Capitolul 10. SQL Dinamic

Aproape orice dezvoltator de aplicaţii cu baze de date îşi doreşte să poată lansa dintr-un bloc procedural o comandă SQL. Această facilitate a fost considerată un lux până prin versiunea 7 a Oracle când producătorii au furnizat un pachet special denumit DBMS_SQL. Deşi un pas important, acest pachet este astăzi demodat, nu numai pentru că necesită etape destul de laborioase în lansarea unei comenzi SQL, ci şi pentru că noile tipuri de date, introduse în versiunile 8x şi 9x nu sunt recunoscute. Astfel încât, pentru prezentul capitol, ne-am propus să discutăm numai despre SQL Dinamic Nativ - NDS (*Native Dynamic SQL*), aşa cum a apărut în Oracle 8i şi se găseşte în PL/SQL-ul "contemporan".

Aria aplicaţiilor NDS este departe de a fi limitată doar la execuţia dintr-un bloc PL/SQL a unor comenzi SQL, iar în cele ce urmează vom parcurge şi câteva exemple de apel dinamic al unor procedurii/funcţii, interogări mono şi multi-linie, recursivitate etc.

10.1. Comenzi DDL

În mod normal, PL/SQL-ul nu permite crearea/ştergerea de tabele sau alte obiecte din baza de date, definirea de restricţii, acordarea de drepturi utilizatorilor şi altor comezi de tip DDL (*Data Definition Language*). Din Oracle 8i, comanda `EXECUTE IMMEDIATE` elimină această limitare. Blocul anonim din listing 10.1 prezintă două variante de creare a tabelei CONCEDII ce conţine planificarea perioadelor de concedii de odihnă pentru fiecare angajat al firmei, atributele fiind marca, data iniţială şi data finală a concediului. Fireşte, concediul poate fi luat pe "bucăţi", cheia primară fiind combinaţia (`Marca`, `DataI`).

Listing 10.1. Două variante de creare a tabelei CONCEDII

```
DECLARE
    tabela VARCHAR2(30) ;
    atribut1 VARCHAR2(50) ;
    restrictii_atribut1 VARCHAR2(500) ;
    atribut2 VARCHAR2(50) ;
    restrictii_atribut2 VARCHAR2(500) ;
    atribut3 VARCHAR2(50) ;
    restrictii_atribut3 VARCHAR2(500) ;
    restrictie1 VARCHAR2(500) ;
    restrictie2 VARCHAR2(500) ;
    restrictie3 VARCHAR2(500) ;

BEGIN

    -- primul mod de creare a tabelei CONCEDII
    EXECUTE IMMEDIATE '
        CREATE TABLE concedii (
                marca INTEGER
```

```
                    CONSTRAINT nn_concedii_marca NOT NULL
                    CONSTRAINT fk_concedii_personal
                        REFERENCES personal (marca)
            , datai DATE
            , dataf DATE
            , CONSTRAINT ck_concedii1 CHECK (dataf >= datai)
            , CONSTRAINT ck_concedii2 CHECK (dataf - datai < 90)
            , CONSTRAINT pk_concedii PRIMARY KEY (marca,datai)
        ) ' ;


    -- urmează a doua varianta

    -- dar mai întâi, ştergem tabela creată mai sus
    EXECUTE IMMEDIATE 'DROP TABLE concedii'  ;

    -- apoi o re-creăm, dar, de date aceasta, folosind variabile
    tabela := 'concedii' ;
    atribut1 := 'marca INTEGER '  ;
    restrictii_atribut1 := ' CONSTRAINT nn_concedii_marca NOT NULL
        CONSTRAINT fk_concedii_personal REFERENCES personal (marca) ' ;
    atribut2 := 'datai DATE '  ;
    restrictii_atribut2 := ' ' ;
    atribut3 := 'dataf DATE' ;
    restrictii_atribut3 := ' ' ;
    restrictie1 := ' CONSTRAINT ck_concedii1 CHECK (dataf >= datai) ' ;
    restrictie2 := ' CONSTRAINT ck_concedii2 CHECK (dataf - datai < 90) ' ;
    restrictie3 := ' CONSTRAINT pk_concedii PRIMARY KEY (marca,datai) ' ;

    EXECUTE IMMEDIATE 'CREATE TABLE ' || tabela || ' ( ' ||
        atribut1 || restrictii_atribut1 ||
        ', ' || atribut2 || restrictii_atribut2 ||
        ', ' || atribut3 || restrictii_atribut3 ||
        ', ' || restrictie1 ||
        ', ' || restrictie2 ||
        ', ' || restrictie3 || ' ) ' ;

END ;
/
```

Comenzile SQL (`CREATE TABLE`, `DROP TABLE`, `INSERT`, `UPDATE` etc.) nu se termină cu punct-virgulă; în schimb, acest caracter marchează finalul comenzii `EXECUTE IMMEDIATE`. Prima variantă este cea în care, între apostrofuri, se inserează întreaga comandă `CREATE`, ca şi cum ar fi scrisă în SQL*Plus sau într-un script. A doua variantă foloseşte variabile, iar argumentul comenzii `EXECUTE IMMEDIATE` este o combinaţie de literali şi variabile. Între cele două variante de `EXECUTE IMMEDIATE` cu `CREATE TABLE` a fost introdusă o comandă pentru ştergerea tabelei pentru a evita declanşarea unei erori datorate existenţei tabelei ce s-ar dori a fi creată. Fără ştergere, a doua execuţie a blocului s-ar fi soldat cu o eroare datorată încercării de a recrea o tabelă care există, după cum arată figura 10.1.

```
SQL>  @f:\oracle_carte\cap10_sql_dinamic\listing10_01.sql
 64  /

PL/SQL procedure successfully completed.

SQL> desc concedii
 Name
 -----------------------------------------------------------
 MARCA
 DATAI
 DATAF

SQL>  @f:\oracle_carte\cap10_sql_dinamic\listing10_01.sql
 64  /
DECLARE
*
ERROR at line 1:
ORA-00955: name is already used by an existing object
ORA-06512: at line 15


SQL>
```

Figura 10.1. Lansarea repetată a blocului PL/SQL ce conţine comanda de creare a tabelei

De aceea, se poate redacta o procedură pentru generalizarea ştergerii oricărei tabele, procedură care, pentru evitarea oricărei erori, verifică în prealabil existenţa tabelei ce urmează a fi eliminată din schemă. În secţiunea dedicată excepţiilor, se preia eventuala `NO_DATA_FOUND` şi printr-un `NULL` elegant nu se execută nimic – vezi listing 10.2.

Listing 10.2. Procedura de ştergere dinamică a unei tabele

```
CREATE OR REPLACE PROCEDURE p_sterge (
    tabela VARCHAR2)
IS
    v_unu NUMBER(1) := 0 ;
BEGIN
    SELECT 1 INTO v_unu FROM USER_TABLES
        WHERE table_name = UPPER(tabela) ;
    EXECUTE IMMEDIATE ' DROP TABLE ' || tabela ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    -- tabela nu există, aşa că e mai bine să n-o ştergem !
    NULL ;
END p_sterge ;
```

Apelul acestei proceduri se realizează de o manieră celei prezentate în blocul anonim din listing 10.3.

Listing 10.3. Apelul procedurii de ştergere

```
BEGIN

    p_sterge ('CONCEDII') ;

    EXECUTE IMMEDIATE '
```

```
        CREATE TABLE concedii (
                marca INTEGER
                        CONSTRAINT nn_concedii_marca NOT NULL
                        CONSTRAINT fk_concedii_personal
                                REFERENCES personal (marca)
                , datai DATE
                , dataf DATE
                , CONSTRAINT ck_concedii1 CHECK (dataf >= datai)
                , CONSTRAINT ck_concedii2 CHECK (dataf - datai < 90)
                , CONSTRAINT pk_concedii3 PRIMARY KEY (marca,datai)
        ) ' ;

END ;
```

Beneficiind de cunoştinţele acumulate cu atâta efort, putem purcede la a construi un pachet, numit, cam pretenţios, PAC_ADMINSTRARE, în care vom combina colecţiile de tip vectori asociativi, pe unul sau două nivele, cu SQL dinamic şi alte ingrediente necesare. Specificaţiile sale constituie subiectul listingului 10.4.

Listing 10.4. Specificaţiile pachetului PAC_ADMINISTRARE

```
CREATE OR REPLACE PACKAGE pac_administrare AUTHID CURRENT_USER
AS

TYPE t_atribute IS TABLE OF VARCHAR2 (200) INDEX BY PLS_INTEGER ;
TYPE t_restrictii_atribute IS TABLE OF t_atribute INDEX BY PLS_INTEGER ;

TYPE t_RefCursor IS REF CURSOR ;

PROCEDURE p_creare_tabele (
    tabela VARCHAR2, v_atribute t_atribute, v_tip_atribute t_atribute,
    v_restrictii_atribute t_restrictii_atribute, v_restrictii_tabela t_atribute ) ;

PROCEDURE p_sterge_tabela (tabela VARCHAR2) ;

PROCEDURE p_dezactiveaza_restrictii (tabela VARCHAR2) ;

PROCEDURE p_dezactiveaza_restrictii (v_restrictii t_atribute ) ;

PROCEDURE p_activeaza_restrictii (tabela VARCHAR2) ;

PROCEDURE p_activeaza_restrictii (v_restrictii t_atribute ) ;

END pac_administrare ;
```

Dacă `t_atribute` este un banal tip de tablou asociativ, componentele sale având o lungime de maxim 200 caractere, `t_restrictii_atribute` reprezintă un tip mai exotic – vector de vectori (nici o legătură cu jargonul manelisto-cartierist !); altfel spus, `t_restrictii` este un soi de matrice; primul indice va desemna linia atributului, iar coloana o restricţie a atributului respectiv. Cele şase proceduri sunt destinate creării şi ştergerii de tabele şi activării/dezactivării de restricţii. De fapt, sunt doar patru proceduri, deoarece s-a apelat la supraîncărcarea P_DEZACTIVEAZA_RESTRICTII şi P_ACTIVEAZA_RESTRICTII. Listingul

10.5 conține impresionantele corpuri ale procedurilor. Cea mai "supraponderală" este procedura de creare dinamică a tabelelor. Aceasta deoarece procedura nu numai că articulează argumentele comenzii `CREATE TABLE`, ci şi pentru că încearcă să dea un nume cât mai sugestiv fiecărei restricții.

Primul dintre cei cinci parametri de intrare, `tabela`, este de tip scalar şi desemnează numele tabelei ce se doreşte a fi creată dinamic. Ceilalți patru, `v_atribute`, `v_tip_atribute`, `v_restrictii_atribute` şi `v_restrictii_tabela` sunt de tip colecție. Operațiunea de creare a tabelei este precedată de apelul procedurii de ştergere, pentru înlăturarea eventualei erori ce apare la încercarea de creare a unei tabele deja existente.

Articularea opțiunilor comenzii `CREATE TABLE` se derulează în doi paşi, unul care trece în revistă toate atributele, iar celălalt care parcurge toate restricțiile definite după ultimul atribut (de obicei, acestea sunt restricțiile la nivel de înregistrare). Primul pas se bazează pe "scanarea" tuturor componentelor din tabloul `v_atribute`. Pentru fiecare atribut, a fost introdusă o secvență iterativă pentru preluarea eventualelor restricții, prilej de folosire a tabeloului bidimensional `v_restrictii_atribute`. Partea ce mai consistentă a acestei iterații ține de atribuirea automată a unui nume pentru restricția curentă. Fideli notației pseudo-ungureşti, restricțiile de tip `CHECK` vor începe cu `CK`, de tip `PRIMARY KEY` cu `PK`, `UNIQUE` cu `UN` şi `REFERENCES` cu `FK`.

Al doilea pas constă într-o altă secvență iterativă dedicată restricțiilor definite după ultimul atribut, restricții care se găsesc în tabloul `v_restrictii_tabela`. Şi aici un spațiu generos este dedicat "botezării" fiecărei restricții în funcție de tipul său. Întrucât pentru un atribut sau pentru o tabelă pot fi declarate mai multe clauze `CHECK`, respectiv `CHECK/UNIQUE/REFERENCES`, s-a recurs la variabilele `ultimul_check`, `ultimul_unique` şi `ultimul_reference` pentru a evita repetările generatoare de erori.

Listing 10.5. Corpul pachetului PAC_ADMINISTRARE

```
CREATE OR REPLACE PACKAGE BODY pac_administrare  AS
-------------------------------------------------------------------------
PROCEDURE p_creare_tabele (
    tabela VARCHAR2, v_atribute t_atribute,
    v_tip_atribute t_atribute,  v_restrictii_atribute t_restrictii_atribute,
    v_restrictii_tabela t_atribute )
IS
    sir1 VARCHAR2 (4000) := ' ' ;
    sir2 VARCHAR2 (2000) := ' ' ;
    ultimul_check NUMBER(2) := 0 ;
    ultimul_unique NUMBER(2) := 0 ;
    ultimul_reference NUMBER(2) := 0 ;
    ultimul_atribut VARCHAR2(30) := NULL ;
    nume_restrictie VARCHAR2(30) ;
 BEGIN
    p_sterge_tabela (tabela) ;
    sir1 := 'CREATE TABLE ' || tabela || ' (' ;

    -- se parcurg atributele
    FOR i IN 1..v_atribute.COUNT LOOP
        -- de la al doilea atribut introducem separatorul virgulă
```

```
        IF i > 1 THEN
            sir1 := sir1 || ', ' ;
        END IF   ;
        sir1 := sir1 || v_atribute (i) || ' ' || v_tip_atribute (i) || ' ' ;

        -- pentru fiecare atribut se trec în revistă restricţiile sale
        FOR j IN 1..99 LOOP
            -- atenţie la formatul EXISTS pentru vectorii bidimensionali !
            IF v_restrictii_atribute(i).EXISTS(j) THEN
                    -- se caută un nume pentru restricţie
                    CASE
                    WHEN v_restrictii_atribute(i)(j) LIKE '%PRIMARY KEY%' THEN
                            nume_restrictie := 'PK_' || tabela ;
                    WHEN v_restrictii_atribute(i)(j) LIKE '%UNIQUE%' THEN
                            nume_restrictie := 'UN_' || tabela || '_' || v_atribute (i) ;
                    WHEN v_restrictii_atribute(i)(j) LIKE '%NOT NULL%' THEN
                            nume_restrictie := 'NN_' || tabela || '_' || v_atribute (i) ;
                    WHEN v_restrictii_atribute(i)(j) LIKE '%REFERENCES%' THEN
                            nume_restrictie := 'FK_' || tabela || '_' || v_atribute (i) ;
                    WHEN v_restrictii_atribute(i)(j) LIKE '%CHECK%' THEN
                            -- sunt posibile mai multe CHECK-uri pentru acelasi atribut
                            IF NVL(ultimul_atribut, ' ' ) =  v_atribute (i) THEN
                                    ultimul_check := ultimul_check + 1 ;
                            ELSE
                                    ultimul_atribut := v_atribute (i) ;
                                    ultimul_check := 1 ;
                            END IF ;
                            nume_restrictie := 'CK_' || tabela || '_' || v_atribute (i) || '_' ||
                                    ultimul_check ;
                    END CASE ;
                    -- se introduce în comanda CREATE TABLE clauza CONSTRAINT
                    sir1 := sir1 || ' CONSTRAINT '|| nume_restrictie || ' ' ||
                             v_restrictii_atribute(i)(j) || ' ' ;
            ELSE
                    EXIT ;
            END IF ;
        END LOOP ;
    END LOOP ;

    -- se trec în revistă eventualele restricţii la nivel de înregistrare
    ultimul_unique := 0 ;
    ultimul_check := 0 ;
    ultimul_reference := 0 ;
    FOR i IN 1..99 LOOP
        IF v_restrictii_tabela.EXISTS(i) THEN
            -- mai întâi se caută un nume pentru restricţie
            CASE
            WHEN v_restrictii_tabela(i) LIKE '%PRIMARY KEY%' THEN
                    nume_restrictie := 'PK_' || tabela ;
            WHEN v_restrictii_tabela(i) LIKE '%UNIQUE%' THEN
                    -- sunt posibile mai multe UNIQUE-uri la nivel de tabelă
                    ultimul_unique := ultimul_unique + 1  ;
                    nume_restrictie := 'UN_' || tabela || '_' || ultimul_unique ;
            WHEN v_restrictii_tabela(i) LIKE '%CHECK%' THEN
                    -- sunt posibile mai multe CHECK-uri
                    ultimul_check := ultimul_check + 1  ;
                    nume_restrictie := 'CK_' || tabela || '_' || ultimul_check ;
            WHEN v_restrictii_tabela(i) LIKE '%REFERENCES%' THEN
                    -- sunt posibile mai multe REFERENCES-uri
                    ultimul_reference := ultimul_reference + 1  ;
```

```
                    nume_restrictie := 'FK_' || tabela || '_' || ultimul_reference ;
                END CASE ;
                -- se introduce clauza CONSTRAINT pentru restricţiile la nivel de înregistrare
                sir2 := sir2 || ', CONSTRAINT '|| nume_restrictie || ' ' || v_restrictii_tabela(i) || ' ' ;
            ELSE
                EXIT ;
            END IF ;
        END LOOP ;

        sir2 := sir2 || ' )' ;

        -- şi acum, crearea !!!
        EXECUTE IMMEDIATE ' ' || sir1 || sir2 ;

END p_creare_tabele  ;


-------------------------------------------------------------------------
PROCEDURE p_sterge_tabela (tabela VARCHAR2)
IS
    v_unu NUMBER(1) := 0 ;
BEGIN
    SELECT 1 INTO v_unu FROM USER_TABLES
        WHERE table_name = UPPER(tabela) ;
    EXECUTE IMMEDIATE ' DROP TABLE ' || tabela ;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    -- tabela nu există, aşa că e mai bine să n-o ştergem !
    NULL ;
END p_sterge_tabela ;

-------------------------------------------------------------------------
PROCEDURE p_dezactiveaza_restrictii (tabela VARCHAR2)
IS
    v_restrictii t_atribute ;
BEGIN
    SELECT constraint_name BULK COLLECT INTO v_restrictii
        FROM user_constraints WHERE table_name = UPPER(RTRIM(tabela)) ;
    FOR i IN 1..v_restrictii.COUNT LOOP
        EXECUTE IMMEDIATE 'ALTER TABLE ' || tabela || ' DISABLE CONSTRAINT '
            || v_restrictii (i) ;
    END LOOP ;
END p_dezactiveaza_restrictii ;

-------------------------------------------------------------------------
PROCEDURE p_dezactiveaza_restrictii (v_restrictii t_atribute )
IS
    tabela VARCHAR2 (50) ;
BEGIN
    FOR i IN 1..v_restrictii.COUNT LOOP
        SELECT table_name INTO tabela FROM user_constraints
            WHERE constraint_name = UPPER(RTRIM(v_restrictii (i))) ;
        EXECUTE IMMEDIATE 'ALTER TABLE ' || tabela || ' DISABLE CONSTRAINT '
            || v_restrictii (i) ;
    END LOOP ;
END p_dezactiveaza_restrictii ;

-------------------------------------------------------------------------
PROCEDURE p_activeaza_restrictii (tabela VARCHAR2)
IS
```

```
    v_restrictii t_atribute ;
BEGIN
    SELECT constraint_name BULK COLLECT INTO v_restrictii
        FROM user_constraints WHERE table_name = UPPER(RTRIM(tabela)) ;
    FOR i IN 1..v_restrictii.COUNT LOOP
            EXECUTE IMMEDIATE 'ALTER TABLE ' || tabela || ' ENABLE CONSTRAINT '
                    || v_restrictii (i) ;
    END LOOP ;
END p_activeaza_restrictii ;

--------------------------------------------------------------------------
PROCEDURE p_activeaza_restrictii (v_restrictii t_atribute )
IS
    tabela VARCHAR2 (50) ;
BEGIN
    FOR i IN 1..v_restrictii.COUNT LOOP
        SELECT table_name INTO tabela FROM user_constraints
            WHERE constraint_name = UPPER(RTRIM(v_restrictii (i))) ;
         EXECUTE IMMEDIATE 'ALTER TABLE ' || tabela || ' ENABLE CONSTRAINT '
            || v_restrictii (i) ;
    END LOOP ;
END p_activeaza_restrictii ;

END pac_administrare ;
```

Celelate proceduri ale pachetului sunt ceva mai simple. Şi pentru P_ACTIVEAZA_RESTRICTII şi pentru P_DEZACTIVEAZA_RESTRICTII s-a recurs s-a supraîncărcare. Artificiul supraîncărcării este util pentru determinarea restricţiilor de activat/dezactivat. Astfel, dacă parametrul de intrare al procedurii este simplu (şir de caractere), atunci este vorbă de o tabelă şi se activează/dezactivează toate restricţiile declarate pentru tabela respectivă. Când parametrul este de tip colecţie (vector asociativ), procedura "pricepe" că este vorba de o listă de restricţii ce trebuie activate sau dezactivate.

Ceea ce rămâne în sarcina procedurilor este ca, atunci când parametrul de intrare este scalar (`tabela`), să extragă din USER_CONSTRAINTS toate restricţiile declarate pentru tabela repectivă, iar atunci când parametrul este de tip colecţie, să extragă, pentru fiecare restricţie în parte, tabela pentru care a fost definită restricţia respectivă. Astfel, se cunosc toţi parametrii comenzilor `ALTER TABLE` *tabelă* `ENABLE CONSTRAINT` *restricţie* şi `ALTER TABLE` *tabelă* `DISABLE CONSTRAINT` *restricţie* care se vor lansa prin `EXECUTE IMMEDIATE`.

Blocul anonim din listing 10.6 ilustrează apelul procedurii de creare "dinamică" a tabelei CONCEDII. Variabilele tablou locale, `v_atrib`, `v_tip_atrib`, `v_restr_atrib` şi `v_restr_tab` sunt de tipurile declarate în specificaţiile pachetului PAC_ADMINSTRARE.

Listing 10.6. Crearea unei tabele folosind pachetul PAC_ADMINISTRARE

```
DECLARE
    v_atrib pac_administrare.t_atribute ;
    v_tip_atrib pac_administrare.t_atribute ;
    v_restr_atrib pac_administrare.t_restrictii_atribute ;
    v_restr_tab pac_administrare.t_atribute ;
```

```
BEGIN

    -- atributele şi restricţiile lor
    v_atrib(1) := 'marca' ;
    v_tip_atrib(1) := 'INTEGER' ;
    v_restr_atrib(1)(1) := 'NOT NULL' ;
    v_restr_atrib(1)(2) := 'REFERENCES personal (marca)' ;

    v_atrib(2) := 'datai' ;
    v_tip_atrib(2) := 'DATE' ;
    v_restr_atrib(2)(1) := 'NOT NULL' ;
    v_restr_atrib(2)(2) := 'CHECK (EXTRACT (YEAR FROM datai) >= 2003) ' ;

    v_atrib(3) := 'dataf' ;
    v_tip_atrib(3) := 'DATE' ;
    v_restr_atrib(3)(1) := 'NOT NULL' ;

    -- restrictiile la nivel de tabela
    v_restr_tab (1) := 'CHECK (dataf >= datai)' ;
    v_restr_tab (2) := 'CHECK (dataf - datai < 90)' ;
    v_restr_tab (3) := 'PRIMARY KEY (marca, datai)' ;

    -- şi acum, marele apel !!!
    pac_administrare.p_creare_tabele ('CONCEDII', v_atrib, v_tip_atrib, v_restr_atrib,
        v_restr_tab) ;

END ;
```

Fireşte, specificarea corectă a tipului fiecărui atribut şi restricţiilor la nivel de câmp sau înregistrare cade în sarcina celui care redactează blocul de apel al procedurii. Lansarea în execuţie a acestui bloc, ca şi vizualizarea ulterioară din SQL*Plus a celor trei atribute ale tabelei CONCEDII, este ilustrată în figura 10.2.

```
SQL>  @f:\oracle_carte\cap10_sql_dinamic\listing10_06.sql
 33  /

PL/SQL procedure successfully completed.

SQL> desc concedii
 Name
 -----------------------------------------------------------
 MARCA
 DATAI
 DATAF

SQL>
```

Figura 10.2. Apelul procedurii de creare a tabelei şi vizualizarea atributelor

Privitor la folosirea procedurilor de activare şi dezactivare de restricţii din schema curentă, avem pregătit un nou bloc, la fel de anonim – vezi listing 10.7. Blocul combină apelul procedurilor din pachet cu secvenţe iterative pentru afişarea stării restricţiilor, tocmai pentru a pune în evidenţă rezultatele procedurilor. După o primă afişare a stării restricţiilor, se apeleaza P_DEZACTIVEAZA_RESTRICTII care dezactivează toate restricţiile la nivel de

atribut şi înregistrare ale tabelei CONCEDII. Se afişează starea după apel, iar apoi se lansează procedura pentru activarea tuturor restricţiilor aceleiaşi tabele. Secvenţa iterativă pentru afişare este urmată de o dezactivare selectivă – numai a restricţiilor PK_CONCEDII şi FK_CONCEDII_MARCA. Cu ultimele puteri, blocul mai afişează odată starea restricţiilor.

Listing 10.7. Activări şi dezactivări de restricţii folosind SQL dinamic

```
DECLARE
    v_restr pac_administrare.t_atribute ;
BEGIN
    DBMS_OUTPUT.PUT_LINE ('INAINTE de dezactivare, restrictiile tabelei CONCEDII:') ;
    FOR rec_1 IN (SELECT constraint_name, status FROM user_constraints
                        WHERE table_name = 'CONCEDII') LOOP
        DBMS_OUTPUT.PUT_LINE ('Restrictie '|| rec_1.constraint_name ||
            ', stare:'||rec_1.status) ;
    END LOOP ;

    -- se dezactivează toate restricţiile tabelei CONCEDII
    pac_administrare.p_dezactiveaza_restrictii('CONCEDII');
    DBMS_OUTPUT.PUT_LINE (' ');
    DBMS_OUTPUT.PUT_LINE ('DUPA dezactivare, restrictiile tabelei CONCEDII:') ;
    FOR rec_1 IN (SELECT constraint_name, status FROM user_constraints
            WHERE table_name = 'CONCEDII') LOOP
        DBMS_OUTPUT.PUT_LINE ('Restrictie '|| rec_1.constraint_name
            || ', stare:'||rec_1.status) ;
    END LOOP ;

    -- se RE-activează toate restricţiile tabelei CONCEDII
    pac_administrare.p_activeaza_restrictii('CONCEDII');
    DBMS_OUTPUT.PUT_LINE (' ');
    DBMS_OUTPUT.PUT_LINE ('Dupa RE-activare, restrictiile tabelei CONCEDII:') ;
    FOR rec_1 IN (SELECT constraint_name, status FROM user_constraints
            WHERE table_name = 'CONCEDII') LOOP
        DBMS_OUTPUT.PUT_LINE ('Restrictie '|| rec_1.constraint_name
                || ', stare:'||rec_1.status) ;
    END LOOP ;

    v_restr (1) := 'PK_CONCEDII' ;
    v_restr (2) := 'FK_CONCEDII_MARCA' ;
    pac_administrare.p_dezactiveaza_restrictii(v_restr) ;

    DBMS_OUTPUT.PUT_LINE (' ');
    DBMS_OUTPUT.PUT_LINE
        ('Dupa dezactivarea celor doua, restrictiile tabelei CONCEDII:') ;
    FOR rec_1 IN (SELECT constraint_name, status FROM user_constraints
                        WHERE table_name = 'CONCEDII') LOOP
            DBMS_OUTPUT.PUT_LINE ('Restrictie '|| rec_1.constraint_name
                || ', stare:'||rec_1.status) ;
    END LOOP ;
END ;
```

Rezultatul SQL*Plus al acestor laborioase operaţiuni este afişat în figura 10.3.

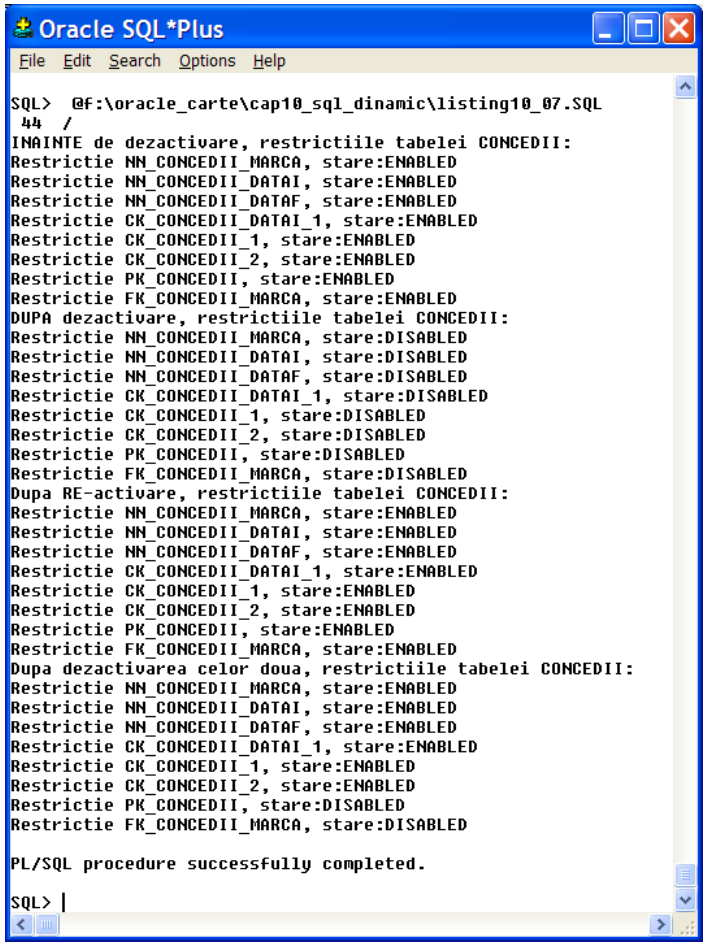

Figura 10.3. Apeluri ale procedurilor de activare/dezactivare a restricţiilor

10.2. Apelul “dinamic” al unei proceduri/funcții

Cel de-al doilea ingredient din SQL dinamic nativ pe care îl discutăm succint în continuare este legat de crearea şi lansarea dintr-o procedură a unul bloc PL/SQL, anonim şi mai puțin anonim. Câteva explicații despre obiectivul urmărit nu strică. În aplicațiile client/server tradiționale, pe două straturi, sesiunea de lucru este descrisă la logarea clientului cu numele utilizator şi parola corespunzătoare, iar obiectele instanțiate în memorie pot fi accesate în limita drepturilor acordate. În aplicațiile web, însă, dialogul utilizatorului cu baza de date este mijlocit de serverul de aplicații care preia pe cont propriu cererile clienților şi negociază de unul singur cu serverul de bază de date. Aceasta

înseamnă că sesiunea server de aplicații – server de bază de date este partajată de mai mulți clienți.

10.2.1. Crearea şi folosirea dinamică a unei pachet

Procedura următoare, P_PACHET_USER (listing 10.8), care primeşte drept parametru de intrare numele utilizatorului, crează un pachet ce conține variabila publică `data_ultimei_modificari`. Interesant este că numele pachetului conține şi numele utilizatorului: PACHET_*utilizator*, ceea ce reprezintă o premieră pentru cartea de față. Practic, procedura verifică existența pachetului, folosind tabela virtuală din dicționar USER_OBJECTS. Blocul inclus preia excepția `NO_DATA_FOUND` care semnalizează inexistența pachetului în schema curentă şi, printr-un `EXECUTE IMMEDIATE` fulgerător, crează dinamic pachetul al cărui nume nu este mai puțin... dinamic. La revenirea în blocul principal se apelează (cum altfel, decât dinamic ?) pachetul, modificându-se valoarea variabilei publice cu data şi ora curentă.

Listing 10.8. Crearea şi apelul dinamic al unui pachet

```
CREATE OR REPLACE PROCEDURE p_pachet_user (utilizator VARCHAR2)
    AUTHID CURRENT_USER
AS
    v_unu NUMBER(1) ;
BEGIN
    BEGIN
        SELECT 1 INTO v_unu
        FROM user_objects
        WHERE object_type='PACKAGE' AND object_name = 'PACHET_'||utilizator ;
    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        EXECUTE IMMEDIATE  'CREATE OR REPLACE PACKAGE pachet_' || utilizator
            || ' AUTHID CURRENT_USER AS data_ultimei_modificari DATE ;
            END pachet_'|| utilizator ||';' ;
    END ;
    EXECUTE IMMEDIATE 'BEGIN pachet_'|| utilizator ||
        '.data_ultimei_modificari := SYSDATE; END ;' ;
END p_pachet_user ;
```

Prin urmare, un bloc PL/SQL poate fi inclus într-o comandă `EXECUTE IMMEDIATE` "liniarizându-l", corectitudinea sintactică a blocului (atenție la terminatorul de comandă, punct-virgulă !) fiind decisivă pentru succesul apelului. Figura 10.4 suprinde în SQL*Plus starea dinaintea apelului procedurii, lansarea de două ori a procedurii (cu argumentele 'MARIN1' şi 'MARIN2'), apelul dinamic al pachetului pentru modificarea valorii variabilei publice `data_ultimei_modificari` şi, în final, afişarea pachetelor nou create din dicționar (USER_OBJECTS).

Se cuvine de remarcat că data creării şi a ultimei modificări a celor două pachete nu indică tocmai adevărul...

Din păcate, variabilele declarate în bloc nu pot fi accesate prin comanda `EXECUTE IMMEDIATE`. Astfel, blocul din listing 10.9 conţine două variabile, `v1` şi `v2`, prima fiind iniţializată chiar din secţiunea declaraţiilor.

```
SQL> SELECT object_name, created, last_ddl_time, status
  2  FROM user_objects WHERE object_type = 'PACKAGE'
  3   AND object_name LIKE 'PACHET_MARIN%'
  4  /

no rows selected

SQL>  EXECUTE p_pachet_user('MARIN1')

PL/SQL procedure successfully completed.

SQL>  EXECUTE p_pachet_user('MARIN2')

PL/SQL procedure successfully completed.

SQL> BEGIN
  2  pachet_marin1.data_ultimei_modificari := SYSDATE + 1;
  3  END;
  4  /

PL/SQL procedure successfully completed.

SQL>  SELECT object_name, created, last_ddl_time, status
  2   FROM user_objects WHERE object_type = 'PACKAGE'
  3    AND object_name LIKE 'PACHET_MARIN%'
  4  /

OBJECT_NAME           CREATED   LAST_DDL_ STATUS
--------------------- --------- --------- -------
PACHET_MARIN1         01-APR-03 01-APR-03 VALID
PACHET_MARIN2         01-APR-03 01-APR-03 VALID

SQL>
```

Figura 10.4. Ilustrarea, în SQL Plus, a folosirii procedurii P_PACHET_USER şi apelului dinamic al unui pachet

Blocul PL/SQL executat dinamic încearcă să atribuie o valoare variabilei `v2`.

Listing 10.9. Domeniul de vizibilitate pentru variabile din bloc PL/SQL dinamic

```
DECLARE
    v1 VARCHAR2(100) := 'Valoare 1' ;
    v2 VARCHAR2(100) ;
BEGIN
    EXECUTE IMMEDIATE ' BEGIN v2:= '''||'Valoare 2'||''' ; END ; ' ;
    DBMS_OUTPUT.PUT_LINE(v2) ;
END ;
```

Lansarea în execuţie prilejuieşte primirea unui mesaj de eroare, după cum se observă în figura 10.5. Mesajul este limpede. Variabila `v2` nu este recunoscută de comanda `EXECUTE IMMEDIATE`; prin urmare toate variabilele incluse în blocul inclus în comandă au regim privat (local).

```
SQL> DECLARE
  2   v1 VARCHAR2(100) := 'Valoare 1' ;
  3   v2 VARCHAR2(100) ;
  4  BEGIN
  5   EXECUTE IMMEDIATE ' BEGIN v2:= '''||'Valoare 2'||''' ; END ; ' ;
  6   DBMS_OUTPUT.PUT_LINE(v2) ;
  7  END ;
  8  /
DECLARE
*
ERROR at line 1:
ORA-06550: line 1, column 8:
PLS-00201: identifier 'V2' must be declared
ORA-06550: line 1, column 8:
PL/SQL: Statement ignored
ORA-06512: at line 5
```

Figura 10.5. Vizibilitatea unei variabile incluse în `EXECUTE IMMEDIATE`

Fireşte, acest neajuns poate fi depăşit dacă se recurge la variabile publice (din pachet). Execuţia listingului 10.10 crează, în prima etapă, specificaţiile unui pachet, PAC_STERGE ce conţine două variabile, inevitabil publice, `v1` şi `v2`. Blocul anonim care urmează pasează, mai întâi, variabilei `v2` valoarea din `v1`, printr-o banală comandă de atribuire, apoi modifică aceeaşi valoare printr-un bloc anonim inclus într-o comandă `EXECUTE IMMEDIATE`.

Listing 10.10. Apelarea dintr-un bloc dinamic a variabilei din pachet

```
CREATE OR REPLACE PACKAGE pac_sterge AS
    v1 VARCHAR2(100) := 'Valoare 1' ;
    v2 VARCHAR2(100) ;
END pac_sterge ;
/
BEGIN
    pac_sterge.v2:= pac_sterge.v1 ;
    EXECUTE IMMEDIATE ' BEGIN pac_sterge.v2 := '"'||'Valoare 2'||'"' ; END ; ' ;
    DBMS_OUTPUT.PUT_LINE(pac_sterge.v2) ;
END ;
```

Rezultatul (figura 10.6) este încurajator.

```
SQL> CREATE OR REPLACE PACKAGE pac_sterge AS
  2   v1 VARCHAR2(100) := 'Valoare 1' ;
  3   v2 VARCHAR2(100) ;
  4  END pac_sterge ;
  5  /

Package created.

SQL>
SQL> BEGIN
  2   pac_sterge.v2:= pac_sterge.v1 ;
  3   EXECUTE IMMEDIATE ' BEGIN pac_sterge.v2:= '''||'Valoare 2'||''' ; END ; ' ;
  4   DBMS_OUTPUT.PUT_LINE(pac_sterge.v2) ;
  5  END ;
  6  /
Valoare 2

PL/SQL procedure successfully completed.

SQL>
```

Figura 10.6. Apelul, din blocul dinamic, a unei variabile publice

10.2.2. Preluarea valorilor unei înregistrări şi generarea dinamică a comenzii INSERT

În cele ce urmează ne propunem un obiectiv ceva mai ambiţios. În aplicaţiile client-server sau web, deseori inserarea, modificarea sau ştergera de înregistrări din tabele nu se realizează direct prin comenzi `INSERT/UPDATE/DELETE` lansate de pe client către serverul de baze de date, ci prin apelul unei proceduri de inserare/modificare/ştergere, procedură căreia i se transmit sub formă de parametri, valorile înregistrării de adăugat sau cheile primare/condiţiile liniilor de modificat sau şters.

Să discutăm cazul tabelei PERSONAL, oprindu-ne doar la operaţiunea de adăugare a unei înregistrări. Interfaţa aplicaţiei preia valorile atributelor şi le pasează sub forma unei variabile compozite (de tip înregistrare – ROWTYPE). Atenţie, însă, dacă valoarea pasată pentru un atribut este NULL, vrem ca în tabelă atributul respectiv să primească valoarea implicită (clauza `DEFAULT` din comanda `CREATE TABLE` sau `ALTER TABLE`). Or, dacă `INSERT` ar avea formatul `INSERT INTO personal VALUES (valoare1, valoare2, ...)`, oricare valoare NULL ar putea intra în conflict cu o eventuală restricţie `NOT NULL` declarată pentru atributului respectiv. De aceea, procedura pe care o vom redacta în continuare va testa, pe rând, valoarea fiecărui atribut. Numai dacă aceasta este nenulă, atributului respectiv va fi inclus în comanda de inserare dinamică. Practic, în final, se va executa o comandă de genul `INSERT INTO personal (atribut_nenul1, atribut_nenul2,...) VALUES (valoare_nenulă1, valoare_nenulă2, ....)`.

Listingul 10.11 conţine atât specificaţiile, cât şi corpul pachetului PAC_INSERT. Variabilele publice sunt:

- `v_enull` care va conţine TRUE dacă valoarea atributului curent este nulă şi FALSE în caz contrar;
- `v_sir` ce conţine valoarea nenulă a a unui atribut sub formă de şir ce urmază a fi inclus în clauza `VALUES`;
- `v_personal` – o copie a tuturor valorilor atributelor pasate la apelul procedurii P_INSERT;

Soluţia propusă se bazează, în bună măsură, pe supraîncărcarea funcţiei F_VAL care, în funcţie de tipul atribului primit, va returna şirul de caractere (inclusiv, apostrofuri, funcţii `TO_DATE`) necesar clauzei `VALUES`. În fine, procedura ce generează comanda `INSERT` se numeşte P_INSERT şi prezintă un parametru de intrare de tip înregistrare (`personal%ROWTYPE`).

Listing 10.11. Pachetul pentru generarea dinamică a comenzii INSERT

```
CREATE OR REPLACE PACKAGE pac_insert AS
    v_enull BOOLEAN := FALSE ;
    v_sir VARCHAR2(100);
```

```
    v_personal personal%ROWTYPE ;
FUNCTION f_val (atribut_ VARCHAR2) RETURN VARCHAR2 ;
FUNCTION f_val (atribut_ NUMBER) RETURN VARCHAR2 ;
FUNCTION f_val (atribut_ DATE) RETURN VARCHAR2 ;
PROCEDURE p_insert (rec_personal personal%ROWTYPE) ;
END pac_insert ;
/

CREATE OR REPLACE PACKAGE BODY pac_insert AS

-- prelucrarea valorilor de tip ŞIR DE CARACTERE
FUNCTION f_val (atribut_ VARCHAR2) RETURN VARCHAR2 IS
BEGIN
    RETURN "" || atribut_  ||"";
END f_val ;

-- prelucrarea valorilor de tip NUMERIC
FUNCTION f_val (atribut_ NUMBER) RETURN VARCHAR2 IS
BEGIN
    RETURN ' ' || atribut_ || ' ' ;
END f_val ;

-- prelucrarea valorilor de tip DATA CALENDARISTICA
FUNCTION f_val (atribut_ DATE) RETURN VARCHAR2 IS
BEGIN
    RETURN ' TO_DATE ('" || TO_CHAR(atribut_,'DD/MM/YYYY') || '", "DD/MM/YYYY") ' ;
END f_val ;

/* procedura de preluare a valorilor unei înregistrări din PERSONAL şi generare
dinamică a comenzii INSERT */
PROCEDURE p_insert (rec_personal personal%ROWTYPE) IS
    sir_atribute VARCHAR2 (2000)  ; -- INSERT INTO personal (atribut_i, atribut_j...)
    sir_valori VARCHAR2 (2000)    ; -- VALUES (valoarea_i, valoare_j, ...)
BEGIN
    v_personal := rec_personal ; -- copierea valorilor în variabila din pachet

    -- se parcurg toate atributele tabelei şi se testează care au valori NULLe
    FOR rec_atribute IN (SELECT column_name, column_id, data_type
            FROM USER_TAB_COLUMNS WHERE TABLE_NAME ='PERSONAL'
            ORDER BY column_id) LOOP
        IF rec_atribute.column_id = 1 THEN --  primul atribut
            sir_atribute := ' INSERT INTO PERSONAL ( ' ;
            sir_valori :=  ' VALUES ( ' ;
        END IF ;

        -- se testează dacă valoarea atributului curent este NULL
        --    artificiul consta in folosirea unui bloc dinamic care atribuie valoarea
        --    TRUE sau FALSE variabilei publice V_ENULL
        EXECUTE IMMEDIATE
            ' BEGIN IF pac_insert.v_personal.'|| rec_atribute.column_name ||
                    ' IS NULL THEN pac_insert.v_enull := TRUE ;
                    ELSE pac_insert.v_enull := FALSE ; END IF; END; ' ;
        IF v_enull = TRUE THEN
            -- se omite din lista atributelor şi cea a valorilor
            NULL ;
        ELSE
            -- valoarea atributului curent nu este NULL, deci trebuie modificate şi
            --          lista atributelor şi cea a valorilor
            IF rec_atribute.column_id > 1 THEN
                    sir_atribute := sir_atribute || ', ' ;
```

```
                sir_valori := sir_valori || ', ' ;
            END IF ;
            -- se adaugă numele în lista atributelor
            sir_atribute := sir_atribute || rec_atribute.column_name ;

            /* un alt artificiu, cel mai interesant: valoarea atributului curent se pasează
               unei funcţii F_VAL din pachet; aceasta va returna variabilei publice V_SIR
               şirul de caractere ce trebuie inclus în clauza VALUES din comanda INSERT;
               întrucât valoarea atributului poate fi numerică, şir de caractere sau
               dată calendaristică, ne bazăm pe supraîncarcarea funcţiei F_VAL */
            EXECUTE IMMEDIATE ' BEGIN pac_insert.v_sir :=
                    pac_insert.f_val ( pac_insert.v_personal.'||
                    rec_atribute.column_name || ') ; END;  ' ;
            -- se adaugă valoarea V_SIR în lista valorilor clauzei VALUES
            sir_valori := sir_valori || pac_insert.v_sir ;
        END IF ;
    END LOOP ;
    sir_atribute := sir_atribute || ' ) ' ;
    sir_valori := sir_valori || ') '  ;

    -- pentru impresie, se afişează cele două părţi ale comenzii INSERT
    DBMS_OUTPUT.PUT_LINE(sir_atribute) ;
    DBMS_OUTPUT.PUT_LINE(sir_valori) ;

    -- în fine, ECCE INSERT-ul !
    EXECUTE IMMEDIATE sir_atribute || ' ' || sir_valori ;
END p_insert ;

END pac_insert ;
/
```

La SQL dinamic s-a recurs în trei puncte ale procedurii. Mai întâi, se testează dacă valoarea atributului curent (`pac_insert.v_personal.'|| rec_atribute.column_name`) este NULLă, rezultatul fiind stocat în variabila publică `v_enull`. Al doilea moment este cel al apelului dinamic al funcției F_VAL, pasându-i-se valoarea atributului curent. Este şi momentul în care intră în acțiune supraîncărcarea, deoarece tipul valorii transmise determină instanța funcției ce va intra în execuție. În fine, ultima comandă `EXECUTE IMMEDIATE` este cea care lansează comanda `INSERT`.

Pentru testarea funcției, apelăm la blocul anonim din listing 10.12. Acesta inițializează valorile atributelor unei înregistrări din PERSONAL şi i le pasează procedurii din pachetul PAC_INSERT.

Listing 10.12. Apelul procedurii de inserare dinamică

```
DECLARE
    linie_personal personal%ROWTYPE ;
BEGIN
    -- initializarea valorii atributelor
    linie_personal.marca := 1011 ;
    linie_personal.numepren := 'Angajat 1011' ;
    linie_personal.compart := NULL ;
    linie_personal.datasv := TO_DATE ('12/05/1997', 'DD/MM/YYYY') ;
    linie_personal.salorar := NULL ;
    linie_personal.salorarco := NULL ;
    linie_personal.colaborator := NULL ;
```

```
    -- apelul procedurii
    pac_insert.p_insert (linie_personal) ;
    COMMIT ;
END ;
/
```

Valorile atributelor `compart`, `salorar`, `salorarco` şi `colaborator` sunt nule, aşă ca acestea ar trebui excluse în formatul comenzii `INSERT`. Apelul blocului de mai sus din SQL*Plus este prezentat în figura 10.7.

```
SQL> DECLARE
  2   linie_personal personal%ROWTYPE ;
  3  BEGIN
  4   -- initializarea valorii atributelor
  5   linie_personal.marca := 1011 ;
  6   linie_personal.numepren := 'Angajat 1011' ;
  7   linie_personal.compart := NULL ;
  8   linie_personal.datasv := TO_DATE ('12/05/1997', 'DD/MM/YYYY') ;
  9   linie_personal.salorar := NULL ;
 10   linie_personal.salorarco := NULL ;
 11   linie_personal.colaborator := NULL ;
 12
 13   -- apelul procedurii
 14   pac_insert.p_insert (linie_personal) ;
 15   commit ;
 16  END ;
 17  /
INSERT INTO PERSONAL ( MARCA, NUMEPREN, DATASV )
VALUES (  1011 , 'Angajat 1011',  TO_DATE ('12/05/1997', 'DD/MM/YYYY') )

PL/SQL procedure successfully completed.

SQL>
```

Figura 10.7. Apelul procedurii de inserare dinamică

Cele două linii afişate la execuţie constituie părţile comenzii `INSERT`. Aşa cum ne-am propues, în listă sunt incluse numai atributele ale căror valori sunt nenule (`marca`, `numepren` şi `datasv`).

10.3. Variabile legate şi comenzi DML dinamice

Procedura P_PACHET_USER se poate redacta şi de maniera prezentată în listing 10.13. Elementul de noutate îl constituie folosirea unei variabile legate. Astfel, conţinutul atribuit variabilei din pachet nu este specificat direct, ci prin variabila legată referită prin `:1`, clauza `USING` fiind cea care face efectiv legătura `:1 - SYSDATE`.

Listing 10.13. O primă variabilă legată

```
CREATE OR REPLACE PROCEDURE p_pachet_user (utilizator VARCHAR2)
        AUTHID CURRENT_USER  AS
    v_unu NUMBER(1) ;
BEGIN
```

```
    BEGIN
        ....
    END ;
    EXECUTE IMMEDIATE 'BEGIN pachet_'|| utilizator || '.data_ultimei_modificari := :1 ;
        END ;'  USING SYSDATE ;
END p_pachet_user ;
```

Fireşte, nimic nu se schimbă în apelul procedurii, după cum reiese şi din figura 10.8. Apelul procedurii crează pachetul PACHET_MARIN3 şi iniţializează variabila din acesta folosind variabila legată.

```
SQL> EXECUTE p_pachet_user('MARIN3')

PL/SQL procedure successfully completed.

SQL> SELECT object_name, created, last_ddl_time, status
  2  FROM user_objects WHERE object_type = 'PACKAGE'
  3   AND object_name LIKE 'PACHET_MARIN%'
  4  /

OBJECT_NAME           CREATED   LAST_DDL_ STATUS
--------------------- --------- --------- -------
PACHET_MARIN1         01-APR-03 01-APR-03 VALID
PACHET_MARIN2         01-APR-03 01-APR-03 VALID
PACHET_MARIN3         01-APR-03 01-APR-03 VALID

SQL> BEGIN
  2  DBMS_OUTPUT.PUT_LINE(pachet_marin3.data_ultimei_modificari);
  3  END;
  4  /
01-APR-03

PL/SQL procedure successfully completed.

SQL>
```

Figura 10.8. Afişarea valorii setate prin folosirea variabilei legate

Să luăm un alt exemplu în care variabile legate se folosesc pentru o comandă `INSERT` "dinamică". Blocul anonim din listing 10.14 prezintă, pentru comparaţie, o variantă de inserare "obişnuită", în care constantele de tip şir de caractere sunt încadrate de două apostrofuri (nu confundaţi cu ghilimelele !) şi o alta, nu neapărat neobişnuită, în care se folosesc patru variabile legate.

Listing 10.14. Inserare dinamică, fără şi cu variabile legate

```
DECLARE
    v_marca personal.marca%TYPE := 1006;
    v_numepren personal.numepren%TYPE := 'Angajat 6' ;
    v_compart personal.compart%TYPE := 'CONTA' ;
    v_datasv personal.datasv%TYPE := TO_DATE ('23/11/1989', 'DD/MM/YYYY') ;
BEGIN
    -- varianta 1 - FĂRĂ variabile legate
    EXECUTE IMMEDIATE 'INSERT INTO personal (marca, numepren, compart, datasv)
        VALUES (1005, "Angajat 5", "IT", TO_DATE("12/05/1992", "DD/MM/YYYY")) ' ;

    -- varianta 2 - CU variabile legate
    EXECUTE IMMEDIATE 'INSERT INTO personal (marca, numepren, compart, datasv)
```

```
        VALUES (:1, :2, :3, :4 ) ' USING v_marca, v_numepren, v_compart, v_datasv ;
    COMMIT ;
END ;
```

Analog inserării, se pot folosi variabile legate în `EXECUTE IMMEDIATE` pentru modificări şi ştergeri de linii. Una dintre cele mai importante restricţii ale variabilelor legate ţine de faptul că acestea nu pot desemna nume de obiecte ale bazei. Modificăm uşor forma blocului din listingul precedent şi încercăm ca în comanda `INSERT` să specificăm numele tabelei PERSONAL printr-o variabilă legată – vezi listing 10.15.

Listing 10.15. Variabilă legată (incorect) pentru specificarea numelui tabelei

```
DECLARE
    v_marca personal.marca%TYPE := 1099;
    v_numepren personal.numepren%TYPE := 'Angajat 99' ;
    v_compart personal.compart%TYPE := 'CONTA' ;
    v_datasv personal.datasv%TYPE := TO_DATE ('20/09/1997', 'DD/MM/YYYY') ;
    v_tabela VARCHAR2 (30) := 'PERSONAL' ;
BEGIN
    -- tentativa nr.3
    EXECUTE IMMEDIATE 'INSERT INTO :1  (marca, numepren, compart, datasv)
        VALUES (:2, :3, :4, :5 ) ' USING v_tabela, v_marca, v_numepren, v_compart, v_datasv ;
END ;
```

Consecinţele sunt dezastroase – figura 10.9. După cum anticipam, PL/SQL se opune energic desemnării numelui oricărui obiect al bazei (în cazul nostru, tabelă) printr-o variabilă legată.

```
SQL> DECLARE
  2   v_marca personal.marca%TYPE := 1099;
  3   v_numepren personal.numepren%TYPE := 'Angajat 99' ;
  4   v_compart personal.compart%TYPE := 'CONTA' ;
  5   v_datasv personal.datasv%TYPE := TO_DATE ('20/09/1997', 'DD/MM/YYYY') ;
  6   v_tabela VARCHAR2 (30) := 'PERSONAL' ;
  7  BEGIN
  8   -- tentativa nr.3
  9   EXECUTE IMMEDIATE 'INSERT INTO :1  (marca, numepren, compart, datasv)
 10    VALUES (:2, :3, :4, :5 ) ' USING v_tabela, v_marca, v_numepren, v_compart, v_datasv ;
 11  END ;
 12  /
DECLARE
*
ERROR at line 1:
ORA-00903: invalid table name
ORA-06512: at line 9
```

Figura 10.9. Folosire eronată a unei variabile legate

Acest gen de situaţie reclamă utilizarea variabilelor pentru desemnarea tabelei de maniera prezentată în paragraful 10.1 – vezi şi listing 10.16.

Listing 10.16. Sintaxa corectă de specificare, prin variabilă, a numelui tabelei

```
DECLARE
    v_marca personal.marca%TYPE := 1099;
    v_numepren personal.numepren%TYPE := 'Angajat 99' ;
    v_compart personal.compart%TYPE := 'CONTA' ;
    v_datasv personal.datasv%TYPE := TO_DATE ('20/09/1997', 'DD/MM/YYYY') ;
```

```
    v_tabela VARCHAR2 (30) := 'PERSONAL' ;
BEGIN
    -- tentativa nr.3
    EXECUTE IMMEDIATE 'INSERT INTO ' || v_tabela ||
          '  (marca, numepren, compart, datasv) VALUES (:1, :2, :3, :4 ) '
        USING v_marca, v_numepren, v_compart, v_datasv ;
END ;
```

10.4. Interogări dinamice mono-linie

Un exemplu tipic de funcţie dinamică cu interogare al cărei rezultat este mono-linie vizează furnizarea numărului de înregistrări dintr-o tabelă specificată[1] - vezi listing 10.15. De data aceasta, clauza folosită este `INTO`, prin care se precizează în ce variabile se va stoca rezultatul interogării.

Listing 10.17. Funcţie ce furnizează numărul de linii dintr-o tabelă specificată

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

Listing 10.18 conţine două blocuri anonime pentru apelul funcţiei. La primul apel se transmite doar numele tabelei – TRANSE_SV. Al doilea apel valorifică şi avantajul specificării unei condiţii pe care trebuie să o îndeplinească liniile tabelei pentru a fi luate în calcul la numărare. Şi în aceste situaţii trebuie mărită atenţia la modul în care sunt introduşi literalii, respectiv modul de folosire a apostrofului.

Listing 10.18. Două blocuri de apel a funcţiei F_NR_LINII

```
BEGIN
DBMS_OUTPUT.PUT_LINE('In tabela ' || 'TRANSE_SV' ||
 ' sunt ' || f_nr_linii('TRANSE_SV', NULL)
 ||' linii ');
END ;
/

BEGIN
DBMS_OUTPUT.PUT_LINE('In tabela ' || 'PERSONAL' ||
' sunt ' || f_nr_linii('PERSONAL', 'compart = ''IT''')
||' angajat in compartimentul IT ');
END ;
/
```

[1] Vezi şi [Fuerstein???] [Fotache??? – sql dinamic]

Figura 10.10 prezintă rezultatele lansării celor două blocuri anonime PL/SQL din SQL*Plus.

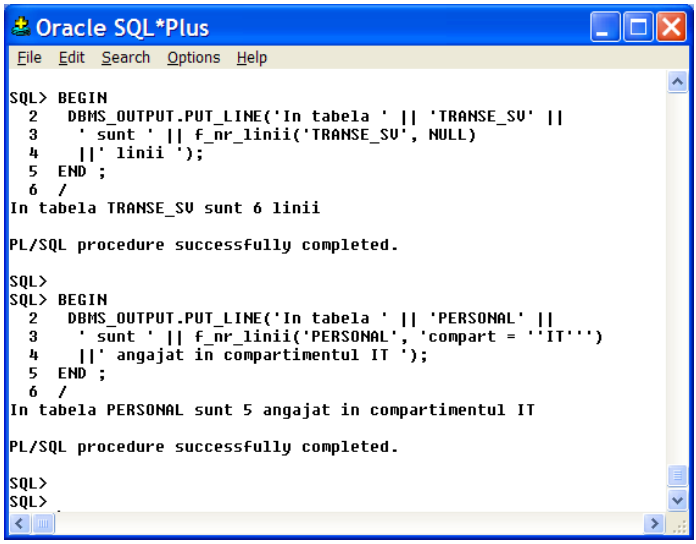

Figura 10.10. Două apeluri a funcţiei F_NR_LINII

Pentru divertisment, ne propunem să obţinem tabela ce are cel mai mare număr de înregistrări din schema curentă, scop pentru care vom redacta o frază `SELECT` ce apelează funcţia de mai sus:

```
SELECT *
FROM      (SELECT table_name AS tabela,
                  f_nr_linii(table_name, NULL) AS nr_inreg
           FROM USER_TABLES ) cite_inreg
           WHERE nr_inreg =
                 (SELECT MAX(nr_inreg)
                  FROM
                  (SELECT f_nr_linii(table_name, NULL)
                       AS nr_inreg
                   FROM USER_TABLES )
                        )
```

Revenim la blocul anonim PL/SQL din listing 10.10. Pentru a modifica valoarea variabilei `v2` din pachetul PAC_STERGE, putem folosi şi un mic artificiu, ca în listing 10.19. Artificiul este interogarea ce "preia" datele din tabela sistem `DUAL`. Expresia din clauza `SELECT` va fi pasată variabilei specificate prin opţiunea `INTO` a comenzii `EXECUTE IMMEDIATE`.

Listing 10.19. O altă modalitate de modificare a variabilei din pachet

```
BEGIN
    pac_sterge.v2:= pac_sterge.v1 ;
    EXECUTE IMMEDIATE 'SELECT '''||'Valoare 2'||''' FROM dual' INTO pac_sterge.v2 ;
```

```
    DBMS_OUTPUT.PUT_LINE(pac_sterge.v2) ;
END ;
```

10.5. 10.5. Interogări multi-linie

Comanda `EXECUTE IMMEDIATE` a fost cea care a rezolvat, până în prezent, toate problemele în care a fost necesar SQL dinamic. A rămas o singură speță, cea a interogărilor ale căror rezultat numără mai multe linii, pentru care for recurge la o formă modificată a comenzii `OPEN` aplicată variabilelor cursor.

10.5.1. Arhivarea tabelei PONTAJE pe ani calendaristici

Pentru cele ce urmează ne propunem crearea unei proceduri destinate arhivării anuale a pontajelor. Deoarece volumul tabelei PONTAJE poate căpăta, în timp, proporții impresionante şi, pe de altă parte, doar rareori se lucrează cu pontaje de pe anii anteriori, mulți responsabili de aplicații preferă arhivarea datelor unui an într-o tabelă de genul *tabelă*_an, şi apoi descongestionarea tabelei "master". Astfel, la începutul anului 2003, administratorul BD salvează toate pontajele care au data de întocmire cuprinsă între 1 ianuarie şi 31 decembrie 2002 într-o tabelă numită PONTAJE_2002, după care se şterg liniile respective din tabela PONTAJE.

Procedura ARHIVARE_PONTAJE din pachetul PAC_ARHIVARE (listing 10.20) pare a rezolva problema la cel mai nepretențios mod cu putință. I se pasează anul (*nnnn*) pentru care se doreşte arhivarea şi, în caz că nu există (verificarea se face apelând la un bloc inclus), se creează tabela PONTAJE_*nnnn* . După verificare/creare, se inserează toate liniile din PONTAJE corespunzătoare anului arhivat, apoi se descongestionează tabela sursă de liniile proaspăt arhivate.

Listing 10.20. Arhivarea simplă a pontajelor dintr-un an

```
CREATE OR REPLACE PACKAGE pac_arhivare AUTHID CURRENT_USER AS
PROCEDURE arhivare_pontaje (an_ salarii.an%TYPE)  ;
END pac_arhivare ;
/
CREATE OR REPLACE PACKAGE BODY pac_arhivare AS
PROCEDURE arhivare_pontaje (an_ salarii.an%TYPE)
IS
    v_unu NUMBER(1) := 0 ;
BEGIN
    -- se verifică dacă există PONTAJE_nnnn
    BEGIN
        SELECT 1 INTO v_unu FROM user_tables WHERE table_name = 'PONTAJE_' || an_ ;
    EXCEPTION
    WHEN NO_DATA_FOUND THEN -- nu există, aşa că se crează tabela
        EXECUTE IMMEDIATE 'CREATE TABLE pontaje_' || an_ ||
                ' AS SELECT * FROM pontaje WHERE 1=2 '  ;

        -- se declară cheia primară a noii tabele
        EXECUTE IMMEDIATE 'ALTER TABLE pontaje_' || an_ ||
                ' ADD CONSTRAINT pk_pontaje_'|| an_ || ' PRIMARY KEY (marca, data) '  ;
    END ;
```

```
        -- se inserează în PONTAJE_nnnn liniile din PONTAJE
        EXECUTE IMMEDIATE 'INSERT INTO pontaje_'|| an_ ||
        ' SELECT * FROM pontaje WHERE TO_NUMBER(TO_CHAR(data, "YYYY")) = :1'
                    USING an_ ;
        -- se şterg din PONTAJE liniile arhivate
        DELETE FROM pontaje WHERE TO_NUMBER(TO_CHAR(data, 'YYYY')) = an_ ;
        COMMIT ;
    END arhivare_pontaje ;

    END pac_arhivare ;
    /
```

Această soluție ridică unele probleme. Dacă, spre exemplu, după ce s-a operat arhivarea, se descoperă că unulele dintre înregistrările din PONTAJE ec au apucat să fie arhivate erau eronate (valorile atributelor `OreLucrate` şi `OreCo` trebuiau inversate), acestea vor fi re-inserate în PONTAJE cu valorile corecte. La apelul următor al procedurii de arhivare, s-ar declanşa însă o eroare datorată violării cheii primare din PONTAJE_2002. Aşa încât modificăm logica procedurii, astfel:

- se verifică existența tabelei PONTAJE_*nnnn*;
- se actualizează eventualele înregistrări din PONTAJE_*nnnn*, dar care prezintă corespondent PONTAJE, modificându-se atributele `OreLucrate`, `OreCo`, `OreNoapte`, `OreAbsNem`;
- se inserează în PONTAJE_*nnnn* liniile din PONTAJE inexistente în tabela arhivă;
- se şterg liniile din PONTAJE inserate sau actualizate în PONTAJE_*nnnn*.

La aşa ambiții, trebuie modificată procedura de arhivare. Dintre variante, alegem una care să folosească SQL dinamic pentru o interogare multi-linie – vezi listing 10.21. Variabila cursor este denumită `v_cursor`. Ideea de bază a noii versiuni este ca, pentru actualizarea liniilor din PONTAJE_*nnnn*, să se încarce în variabila cursor toate înregistrările care sunt prezente în PONTAJE_*nnnn* de la o arhivare mai veche, dar care apar şi în PONTAJE (se consideră că odată ce re-apar în PONTAJE, înregistrările trebuie actualizate în arhivă).

Practic, sâmburele de SQL-ul dinamic este legat de deschiderea variabilei cursor (`OPEN v_cursor FOR ...`) prin care în fraza `SELECT` se specifică denumirea tabelei din care se extrag datele la momentul execuției procedurii. În rest, încărcarea unei linii din cursor şi secvența iterativă de parcurgere a tuturor liniilor din variabila cursor nu prezintă diferențe semnificative față de un cursor "clasic".

Listing 10.21. Interogarea dinamică multi-linie

```
CREATE OR REPLACE PACKAGE pac_arhivare AUTHID CURRENT_USER AS
PROCEDURE arhivare_pontaje (an_ salarii.an%TYPE)  ;
END pac_arhivare ;
/

CREATE OR REPLACE PACKAGE BODY pac_arhivare AS
PROCEDURE arhivare_pontaje (an_ salarii.an%TYPE)
IS
```

```
    TYPE t_refcursor IS REF CURSOR ; -- declararea tipului variabilă-cursor
    v_cursor t_refcursor ;                  -- aceasta este variabila cursor
    rec_cursor pontaje%ROWTYPE ;    -- variabila-înregistrare pentru stocarea unei linii
                                            -- din variabila-cursor
    v_unu NUMBER(1) := 0 ;
    TYPE t_pontaje IS RECORD (orelucrate pontaje.orelucrate%TYPE, oreco pontaje.oreco%TYPE,
        orenoapte pontaje.orenoapte%TYPE, oreabsnem pontaje.oreabsnem%TYPE ) ;
    v_pontaje t_pontaje ;
BEGIN
    BEGIN
        SELECT 1 INTO v_unu FROM user_tables WHERE table_name = 'PONTAJE_' || an_ ;
    EXCEPTION
    WHEN NO_DATA_FOUND THEN -- nu există, aşa că se crează tabela
        EXECUTE IMMEDIATE 'CREATE TABLE pontaje_' || an_ ||
            ' AS SELECT * FROM pontaje WHERE 1=2 ' ;
        -- se declară cheia primară a noii tabele
        EXECUTE IMMEDIATE 'ALTER TABLE pontaje_' || an_ ||
            ' ADD CONSTRAINT pk_pontaje_'|| an_ || ' PRIMARY KEY (marca, data) ' ;
    END ;

    -- se extrag eventualele linii din PONTAJE_nnnn care există în PONTAJE pentru a fi actualizate
    OPEN v_cursor FOR 'SELECT marca, data FROM pontaje_'||an_||
        ' WHERE (marca, data) IN (SELECT marca, data FROM pontaje) ' ;
    LOOP
        FETCH v_cursor INTO rec_cursor ;
        EXIT WHEN v_cursor%NOTFOUND ;
        EXECUTE IMMEDIATE 'SELECT orelucrate, oreco, orenoapte, oreabsnem
                FROM pontaje WHERE marca = :1 AND data = :2 '
                INTO v_pontaje USING rec_cursor.marca, rec_cursor.data ;

        EXECUTE IMMEDIATE 'UPDATE pontaje_' || an_ ||  ' SET orelucrate = :1 , oreco = :2,
            orenoapte = :3, oreabsnem = :4 WHERE marca= :5 AND data = :6 '
            USING v_pontaje.orelucrate, v_pontaje.oreco, v_pontaje.orenoapte,
                    v_pontaje.oreabsnem, rec_cursor.marca, rec_cursor.data ;
    END LOOP ;
    CLOSE v_cursor ;

    -- se insereaza în PONTAJE_nnnn liniile din PONTAJE ce nu există în prima tabelă
    EXECUTE IMMEDIATE 'INSERT INTO pontaje_'|| an_ ||
        ' SELECT * FROM pontaje WHERE TO_NUMBER(TO_CHAR(data, "YYYY")) = :1
            AND (marca, data) NOT IN (SELECT marca, data FROM pontaje_'||an_|| ') ' USING an_ ;

    -- se şterg din PONTAJE liniile arhivate
    DELETE FROM pontaje WHERE TO_NUMBER(TO_CHAR(data, 'YYYY')) = an_ ;

    COMMIT ;
END arhivare_pontaje ;

END pac_arhivare ;
/
```

Merită remarcată, tot ca element de relativă noutate, şi folosirea simultană, în prima comandă `EXECUTE IMMEDIATE` din buclă, a clauzelor `INTO` şi `USING`. Prima serveşte la indicarea variabilei (variabilelor) în care se va salva rezultatul interogării dinamice, iar `USING` specifică variabilele legate, ale căror valori vor substitui parametrii `:1` şi `:2`. Cea de-a doua comandă `EXECUTE IMMEDIATE` din secvenţa iterativă foloseşte variabile legate deopotrivă pentru indicarea

valorilor de preluat (în clauza `SET` a comenzii `UPDATE`) şi predicatului de selecţie (clauza `WHERE`).

10.5.2. Fuzionarea tabelelor arhivă

Arhivarea pontajelor pe ani calendaristici este benefică pentru descongestionarea tabelei PONTAJE, ţinând seama că, de obicei, se editează pontajele de pe luna curentă şi, eventual, luna anterioară. Dacă însă baza de date este operaţională din 1999, iar conducerea firmei doreşte o analiză a pontajelor pentru un interval calendaristic cuprins între 1 februarie 1999 şi 20 februarie 2001, este necesară fuzionarea înregistrărilor a trei tabele: PONTAJE_1999 (pentru înregistrările de după 1 februarie), PONTAJE_2000 şi PONTAJE_2001 (numai înregistrările cuprinse între 1 ianuarie şi 20 februarie). De aceea, este nevoie de o procedură specială care să fuzioneze oricâte tabele, în funcţie de intervalul ales.

Procedura FUZIUNE_PONTAJE este inclusă în pachetul PAC_ARHIVARE şi prezintă doi parametri de intrare, `data_initiala` şi `data_finala`. Fuziunea se realizează într-o tabelă tampon numită PONTAJE_TEMP. Blocul inclus la începutul secţiunii executabile testează existenţa acestei tabele tampon şi, dacă este cazul, o crează. Secvenţa iterativă se execută pentru fiecare an calendaristic din interval. La fiecare an se folosesc două variabile. O variabilă indică data iniţială pentru anul respectiv – `data_in_an_crt` (data iniţială an curent) – care este 1 ianuarie pentru toţi anii din interval, cu excepţia primului, şi o altă variabilă – `data_sf_an_crt` – care este 31 decembrie, cu excepţia ultimului an.

Listing 10.22. Fuziunea tabelelor PONTAJE_*nnnn*

```
CREATE OR REPLACE PACKAGE pac_arhivare AUTHID CURRENT_USER AS
PROCEDURE arhivare_pontaje (an_ salarii.an%TYPE)   ;
PROCEDURE fuziune_pontaje (data_initiala DATE, data_finala DATE) ;
END pac_arhivare ;
/

CREATE OR REPLACE PACKAGE BODY pac_arhivare AS

PROCEDURE fuziune_pontaje (data_initiala DATE, data_finala DATE)
IS
    an_inceput NUMBER (4) ;  -- primul an din intervalul de fuziune
    an_sfirsit NUMBER (4) ;  -- ultimul an din intervalul de fuziune
    data_in_an_crt DATE ;   -- data de început din anul curent
    data_sf_an_crt DATE ;   -- data de final din anul curent
    TYPE rc_tabela IS REF CURSOR ;
    c_tabela rc_tabela ;
    tab_ VARCHAR2(30) ;
BEGIN
    -- folosim o tabelă - PONTAJE_TEMP, pe care o creăm (dacă e nevoie)
    BEGIN
        SELECT table_name INTO tab_ FROM user_tables
            WHERE table_name = 'PONTAJE_TEMP' ;
        EXECUTE IMMEDIATE 'DELETE FROM ' || tab_ ;
    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        EXECUTE IMMEDIATE 'CREATE TABLE pontaje_temp AS SELECT * FROM pontaje
```

```
            WHERE 1=2' ;
    END ;
    -- după acest bloc, PONTAJE_TEMP există şi n-are înregistrări

    -- extragerea anilor iniţial şi final ai intervalului calendaristic
    an_inceput := TO_NUMBER(TO_CHAR(data_initiala, 'YYYY')) ;
    an_sfirsit := TO_NUMBER(TO_CHAR(data_finala, 'YYYY')) ;

    FOR i IN an_inceput .. an_sfirsit LOOP          -- bucla executată pentru fiecare an din interval
        tab_ := 'PONTAJE_' || i ;
        -- cursorul C_TABELA conţine cel mult o linie, dacă există tabela PONTAJE_nnnn,
        --    unde nnnn = i
        OPEN c_tabela FOR 'SELECT table_name FROM user_tables
            WHERE table_name = :1 ' USING tab_ ;
        FETCH c_tabela INTO tab_ ;

        IF c_tabela%ROWCOUNT > 0 THEN
            -- există tabelă-arhivă pentru acest an
            -- pentru anul curent, datele iniţiale şi finale implicite sunt 1 ian. si 31 dec.
              data_in_an_crt := TO_DATE('01/01/'||i, 'DD/MM/YYYY') ;
              data_sf_an_crt := TO_DATE('31/12/'||i, 'DD/MM/YYYY') ;

            -- dacă e primul an din interval, data iniţiala ia valoarea  DATA_IN
            IF i = an_inceput THEN
                    data_in_an_crt := data_initiala ;
            END IF ;

            -- dacă e ultimul an din interval, data finală ia valoarea DATA_SF
            IF i = an_sfirsit THEN          -- este ultimul an din interval
                    data_sf_an_crt := data_finala ;
            END IF  ;

            -- inserarea liniilor din PONTAJE_nnnn în PONTAJE_TEMP
            EXECUTE IMMEDIATE 'INSERT INTO pontaje_temp SELECT * FROM pontaje_' ||
                    i || ' WHERE data BETWEEN :1 AND :2 ' USING data_in_an_crt, data_sf_an_crt ;
        END IF ;
    END LOOP ;
    COMMIT;
END ;

-------------------------------------------------------------------------------
PROCEDURE arhivare_pontaje (an_ salarii.an%TYPE)
IS
    ... vezi listing anterior
END arhivare_pontaje ;

END pac_arhivare ;
```

10.5.3. SQL dinamic "înmănunchiat"

Revenim la această traducere exotică (pe care am lansat-o cu mult tam-tam în capitolul 8) pentru a introduce câteva opţiuni care ar putea ameliora sensibil performanţele blocurilor în care este folosit SQL-ul dinamic. Reamintim celor care au trecut în mare viteză prin sau pe lângă finalul capitolului 8 că înmănunchierea în PL/SQL presupune încărcărea sau parcurgerea unei variabile de tip colecţie – vector asociativ, tabel încapsulat, vector cu mărime variabilă - dintr-o singură

mişcare, asffel încât transferul repetat al controlului, în cadrul unei secvenţe repetitive, între motorul SQL şi motorul PL/SQL este eliminat.

Trei comenzi constituie baza SQL-ului dinamic înmănunchiat: `EXECUTE IMMEDIATE`, `FETCH` şi `FORALL`. Listing-ul 10.23 ilustrează două dintre acestea. Pentru a le pune în valoare, schimbăm logica procedurii de arhivare. Astfel, valorile cheilor primare ale eventualelor înregistrări din PONTAJE_*nnnn* sunt salvate în două variabile de tip colecţie – tablourile asociative `v_marca` şi `v_data` – folosindu-se opţiunea `BULK COLLECT INTO`. Ulterior, în loc de a parcurge printr-o secvenţă iterativă toate componentele celor două variabile-masiv, actualizarea liniilor din PONTAJE_nnnn este realizată printr-o singură comandă, întrebuinţându-se comanda `FORALL... EXECUTE IMMEDIATE`.

Listing 10.23. Folosirea clauzelor BULK COLLECT şi FORALL

```
CREATE OR REPLACE PACKAGE pac_arhivare AUTHID CURRENT_USER AS
PROCEDURE arhivare_pontaje (an_ salarii.an%TYPE)   ;
END pac_arhivare ;
/

CREATE OR REPLACE PACKAGE BODY pac_arhivare AS
PROCEDURE arhivare_pontaje (an_ salarii.an%TYPE)
IS
    v_unu NUMBER(1) := 0 ;
    TYPE t_marca IS TABLE OF pontaje.marca%TYPE INDEX BY PLS_INTEGER ;
    v_marca t_marca ;
    TYPE t_data IS TABLE OF pontaje.data%TYPE INDEX BY PLS_INTEGER ;
    v_data t_data ;
BEGIN
    BEGIN
        ... identic listingului 10.21
    END ;

    -- se salvează în doi vectori cheile primare ale înregistrărilor ce urmează a fi actualizate
    EXECUTE IMMEDIATE 'SELECT marca, data FROM pontaje_'||an_ || ' WHERE (marca, data) IN
        (SELECT marca, data FROM pontaje) ' BULK COLLECT INTO v_marca, v_data ;

    FORALL i IN 1..v_marca.COUNT EXECUTE IMMEDIATE
            'UPDATE pontaje_'|| an_ || ' p1
        SET (orelucrate, oreco, orenoapte, oreabsnem) =
            (SELECT orelucrate, oreco, orenoapte, oreabsnem FROM pontaje  p2
              WHERE p2.marca = p1.marca AND p1.data = p2.data)
        WHERE marca = :1 AND data = :2 ' USING v_marca(i), v_data(i) ;

    -- se insereaza în PONTAJE_nnnn liniile din PONTAJE ce nu există în prima tabelă
    ... şi apoi se şterg din PONTAJE liniile arhivate
    ... vezi listing 10.21

    COMMIT ;
END arhivare_pontaje ;

END pac_arhivare ;
```