Fiola de SQL (25)

SQL dinamic în Oracle 8i și 9i

– *Marin Fotache*

Oricât de îndemnânatici am fi în materie de SQL, oricât de darnic ar fi dialectul SQL al SGBD-ului în care lucrăm cu: opțiuni privind ierarhii, OLAP, subconsultări în clauze FROM, interogări scalare și alte bunătățuri, rămân destule probleme în care fraza SELECT trebuie completată de ingrediente procedurale. Este, deci, momentul să „dinamizăm" SQL-ul, adică să apelăm la SQL dinamic.

Cel mai adesea, SQL-ul dinamic este nesar atunci când anumite clauze, precum numele atributelor, condițiile de selecție, grupare etc. se cunosc abia în momentul lansării în execuție a comenzii respective (de creare/ștergere a unei tabele, de actualizare și, mai ales, de extragere a rezultatelor printr-o frază SELECT). Deși subiect al standardizării, chestiunile legale de SQL dinamic sunt specifice limbajului de programare al fiecărui SGBD, pe alocuri existând diferențe sensibile.

Listing 1 – Procedură pentru ștergerea din PL/SQL a unei tabele

```
CREATE OR REPLACE PROCEDURE sql_dinamic_stergere_tabela
   (tabela_ IN VARCHAR2)
   IS
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE ' || tabela_ ;
END;
/
```

Listing 2 – Versiune ameliorată 1 a procedurii de ștergere

```
CREATE OR REPLACE PROCEDURE sql_dinamic_stergere_tabela
   (tabela_ IN VARCHAR2)
   IS
   x_ SMALLINT ;
BEGIN
    SELECT 1 INTO x_ FROM USER_TABLES
        WHERE TABLE_NAME = UPPER(tabela_) ;
    EXECUTE IMMEDIATE 'DROP TABLE ' || tabela_ ;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR
             (-20003, 'Tabela ' || tabela_ ||
      ' nu exista in schema curenta !') ;
END;
```

Listing 3 – Versiune ameliorată 2 a procedurii de ștergere

```
CREATE OR REPLACE PROCEDURE sql_dinamic_stergere_tabela
   (tabela_ IN VARCHAR2)
IS
   x_ SMALLINT := 0 ;
BEGIN
    BEGIN
        SELECT 1 INTO x_ FROM USER_TABLES
      WHERE TABLE_NAME = UPPER(tabela_)  ;

        EXECUTE IMMEDIATE 'DROP TABLE ' || tabela_ ;
    EXCEPTION
       WHEN NO_DATA_FOUND THEN
           DBMS_OUTPUT.PUT_LINE ('Tabela ' || tabela_ ||
      ' nu exista in schema curenta !') ;
    END    ;
END;
```

Pachetul DBMS_SQL

Cu versiunea 7 și PL/SQL 2.1, Oracle deschide porțile, chiar dacă nu foarte larg, SQL-ului dinamic, prin livrarea pachetului-sistem DBMS_SQL. Principalele limite ale pachetului sunt:

- este foarte complicat;
- nu recunoaște tipurile de date apărute după Oracle 7;
- este lent.

Sunt suficiente motive pentru a nu ne mai osteni a-l discuta și să ne concentrăm pe o facilitate apărută în Oracle 8*i* – *Native Dynamic SQL* (NDS) implementată în PL/SQL și regăsită în principal sub forma noii comenzi EXECUTE IMMEDIATE și a ameliorării altei comenzi – OPEN FOR.EXECUTE IMMEDIATE lansează imediat în execuție, după cum sugerează și numele său, o comandă *SQL*, fie *DDL* (CREATE TABLE, CREATE INDEX etc.) fie DML - cu excepția frazelor SELECT ce returnează mai multe linii. Pentru acestă din urmă situație sunt necesare soluții bazate pe OPEN FOR.

EXECUTE IMMEDIATE

Să începem cu un exemplu supărător de simplist – ștergerea unei tabele dintr-un bloc PL/SQL. Până la apariția pachetului DBMS_SQL această operațiune nu putea fi lansată dintr-o procedură sau funcție stocată. Folosind EXECUTE IMEDIATE se poate crea o procedură, pe care am denumit-o, pe scurt, SQL_DINAMIC_STERGERE_TABELA (vezi listing 1).

Ca orice procedură, aceasta poate fi apelată dintr-o altă procedură, funcție sau bloc anonim de genul:

```
BEGIN
     SQL_DINAMIC_STERGERE_TABELA ('FACTURI_1999') ;
 END ;
/
```

Listing 4 – Bloc anonim pentru arhivarea facturilor dintr-un an într-o tabelă separată

```
DECLARE
   an_ INTEGER := 2001 ;
BEGIN
   EXECUTE IMMEDIATE
   'CREATE TABLE facturi_' || an_ ||
       ' AS
        SELECT * FROM facturi
        WHERE TO_CHAR(datafact,''||''''||'YYYY'||''''||'')
                      = ''||an_  ;

   DELETE FROM facturi
      WHERE TO_CHAR(datafact,'YYYY') = an_  ;
   COMMIT ;
END ;
```

Pentru a semnaliza printr-o eroare situația în care tabela-argument nu există în schema curentă și, implicit, ștergerea nu poate avea loc, procedura poate fi schimbată astfel (listing 2).

Sunt și situații în care, pentru a nu perturba mersul aplicației, procedura nu trebuie să declanșeze eroarea, așa că e mai nimerită varianta din listing 3.

În blocul principal a fost inclus un al doilea bloc care preia excepția, așa că, în cazul inexistenței tabelei-argument, se afișează doar un mesaj de eroare.

Pentru cele ce urmează ne propunem crearea unei proceduri destinate arhivării anuale a facturilor. Deoarece volumul tabelei `FACTURI` este foarte mare, și doar rareori se lucrează cu date de pe anii anteriori, unii responsabili de aplicații preferă arhivarea datelor unui an într-o tabelă de genul `tabelă_an`, și apoi descongestionarea tabelei „master". Astfel, la începutul anului 2002, administratorul BD salvează toate facturile care au data de întocmire între 1 ianuarie și 31 decembrie 2001 într-o tabelă numită `FACTURI_2001`, după care șterge liniile respective din tabela FACTURI. Dacă se lucrează cu un bloc anonim (listing 4), la începutul căruia se inițializează variabila `an_`, lucrurile funcționează fără necazuri.

Cum lucrurile par a fi generalizabile, nimic nu e mai la îndemână decât crearea unei proceduri în care `an_` să fie parametru de intrare, iar restul comenzilor să fie absolut identice – vezi listing 5:

Ei bine, la apelul acestei proceduri printr-un bloc anonim:

```
BEGIN
   arhivare_facturi (1996) ;
END ;
/
```

obținem un mesaj ciudat, precum că nu sunt suficiente privilegii. Remedierea acestei situații necesită introducerea clauzei `AUTHID CURRENT_USER` în procedură, ce trebuie plasată imediat după lista parametrilor (listing 6).

Pentru ilustrarea folosirii NDS în cazul interogărilor ce obțin o singură linie, Steven Feuerstein prezintă o funcție PL/SQL care preia numele unei tabele și condiția de selecție a liniilor, returnând numărul liniilor ce îndeplinesc condiția respectivă. Iată conținutul acesteia în listing 7.

Dacă, dintr-un bloc PL/SQL dorim să obținem câte linii are factura 1117, putem apela funcția astfel:

```
BEGIN
/*
...
*/
   DBMS_OUTPUT.PUT_LINE (nds_nr_inregistrari('LINIIFACT',
   'nrfact=1117'));
END ;
```

Listing 5 – Procedură pentru generalizarea arhivării facturilor

```
CREATE OR REPLACE PROCEDURE arhivare_facturi
   (an_ IN INTEGER)
IS
BEGIN
   EXECUTE IMMEDIATE
   'CREATE TABLE facturi_' || an_ ||
      ' AS
       SELECT * FROM facturi
       WHERE TO_CHAR(datafact,''||''''||'YYYY'||''''||'')
                    = '||an_  ;

   DELETE FROM facturi
      WHERE TO_CHAR(datafact,'YYYY') = an_  ;
   COMMIT ;

END ;
```

Numărul clienților ce au sediul în municipiul Iași (cod poștal 6600) se obține cu:

```
BEGIN
/*
...
*/
   DBMS_OUTPUT.PUT_LINE (nds_nr_inregistrari('CLIENTI',
      'codpost=''''||'6600'||''''')) ;
END ;
```

Observați modul în care au fost introduși literalii, respectiv modul de folosire a apostrofului.

Care este tabela ce conține cele mai multe înregistrări din schema curentă ?

Este o problemă în care vom folosi în egală măsură dicționarul de date, din care vom extrage numele fiecărei tabele, și funcția de mai sus, căreia îi vom transmite, pe rând, numele tabelei, urmând ca aceasta să returneze numărul liniilor - vezi listing 8.

Răspunsul furnizat este parțial corect, deoarece pot exista mai multe tabele cu același număr maxim de înregistrări, iar blocul anonim de mai sus afișează doar una. Rezolvarea vine de la o frază SQL care folosește funcția la modul următor:

```
SELECT *
FROM
   (SELECT TABLE_NAME AS tabela,
      nds_nr_inregistrari(table_name, NULL) AS nr_inreg
```

Listing 6 – Clauza AUTHID CURRENT_USER pentru a se permite crearea tabelelor

```
CREATE OR REPLACE PROCEDURE arhivare_facturi
   (an_ IN INTEGER)
   AUTHID CURRENT_USER
IS
BEGIN
   EXECUTE IMMEDIATE
   'CREATE TABLE facturi_' || an_ ||
      ' AS
       SELECT * FROM facturi
       WHERE TO_CHAR(datafact,''||''''||'YYYY'||''''||'')
   = '||an_  ;

   DELETE FROM facturi2
      WHERE TO_CHAR(datafact,'YYYY') = an_  ;
   COMMIT ;
END ;
```

Listing 7 – Funcție ce întoarce numărul de înregistrări pentru o tabelă dată

```
CREATE OR REPLACE FUNCTION nds_nr_inregistrari
  (
  tabela IN VARCHAR2,
  conditie IN VARCHAR2
  )
RETURN INTEGER
IS
    rezultat INTEGER;
BEGIN
  EXECUTE IMMEDIATE
   'SELECT COUNT(*)
     FROM ' || tabela ||
   ' WHERE ' || NVL (conditie, '1=1')
   INTO rezultat ;
  RETURN rezultat ;
END;
```

Listing 8 – Bloc anonim ce afișează una dintre tabelele cu număr maxim de înregistrări

```
DECLARE
   CURSOR c_tabele IS SELECT TABLE_NAME FROM USER_TABLES;
   rec_tabele c_tabele%ROWTYPE ;
   nr_max INTEGER := 0 ;
   nr INTEGER ;
   tabela VARCHAR2(30) ;
BEGIN
   FOR rec_tabele IN c_tabele LOOP
      nr := nds_nr_inregistrari(rec_tabele.table_name,
                                NULL) ;
      IF nr > nr_max THEN
         nr_max := nr ;
         tabela := rec_tabele.table_name ;
      END IF ;
   END LOOP ;
   DBMS_OUTPUT.PUT_LINE ('Tabela: '||tabela||' - nr:
                         '||nr_max) ;
END ;a
```

```
    FROM USER_TABLES ) cite_inreg
WHERE nr_inreg =
   (SELECT MAX(nr_inreg)
   FROM
      (SELECT nds_nr_inregistrari(table_name, NULL) AS
                                                  nr_inreg
       FROM USER_TABLES )
   )
```

Dacă suntem interesați ca rezultatul corect (ce afișează toate tabelele cu număr maxim de înregistrări) să fie obținut din PL/SQL, se poate crea o procedură de forma blocului (anonim) prezentat în listing 9:

Practic, fraza `SELECT` izbăvitoare va fi declarată în întregime ca sursă a unui cursor, bucla afișând toate liniile acestuia.

Fuziunea tabelelor-arhivă

Ceva mai înainte, prezentam o tehnică de rupere și salvare din tabela `FACTURI` a liniilor referitoare la un an calendaristic. Astfel, să presupunem că baza de date este operațională din 1995, iar conducerea firmei dorește o analiză a vânzărilor pentru întreaga perioadă 1995-2001. Alteori, analiza se poate face pe ultimii doi, trei s.a.m.d. ani. În toate aceste cazuri, este necesară fuzionarea înregistrărilor a două sau mai multe tabele, tabele ale căror nume se termină cu anul pe care îl reprezintă: `FACTURI_1995, ..., FACTURI_2001`. Sarcina fuzionării tabelelor, pentru un interval la alegere, constituie obiectul de activitate al procedurii `FUZ_FACT` din listing 10.

Listing 9 – Bloc anonim ce afișează corect (toate) tabelele cu număr maxim de înregistrări

```
DECLARE
   CURSOR c_rezultat IS
      SELECT *
      FROM
           (SELECT TABLE_NAME AS tabela,
              nds_nr_inregistrari(table_name, NULL)
              AS nr_inreg
            FROM USER_TABLES
       ) cite_inreg
      WHERE nr_inreg =
            (SELECT MAX(nr_inreg)
                  FROM
               (SELECT nds_nr_inregistrari(table_name,
                                           NULL)
                        AS nr_inreg
                  FROM USER_TABLES )
            ) ;
   rec_rezultat c_rezultat%ROWTYPE ;
BEGIN
   FOR rec_rezultat IN c_rezultat LOOP
      DBMS_OUTPUT.PUT_LINE ('Tabela:
   '||rec_rezultat.tabela||
         ' - nr: '|| rec_rezultat.nr_inreg) ;
   END LOOP ;
END ;
```

Pentru a nu altera conținutul tabelei `FACTURI`, fuziunea se face într-o tabelă „de manevră" botezată `FACTURI_TEMP`. Procedura preia doi parametri de intrare, anul de început și de final ai intervalului care interesează. Cursorul `tabele` va conține toate denumirile de tabele (din schema curentă) al căror nume începe cu `FACTURI_`, iar următoarele patru caractere sunt cuprinse între anul inițial și cel final. În corpul procedurii se testează existența tabelei-tampon. Dacă aceasta nu există, se crează dinamic, copiindu-se atributele din tabela `FACTURI`. Faptul că predicatul clauzei `WHERE` a comenzii `CREATE TABLE` are valoarea logică fals ne scutește de apariția unor linii nedorite în tabelă. Dacă `FACTURI_TEMP` există deja, se golește, pentru a putea fi populată cu liniile corespunzătoare facturilor din intervalul selectat. În bucla `FOR rec_tabele IN tabele...` se execută repetat comanda de inserare, pentru fiecare linie a cursorului. Facturile existente în baza de date pentru perioada 1996-2001 se obțin în tabela `FACTURI_TEMP` prin lansarea procedurii după cum urmează:

```
BEGIN
      fuz_fact(1996, 2001) ;
END ;
/
```

Listing 10 – Procedură pentru reconstituirea facturilor pentru un interval de ani dat

```
CREATE OR REPLACE PROCEDURE fuz_fact
   (an_in IN INTEGER,
    an_sf IN INTEGER)
   AUTHID CURRENT_USER
AS
   CURSOR tabele IS
   SELECT TABLE_NAME AS tabela
   FROM USER_TABLES
   WHERE TABLE_NAME LIKE 'FACTURI_%' AND
      SUBSTR(RPAD(TABLE_NAME,25),9,4)
      BETWEEN TO_CHAR(an_in) AND TO_CHAR(an_sf) ;

   rec_tabele tabele%ROWTYPE ;
   tab_ VARCHAR2(30) ;
BEGIN
   BEGIN
         SELECT TABLE_NAME
      INTO tab_
      FROM USER_TABLES
      WHERE TABLE_NAME = 'FACTURI_TEMP' ;

      EXECUTE IMMEDIATE
         'DELETE FROM ' || tab_ ;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         EXECUTE IMMEDIATE
         'CREATE TABLE facturi_temp AS
          SELECT * FROM facturi WHERE 1=2' ;
   END ;

   FOR rec_tabele IN tabele LOOP
      EXECUTE IMMEDIATE
         'INSERT INTO facturi_temp
         (SELECT *
         FROM ' || rec_tabele.tabela || ') '  ;
   END LOOP ;
   COMMIT ;
END ;
```

OPEN FOR și cursoare pentru interogări multi-linie

Pentru comenzile DDL sau DML care privesc (sau extrag) o singură linie, comanda EXECUTE IMMEDIATE este operațională în cea mai mare parte a situațiilor. Opțiunea OPEN FOR de deschidere a variabilelor-cursor rezolvă și numeroasele cazuri din NDS în care rezultatul unei fraze SELECT este un set de înregistrări.

Revenim la problema fuzionării tabelelor FACTURI_nnnn pentru un interval dat sub forma a doi parametri (ani). Procedura FUZ_FACT nu rezolvă o problemă care, deși cu șanse mici de a se produce, este una delicată. Dacă în schema curentă există și o tabelă cu numele FACTURI_199A, iar intervalul specificat este 1998-2000, atunci '199A' este cuprins între '1999' și '2000', iar această tabelă este inclusă în cursorul c_tabele, ceea ce evident reprezintă o regretabilă eroare. Procedura următoare, FUZ_FACT1, din listing 11, este prima din acest material care apelează, deși oarecum forțat, la un cursor deschis prin clauza OPEN FOR.

Primul pas într-o interogare dinamică multi-linie este definirea variabilei cursor ca instanță a tipului REF CURSOR (liniile 5 și 6). Deschiderea variabilei cursor se face repetat, pentru fiecare an al intervalului de referință. Clauza FOR asigură includerea în setul de înregistrări al cursorului a numelui tabelei (FACTURI_xxxx, undex *xxxx* = *i*), când în schema curentă există o asemenea tabelă. Acesta este motivul pentru care spuneam că folosirea termenului *interogare multi-linie* este exagerată: într-o schemă nu pot exista două tabele cu nume identic, așa că variabila cursor poate avea cel mult o linie. Când nu există FACTURI_xxxx, c_tabela%NOTFOUND are valoarea logică adevărat, iar deschiderea nu mai are loc.

Pentru exemplificarea unei interogări multi-linie autentice, schimbăm logica procedurii, de fapt, creăm o alta, numită fuz_fact2 al cărei conținut este cel din listing 12.

De această dată, cunoscând intervalul de referință – spre exemplu 1999-2001 – vrem să construim dinamic lista folosită ca argument în clauza WHERE:

```
SELECT TABLE_NAME AS tabela
FROM USER_TABLES
WHERE SUBSTR(RPAD(TABLE_NAME,25),9,4) IN
   ('1999','2000','2001') ;
```

Variabila-cursor c_tabela are toate șansele să conțină mai multe linii, iar INSERT-ul dinamic (prin EXECUTE IMMEDIATE) populează tabela FACTURI_TEMP cu înregistrările corecte.

Listing 11 – Primul exemplu de folosire a OPEN FOR...

```
CREATE OR REPLACE PROCEDURE fuz_fact1
   (an_in IN INTEGER,
    an_sf IN INTEGER)
   AUTHID CURRENT_USER
AS
   TYPE rc_tabela IS REF CURSOR ;
   c_tabela rc_tabela ;
   v_tabela USER_TABLES.TABLE_NAME%TYPE ;
   tab_ VARCHAR2(30) ;
BEGIN
   BEGIN
      SELECT TABLE_NAME
      INTO tab_
      FROM USER_TABLES
      WHERE TABLE_NAME = 'FACTURI_TEMP' ;

      EXECUTE IMMEDIATE
         'DELETE FROM ' || tab_ ;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         EXECUTE IMMEDIATE
         'CREATE TABLE facturi_temp AS
          SELECT * FROM facturi WHERE 1=2' ;
   END ;

   FOR i IN an_in..an_sf LOOP

      OPEN c_tabela FOR
         'SELECT TABLE_NAME AS tabela
          FROM USER_TABLES
          WHERE TABLE_NAME = ' ||''''||
            'FACTURI_' || TO_CHAR(i)||'''' ;

      FETCH c_tabela INTO v_tabela ;
      IF c_tabela%NOTFOUND THEN
         -- nu exista in schema tabela pentru acest an
         NULL ;
      ELSE
         EXECUTE IMMEDIATE
            'INSERT INTO facturi_temp
            (SELECT *
             FROM ' || v_tabela || ') '  ;
      END IF ;
   END LOOP ;
   COMMIT ;
END ;
```

Listing 12 – Cursor multi-linie deschis cu OPEN FOR...

```
CREATE OR REPLACE PROCEDURE fuz_fact2
   (an_in IN INTEGER,
    an_sf IN INTEGER)
   AUTHID CURRENT_USER
AS
   TYPE rc_tabela IS REF CURSOR ;
   c_tabela rc_tabela ;
   v_tabela USER_TABLES.TABLE_NAME%TYPE ;
   tab_ VARCHAR2(30) ;
   lista_ VARCHAR2(500) ;
BEGIN
   BEGIN
      SELECT TABLE_NAME
      INTO tab_
      FROM USER_TABLES
      WHERE TABLE_NAME = 'FACTURI_TEMP' ;
      EXECUTE IMMEDIATE
         'DELETE FROM ' || tab_ ;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         EXECUTE IMMEDIATE
         'CREATE TABLE facturi_temp AS
          SELECT * FROM facturi2 WHERE 1=2' ;
   END ;
   lista_ := '(' || '''' || an_in || '''' ;

   FOR i IN an_in+1..an_sf LOOP
      lista_ := lista_ || ',' || '''' || i || '''' ;
   END LOOP ;
   lista_ := lista_ || ')' ;

   OPEN c_tabela FOR
      'SELECT TABLE_NAME AS tabela
       FROM USER_TABLES
       WHERE SUBSTR(RPAD(TABLE_NAME,10),1,8) = ' || ''''  ||
         'FACTURI_' ||''''  ||
         ' AND SUBSTR(RPAD(TABLE_NAME,25),9,4) IN ' ||
                                              lista_ ;

   LOOP
      FETCH c_tabela INTO v_tabela ;
      EXIT WHEN c_tabela%NOTFOUND ;
      EXECUTE IMMEDIATE
         'INSERT INTO facturi_temp
         (SELECT *
          FROM ' || v_tabela || ') '  ;
   END LOOP ;
   COMMIT ;
END ;
```

Listing 13 – Interval de fuziune a facturilor specificat prin două date calendaristice

```
CREATE OR REPLACE PROCEDURE fuz_fact3
    (data_in IN DATE,
     data_sf IN DATE)
    AUTHID CURRENT_USER
AS
    an_in INTEGER ;
    an_sf INTEGER ;

    data_in_an_crt DATE ;
    data_Sf_an_crt DATE ;

    TYPE rc_tabela IS REF CURSOR ;
    c_tabela rc_tabela ;
    v_tabela USER_TABLES.TABLE_NAME%TYPE ;

    tab_ VARCHAR2(30) ;
    sir_ VARCHAR2(5000) ;

BEGIN

    -- celebra secventa de creare/golire a tabelei FACT_TEMP
    BEGIN
      SELECT TABLE_NAME
      INTO tab_
      FROM USER_TABLES
      WHERE TABLE_NAME = 'FACTURI_TEMP' ;
          EXECUTE IMMEDIATE
          'DELETE FROM ' || tab_ ;
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
          EXECUTE IMMEDIATE
          'CREATE TABLE facturi_temp AS
           SELECT * FROM facturi WHERE 1=2' ;
    END ;
    -- END OF celebra secventa de creare/golire a tabelei
    FACT_TEMP

    -- extragerea anilor initial si final ai intervalului
    calendaristic
    an_in := TO_NUMBER(TO_CHAR(data_in, 'YYYY')) ;
    an_sf := TO_NUMBER(TO_CHAR(data_sf, 'YYYY')) ;


    FOR i IN an_in .. an_sf LOOP  -- bucla executata pentru
                  -- fiecare an din interva

      -- cursorul C_TABELA contine cel mult o linie,
      -- daca exista tabela FACTURA_nnnn, unde nnnn = i
      OPEN c_tabela FOR
          'SELECT TABLE_NAME AS tabela
           FROM USER_TABLES
           WHERE TABLE_NAME = ' ||''''||
             'FACTURI_' || TO_CHAR(i)||'''' ;

      IF c_tabela%ROWCOUNT > 0 THEN        -- exista tabela
    pentru acest an

          -- pentru anul curent datele initiale si finale
          -- sunt, implicit 1 ianuarie,
            respectiv 31 decembrie
          data_in_an_crt := TO_DATE('01/01/'||i,
    'DD/MM/YYYY') ;
          data_sf_an_crt := TO_DATE('31/12/'||i,
    'DD/MM/YYYY') ;

          -- daca e primul an din interval, data initiala
          -- ia valoarea primului parametru de intrare -
    DATA_IN
          IF i = an_in THEN
             data_in_an_crt := data_in ;
          END IF ;

          -- daca e ultimul an din interval, data finala
          --  ia valoarea celui de-al doilea parametru de
          --  intrare - DATA_SF
          IF i = an_sf THEN   -- este ultimul an din interval
             data_sf_an_crt := data_sf ;
          END IF ;

          -- se incarca prima (si singura) inregistrare a
          --  cursorului
          FETCH c_tabela INTO v_tabela ;

          -- fraza SELECT care va fi argumentul comenzii
          --  INSERT pentu tabela curenta
          sir_ := ' SELECT * FROM ' || v_tabela ||
              ' WHERE datafact BETWEEN :data_initiala AND
                 :data_finala ' ;

          -- executia comezii INSERT si <<legarea>> celor
    doua
          -- variabile
          EXECUTE IMMEDIATE
             'INSERT INTO facturi_temp ' || sir_
             USING data_in_an_crt, data_sf_an_crt ;
      END IF ;
    END LOOP ;
END ;
```

Variabile legate și NDS

Pentru exprimarea condițiilor din clauzele `WHERE` ale comenzilor `OPEN FOR` în care rezultatul este multi-linie, se recomandă folosirea variabilelor legate. La fel și pentru `EXECUTE IMMEDIATE`, atunci când aceasta privește o comandă `UPDATE` (expresia de atribuire din `SET`). Între concatenare și atribuire, prima este considerată mai rapidă și mai ușor de întreținut. Din păcate, legarea poate privi numai valori (în fraze `SELECT` sau `UPDATE`), fiind interzisă în specificarea numelor de tabele sau atribute, în general în desemnarea numelor de obiecte din schema bazei.

Complicăm un pic discuția referitoare la analiza datelor pe mai mulți ani. Până acum preluam anii de început și de sfârșit și fuzionam întregul conținut al tabelelor corepspunzătoare fiecărui an din interval. Ce se întâmplă însă atunci când intervalul pentru care trebuie analizate vânzările este 14 august 1996 – 28 noiembrie 2000 ? Este evident că procedura ce fuziunează tabelele trebuie să preia, în noile condiții, doi parametri de tip dată calendaristică. Procedura `fuz_fact3` din listing 13 folosește o buclă care se execută pentru fiecare an calendaristic din interval. La fiecare an se folosesc două variabile. O variabilă indică data inițială pentru anul respectiv – `data_in_an_crt` (data inițială an curent) – care este 1 ianuarie pentru toți anii din interval, cu excepția primului și o altă variabilă – `data_sf_an_crt` – care este 31 decembrie, cu excepția ultimului an.

Comanda `EXECUTE IMMEDIATE`, prin care se face inserarea în `FACTURI_TEMP` a liniilor din tabela `FACTURI_nnnn` (unde *nnnn* = i), folosește două variabile legate, care precizează intervalul calendaristic selectat pentru anul curent (din buclă).

Cam acestea ar fi coordonatele SQL-ului dinamic post-Oracle 8*i*. Într-una din fiolele următoare vom urmări câteva chestiuni specifice acestei teme în Visual FoxPro și PostgreSQL. Cu bine !

Marin Fotache este ConferențiarDr. la Catedra de Informatică Economică, UAIC Iași, Facultatea de Economie și Administrarea Afacerilor. Poate fi contactat pe email la: fotache@uaic.ro. ■ 98

Bibliografie

- Steven Feuerstein – Discovering the Wonders of Native Dynamic SQL, Oracle Magazine, Nov-Dec 2000 ;
- A Dynamic Approach to Multirow Queries, Oracle Magazine, Jan-Feb 2001;
- Advanced Topics in Native Dynamic SQL, Oracle Magazine, Mar-Apr 2001