8. Folosirea dinamică a comenzilor *SQL* în *PL/SQL*

SQL dinamic este o parte integrantă a limbajului *SQL* care permite folosirea dinamică a comenzilor sale în proceduri stocate sau în blocuri anonime. Spre deosebire de comenzile statice, care nu se schimbă în timp real, comenzile dinamice se schimbă de la o execuţie la alta. Comenzile dinamice *SQL* pot depinde de anumite valori de intrare furnizate de utilizator sau de procesarea realizată în programul aplicaţie. Ele nu sunt incorporate în programul sursă, ci sunt depuse în şiruri de caractere.

SQL dinamic este o tehnică de programare care permite construirea dinamică a comenzilor la momentul execuţiei. Textul comenzii nu este cunoscut la compilare. De exemplu, se creează o procedură care operează asupra unui tabel al cărui nume este cunoscut doar când se execută procedura. În momentul compilării este cunoscută definiţia tabelelor, dar nu şi numele acestora. Există aplicaţii (de exemplu, legate de *data warehouse*) în care la fiecare unitate de timp (de exemplu, sfert de oră) sunt generate noi tabele, toate având aceeaşi structură.

Utilitatea tehnicii *SQL* dinamic este justificată de motive majore, dintre care se remarcă:

- necesitatea de a executa în *PL/SQL*, comenzi *SQL* care nu pot fi apelate în codul *PL/SQL* (de exemplu, *CREATE*, *DROP*, *GRANT*, *REVOKE*, *ALTER SESSION*, *SET ROLE*);
- necesitatea unei flexibilităţi în tratarea comenzilor (de exemplu, posibilitatea de a avea diferite condiţii în clauza *WHERE* a comenzii *SELECT*);
- necunoaşterea completă, la momentul implementării, a comenzii *SQL* care trebuie executată.

Pentru execuţia dinamică a comenzilor *SQL* în *PL/SQL* există două tehnici:

- utilizarea pachetului *DBMS_SQL*;
- *SQL* dinamic nativ.

Dacă s-ar face o comparaţie între *SQL* dinamic nativ şi funcţionalitatea pachetului *DBMS_SQL*, se poate sublinia că *SQL* dinamic nativ este mai uşor de utilizat, mai rapid şi are avantajul că suportă tipuri definite de utilizator.

Pachetul *DBMS_SQL*, în raport cu *SQL* dinamic nativ:

- poate fi folosit în programe *client-side*;
- suportă comenzi *SQL* mai mari de 32 *KB*;
- permite încărcarea înregistrărilor (procedura *FETCH_ROWS*);

- acceptă comenzi cu clauza *RETURNING* pentru reactualizarea şi ştergerea de linii multiple;
- suportă posibilităţile oferite de comanda *DESCRIBE* (procedura *DESCRIBE_COLUMNS*);
- analizează validitatea unei comenzi *SQL* o singură dată (procedura *PARSE*), permiţând ulterior mai multe utilizări ale comenzii pentru diferite mulţimi de argumente.

SQL dinamic nativ a fost introdus în *Oracle8i*, asigurând plasarea de comenzi *SQL* dinamic în codul *PL/SQL*. Comanda de bază utilizată pentru procesarea dinamică nativă a comenzilor *SQL* şi a blocurilor *PL/SQL* este *EXECUTE IMMEDIATE*, care are următoarea sintaxă:

EXECUTE IMMEDIATE *şir_dinamic*
 [***INTO*** {*def_variabila* [, *def_variabila ...*] | *record*}]
 [***USING*** [***IN*** | ***OUT*** | ***IN OUT***] *argument_bind*
 [, [***IN*** | ***OUT*** | ***IN OUT***] *argument_bind ...*]]
 [{***RETURNING*** | ***RETURN***}
 INTO *argument_bind* [, *argument_bind ...*]];

Atributul *şir_dinamic* este o expresie (şir de caractere) care reprezintă o comandă *SQL* (fără caracter de terminare) sau un bloc *PL/SQL* (având caracter de continuare); *def_variabila* reprezintă variabila în care se stochează valoarea coloanei selectate; *record* reprezintă înregistrarea în care se depune o linie selectată; *argument_bind*, dacă se referă la valori de intrare (*IN*) este o expresie (comandă *SQL* sau bloc *PL/SQL*), iar dacă se referă la valori de ieşire (*OUT*) este o variabilă ce va conţine valoarea selectată de comanda *SQL* sau de blocul *PL/SQL*.

Clauza *INTO* este folosită pentru cereri care întorc o singură linie, iar clauza *USING* pentru a reţine argumentele de legătură. Pentru procesarea unei cereri care returnează mai multe linii sunt necesare instrucţiunile *OPEN...FOR*, *FETCH* şi *CLOSE*. Prin clauza *RETURNING* sunt precizate variabilele care conţin rezultatele.

***Observaţii*:**

- *SQL* dinamic suportă toate tipurile *SQL*, dar nu acceptă tipuri de date specifice *PL/SQL* (unica excepţie este tipul *RECORD* care poate să apară în clauza *INTO*).
- În subprogramele *PL/SQL* pot să fie executate dinamic comenzi *SQL* care se referă la obiecte aparţinând unei baze de date distante.
- În anumite situaţii, o comandă *LDD* poate crea o interblocare. De exemplu, o procedură poate genera o interblocare dacă în corpul

procedurii există o comandă care şterge chiar procedura respectivă. Prin urmare, niciodată nu pot fi utilizate comenzile *ALTER* sau *DROP* referitoare la un subprogram sau pachet în timp ce se lucrează cu pachetul sau subprogramul respectiv.

***Exemplu*:**

Să se construiască o procedură care poate şterge orice tabel din baza de date. Numele tabelului şters este transmis ca parametru acestei proceduri.

```
CREATE PROCEDURE sterge_tabel (nume_tabel IN
VARCHAR2) AS
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE :tab'
    USING nume_tabel;
END;
```

La execuţia acestei proceduri va fi semnalată o eroare, deoarece nu pot fi utilizate variabile de legătură ca argument, pentru a transmite numele obiectelor dintr-o schemă. Prin urmare comanda corectă este:

```
EXECUTE IMMEDIATE 'DROP TABLE ' || nume_tabel;
```

***Exemplu*:**

Valoarea *null* nu poate să apară în clauza *USING*. Comanda următoare este incorectă.

```
EXECUTE IMMEDIATE 'UPDATE fotografie SET valoare =
:x' USING null;
```

Totuşi, dacă este necesară folosirea valorii *null*, pot fi utilizate variabile neiniţializate. O soluţie pentru corectarea erorii anterioare este dată de secvenţa:

```
DECLARE
  val_null  NUMBER;
BEGIN
  EXECUTE IMMEDIATE 'UPDATE fotografie SET valoare =
:x' USING val_null;
END;
```

***Exemplu*:**

Să se obţină numărul fotografiilor din expozitii a căror valoare depăşeşte o limită precizată (cerere *single-row*).

```
CREATE OR REPLACE FUNCTION numar_fotografii
(val_fotografie NUMBER)
  RETURN NUMBER AS
  sir_cerere  VARCHAR2(500);
```

```
  num_foto    NUMBER;
BEGIN
  sir_cerere :=
    'SELECT COUNT(*) FROM fotografie ' ||
      'WHERE valoare >= :alfa';
  EXECUTE IMMEDIATE sir_cerere
    INTO  num_foto
    USING val_fotografie;
  RETURN num_foto;
END;
```

***Exemplu*:**

Exemplul care urmează furnizează o modalitate de utilizare corectă a argumentelelor în clauza *USING*.

a) Pentru comenzi *SQL*, asocierea cu argumentele de legătură (*bind*) din clauza *USING* este prin poziţie.

```
sql_com := 'INSERT INTO alfa VALUES (:x, :x, :y,
:x)';
EXECUTE IMMEDIATE sql_com USING a, a, b, a
```

b) Pentru blocuri *PL/SQL* executate dinamic, asocierea cu argumentele *bind* din clauza *USING* se face prin nume.

```
DECLARE
  x  NUMBER := 7;
  y  NUMBER := 23;
  v_bloc  VARCHAR2(200);
BEGIN
  v_bloc := 'BEGIN calcule(:a, :a, :b, :a); END;';
  EXECUTE IMMEDIATE v_bloc USING x, y;
END;
```

În exemplul care urmează va fi ilustrat modul de utilizare a comenzii *EXECUTE IMMEDIATE* pentru executarea comenzilor *LDD*, comenzilor *LMD* de reactualizare şi a blocurilor *PL/SQL* anonime. Şirul care trebuie executat poate fi un literal inclus între apostrofuri (de exemplu, *CREATE TABLE* sau *DROP TABLE*) sau poate fi un şir de caractere (de exemplu, blocuri anonime). Caracterul „;“ nu trebuie inclus decât pentru blocurile anonime.

***Exemplu*:**

```
DECLARE
  v_sql_sir     VARCHAR2(200);
  v_plsql_bloc  VARCHAR2(200);
```

```
 BEGIN
  -- creare tabel
  EXECUTE IMMEDIATE
    'CREATE TABLE model_tabel (col1 VARCHAR(30))';
  FOR contor IN 1..10 LOOP
    v_sql_sir :=
      'INSERT INTO model_tabel
        VALUES (''Linia ' || contor || ''')';
    EXECUTE IMMEDIATE v_sql_sir;
  END LOOP;
  -- tipareste continut tabel utilizand un bloc
anonim
  v_plsql_bloc :=
    'BEGIN
      FOR cont IN (SELECT * FROM model_tabel) LOOP
        DBMS_OUTPUT.PUT_LINE (cont.col1);
      END LOOP;
     END;';
  -- executie bloc anonim
  EXECUTE IMMEDIATE v_plsql_bloc;
  -- sterge tabel
  EXECUTE IMMEDIATE 'DROP TABLE model_tabel';
END;
```

Comanda *EXECUTE IMMEDIATE* poate fi utilizată şi pentru execuţia unor comenzi în care intervin variabile *bind*. Exemplul următor ilustrează această situaţie, marcând şi modul de folosire a clauzei *USING*.

Exemplu:

```
DECLARE
  v_sql_sir     VARCHAR2(200);
  v_plsql_bloc  VARCHAR2(200);
BEGIN
  v_sql_sir :=
    'INSERT INTO fotografie (cod_fotografie, titlu,
valoare)
      VALUES (:cod, :descriere, :val)';
  EXECUTE IMMEDIATE v_sql_sir USING
      '17', 'Sarbatoare', 1500;
  v_plsql_bloc :=
    'BEGIN
      UPDATE artist SET nume = ''Popescu''
      WHERE  cod_artist = :xx;
```

```
     END;';
  EXECUTE IMMEDIATE v_plsql_bloc USING '100';
END;
```

Pentru executarea cererilor multiple este necesară o abordare similară celei descrise în cazul cursoarelor dinamice, prin utilizarea comenzilor *OPEN...FOR*, *FETCH* şi *CLOSE*. În exemplul care urmează este prezentată maniera în care pot fi executate diferite cereri, utilizând *SQL* dinamic nativ.

Exemplu:

```
CREATE OR REPLACE PACKAGE nativ AS
  TYPE t_ref  IS REF CURSOR;
  FUNCTION foto_cerere (p_clauza IN VARCHAR2)
    RETURN t_ref;
  FUNCTION foto_alta_cerere (p_gen IN VARCHAR2)
    RETURN t_ref;
END nativ;

CREATE OR REPLACE PACKAGE BODY nativ AS
  FUNCTION foto_cerere (p_clauza IN VARCHAR2)
    RETURN t_ref IS
    v_retur_cursor  t_ref;
    v_sql_comanda   VARCHAR2(500);
  BEGIN
    v_sql_comanda := 'SELECT * FROM fotografie ' ||
p_clauza;
    OPEN v_retur_cursor FOR v_sql_comanda;
    RETURN v_retur_cursor;
  END foto_cerere;

  FUNCTION foto_alta_cerere (p_gen IN VARCHAR2)
    RETURN t_ref IS
    v_retur_cursor  t_ref;
    v_sql_comanda   VARCHAR2(500);
  BEGIN
    v_sql_comanda := 'SELECT * FROM fotografie WHERE
gen = :s';
    OPEN v_retur_cursor FOR v_sql_comanda USING
p_gen;
    RETURN v_retur_cursor;
  END foto_alta_cerere;
END nativ;
```

```
 DECLARE
  v_fotografie   fotografie%ROWTYPE;
  v_foto_cursor  nativ.t_ref;
BEGIN
  -- deschide cursor
  v_foto_cursor :=
    nativ.foto_cerere ('WHERE valoare < 1000000');
  -- parcurge cursor si tipareste rezultate
  DBMS_OUTPUT.PUT_LINE ('Urmatoarele fotografii au
valoarea mai mica de un milion de dolari');
  LOOP
    FETCH v_foto_cursor INTO v_fotografie;
    EXIT WHEN v_foto_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_fotografie.titlu || ' ' ||
                         v_fotografie.gen);
  END LOOP;
  CLOSE v_foto_cursor;
  -- se procedeaza similar pentru functia
foto_alta_cerere
  -- deschide cursor
  v_foto_cursor :=
    nativ.foto_alta_cerere ('Natura');
  -- parcurge cursor si tipareste rezultate
  DBMS_OUTPUT.PUT_LINE ('Urmatoarele fotografii
apartin genului natura');
  LOOP
    FETCH v_foto_cursor INTO v_fotografie;
    EXIT WHEN v_foto_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE (v_fotografie.titlu  || ' '
|| v_fotografie.valoare);
  END LOOP;
  CLOSE v_foto_cursor;
END;
```

Comanda *EXECUTE IMMEDIATE* poate fi utilizată pentru cereri care întorc o singură linie, cu sau fără variabile *bind*. Următorul exemplu prezintă, în trei cazuri, modul de implementare a acestei posibilităţi.

***Exemplu*:**

```
DECLARE
  v_sql_cerere    VARCHAR2(200);
  v_expozitie     expozitie%ROWTYPE;
```

```
  v_nume_expozitie expozitie.nume_expozitie%TYPE;
BEGIN
  -- selectare in variabila
  v_sql_cerere :=
    'SELECT nume_expozitie ' ||
    'FROM   expozitie ' ||
    'WHERE  cod_expozitie = ''3''';
  EXECUTE IMMEDIATE v_sql_cerere INTO
v_nume_expozitie;
  DBMS_OUTPUT.PUT_LINE (v_nume_expozitie);
  -- selectare in inregistrare, utilizand variabile
bind
  v_sql_cerere :=
    'SELECT * ' ||
    'FROM   expozitie ' ||
    'WHERE nume_expozitie = :nume_expozitie';
  EXECUTE IMMEDIATE v_sql_cerere INTO v_expozitie
USING v_nume_expozitie;
  DBMS_OUTPUT.PUT_LINE (v_expozitie.nume_cladire);
  -- incarcarea unei noi linii va genera eroarea ORA-
1422
  v_sql_cerere :=
    'SELECT * FROM expozitie';
  EXECUTE IMMEDIATE v_sql_cerere INTO v_expozitie;
END;
```

Structura *Bulk bind* permite executarea cererilor care returnează mai multe linii, toate liniile returnate putând fi obţinute printr-o singură operaţie. Puterea acestei structuri poate fi combinată cu facilităţile oferite de *SQL* dinamic. Utilizând comenzile *FETCH*, *EXECUTE IMMEDIATE*, *FORALL*, clauzele *RETURNING INTO*, *COLLECT INTO* şi atributul cursor *%BULK_ROWCOUNT* se pot construi comenzi *SQL* care se execută dinamic utilizând tehnica *bulk bind*. Comenzile vor avea o structură adaptată pentru rezolvarea dinamică a comenzilor *SQL.*

În acest caz, comanda *FETCH* are forma sintactică:

FETCH *cursor_dinamic*
BULK COLLECT INTO *variabila* [, *variabila ...*];

De remarcat că dacă numărul variabilelor este mai mare decât numărul de coloane, *Oracle* va declanşa o eroare.

Comanda *FORALL* va avea următoarea structură modificată:

FORALL *index* ***IN*** *lim_inf .. lim_sup*
EXECUTE IMMEDIATE *şir_dinamic*
USING *argument_bind* | *argument_bind*(*index*)
[, *argument_bind* | *argument_bind*(*index*) ...]
[{***RETURNING*** | ***RETURN***} ***BULK COLLECT***
INTO *argument_bind* [, *argument_bind* ...]];

Atributul *şir_dinamic* acceptă comenzile *INSERT*, *UPDATE* şi *DELETE*, dar nu comanda *SELECT*.

Liniile de valori returnate de comanda *EXECUTE IMMEDIATE* pot fi depuse într-o colecţie. Comanda îşi modifică structura în următoarea manieră:

EXECUTE IMMEDIATE *şir_dinamic*
[[***BULK COLLECT***] ***INTO*** *variabila* [, *variabila* ...]
[***USING*** *argument_bind* [, *argument_bind* ...]]
[{***RETURNING*** | ***RETURN***}
BULK COLLECT INTO *argument_bind* [, *argument_bind* ...]];

***Exemplu*:**

Exemplul care urmează arată modul de utilizare a clauzei *BULK COLLECT* în comenzile *FETCH* şi *EXECUTE IMMEDIATE*.

```
DECLARE
  TYPE  foto_ref IS REF CURSOR;
  TYPE  tab_num IS TABLE OF NUMBER;
  TYPE  caracter_tab IS TABLE OF VARCHAR2(30);
  foto  foto_ref;
  num1  tab_num;
  car1  caracter_tab;
  num2  tab_num;
BEGIN
  OPEN foto FOR 'SELECT cod_fotografie, valoare FROM
fotografie';
  FETCH foto BULK COLLECT INTO car1, num1;
  CLOSE foto;
  EXECUTE IMMEDIATE 'SELECT valoare FROM fotografie'
    BULK COLLECT INTO num2;
END;
```

Numai comenzile *INSERT*, *UPDATE* şi *DELETE* pot avea variabile *bind* ca argumente *OUT*.

***Exemplu*:**

```
DECLARE
  TYPE tab IS TABLE OF VARCHAR2(60);
```

```
  v_val    tab;
  bun      NUMBER := 100000;
  com_sql  VARCHAR2(200);
BEGIN
  com_sql := 'UPDATE fotografie SET valoare = :1
              RETURNING titlu INTO :2';
  EXECUTE IMMEDIATE com_sql
    USING bun RETURNING BULK COLLECT INTO v_val;
END;
```

Pentru a utiliza variabile *bind* ca intrări într-o comandă *SQL* (diferită de *SELECT*) se pot folosi comanda *FORALL* şi clauza *USING*.

***Exemplu*:**

```
DECLARE
  TYPE  num  IS TABLE OF NUMBER;
  TYPE  caractere  IS TABLE OF VARCHAR2(30);
  num1  num;
  car1  caractere;
BEGIN
  num1 := num(1, 2, 3, 4, 5);
  FORALL i IN 1..5
    EXECUTE IMMEDIATE
       'UPDATE fotografie SET valoare = valoare*1.1
        WHERE  cod_fotografie = :1
        RETURNING titlu INTO :2'
      USING num1(i) RETURNING BULK COLLECT INTO car1;
  …
END;
```

SQL dinamic poate fi apelat şi din cadrul altor limbaje. Limbajele *C*/*C++* apelează *SQL* dinamic prin *OCI* (*Oracle Call Interface*) sau poate fi utilizat un precompilator *Pro*C*/*C++* pentru adăugarea comenzilor dinamice *SQL* la codul *C*. Limbajul *Cobol* poate utiliza comenzi dinamice *SQL* folosind un precompilator *Pro*Cobol*, iar limbajul *Java* prin intermediul lui *JDBC* (interfaţa pentru conectarea limbajului la baze de date relaţionale).