7. Modularizarea aplicaţiilor prin utilizarea pachetelor

Pachetul (*package*) permite încapsularea într-o unitate logică în baza de date a procedurilor, funcţiilor, cursoarelor, tipurilor, constantelor, variabilelor şi excepţiilor. Pachetele sunt unităţi de program care sunt compilate, depanate şi testate. Ele reprezintă obiecte ale bazei de date care grupează tipuri, obiecte şi subprograme *PL/SQL* cu o legătură logică între ele.

Pachetele stocate permit gruparea procedurilor şi funcţiilor înrudite, precum şi partajarea variabilelor de către acestea. Împachetarea subprogramelor prezintă beneficii importante. Când este referit un pachet (când este apelată pentru prima dată o construcţie a pachetului), întregul pachet este încărcat în zona globală sistem (*SGA*) şi este pregătit pentru execuţie. Plasarea pachetului în *SGA* prezintă avantajul vitezei de execuţie, deoarece *server*-ul nu mai trebuie să aducă informaţia despre pachet de pe disc, aceasta fiind deja în memorie. Prin urmare, apeluri ulterioare ale unor construcţii din acelaşi pachet nu solicită operaţii *I*/*O* de pe disc.

Dacă apar proceduri şi funcţii înrudite care trebuie să fie executate împreună, este convenabil ca acestea să fie grupate într-un pachet stocat. Este de remarcat că în memorie există o singură copie a unui pachet, pentru toţi utilizatorii.

Spre deosebire de subprograme, pachetele nu pot fi apelate, nu pot transmite parametri şi nu pot fi încuibărite.

În mod uzual un pachet are două părţi, fiecare fiind stocată separat în dicţionarul datelor.

- Specificaţia pachetului (*package specification*) reprezintă partea „vizibilă“, adică interfaţa cu aplicaţii sau cu alte unităţi program. În această parte se descrie tot ce este necesar utilizatorului pentru a folosi pachetul. Se declară tipuri, constante, variabile, excepţii, cursoare şi subprograme.
- Corpul pachetului (*package body*) reprezintă partea „acunsă“, mascată de restul aplicaţiei, adică realizarea specificaţiei. Corpul defineşte cursoare şi subprograme, implementând specificaţia. Obiectele conţinute în corpul pachetului sunt fie private, fie publice.

Prin urmare, specificaţia defineşte interfaţa utilizatorului cu pachetul, iar corpul pachetului conţine codul care implementează operaţiile definite în specificaţie. Crearea unui pachet se face în două etape care corespund creării părţilor acestuia.

Un pachet poate cuprinde fie doar partea de specificaţie, fie atât

specificaţia, cât şi corpul pachetului. În cazul în care conţine doar specificaţia, pachetul va cuprinde numai definiţii de tipuri şi declaraţii de date. Corpul pachetului poate fi schimbat fără a fi necesară schimbarea specificaţiei pachetului. Dacă specificaţia este schimbată, aceasta invalidează automat corpul pachetului, deoarece corpul depinde de specificaţie. Specificaţia şi corpul pachetului sunt unităţi compilate separat. Corpul unui pachet poate fi compilat doar după ce specificaţia acestuia a fost compilată cu succes.

Un pachet are următoarea formă generală:

CREATE PACKAGE *nume_pachet* {***IS*** | ***AS***} *-- specificaţia*
/ interfaţa utilizator, care conţine: declaraţii de tipuri şi obiecte publice, specificaţii de subprograme */*
END [*nume_pachet*];
CREATE PACKAGE BODY *nume_pachet* {***IS*** | ***AS***} *-- corpul*
/ implementarea, care conţine: declaraţii de obiecte şi tipuri private, corpuri de subprograme specificate în partea de interfaţă */*
[***BEGIN***]
/ instrucţiuni de iniţializare, executate o singură dată, atunci când pachetul este invocat prima oară într-o sesiune a unui utilizator */*
END [*nume_pachet*];

Specificaţia unui pachet

Specificaţia unui pachet cuprinde declararea procedurilor, funcţiilor, constantelor, variabilelor şi excepţiilor care pot fi accesibile utilizatorilor prin intermediul pachetului, adică declararea obiectelor de tip *PUBLIC* din pachet. Acestea pot fi utilizate în proceduri sau comenzi care nu aparţin pachetului, dar care au privilegiul *EXECUTE* asupra acestuia.

Variabilele declarate în specificaţia unui pachet sunt globale pachetului şi sesiunii. Ele sunt iniţializate implicit cu valoarea *null*, dacă nu este specificată explicit o altă valoare.

Crearea specificaţiei se face prin comanda:

CREATE [***OR REPLACE***] ***PACKAGE*** [*schema*.]*nume_pachet*
[***AUTHID*** {***CURRENT_USER*** | ***DEFINER***}]
{***IS*** | ***AS***}
specificaţie_pachet;

Opţiunea *specificaţie_pachet* poate include declaraţii de tipuri, variabile, cursoare, excepţii, funcţii, proceduri, clauze *PRAGMA* etc. În secţiunea declarativă, un obiect trebuie să fie declarat înainte de a fi referit. Opţiunea *OR REPLACE* se specifică dacă există deja corpul pachetului.

Clauza *AUTHID* specifică faptul că subprogramele pachetului se execută

cu drepturile proprietarului (implicit) sau ale utilizatorului curent. De asemenea, această clauză precizează dacă referinţele la obiecte sunt rezolvate în schema proprietarului pachetului sau în cea a utilizatorului curent.

Corpul unui pachet

Corpul unui pachet conţine codul *PL/SQL* pentru obiectele declarate în specificaţia acestuia şi obiectele private ale pachetului. De asemenea, corpul poate include o secţiune declarativă în care sunt specificate definiţii locale de tipuri, variabile, constante, proceduri şi funcţii. Obiectele private sunt vizibile numai în interiorul corpului pachetului şi pot fi accesate numai de către funcţiile şi procedurile din pachetul respectiv.

Corpul pachetului este opţional şi nu este necesar să fie creat dacă specificaţia pachetului nu conţine declaraţii de proceduri sau funcţii.

Este importantă ordinea în care subprogramele sunt definite în interiorul corpului pachetului. O variabilă trebuie declarată înainte de a fi referită de altă variabilă sau subprogram, iar un subprogram privat trebuie declarat sau definit înainte de a fi apelat de alte subprograme. Declaraţiile publice (interfaţa operaţională) sunt vizibile aplicaţiei, spre deosebire de declaraţiile private (implementarea efectivă a corpului) care sunt mascate aplicaţiei.

Comanda care permite crearea corpului unui pachet are sintaxa:

CREATE [***OR REPLACE***] ***PACKAGE BODY*** [*schema.*]*nume_pachet*
{***IS*** | ***AS***}
corp_pachet;

Un pachet este instanţiat atunci când este apelat prima dată. Aceasta presupune că pachetul este citit de pe disc, adus în memorie şi este executat codul compilat al subprogramului apelat. De asemenea, în acest moment se alocă memorie tuturor variabilelor definite în pachet.

În multe cazuri este necesar să se facă o iniţializare atunci când pachetul este instanţiat prima dată într-o sesiune. Aceasta se realizează prin adăugarea unei secţiuni de iniţializare (opţională) în corpul pachetului, secţiune încadrată între cuvintele cheie *BEGIN* şi *END*. Secţiunea conţine un cod de iniţializare care este executat atunci când pachetul este invocat pentru prima dată.

Crearea pachetului face ca acesta să fie disponibil utilizatorului care l-a creat şi oricărui alt utilizator ce deţine privilegiul *EXECUTE*.

Referinţa la un tip sau obiect specificat în pachet se face prefixând numele acestuia cu numele pachetului. În corpul pachetului, obiectele din specificaţie pot fi referite fără a le prefixa cu numele pachetului.

Procesul de creare a specificaţei şi corpului unui pachet urmează un algoritm similar celui întâlnit în crearea subprogramelor *PL/SQL* independente. În ambele cazuri sunt verificate erorile sintactice şi semantice, iar modulul este

depus în dicţionarul datelor. După ce a fost depus corpul pachetului, sunt verificate instrucţiunile *SQL* individuale, adică se cercetează dacă obiectele referite există şi dacă utilizatorul le poate accesa. Sunt comparate declaraţiile de subprograme din specificaţia pachetului cu cele din corpul pachetului (dacă au acelaşi număr şi tip de parametri). Orice eroare detectată la compilarea specificaţiei sau a corpului pachetului este marcată în dicţionarul datelor.

După ce specificaţia şi corpul pachetului sunt compilate, ele devin obiecte în schema curentă. În vizualizarea *USER_OBJECTS* din dicţionarul datelor, vor fi create două noi linii:

OBJECT_TYPE *OBJECT_NAME*
PACKAGE *nume_pachet*
PACKAGE BODY *nume_pachet*

Modificarea şi suprimarea pachetelor

Modificarea unui pachet presupune recompilarea sa (pentru a putea modifica metoda de acces şi planul de execuţie) şi se realizează prin comanda:

ALTER PACKAGE [*schema.*]*nume_pachet*
COMPILE [{***PACKAGE*** | ***BODY***}];

Schimbarea corpului pachetului nu cere recompilarea construcţiilor dependente, în timp ce schimbările în specificaţia acestuia solicită recompilarea fiecărui subprogram stocat care face referinţă la pachetul respectiv. Din acest motiv, este bine ca specificaţia să conţină cât mai puţine construcţii.

Dacă se doreşte modificarea sursei, utilizatorul poate recrea pachetul (folosind opţiunea *OR REPLACE*) înlocuindu-l pe cel existent.

Suprimarea unui pachet se realizează prin comanda:

DROP PACKAGE [*schema*.]*nume_pachet* [{***PACKAGE*** | ***BODY***}];

Dacă în cadrul comenzii apare opţiunea *BODY*, este distrus doar corpul pachetului, în caz contrar sunt suprimate atât specificaţia, cât şi corpul acestuia. Dacă pachetul este suprimat, toate obiectele dependente de acesta devin invalide. Dacă este distrus numai corpul, toate obiectele dependente de acesta rămân valide. În schimb, nu pot fi apelate subprogramele declarate în specificaţia pachetului sau cele dependente de acestea, până când nu este recreat corpul pachetului.

Pentru ca un utilizator să poată suprima un pachet trebuie fie ca pachetul să aparţină schemei utilizatorului, fie ca utilizatorul să aibă privilegiul sistem *DROP ANY PROCEDURE*.

Apelarea unui pachet se realizează în funcţie de mediul (*SQL* sau *PL/SQL*) care solicită un obiect din pachetul respectiv.

În *PL/SQL* se face prin referirea:

nume_pachet.nume_componentă [(*listă_de_argumente*)];

În *SQL* se face prin comanda:

EXECUTE *nume_pachet.nume_componentă* [(*listă_de_argumente*)].

***Exemplu*:**

Să se creeze un pachet ce include o procedură prin care se verifică dacă o combinaţie specificată dintre atributele *cod_artist* şi *gen* este o combinaţie care există în tabelul *fotografie*.

```
CREATE PACKAGE verif_pachet IS
  PROCEDURE verifica
      (p_idartist IN fotografie.cod_artist%TYPE,
       p_gen      IN fotografie.gen%TYPE);
END verif_pachet;
/
CREATE OR REPLACE PACKAGE BODY verif_pachet
IS
  i NUMBER := 0;
  CURSOR c_fotografie IS
    SELECT cod_artist, gen
    FROM   fotografie;
  TYPE     fotografie_table_tip     IS     TABLE     OF
c_fotografie%ROWTYPE
    INDEX BY BINARY_INTEGER;
  art_gen fotografie_table_tip;
  PROCEDURE verifica
      (p_idartist IN fotografie.cod_artist%TYPE,
       p_gen      IN fotografie.gen%TYPE)
  IS
  BEGIN
    FOR k IN art_gen.FIRST..art_gen.LAST LOOP
      IF p_idartist = art_gen(k).cod_artist
         AND p_gen = art_gen(k).gen THEN
           RETURN;
      END IF;
    END LOOP;
    RAISE_APPLICATION_ERROR(-20777,'combinatie
eronata');
  END verifica;
BEGIN
  FOR v_foto IN c_fotografie LOOP
    art_gen(i) := v_foto;
```

```
    i := i+1;
  END LOOP;
END verif_pachet;
/
```

Apelarea în *PL/SQL* a unui obiect (*verifica*) din pachet se face prin:

```
verif_pachet.verifica (100, 'Natura');
```

Utilizarea în *SQL* a unui obiect (*verifica*) din pachet se face prin:

```
EXECUTE verif_pachet.verifica (100, 'Natura');
```

Una dintre posibilităţile interesante oferite de pachetele *PL/SQL* este aceea de a crea proceduri/funcţii *overload*. Acest tip de programare este folositor atunci când este necesar un singur subprogram care să execute aceeaşi operaţie pe obiecte de tipuri diferite (diferite tipuri de parametri de intrare). Atunci când este apelat un subprogram *overload* sistemul decide, pe baza tipului şi numărului de parametri, instanţa care trebuie să fie executată. Numai subprogramele locale sau care aparţin unui pachet pot fi *overload*.

***Observaţii*:**

- Un declanşator nu poate apela o procedură sau o funcţie ce conţine una dintre comenzile *COMMIT*, *ROLLBACK*, *SAVEPOINT*. Prin urmare, pentru flexibilitatea apelului (de către declanşatori) subprogramelor conţinute în pachete, trebuie verificat că nici una dintre procedurile sau funcţiile pachetului nu conţin aceste comenzi.
- Procedurile şi funcţiile conţinute într-un pachet pot fi referite din fişiere *SQL*Plus*, din subprograme stocate *PL/SQL*, din aplicaţii *client* (de exemplu, *Oracle Forms* sau *Power Builder*), din declanşatori (bază de date), din programe aplicaţie scrise în limbaje *LG3*.
- Într-un pachet nu pot fi referite variabile gazdă.
- Într-un pachet, mai exact în corpul acestuia, sunt permise declaraţii *forward*. Posibilităţile oferite de aceste declaraţii au fost comentate în capitolul destinat subprogramelor.

Dacă un subprogram ce aparţine unui pachet este apelat de un subprogram *stand-alone* trebuie remarcat că:

- dacă se schimbă doar corpul pachetului, subprogramul care referă o construcţie a pachetului rămâne valid;
- dacă specificaţia pachetului se schimbă, atunci atât subprogramul care referă o construcţie a pachetului, precum şi corpul pachetului sunt invalidate.

Dacă un subprogram stocat referit de un pachet se schimbă, atunci întregul corp al pachetului este invalidat, dar specificaţia pachetului rămâne validă.

Pachete predefinite (opțional)

PL/SQL conţine pachete predefinite (care sunt deja compilate în baza de date) utilizate pentru dezvoltarea de aplicaţii. Aceste pachete adaugă noi funcţionalităţi limbajului, precum protocoale de comunicaţie, acces la fişierele sistemului etc. Apelarea unor proceduri din aceste pachete se face prin prefixarea numelui procedurii cu numele pachetului.

Dintre cele mai importante pachete predefinite se remarcă:

- *DBMS_OUTPUT* (permite afişarea de informaţii);
- *DBMS_DDL* (furnizează accesul la anumite comenzi *LDD* care pot fi folosite în programe *PL/SQL*);
- *UTL_FILE* (permite citirea din fişierele sistemului de operare, respectiv scrierea în astfel de fişiere);
- *UTL_HTTP* (foloseşte protocolul *HTTP* pentru accesarea din *PL/SQL* a datelor publicate pe *Internet*);
- *UTL_TCP* (permite aplicaţiilor *PL/SQL* să comunice cu *server*-e externe utilizând protocolul *TCP/IP*);
- *DBMS_SQL* (accesează baza de date folosind *SQL* dinamic);
- *DBMS_JOB* (permite planificarea pentru execuţie a programelor *PL/SQL* şi execuţia acestora);
- *DBMS_PIPE* (permite operaţii de comunicare între două sau mai multe procese conectate la aceeaşi instanţă *Oracle*);
- *DBMS_LOCK* (permite folosirea exclusivă sau partajată a unei resurse);
- *DBMS_SNAPSHOT* (permite exploatarea clişeelor);
- *DBMS_UTILITY* (oferă utilităţi *DBA*, analizează obiectele unei scheme particulare, verifică dacă *server*-ul lucrează în mod paralel etc.);
- *DBMS_LOB* (realizează accesul la date de tip *LOB*, permiţând compararea datelor *LOB*, adăugarea de date la un *LOB*, copierea datelor dintr-un *LOB* în altul, ştergerea unor porţiuni din date *LOB*, deschiderea, închiderea şi regăsirea de informaţii din date *BFILE* etc).
- *DBMS_STANDARD* este un pachet predefinit fundamental prin care se declară tipurile, excepţiile, subprogramele care pot fi utilizate automat în toate programele *PL/SQL*. Conţinutul pachetului este vizibil tuturor aplicaţiilor. Pentru referirea componentelor sale nu este necesară prefixarea cu numele pachetului. De exemplu, utilizatorul poate folosi în aplicaţia sa ori de câte ori are nevoie funcţia *ABS* (*x*) din pachetul *DBMS_STANDARD*, care reprezintă valoarea absolută a numărului *x*, fără a prefixa numele funcţiei cu numele pachetului.

Pachetul *DBMS_OUTPUT*

Un pachet frecvent utilizat este *DBMS_OUTPUT*, care permite afişarea de informaţii atunci când se execută un program *PL/SQL*. Pachetul este util pentru depanarea procedurilor stocate şi a declanşatorilor sau pentru generarea de rapoarte.

DBMS_OUTPUT lucrează cu un *buffer* (conţinut în *SGA*) în care poate fi scrisă informaţie utilizând procedurile *PUT*, *PUT_LINE* şi *NEW_LINE*. Această informaţie poate fi regăsită folosind procedurile *GET_LINE* şi *GET_LINES*. Procedura *DISABLE* dezactivează toate apelurile la pachetul *DBMS_OUTPUT* (cu excepţia procedurii *ENABLE*) şi curăţă *buffer*-ul de orice informaţie.

Inserarea în *buffer* a unui sfârşit de linie se face prin procedura *NEW_LINE*. Procedura *PUT* depune (scrie) informaţie în *buffer*, informaţie care poate fi de tip *NUMBER*, *VARCHAR2* sau *DATE*. *PUT_LINE* are acelaşi efect ca procedura *PUT*, dar inserează şi un sfârşit de linie. Procedurile *PUT* şi *PUT_LINE* sunt *overload*, astfel încât informaţia poate fi scrisă în format nativ.

Procedura *GET_LINE* regăseşte o singură linie de informaţie (de dimensiune maximă 255) din *buffer*, dar sub formă de şir de caractere. Procedura *GET_LINES* regăseşte mai multe linii (*nr_linii*) din *buffer* şi le depune într-un tablou (*nume_tab*) *PL/SQL* care are elemente de tip şir de caractere. Valorile sunt plasate în tabel începând cu linia zero. Specificaţia este următoarea:

```
TYPE string255_table IS TABLE OF VARCHAR2(255)
  INDEX BY BINARY_INTEGER;
PROCEDURE GET_LINES
  (nume_tab OUT string255_table, nr_linii IN OUT
INTEGER);
```

Parametrul *nr_linii* este şi parametru de tip *OUT*, deoarece numărul liniilor solicitate poate să nu corespundă cu numărul de linii din *buffer*. De exemplu, pot fi solicitate 10 linii, iar în *buffer* să fie doar 6 linii. Atunci doar primele 6 linii din tabel sunt returnate.

Dezactivarea referirilor la pachet se poate realiza prin procedura *DISABLE*, iar activarea referirilor se face cu ajutorul procedurii *ENABLE*. Aceasta are un parametru de intrare opţional, dimensiunea *buffer*-ului, care implicit este 2000. Procedura *ENABLE* poate fi apelată de mai multe ori într-o sesiune, iar dimensiunea *buffer*-ului va fi setată la cea mai mare valoare dată de aceste apeluri. Prin urmare, dimensiunea nu este întotdeauna dată de ultimul apel la procedură.

***Exemplu*:**

Următorul exemplu plasează în *buffer* (apelând de trei ori procedura *PUT*)

toate informaţiile într-o singură linie.

```
DBMS_OUTPUT.PUT(:fotografie.valoare||:
fotografie.cod_artist);
DBMS_OUTPUT.PUT(:fotografie.cod_fotografie);
DBMS_OUTPUT.PUT(:fotografie.cod_expozitie);
```

Informaţia respectivă va fi găsită printr-un singur apel *GET_LINE*, dacă aceste trei comenzi sunt urmate de:

```
DBMS_OUTPUT.NEW_LINE;
```

Altfel, nu se va vedea nici un efect al acestor comenzi deoarece *PUT* plasează informaţia în *buffer*, dar nu adaugă sfârşit de linie.

Atunci când este utilizat pachetul *DBMS_OUTPUT*, pot să apară erorile *buffer overflow* şi *line length overflow*. Tratarea acestor erori se face apelând procedura *RAISE_APPLICATION_ERROR* din pachetul *DBMS_STANDARD*.

Pachetul *DBMS_SQL*

Pachetul *DBMS_SQL* permite folosirea dinamică a comenzilor *SQL* în proceduri stocate sau în blocuri anonime şi analiza gramaticală a comenzilor *LDD*. Aceste comenzi nu sunt încorporate în programul sursă, ci sunt depuse în şiruri de caractere. O comandă *SQL* dinamică este o instrucţiune *SQL* care conţine variabile ce se pot schimba în timpul execuţiei.

Pot fi utilizate instrucţiuni *SQL* dinamice pentru:

- a crea o procedură care operează asupra unui tabel al cărui nume nu este cunoscut decât în momentul execuţiei;
- a scrie şi executa o comandă *LDD*;
- a scrie şi executa o comandă *GRANT*, *ALTER SESSION* etc.

În *PL/SQL* aceste comenzi nu pot fi executate static. Pachetul *DBMS_SQL* permite, de exemplu, ca într-o procedură stocată să apară comanda *DROP TABLE*. Evident, folosirea acestui pachet pentru a executa comenzi *LDD* poate genera interblocări. De exemplu, dacă pachetul este utilizat pentru a şterge o procedură care, însă, este folosită ulterior ştergerii ei.

SQL dinamic suportă toate tipurile de date *SQL*, dar nu şi pe cele specifice *PL/SQL*. Unica excepţie o constituie faptul că o înregistrare *PL/SQL* poate să apară în clauza *INTO* a comenzii *EXECUTE IMMEDIATE.*

Orice comandă *SQL* (pentru a fi executată) trebuie să treacă prin nişte etape, cu observaţia că unele dintre acestea pot fi evitate. Etapele presupun:

- analizarea gramaticală a comenzii, adică verificarea sintactică a comenzii, validarea acesteia, asigurarea că toate referinţele la obiecte sunt corecte şi că există privilegiile referitoare la acele obiecte (*parse*);
- obţinerea de valori pentru variabilele de legătură din comanda *SQL*

(*binding variables*);

- executarea comenzii (*execute*);
- selectarea liniilor rezultatului (se referă la cereri, nu la operaţii de reactualizare);
- încărcarea acestor linii (*fetch*).

Dintre subprogramele pachetului *DBMS_SQL*, care permit implementarea etapelor amintite anterior se remarcă:
OPEN_CURSOR (deschide un nou cursor, adică se stabileşte o zonă de memorie în care este procesată comanda *SQL*);

- *PARSE* (stabileşte validitatea comenzii *SQL*, adică instrucţiunea este verificată sintactic şi asociată cursorului deja deschis);
- *BIND VARIABLE* (leagă valoarea dată de variabila corespunzătoare din instrucţiunea *SQL* analizată);
- *EXECUTE* (execută comanda *SQL* şi returnează numărul de linii procesate);
- *FETCH_ROWS* (regăseşte o linie pentru un cursor specificat, iar pentru mai multe linii se foloseşte un *LOOP*);
- *CLOSE_CURSOR* (închide cursorul specificat).

***Exemplu*:**

Să se construiască o procedură care foloseşte *SQL* dinamic pentru a şterge liniile unui tabel specificat (*num_tab*). Subprogramul furnizează ca rezultat numărul liniilor şterse (*nr_lin*).

```
CREATE OR REPLACE PROCEDURE sterge_linii
  (num_tab IN VARCHAR2, nr_lin OUT NUMBER) AS
  nume_cursor INTEGER;
BEGIN
  nume_cursor := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE (nume_cursor, 'DELETE FROM ' ||
                  num_tab, DBMS_SQL.V7);
  nr_lin := DBMS_SQL.EXECUTE (nume_cursor);
  DBMS_SQL.CLOSE_CURSOR (nume_cursor);
END;
```

Prin argumentul *DBMS_SQL.V7* este specificată versiunea 7 a sistemului şi el reprezintă modul în care *Oracle* tratează execuţia comenzilor *SQL*.

Ştergerea efectivă a liniilor tabelului *fotografie* se realizează prin secvenţa:

```
VARIABLE linii_sterse NUMBER
EXECUTE sterge_linii ('fotografie', :linii_sterse)
PRINT linii_sterse
```

Pentru a executa o instrucţiune *SQL* dinamic poate fi utilizată şi comanda *EXECUTE IMMEDIATE*, care va fi analizată în capitolul următor. Comanda poate fi folosită atât pentru interogările ce returnează o singură linie, cât şi pentru cele care returnează mai multe linii.

Pachetul *DBMS_DDL*

Pachetul *DBMS_DDL* furnizează accesul la anumite comenzi *LDD* care pot fi folosite în subprograme *PL/SQL* stocate. În felul acesta pot fi accesate (în *PL/SQL*) comenzile *ALTER* sau *ANALYZE*.

Pachetul include procedura *ALTER_COMPILE* care permite recompilarea explicită a unui program modificat (procedură, funcţie, declanşator, pachet, corp de pachet). Ea are următoarea formă sintactică:

ALTER_COMPILE (*tip_obiect*, *nume_schema*, *nume_obiect*);

Procedura este echivalentă cu instrucţiunea *SQL*:

ALTER PROCEDURE | ***FUNCTION*** | ***PACKAGE*** [*schema.*] *nume*
COMPILE [***BODY***]

Cu ajutorul procedurii *ANALYZE_OBJECT* poate fi analizat un obiect de tip *table*, *cluster* sau *index*. Procedura furnizează statistici referitoare la obiectele amintite. De exemplu, se pot obţine numărul liniilor sau numărul de blocuri ale unui tabel, lungimea medie a unei linii, numărul valorilor distincte ale unei coloane, numărul elementelor *null* dintr-o coloană, distribuţia datelor (histograma) etc.

Procedura are forma sintactică:

ANALYZE_OBJECT (*tip_obiect*, *nume_schema*, *nume_obiect*, *metoda*, *număr_linii_estimate*, *procent*, *opţiune_metoda*, *nume_partiţie*);

Metodele care pot fi utilizate în procedura *ANALYZE_OBJECT* sunt *COMPUTE*, *ESTIMATE* sau *DELETE*. Prin aceste metode se cuantifică distribuţia datelor şi caracteristicile de stocare. *DELETE* determină ştergerea statisticilor (depuse în *DD*) referitoare la obiectul analizat. *COMPUTE* calculează statistici referitoare la un obiect analizat şi le depune în *DD*, iar *ESTIMATE* estimează statistici. Statisticile calculate sau estimate sunt utilizate pentru optimizarea planului de execuţie a comenzilor *SQL* care accesează obiectele analizate.

Procedura este echivalentă cu instrucţiunea:

ANALYZE TABLE | ***CLUSTER*** | ***INDEX*** [*nume_schema.*]*nume_obiect*
[*metoda*] ***STATISTICS*** [***SAMPLE*** *n*] [***ROWS*** | ***PERCENT***]]

Dacă *nume_schema* este *null*, atunci se presupune că obiectul aparţine schemei curente. Pentru *tip_obiect* sunt permise variantele *table*, *index* sau

cluster. Parametrul *procent* reprezintă procentajul liniilor de estimat şi este ignorat dacă este specificat numărul liniilor de estimat. În mod implicit, ultimele patru argumente ale procedurii iau valoarea *null*.

Argumentul *opţiune_metoda* poate avea forma:

[***FOR TABLE***] [***FOR ALL INDEXES***]

[***FOR ALL*** [***INDEXED***] ***COLUMNS***] [***SIZE*** *n*]

Pentru metoda *ESTIMATE* trebuie să fie prezentă una dintre aceste opţiuni.

***Exemplu*:**

Utilizând pachetul *DBMS_DDL* şi metoda *COMPUTE*, să se creeze o procedură care analizează un obiect furnizat ca parametru.

```
CREATE OR REPLACE PROCEDURE analiza
    (p_obiect_tip  IN VARCHAR2,
     p_obiect_nume IN VARCHAR2)
IS
BEGIN
  DBMS_DDL.ANALYZE_OBJECT(p_obiect_tip, USER,
                          UPPER(p_obiect_nume),
'COMPUTE');
END;
```

Procedura se testează (relativ la tabelul *fotografie*) în felul următor:

```
EXECUTE analiza ('TABLE', 'fotografie')
SELECT LAST_ANALYZED
FROM   USER_TABLES
WHERE  TABLE_NAME = 'FOTOGRAFIE';
```

Pachetul *DBMS_JOB*

Pachetul *DBMS_JOB* este utilizat pentru planificarea programelor *PL/SQL* în vederea execuţiei. Cu ajutorul acestui pachet se pot executa programe *PL/SQL* la momente determinate de timp, se pot şterge sau suspenda programe din lista de planificări pentru execuţie, se pot rula programe de întreţinere în timpul perioadelor de utilizare scăzută etc.

Dintre subprogramele acestui pachet se remarcă:

- *SUBMIT* – adaugă un nou *job* în coada de aşteptare a *job*-urilor;
- *REMOVE* – şterge un *job* specificat din coada de aşteptare a *job*-urilor;
- *RUN* – execută imediat un *job* specificat;
- *BROKEN* – dezactivează execuţia unui *job* care este marcat ca *broken* (implicit, orice *job* este *not broken*, iar un *job* marcat *broken* nu se execută);

- *CHANGE* – modifică argumentele *WHAT*, *NEXT_DATE*, *INTERVAL*;
- *WHAT* – modifică descrierea unui *job* specificat;
- *NEXT_DATE* – modifică momentul următoarei execuţii a unui *job*;
- *INTERVAL* – modifică intervalul între diferite execuţii ale unui *job*.

Fiecare dintre subprogramele pachetului are argumente specifice. De exemplu, procedura *DBMS_JOB.SUBMIT* are ca argumente:

- *JOB* – de tip *OUT*, un identificator pentru *job* (*BINARY_INTEGER*);
- *WHAT* – de tip *IN*, codul *PL/SQL* care va fi executat ca un *job* (*VARCHAR2*);
- *NEXT_DATE* – de tip *IN*, data următoarei execuţii a *job*-ului (implicit este *SYSDATE*);
- *INTERVAL* – de tip *IN*, funcţie care furnizează intervalul dintre execuţiile *job*-ului (*VARCHAR2*, implicit este *null*);
- *NO_PARSE* – de tip *IN*, variabilă logică care indică dacă *job*-ul trebuie analizat gramatical (*BOOLEAN*, implicit este *FALSE*).

Dacă unul dintre parametri *WHAT*, *INTERVAL* sau *NEXT_DATE* are valoarea *null*, atunci este folosită ultima valoare asignată acestora.

***Exemplu*:**

Să se utilizeze pachetul *DBMS_JOB* pentru a plasa pentru execuţie în coada de aşteptare a *job*-urilor, procedura *verifica* din pachetul *verif_pachet.*

```
VARIABLE num_job NUMBER
BEGIN
  DBMS_JOB.SUBMIT(
    job => :num_job,
    what => 'verif_pachet.verifica(100,''Natura'');',
    next_date => TRUNC(SYSDATE+1),
    interval => 'TRUNC(SYSDATE+1)');
  COMMIT;
END;
/
PRINT num_job
```

Vizualizarea *DBA_JOBS* din dicţionarul datelor furnizează informaţii referitoare la starea *job*-urilor din coada de aşteptare, iar vizualizarea *DBA_JOBS_RUNNING* conţine informaţii despre *job*-urile care sunt în curs de execuţie. Vizualizările pot fi consultate doar de utilizatorii care au privilegiul *SYS.DBA_JOBS*.

Exemplu:

```
SELECT JOB, LOG_USER, NEXT_DATE, BROKEN, WHAT
```

```
FROM    DBA_JOBS;
```

Pachetul *UTL_FILE*

Pachetul *UTL_FILE* permite programului *PL/SQL* citirea din fişierele sistemului de operare, respectiv scrierea în aceste fişiere. El este utilizat pentru exploatarea fişierelor text.

Pachetul procesează fişierele într-o manieră clasică:

- verifică dacă fişierul este deschis (funcţia *IS_OPEN*);
- dacă fişierul nu este deschis, îl deschide şi returnează un *handler* de fişier (de tip *UTL_FILE.FILE_TYPE*) care va fi utilizat în următoarele operaţii *I*/*O* (funcţia *FOPEN*);
- procesează fişierul (citire/scriere din/în fişier);
- închide fişierul (procedura *FCLOSE*).

Prin procedura *GET_LINE*, pachetul *UTL_FILE* citeşte o linie de text din fişierul deschis pentru citire şi o plasează într-un *buffer* de tip şir de caractere, iar prin procedurile *PUT* şi *PUT_LINE* scrie un text din *buffer* în fişierul deschis pentru scriere sau adăugare.

Utilizarea componentelor acestui pachet pentru procesarea fişierelor sistemului de operare poate declanşa următoarele excepţii:

- *INVALID_PATH* – numele sau locaţia fişierului sunt invalide;
- *INVALID_MODE* – parametrul *OPEN_MODE* (prin care se specifică dacă fişierul este deschis pentru citire, scriere, adăugare) este invalid;
- *INVALID_FILEHANDLE* – *handler*-ul de fişier obţinut în urma deschiderii este invalid;
- *INVALID_OPERATION* – operaţie invalidă asupra fişierului;
- *READ_ERROR* – o eroare a sistemului de operare a apărut în timpul operaţiei de citire;
- *WRITE_ERROR* – o eroare a sistemului de operare a apărut în timpul operaţiei de scriere;
- *INTERNAL_ERROR* – o eroare nespecificată a apărut în *PL/SQL*.

Pachetele *DBMS_PIPE* şi *DBMS_ALERT*

Pachetul *DBMS_PIPE* permite operaţii de comunicare între două sau mai multe sesiuni conectate la aceeaşi bază de date. De exemplu, pachetul poate fi utilizat pentru comunicarea dintre o procedură stocată şi un program *Pro*C*. Comunicarea se face prin conducte (*pipe*). O conductă este o zonă de memorie utilizată de un proces pentru a transmite informaţie altui proces. Informaţia trimisă prin conductă este depusă într-un *buffer* din *SGA*. Toate informaţiile din conductă sunt pierdute atunci când instanţa este închisă.

Conductele sunt asincrone, ele operând independent de tranzacţii. Dacă un

anumit mesaj a fost transmis, nu există nici o posibilitate de oprire a acestuia, chiar dacă sesiunea care a trimis mesajul este derulată înapoi (*rollback*).

Pachetul *DBMS_PIPE* este utilizat pentru a trimite mesaje în conductă (*DBMS_PIPE.SEND_MESSAGE*), mesaje ce constau din date de tip *VARCHAR2*, *NUMBER*, *DATE*, *RAW* sau *ROWID*. Tipurile obiect definite de utilizator şi colecţiile nu sunt acceptate de acest pachet.

De asemenea, pachetul poate realiza primirea de mesaje din conductă în *buffer*-ul local (*DBMS_PIPE.RECEIVE_MESSAGE*), accesarea următorului articol din *buffer* (*DBMS_PIPE.UNPACK_MESSAGE*), crearea unei noi conducte (*DBMS_PIPE*. *CREATE_PIPE*) etc.

DBMS_ALERT este similar pachetului *DBMS_PIPE*, fiind utilizat tot pentru comunicarea dintre sesiuni conectate la aceeaşi bază de date. Există totuşi câteva deosebiri esenţiale.

- *DBMS_ALERT* asigură o comunicare sincronă.
- Un mesaj trimis prin *DBMS_PIPE* este primit de către un singur destinatar (cititor), chiar dacă există mai mulţi pe conductă, pe când cel trimis prin *DBMS_ALERT* poate fi primit de mai mulţi cititori simultan.
- Dacă două mesaje sunt trimise printr-o conductă (înainte ca ele să fie citite), ambele vor fi primite de destinatar prin *DBMS_PIPE*. În cazul pachetului *DBMS_ALERT*, doar cel de al doilea mesaj va fi primit.

Pachete predefinite furnizate de *Oracle9i*

Oracle9i furnizează o varietate de pachete predefinite care simplifică administrarea bazei de date şi oferă noi funcţionalităţi legate de noile caracteristici ale sistemului. Dintre pachetele introduse în versiunea *Oracle9i* se remarcă:

- *DBMS_REDEFINITION* – permite reorganizarea *online* a tabelelor;
- *DBMS_LIBCACHE* – permite extragerea de comenzi *SQL* şi *PL/SQL* dintr-o instanţă distantă într-una una locală (vor fi compilate local, dar nu executate);
- *DBMS_LOGMNR_CDC_PUBLISH* – realizează captarea schimbărilor din tabelele bazei de date (identifică datele adăugate, modificate sau şterse şi editează aceste informaţii într-o formă utilizabilă în aplicaţii);
- *DBMS_LOGMNR_CDC_SUBSCRIBE* – face posibilă vizualizarea şi interogarea schimbărilor din datele care au fost captate cu pachetul *DBMS_LOGMNR_CDC_PUBLISH*;
- *DBMS_METADATA* – furnizează informaţii despre obiectele bazei de date;
- *DBMS_RESUMABLE* – permite setarea limitelor de spaţiu şi timp pentru

o operaţie specificată, operaţia fiind suspendată dacă sunt depăşite aceste limite;

- *DBMS_XMLQUERY*, *DBMS_XMLSAVE*, *DBMS_XMLGEN* – permit prelucrarea şi conversia datelor *XML* (*XMLGEN* converteşte rezultatul unei cereri *SQL* în format *XML, XMLQUERY* este similară lui *XMLGEN,* doar că este scrisă în *C,* iar *XMLSAVE* face conversia din format *XML* în date ale bazei);
- *DBMS_LDAP* – furnizează funcţii şi proceduri pentru accesarea datelor de pe *server*-e *LDAP* (*Lightweight Directory Access Protocol*);
- *UTL_ENCODE* – permite codificarea şi decodificarea într-un format standard a unei date de tip *RAW*, astfel încât datele să poată fi transferate între diferite gazde (*host*-uri);
- *UTL_INADDR* – returnează numele unei gazde locale sau distante a cărei adresă *IP* este cunoscută şi reciproc, returnează adresa *IP* a unei gazde căreia i se cunoaşte numele (de exemplu, *www.oracle.com*);
- *DBMS_AQELM* – furnizează proceduri şi funcţii pentru gestionarea configuraţiei cozilor de mesaje asincrone prin *e-mail* şi *HTTP*;
- *DBMS_FGA* – asigură întreţinerea unor funcţii de securitate;
- *DBMS_FLASHBACK* – permite trecerea la o versiune a bazei de date corespunzătoare unei unităţi de timp specificate sau unui *SCN* (*system change number*) dat, în felul acesta putând fi recuperate linii şterse sau mesaje *e-mail* distruse;
- *DBMS_TRANSFORM* – furnizează subprograme ce permit transformarea unui obiect (expresie *SQL* sau funcţie *PL/SQL*) de un anumit tip (sursă) într-un obiect având un tip (destinaţie) specificat;
- *DBMS_WM* – oferă funcţionalităţi legate de *Workspace Manager*. Gestiunea spaţiului de lucru se referă la abilitatea bazei de date de a păstra diferite versiuni ale aceleaşi linii în unul sau mai multe spaţii de lucru (virtuale), utilizatorii bazei putând modifica independent aceste versiuni. De exemplu, pachetul oferă posibilitatea de a modifica descrierea unui punct de salvare, de a şterge puncte de salvare, de a lista privilegiile pe care le are utilizatorul curent referitor la un anumit spaţiu de lucru, de a determina dacă există conflicte între spaţii de lucru etc.