile 2 in einer APEX-Collection abgelegt.

```
declare
```

```
  l_blob blob;
  l_anz number := 0;
  l_last_row number := -1;
  l_new_row boolean := false;
  l_tno varchar2(4000) := null;
  cursor csr_xlsx(cp_blob blob) is
    select
      sheet_nr,
      row_nr,
      col_nr,
      cell_type,
      decode(cell_type,
        'S', string_val,
        'N', to_char(number_val),
        'D', to_char(date_val, 'dd.mm.yyyy hh24:mi:ss'),
        'N/A') cell_value
    from table( as_read_xlsx.read( cp_blob ))
    where sheet_nr = 1
    and row_nr > 1
    order by sheet_nr, row_nr, col_nr;
begin
  ...
  if ( l_blob IS NOT NULL) then
    -- Collection fuer temporaere Aufnahme des XLSX Inhalts anlegen
    apex_collection.create_or_truncate_collection(
      p_collection_name => 'MY_UPLOAD'
    );
    for rec_xlsx in csr_xlsx(l_blob) loop
      if (rec_xlsx.row_nr > l_last_row) then
        if (l_last_row > 1) then
          -- neue Zeile, zwischengespeicherte Werte sichern
          APEX_COLLECTION.ADD_MEMBER(
            p_collection_name => 'MY_UPLOAD',
            p_c001 => l_tno
          );
        end if;
        l_last_row := rec_xlsx.row_nr;
      end if;
      if (rec_xlsx.col_nr) = 1 then
        l_tno := rec_xlsx.cell_value;
      end if;
    end loop;
    -- Werte der letzten Zeile sichern
    APEX_COLLECTION.ADD_MEMBER(
      p_collection_name => 'MY_UPLOAD',
      p_c001 => l_tno
    );
  end if;
...
```

Der Inhalt der APEX-Collection wird im Interactive Report dann mit dem SQL-Statement angezeigt:

```
SELECT c001
FROM APEX_collections
WHERE collection_name = 'MY_UPLOAD';
```

Schlußbetrachtung

Die Anforderungen im Projekt mit Bezug auf den Import und Export von Exceldateien konnten mittels der im APEX vorhandenen Funktionalität und den PL/SQL Packages AS_READ und AS_READ_XLSX von Anton Scheffer erfolgreich umgesetzt werden. Neben der Oracle Datenbank und APEX musste kein zusätzliches Tool beschafft werden, was einen sehr positiven Einfluss auf die Projekt- und Betriebskosten der entstandenen Applikation hatte.

Quellen

- AS_XLSX und AS_READ_XLSX
  - https://technology.amis.nl/2011/02/19/create-an-excel-file-with-plsql/
  - https://technology.amis.nl/2013/01/19/read-a-excel-xlsx-with-plsql/
- APEX_IR
  - https://docs.oracle.com/database/121/AEAPI/apex_ir.htm#AEAPI29337
  - http://deneskubicek.blogspot.de/2013/05/getting-interactive-report-query.html
- Download Link
  - http://davidsgale.com/apex-how-to-download-a-file

Kontaktadresse:

Henner Hucke
DATAGROUP Enterprise Services GmbH
c/o Roll-Royce Deutschland
Eschenweg 11
D-15827 Dahlewitz

Telefon: +49 (0) 33708 6 1494
E-Mail henner.hucke@datagroup.de
Internet: https://www.datagroup.de