

## Pachete în *PL/SQL*

Pachetul (*package*) permite încapsularea într-o unitate logică în baza de date a procedurilor, funcțiilor, cursorilor, tipurilor, constantelor, variabilelor și excepțiilor.

Pachetele sunt unități de program care sunt compilate, depanate și testate, sunt obiecte ale bazei de date care grupează tipuri, obiecte și subprograme *PL/SQL* având o legătură logică între ele.

De ce sunt importante?

Atunci când este referențiat un pachet (când este apelată pentru prima dată o construcție a pachetului), întregul pachet este încărcat în *SGA*, zona globală a sistemului, și este pregătit pentru execuție. Plasarea pachetului în *SGA* (zona globală sistem) reprezintă avantajul vitezei de execuție, deoarece *server*-ul nu mai trebuie să aducă informația despre pachet de pe disc, aceasta fiind deja în memorie. Prin urmare, apeluri ulterioare ale unor construcții din același pachet, nu solicită operații *I/O* de pe disc. De aceea, ori de câte ori apare cazul unor proceduri și funcții înrudite care trebuie să fie executate împreună, este convenabil ca acestea să fie grupate într-un pachet stocat. Este de subliniat că în memorie există o singură copie a unui pachet, pentru toți utilizatorii.

Spre deosebire de subprograme, pachetele nu pot:

- fi apelate,
- transmite parametri,
- fi încuibărite.

Un pachet are două părți, fiecare fiind stocată separat în dicționarul datelor.

- Specificarea pachetului (*package specification*) – partea „vizibilă”, adică interfața cu aplicații sau cu alte unități program. Se declară tipuri, constante, variabile, excepții, cursori și subprograme folosite de utilizatorul.
- Corpul pachetului (*package body*) – partea „acunsă”, mascată de restul aplicației, adică realizarea specificației. Corpul definește cursori și subprograme, implementând specificația. Obiectele conținute în corpul pachetului sunt fie private, fie publice.

Prin urmare, specificația definește interfața utilizatorului cu pachetul, iar corpul pachetului conține codul care implementează operațiile definite în specificație. Crearea unui pachet se face în două etape care presupun crearea specificației pachetului și crearea corpului pachetului.

Un pachet poate cuprinde, fie doar partea de specificație, fie specificația și corpul pachetului. Dacă conține doar specificația, atunci evident pachetul conține doar definiții de tipuri și declarații de date.

Corpul pachetului poate fi schimbat fără schimbarea specificației pachetului. Dacă specificația este schimbată, aceasta invalidează automat corpul pachetului, deoarece corpul depinde de specificație.

Specificația și corpul pachetului sunt unități compilate separat. Corpul poate fi compilat doar după ce specificația a fost compilată cu succes.

Un pachet are următoarea formă generală:

```
CREATE PACKAGE nume_pachet {IS / AS} -- specificația
    /* interfața utilizator, care conține: declarații de tipuri și obiecte
       publice, specificații de subprograme */
END [nume_pachet];
CREATE PACKAGE BODY nume_pachet {IS / AS} -- corpul
    /* implementarea, care conține: declarații de obiecte și tipuri private,
       corpuri de subprograme specificate în partea de interfață */
[BEGIN]
    /* instrucțiuni de inițializare, executate o singură dată când
       pachetul este invocat prima oară de către sesiunea utilizatorului */
END [nume_pachet];
```

## Specificația unui pachet

Specificația unui pachet cuprinde declararea procedurilor, funcțiilor, constantelor, variabilelor și excepțiilor care pot fi accesibile utilizatorilor, adică declararea obiectelor de tip *PUBLIC* din pachet. Acestea pot fi utilizate în proceduri sau comenzi care nu aparțin pachetului, dar care au privilegiul *EXECUTE* asupra acestuia.

Variabilele declarate în specificația unui pachet sunt globale pachetului și sesiunii. Ele sunt inițializate (implicit) prin valoarea *NULL*, evident dacă nu este specificată explicit o altă valoare.

```
CREATE [OR REPLACE] PACKAGE [schema.]nume_pachet  
[AUTHID {CURRENT_USER / DEFINER}]  
{IS / AS}  
specificație_PL/SQL;
```

*Specificație\_PL/SQL* poate include declarații de tipuri, variabile, cursoare, excepții, funcții, proceduri, pragma etc. În secțiunea declarativă, un obiect trebuie declarat înainte de a fi referit.

Opțiunea *OR REPLACE* este specificată dacă există deja corpul pachetului. Clauzele *IS* și *AS* sunt echivalente, dar dacă se folosește *PROCEDURE BUILDER* este necesară opțiunea *IS*.

Clauza *AUTHID* specifică faptul ca subprogramele pachetului se execută cu drepturile proprietarului (implicit) sau ale utilizatorului curent. De asemenea, această clauză precizează dacă referințele la obiecte sunt rezolvate în schema proprietarului subprogramului sau a utilizatorului curent.

## Corpul unui pachet

Corpul unui pachet conține codul *PL/SQL* pentru obiectele declarate în specificația acestuia și obiectele private pachetului. De asemenea, corpul poate include o secțiune declarativă în care sunt specificate definiții locale de tipuri, variabile, constante, proceduri și funcții locale. Obiectele private sunt vizibile numai în interiorul corpului pachetului și pot fi accesate numai de către funcțiile și procedurile din pachetul respectiv. Corpul pachetului este opțional și nu este necesar să fie creat dacă specificația pachetului nu conține declarații de proceduri sau funcții.

Este importantă ordinea în care subprogramele sunt definite în interiorul corpului pachetului. O variabilă trebuie declarată înainte ca să fie referită de altă variabilă sau subprogram, iar un subprogram privat trebuie declarat sau definit înainte de a fi apelat de alte subprograme.

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

Un pachet este instanțiat când este apelat prima dată. Aceasta presupune că pachetul este citit de pe disc în memorie și este executat codul compilat a subprogramului apelat. În acest moment, memoria este alocată tuturor variabilelor definite în pachet.

În multe cazuri este necesar să se facă o inițializare atunci când pachetul este instanțiat prima dată într-o sesiune. Aceasta se realizează prin adăugarea unei secțiuni de inițializare (opțională) în corpul pachetului secțiune încadrată între cuvintele cheie *BEGIN* și *END*. Secțiunea conține un cod de inițializare care este executat atunci când pachetul este invocat pentru prima dată.

Crearea pachetului face ca acesta să fie disponibil pentru utilizatorul care l-a creat sau orice cont de utilizator căruia i s-a acordat privilegiul *EXECUTE*.

Referința la o declarație sau la un obiect specificat în pachet se face prefixând numele obiectului cu numele pachetului. În corpul pachetului, obiectele din specificație pot fi referite fără a specifica numele pachetului.

Procesul de creare a specificației și corpului unui pachet urmează același algoritm ca cel întâlnit în crearea subprogramelor *PL/SQL* independente.

- sunt verificate erorile sintactice și semantice, iar modulul este depus în dicționarul datelor;
- sunt verificate instrucțiunile *SQL* individuale, adică dacă obiectele referite există și dacă utilizatorul le poate accesa;
- sunt comparate declarațiile de subprograme din specificația pachetului cu cele din corpul pachetului (dacă au același număr și tip de parametri). Orice eroare detectată la compilarea specificației sau a corpului pachetului este marcată în dicționarul datelor.

După ce specificația și corpul pachetului sunt compilate, ele devin obiecte în schema curentă. În vizualizarea *USER\_OBJECTS* din dicționarul datelor, vor fi două noi linii:

<u><b>OBJECT TYPE</b></u>	<u><b>OBJECT NAME</b></u>
<b>PACKAGE</b>	<b>nume_pachet</b>
<b>PACKAGE BODY</b>	<b>nume_pachet</b>

## Modificarea și suprimarea pachetelor

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

***ALTER PACKAGE [schema.]nume\_pachet COMPILE [PACKAGE / BODY]***

Schimbarea corpului pachetului nu cere recompilarea construcțiilor dependente, în timp ce schimbări în specificația pachetului solicită recompilarea fiecărui subprogram stocat care referențiază pachetul.

Dacă se dorește modificarea sursei, utilizatorul poate recrea pachetul (cu opțiunea *REPLACE*) pentru a-l înlocui pe cel existent.

***DROP PACKAGE [schema.]nume\_pachet [PACKAGE / BODY]***

Dacă în cadrul comenzii apare opțiunea *BODY* este distrus doar corpul pachetului, în caz contrar sunt distruse atât specificația, cât și corpul pachetului. Dacă pachetul este distrus, 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 specificația pachetului, până când nu este recreat corpul pachetului.

Pentru ca un utilizator să poată distruge un pachet trebuie ca fie pachetul să aparțină schemei utilizatorului, fie utilizatorul să posede privilegiul de sistem *DROP ANY PROCEDURE*.

Una din posibilitățile interesante oferite de pachetele *PL/SQL* este aceea de a crea proceduri/funcții *overload*. Procesul implică definirea unui număr de proceduri cu același nume, dar care diferă prin numărul și tipul parametrilor pe care le folosesc în fiecare instanță a procedurii implementată separat în corpul pachetului. Acest tip de programare este folositor când este necesară o singură funcție care să execute aceeași operație pe obiecte de tipuri diferite (diferite tipuri de parametri de intrare). Când este apelată o procedură *overload* sistemul decide pe baza tipului și numărului de parametri care instanță a procedurii va fi executată. Numai subprogramele locale sau aparținând unui pachet pot fi *overload*. Subprogramele *stand-alone* nu pot fi *overload*.

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

1) În *PL/SQL* se face prin referirea:

***nume\_pachet.nume\_componentă [(listă\_de\_argumente)];***

2) În *SQL\*Plus* 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 combinație specificată dintre atributele *cod\_artist* și *stil* este o combinație care există în tabelul *opera*.

```

CREATE PACKAGE verif_pachet IS
    PROCEDURE verifica
        (p_idartist IN opera.cod_artist%TYPE,
         p_stil      IN opera.stil%TYPE);
END verif_pachet;
/
CREATE OR REPLACE PACKAGE BODY verif_pachet IS
    i NUMBER := 0;
    CURSOR opera_cu IS
        SELECT cod_artist, stil
        FROM    opera;
    TYPE opera_table_tip IS TABLE OF opera_cu%ROWTYPE
        INDEX BY BINARY INTEGER;
    art_stil opera_table_tip;
    PROCEDURE verifica
        (p_idartist IN opera.cod_artist%TYPE,
         p_stil      IN opera.stil%TYPE);
    IS
    BEGIN
        FOR k IN art_stil.FIRST..art_stil.LAST LOOP
            IF p_idartist = art_stil(k).cod_artist
                AND p_stil = art_stil(k).stil THEN
                RETURN;
            END IF;
        END LOOP;
        RAISE_APPLICATION_ERROR (-20777, 'nu este buna
                                         combinatia');

        END verifica;
    BEGIN
        FOR ope_in IN opera_cu LOOP
            art_stil(i) := ope_in;
            i := i+1;
        END LOOP;
    END verif_pachet;
/

```

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

```
verif_pachet.verifica (7935, 'impresionism');
```

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

```
EXECUTE verif_pachet.verifica (7935, 'impresionism')
```

**Observații:**

- Un declanșator nu poate apela o procedură sau o funcție ce conține comenzile *COMMIT*, *ROLLBACK*, *SAVEPOINT*. Prin urmare, pentru flexibilitatea apelului (de către declanșatori) subprogramelor conținute în pachete, trebuie verificat că nici una din procedurile sau funcțiile pachetului nu conțin aceste comenzi.
- Procedurile și funcțiile conținute într-un pachet pot fi referite din fișiere *iSQL\*Plus*, din subprograme stocate *PL/SQL*, din aplicații client (de exemplu, *Oracle Forms* sau *Power Builder*), din declanșatori (bază de date), din programe aplicație scrise în limbaje de generația a 3-a.
- Într-un pachet nu pot fi referite variabile gazdă.
- Într-un pachet, mai exact în corpul acestuia, sunt permise declarații *forward*.
- Funcțiile unui pachet pot fi utilizate (cu restricții) în comenzi *SQL*.

Dacă un subprogram dintr-un pachet este apelat de un subprogram *stand-alone* trebuie remarcat că:

- dacă corpul pachetului se schimbă, dar specificația pachetului nu se schimbă, atunci subprogramul care referă o construcție a pachetului rămâne valid;
- dacă specificația pachetului se schimbă, atunci subprogramul care referă o construcție a pachetului, precum și corpul pachetului sunt invalidate.

Dacă un subprogram *stand-alone* referit de un pachet se schimbă, atunci întregul corp al pachetului este invalidat, dar specificația pachetului rămâne validă.

**Pachete predefinite**

*PL/SQL* conține pachete predefinite utilizabile pentru dezvoltare de aplicații și care sunt deja compilate în baza de date. Aceste pachete adaugă noi funcționalități limbajului, protocoale de comunicație, acces la fișierele sistemului etc. Apelarea unor proceduri din aceste pachete solicită prefixarea numelui procedurii cu numele pachetului.

Dintre cele mai importante pachete predefinite se remarcă:

- *DBMS\_OUTPUT* (permite afișarea de informații);
- *DBMS\_DDL* (furnizează accesul la anumite comenzi *DDL* 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 *HTTP* pentru accesarea din *PL/SQL* a datelor de pe *Internet*);
- *UTL\_TCP* (permite aplicaţiilor *PL/SQL* să comunice cu *server-e* externe utilizând protocolul *TCP/IP*);
- *DBMS\_JOB* (permite planificarea programelor *PL/SQL* pentru execuţie şi execuţia acestora);
- *DBMS\_SQL* (accesează baza de date folosind *SQL* dinamic);
- *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 clişeeleor);
- *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 sunt utilizabile automat în 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 ori de câte ori are nevoie în aplicaţia sa funcţia *ABS* (*x*), aparţinând pachetului *DBMS\_STANDARD*, care reprezintă valoarea absolută a numărului *x*, fără a prefixa numele funcţiei cu numele pachetului.

### **Pachetul *DBMS\_OUTPUT***

*DBMS\_OUTPUT* permite afişarea de informaţii atunci când se execută un program *PL/SQL* (trimite mesajele din orice bloc *PL/SQL* într-un buffer în *BD*).

*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) informație în *buffer*, informație care este de tipul *NUMBER*, *VARCHAR2* sau *DATE*. *PUT\_LINE* are același efect ca procedura *PUT*, dar inserează și un sfârșit de linie. Procedurile *PUT* și *PUT\_LINE* sunt *overload*, astfel încât informația poate fi scrisă în format nativ (*VARCHAR2*, *NUMBER* sau *DATE*).

Procedura *GET\_LINE* regăsește o singură linie de informație (de dimensiune maximă 255) din *buffer* (dar sub formă de șir de caractere). Procedura *GET\_LINES* regăsește mai multe linii (*nr\_linii*) din *buffer* și le depune într-un tablou (*nume\_tab*) *PL/SQL* având tipul șir de caractere. Valorile sunt plasate în tabel începând cu linia zero. Specificația 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 și parametru de tip *OUT*, deoarece numărul liniilor solicitate poate să nu coincidă cu numărul de linii din *buffer*. De exemplu, pot fi solicitate 10 linii, iar în *buffer* sunt doar 6 linii. Atunci doar primele 6 linii din tabel sunt definite.

Dezactivarea referirilor la pachet se poate realiza prin procedura *DISABLE*, iar activarea referirilor se face cu ajutorul procedurii *ENABLE*.

### **Exemplu:**

Următorul exemplu plasează în *buffer* (apelând de trei ori procedura *PUT*) toate informațiile într-o singură linie.

```
DBMS_OUTPUT.PUT(:opera.valoare||:opera.cod_artist);
DBMS_OUTPUT.PUT(:opera.cod_opera);
DBMS_OUTPUT.PUT(:opera.cod_galerie);
```

Dacă aceste trei comenzi sunt urmate de comanda

```
DBMS_OUTPUT.NEW_LINE;
```

atunci informația respectivă va fi găsită printr-un singur apel *GET\_LINE*. Altfel, nu se va vedea nici un efect al acestor comenzi deoarece *PUT* plasează informația în *buffer*, dar nu adaugă sfârșit de linie.

Când este utilizat pachetul *DBMS\_OUTPUT* pot să apară erorile *buffer overflow* și *line length overflow*. Tratarea acestor erori se face apelând procedura *RAISE\_APPLICATION\_ERROR* din pachetul standard *DBMS\_STANDARD*.

## Pachetul *DBMS\_SQL*

Pachetul *DBMS\_SQL* permite folosirea dinamică a comenzilor *SQL* în proceduri stocate sau în blocuri anonime și analiza gramaticală a comenzilor *LDD*. Aceste comenzi nu sunt incorporate în programul sursă, ci sunt depuse în șiruri de caractere. O comandă *SQL* dinamică este o instrucțiune *SQL* care conține variabile ce se pot schimba în timpul execuției. De exemplu, pot fi utilizate instrucțiuni *SQL* dinamice pentru:

- a crea o procedură care operează asupra unui tabel al cărui nume nu este cunoscut decât în momentul execuției;
- a scrie și executa o comandă *LDD*;
- a scrie și 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 folosești comanda *DROP TABLE*. Evident, folosirea acestui pachet pentru a executa comenzi *LDD* poate genera interblocări. De exemplu, pachetul este utilizat pentru a șterge o procedură care însă este utilizată.

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

Orice comandă *SQL* trebuie să treacă prin niște etape, cu observația că anumite etape pot fi evitate. Etapele presupun: analizarea gramaticală a comenzii, adică verificarea sintactică a comenzii, validarea acesteia, asigurarea că toate referințele la obiecte sunt corecte și asigurarea că există privilegiile referitoare la acele obiecte (*parse*); obținerea de valori pentru variabilele de legătură din comanda *SQL* (*binding variables*); executarea comenzii (*execute*); selectarea rândurilor rezultatului și încărcarea acestor rânduri (*fetch*).

Dintre subprogramele pachetului *DBMS\_SQL*, care permit implementarea etapelor amintite anterior se remarcă:

- *OPEN\_CURSOR* (deschide un nou cursor, adică se stabilește o zonă de memorie în care este procesată comanda *SQL*);
- *PARSE* (stabilește validitatea comenzii *SQL*, adică se verifică sintaxa instrucțiunii și se asociază cursorului deschis);
- *BIND\_VARIABLE* (leaga valoarea data de variabila corespunzătoare din comanda *SQL* analizată)
- *EXECUTE* (execută comanda *SQL* și returnează numărul de linii procesate);

- *FETCH\_ROWS* (regăsește o linie pentru un cursor specificat, iar pentru mai multe linii folosește un *LOOP*);
- *CLOSE\_CURSOR* (î închide cursorul specificat).

Să se construiască o procedură care folosește *SQL* dinamic pentru a șterge liniile unui tabel specificat (*num\_tab*). Subprogramul furnizează ca rezultat numărul liniilor șterse (*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;
```

Argumentul *DBMS\_SQL.V7* reprezintă modul (versiunea 7) în care *Oracle* tratează comenzile *SQL*. Ștergerea efectivă a liniilor tabelului *opera* se realizează:

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

Pentru a executa o instrucțiune *SQL* dinamic poate fi utilizată și comanda *EXECUTE IMMEDIATE*. Comanda conține o clauză opțională *INTO* care este utilizabilă pentru interogări ce returnează o singură linie. Pentru o cerere care returnează mai multe linii trebuie folosite comenzile *OPEN FOR*, *FETCH*, *CLOSE*.

```
CREATE OR REPLACE PROCEDURE sterge_linii
  (num_tab IN VARCHAR2, nr_lin OUT NUMBER)
IS
BEGIN
  EXECUTE IMMEDIATE 'DELETE FROM' || num_tab;
  nr_lin := SQL%ROWCOUNT;
END;
```

Procedura se poate apela printr-o secvență identică cu cea prezentată anterior.

### **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 programului modificat (procedură, funcție, declanșator, pachet, corp pachet).

***ALTER\_COMPILE*** (*tip\_obiect*, *nume\_schema*, *nume\_obiect*);

Procedura este echivalentă cu instrucțiunea *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 obține numărul liniilor unui tabel, numărul de blocuri ale unui tabel, lungimea medie a unei linii, numărul valorilor distincte ale unei col., numărul elementelor *null* dintr-o coloană, distribuția datelor (histograma) etc.

***ANALYZE\_OBJECT*** (*tip\_obiect*, *nume\_schema*, *nume\_obiect*, *metoda*, *număr\_linii\_estimate*, *procent*, *opțiune\_metoda*, *nume\_partiție*);

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

Procedura este echivalentă cu instrucțiunea:

***ANALYZE TABLE / CLUSTER / INDEX*** [*nume\_schema*] *nume\_obiect* [*metoda*] *STATISTICS* [*SAMPLE n*] [*ROWS / PERCENT*]

Dacă *nume\_schema* este *null*, atunci se presupune că este vorba de schema curentă. Dacă *tip\_obiect* este diferit de *table*, *index* sau *cluster*, se declanșează eroarea *ORA - 20001*. Parametrul *procent* reprezintă procentajul liniilor de estimat și este ignorat dacă este specificat numărul liniilor de estimat (*număr\_linii\_estimate*). Implicit, ultimele patru argumente ale procedurii iau valoarea *null*.

Argumentul *opțiune\_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 opțiuni.

### ***Exemplu:***

Utilizând pachetul *DBMS\_DDL* și metoda *COMPUTE*, să se creeze o procedură care analizează un obiect furnizat ca parametru acestei proceduri.

```

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

```

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

```

EXECUTE analiza ('TABLE', 'opera')
SELECT LAST_ANALYZED
FROM   USER_TABLES
WHERE  TABLE_NAME = 'opera';

```

### Pachetul *DBMS\_JOB*

Pachetul *DBMS\_JOB* este utilizat pentru planificarea programelor *PL/SQL* în vederea execuției. Cu ajutorul acestui pachet se pot executa programe *PL/SQL* la momente determinate de timp, se pot șterge sau suspenda programe din lista de planificări pentru execuție, se pot rula programe de întreținere în timpul perioadelor de utilizare scăzută etc.

Dintre subprogramele acestui pachet se remarcă:

- *SUBMIT* – adaugă un nou *job* în coada de așteptare a *job*-urilor;
- *REMOVE* – șterge un *job* specificat din coada de așteptare a *job*-urilor;
- *RUN* – execută imediat un *job* specificat;
- *BROKEN* – dezactivează execuția 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* – furnizează descrierea unui *job* specificat;
- *NEXT\_DATE* – dă momentul următoarei execuții a unui *job*;
- *INTERVAL* – furnizează intervalul între diferite execuții 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 execuții a *job*-ului (implicit este *SYSDATE*);
- *INTERVAL* – de tip *IN*, funcție care furnizează intervalul dintre execuțiile *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 execuție în coada de așteptare a *job*-urilor, procedura *verifica* din pachetul *verif\_pachet*.

```
VARIABLE num_job NUMBER
BEGIN
    DBMS_JOB.SUBMIT(
        job => :num_job,
        what                                     =>
'verif_pachet.verifica(8973, 'impresionism')';
        next_date => TRUNC(SYSDATE+1),
        interval => 'TRUNC(SYSDATE+1)';
    COMMIT;
END;
/
PRINT num_job
```

Vizualizarea *DBA\_JOBS* din dicționarul datelor furnizează informații referitoare la starea *job*-urilor din coada de așteptare, iar vizualizarea *DBA\_JOBS\_RUNNING* conține informații despre *job*-urile care sunt în curs de execuție. 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.

Folosind componentele acestui pachet (funcţiile *FOPEN* şi *IS\_OPEN*; procedurile *GET\_LINE*, *PUT*, *PUT\_LINE*, *PUTF*, *NEW\_LINE*, *FCLOSE*, *FCLOSEALL*, *FFLUSH*) se pot deschide fişiere, obține text din fişiere, scrie text în fişiere, închide fişiere.

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* sau *FCLOSEALL*).

Funcţia *IS\_OPEN* verifică dacă un fişier este deschis. Are antetul:

```
FUNCTION IS_OPEN (handler_fisier IN FILE_TYPE)
RETURN BOOLEAN;
```

Funcţia *FOPEN* deschide un fişier şi returnează un *handler* care va fi utilizat în următoarele operaţii *I/O*. Parametrul *open\_mode* este un string care specifică modul cum a fost deschis fişierul.

```
FUNCTION FOPEN (locatia      IN VARCHAR2,
                nume_fisier  IN VARCHAR2,
                open_mode    IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

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 excepţii, dintre care remarcăm:

- *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* – operație invalidă asupra fișierului;
- *READ\_ERROR* – o eroare a sistemului de operare a apărut în timpul operației de citire;
- *WRITE\_ERROR* – o eroare a sistemului de operare a apărut în timpul operației de scriere;
- *INTERNAL\_ERROR* – o eroare nespecificată a apărut în *PL/SQL*.

Excepțiile trebuie prefixate cu numele pachetului. Evident, pot apărea și erorile *NO\_DATA\_FOUND* și *VALUE\_ERROR*.

### **Pachetele *DBMS\_PIPE* și *DBMS\_ALERT***

Pachetul *DBMS\_PIPE* permite operații de comunicare între două sau mai multe sesiuni conectate la aceeași bază de date. De exemplu, pachetul poate fi utilizat pentru comunicarea dintre o procedură stocată și un program *Pro\*C*. Comunicarea se face prin conducte (*pipe*). O conductă este o zonă de memorie utilizată de un proces pentru a transmite informație altui proces. Informația trimisă prin conductă este depusă într-un *buffer* din *SGA*. Toate informațiile din conductă sunt pierdute atunci când instanța este închisă.

Conductele sunt asincrone, ele operând independent de tranzacții. 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 și colecțiile 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 aceeași bază de date. Există totuși câteva deosebiri esențiale.

- *DBMS\_ALERT* asigură o comunicare sincronă.
- Un mesaj trimis prin *DBMS\_PIPE* este primit de un singur destinatar (cititor) chiar dacă există mai mulți pe conductă, pe când cel trimis prin *DBMS\_ALERT* poate fi primit de mai mulți 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 2-lea mesaj va fi primit.



## ***Pachete predefinite furnizate de Oracle9i***

*Oracle9i* furnizează o varietate de pachete predefinite care simplifică administrarea bazei de date și oferă noi funcționalități 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* și *PL/SQL* dintr-o instanță distantă într-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 șterse și editează aceste informații într-o formă utilizabilă în aplicații);
- *DBMS\_LOGMNR\_CDC\_SUBSCRIBE* – face posibilă vizualizarea și interogarea schimbărilor din datele care au fost captate cu pachetul *DBMS\_LOGMNR\_CDC\_PUBLISH*;
- *DBMS\_METADATA* – furnizează informații despre obiectele bazei de date;
- *DBMS\_RESUMABLE* – permite setarea limitelor de spațiu și timp pentru o operație specificată, operația fiind suspendată dacă sunt depășite aceste limite;
- *DBMS\_XMLQUERY*, *DBMS\_XMLSAVE*, *DBMS\_XMLGEN* – permit prelucrarea și conversia datelor *XML* (*XMLGEN* convertește 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);
- *UTL\_INADDR* – returnează numele unei gazde locale sau distante a cărei adresă *IP* este cunoscută și reciproc, returnează adresa *IP* a unei gazde căreia i se cunoaște numele (de exemplu, *www.oracle.com*);
- *DBMS\_AQELM* – furnizează proceduri și funcții pentru gestionarea configurației cozilor de mesaje asincrone prin *e-mail* și *HTTP*;
- *DBMS\_FGA* – asigură întreținerea unor funcții de securitate;
- *DBMS\_FLASHBACK* – permite trecerea la o versiune a bazei de date corespunzătoare unei unități de timp specificate sau unui *SCN* (*system change number*) dat, în felul acesta putând fi recuperate linii șterse sau mesaje *e-mail* distruse;
- *DBMS\_TRANSFORM* – furnizează subprograme ce permit transformarea unui obiect (expresie *SQL* sau funcție *PL/SQL*) de un anumit tip (sursă) într-un obiect având un tip (destinație) specificat;