

CUPRINS

7. <i>PL/SQL</i> – Pachete.....	2
7.1. Definirea pachetelor	3
7.1.1. Specificația pachetului	6
7.1.2. Corpul pachetului	6
7.1.3. Instanțierea pachetului	7
7.2. Modificare,ștergerea și modificarea pachetelor.....	8
7.3. Pachete predefinite	11
7.3.1. Pachetul <i>STANDARD</i>	12
7.3.2. Pachetul <i>DBMS_OUTPUT</i>	13
7.3.3. Pachetul <i>DBMS_JOB</i>	13
7.3.4. Pachetul <i>UTL_FILE</i>	15
7.3.5. Pachetul <i>DBMS_SQL</i>	16
7.3.6. Pachetul <i>DBMS_DDL</i>	19
Bibliografie	22

7. PL/SQL – Pachete

- Un pachet (*package*) permite încapsularea într-o unitate logică a:
 - constantelor și variabilelor;
 - tipurilor și excepțiilor;
 - cursorilor;
 - procedurilor și funcțiilor.
- Pachetele sunt:
 - unități de program compilate;
 - obiecte ale bazei de date care grupează tipuri, obiecte și subprograme *PL/SQL* având o legătură logică între ele.



Ce fel de subprograme integrăm într-un pachet?

Importanța pachetelor

- Atunci când este referențiat un pachet (atunci când este apelată pentru prima dată o construcție a pachetului), întregul pachet este încărcat în *SGA* (zona globală sistem) și este pregătit pentru execuție.
- Plasarea pachetului în *SGA* reprezintă avantajul vitezei de execuție, deoarece *server*-ul nu mai trebuie să aducă informația despre pachet de pe disc, aceasta fiind deja în memorie.
 - Apelurile 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.
 - În memorie există o singură copie a unui pachet (pentru toți utilizatorii).
- Subprogramelor *overload* pot deveni stocate prin intermediul pachetelor.

Diferența față de subprograme

- Spre deosebire de subprograme, pachetele nu pot:
 - fi apelate;
 - transmite parametri;
 - fi imbricate.

7.1. Definirea pachetelor

- Un pachet conține două părți, fiecare fiind stocată separat în dicționarul datelor:
 - specificația (*package specification*)
 - este partea „vizibilă” a pachetului, adică interfața cu aplicațiile sau cu alte unități program;
 - poate conține declarații de tipuri, constante, variabile, excepții, cursoare și subprograme (care vor fi vizibile altor aplicații);
 - se declară prima (înaintea corpului pachetului).
 - corpul (*package body*)
 - este partea „ascunsă” a pachetului, mascată de restul aplicației;
 - conține codul care implementează obiectele definite în specificație (cursoarele și subprogramele) și, de asemenea, obiecte și declarații proprii;
 - poate conține obiecte publice (declarate în specificație) sau private (nu sunt declarate în specificație).

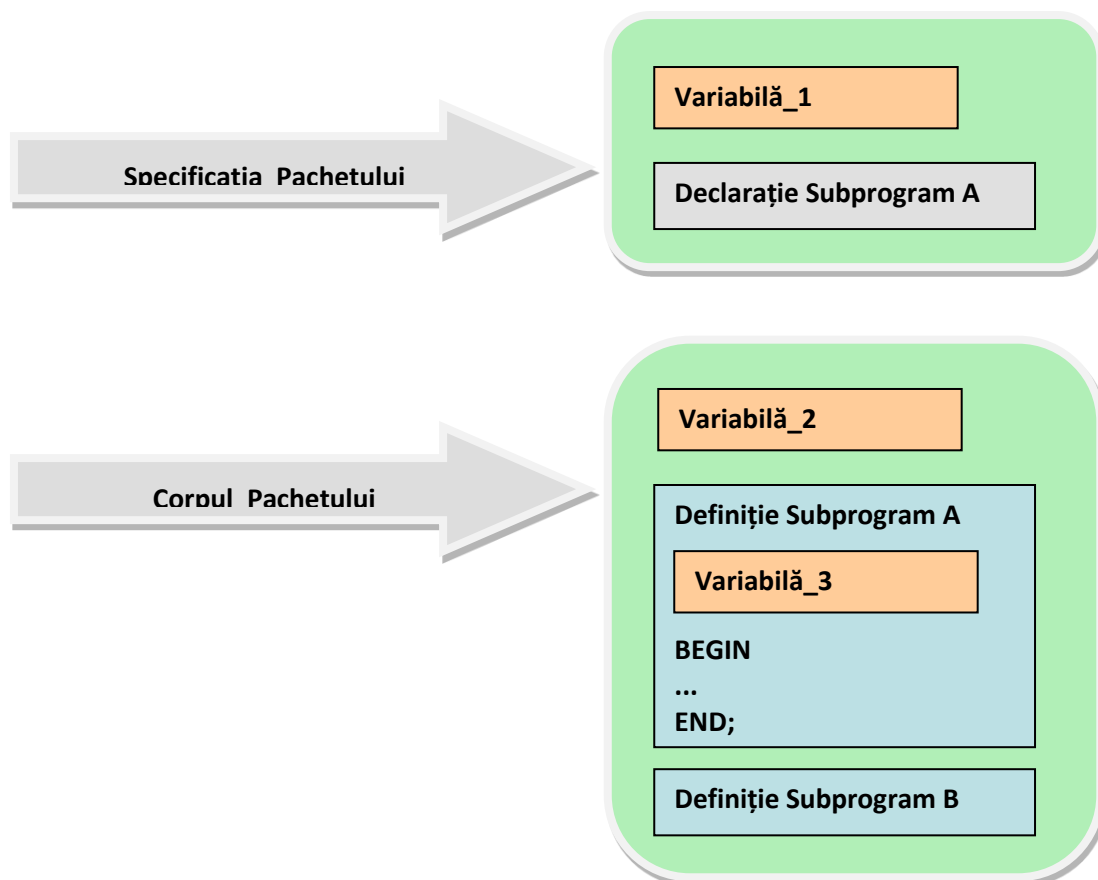


Fig. 7.1. Pachete *PL/SQL*

- Definirea unui pachet se realizează în două etape:
 - crearea specificației pachetului;
 - crearea corpului pachetului.



- ❖ Se pot defini pachete care cuprind doar partea de specificație?



- ❖ Se pot defini pachete care cuprind doar corpul pachetului?

- ❖ Ce situații impun definirea atât a specificației, cât și a corpului pachetului?



- ❖ Corpul pachetului poate fi schimbat fără a modifica specificația acestuia?



- ❖ 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.

- Pentru a crea un pachet în schema personală este necesar privilegiul sistem *CREATE PROCEDURE*, iar pentru a crea un pachet în altă schemă este necesar privilegiul sistem *CREATE ANY PROCEDURE*.
- Un pachet definit este disponibil pentru utilizatorul care l-a creat sau orice utilizator căruia i s-a acordat privilegiul *EXECUTE* asupra pachetului respectiv.

- Sintaxa:

```

CREATE PACKAGE nume_pachet
{IS | AS} -- specificația pachetului
    -- declarații de tipuri și obiecte publice,
    -- specificații de cursoare și subprograme
END [nume_pachet];
/

CREATE PACKAGE BODY nume_pachet
{IS | AS} -- corpul pachetului
    /* declarații de obiecte și tipuri private,
    corpurile cursoarelor și subprogramelor precizate
    în specificație */
[BEGIN
    /* instrucțiuni de inițializare, executate o singură
    dată, atunci când pachetul este invocat prima
    dată în sesiunea utilizatorului */
]
END [nume_pachet];
/

```

- 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, acestea devin obiecte în schema curentă.
 - În vizualizarea *USER_OBJECTS* din dicționarul datelor, vor apărea două noi linii:

OBJECT_TYPE	OBJECT_NAME
PACKAGE	nume_pachet
PACKAGE BODY	nume_pachet

7.1.1. Specificația pachetului

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



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

- Sintaxa:

```
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 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.

7.1.2. Corpul pachetului

- 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.



- ❖ Ordinea în care subprogramele sunt definite în interiorul corpului pachetului este importantă.
- ❖ 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.

- Sintaxa:

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

7.1.3. Instanțierea pachetului

- Un pachet este instanțiat atunci când este apelat prima dată.
 - Pachetul este citit de pe disc, depus în memorie și este executat codul compilat al subprogramului apelat.
 - În acest moment, memoria este alocată tuturor variabilelor definite în pachet.
- În multe cazuri, atunci când pachetul este instanțiat prima dată într-o sesiune, sunt necesare anumite inițializări.
 - 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.
- 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 sunt referite fără a specifica numele pachetului.

7.2. Modificarea, ștergerea și utilizarea pachetelor

Modificarea pachetului

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



- ❖ Schimbarea corpului pachetului nu impune recompilarea construcțiilor dependente.
 - ❖ Schimbările în specificația pachetului necesită recompilarea fiecărui subprogram stocat care referențiază pachetul.
- Folosind comanda *ALTER PACKAGE* se pot recompila explicit specificația pachetului, corpul pachetului sau ambele module.
 - Recompilarea explicită elimină necesitatea recompilării implicite a pachetului la *run-time*.
 - Deoarece într-un pachet toate obiectele sunt stocate ca o unitate, comanda *ALTER PACKAGE* determină recompilarea tuturor obiectelor pachetului.
 - Dacă în timpul recompilării apar erori, atunci este întors un mesaj, iar pachetul devine invalid.
 - Pentru a consulta mesajele erorilor se poate utiliza comanda *SQL*Plus SHOW ERRORS*.



Funcțiile și procedurile dintr-un pachet nu pot fi recompilate individual folosind comenzile *ALTER FUNCTION* sau *ALTER PROCEDURE*.

- Sintaxa:

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

- Clauza *PACKAGE* determină recompilarea atât a specificației, cât și a corpului pachetului.
 - Este opțiune implicită.
- Clauza *SPECIFICATION* determină recompilarea specificației pachetului.
 - După modificarea specificației unui pachet se poate dori recompilarea acesteia, pentru a verifica dacă apar erori de compilare.

- Atunci când se recompilează specificația pachetului, baza de date invalidează orice obiect local care depinde de acea specificație (ca de exemplu, proceduri care invocă proceduri sau funcții din pachet).
- Corpul pachetului depinde de asemenea de specificație. În lipsa unei recompilări explicite, baza de date va recompila implicit obiectele dependente.
- Clauza *BODY* determină doar recompilarea corpului pachetului.
 - Recompilarea corpului pachetului nu invalidează obiectele care depind de specificație.

Ștergerea pachetului

- Sintaxa:

```
DROP PACKAGE [BODY] [schema.]nume_pachet;
```

- Dacă în cadrul comenzii apare opțiunea *BODY*, atunci este șters doar corpul pachetului.
- Dacă se omite opțiunea *BODY*, atunci sunt șterse atât specificația, cât și corpul pachetului.
- Dacă pachetul este șters, toate obiectele dependente de acesta devin invalide. Dacă este șters 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 șterge un pachet trebuie ca pachetul să aparțină schemei utilizatorului sau utilizatorul să posede privilegiul sistem *DROP ANY PROCEDURE*.

Utilizarea pachetului

- Se realizează în funcție de mediul (*SQL* sau *PL/SQL*) care solicită un obiect din pachetul respectiv.
 - Referirea unui obiect din pachet se realizează prefixând numele acestuia cu numele pachetului.
 - De exemplu, invocarea unei proceduri definite într-un pachet se realizează în:
 - *PLSQ/SQL*, prin comanda
nume_pachet.nume_procedură[(listă_argumente)];
 - *SQL*Plus*, prin comanda:
EXECUTE nume_pachet.nume_procedură [(listă_argumente)]

Exemplul 7.1 - vezi curs

Exemplul 7.2 - vezi curs

Comenzile *LCD* - *COMMIT*, *ROLLBACK*, *SAVEPOINT*



- ❖ Un *trigger* nu poate apela un subprogram care conține comenzile *COMMIT*, *ROLLBACK*, *SAVEPOINT*. Prin urmare, pentru flexibilitatea apelului de către *trigger*-i a subprogramelor conținute în pachete, niciun subprogram al pachetului nu trebuie să conțină aceste comenzi.
- ❖ Într-un pachet nu pot fi referite variabile gazdă.
- ❖ Într-un pachet sunt permise declarații *forward*.

Invalidarea modulelor *PL/SQL*



- ❖ Dacă un subprogram independent apelează un subprogram definit într-un pachet, atunci apar următoarele situații:
 - atunci când corpul pachetului este modificat, dar specificația acestuia nu, subprogramul care referă o construcție a pachetului rămâne valid;
 - dacă specificația pachetului este modificată, atunci subprogramul care referă o construcție a pachetului, precum și corpul pachetului devin invalide.
- ❖ Dacă un subprogram independent referit de un pachet se modifică, atunci întregul corp al pachetului devine invalid, dar specificația pachetului rămâne validă.

Exemplul 7.3 - vezi curs

Exemplul 7.4 - vezi curs

Exemplul 7.5 - vezi curs

Exemplul 7.6 - vezi curs



- ❖ Un cursor declarat în specificația unui pachet este un tip de variabilă globală și respectă aceleași reguli privind persistența ca și celelalte variabile.
- ❖ Statusul unui cursor nu este definit de o singură valoare (ca în cazul variabilelor), ci din următoarele atribute:
 - *%ISOPEN* (dacă este deschis sau închis);
 - *%ROWCOUNT* (dacă este deschis, numărul de linii încărcate);
 - *%FOUND* sau *%NOTFOUND* (dacă ultimul *FETCH* a avut succes).

7.3. Pachete predefinite

- *PL/SQL* conține o serie de pachete predefinite utile în dezvoltarea aplicațiilor.
- Pachetele predefinite adaugă noi funcționalități limbajului, protocoale de comunicație, acces la fișierele sistemului etc.
- Exemple de pachete predefinite:
 - *STANDARD*
 - definește mediul *PL/SQL*
 - conține funcțiile predefinite
 - *DBMS_STANDARD*
 - facilități ale limbajului utile pentru interacțiunea aplicației cu *server-ul Oracle*
 - *DBMS_OUTPUT*
 - permite afișarea de informații
 - *DBMS_DDL*
 - permite accesarea anumitor comenzi *DDL* din *PL/SQL*; în plus, oferă operații speciale de administrare
 - *DBMS_JOB*
 - permite planificarea și gestiunea *job*-urilor
 - *DBMS_SQL*
 - oferă o interfață pentru a putea utiliza *SQL* dinamic
 - *DBMS_PIPE*
 - permite operații de comunicare între două sau mai multe sesiuni conectate la aceeași instanță *Oracle*
 - *DBMS_LOCK*
 - este utilizat pentru a cere, a modifica sau a elibera blocările din baza de date
 - permite folosirea exclusivă sau partajată a unei resurse
 - *DBMS_MVIEW* / *DBMS_SNAPSHOT*
 - permite exploatarea vizualizărilor materializate
 - *DBMS_UTILITY*
 - oferă utilități *DBA*, permite analiza obiectelor unei scheme, verifică dacă *server-ul* lucrează în mod paralel etc.
 - *DBMS_LOB*
 - oferă mecanisme de acces și prelucrare a datelor de tip *LOB*
 - permite compararea datelor de tip *LOB*, adăugarea de date la un *LOB*, copierea datelor dintr-un *LOB* în altul, ștergerea unor porțiuni din date *LOB*, deschiderea, închiderea și regăsirea de informații din date *BFILE* etc.

- *UTL_FILE*
 - permite citirea/scrierea din/în fişierele text ale sistemului de operare
- *UTL_MAIL*
 - permite crearea şi trimiterea unui *e-mail*
- *UTL_HTTP*
 - permite utilizarea protocolului *HTTP* pentru a accesa date de pe *Internet*
- *UTL_TCP*
 - permite aplicaţiilor *PL/SQL* să comunice cu *server-e* externe utilizând protocolul *TCP/IP*

7.3.1. Pachetul *STANDARD*

- Este un pachet predefinit fundamental în care se declară tipurile, excepţiile, subprogramele care sunt utilizabile automat în programele *PL/SQL*.
- Conţine funcţiile predefinite (de exemplu, *UPPER*, *ABS*, *TO_CHAR* etc.).
- Conţinutul acestui pachet este vizibil tuturor aplicaţiilor.
- Pentru referirea componentelor acestui pachet nu trebuie utilizată prefixarea cu numele pachetului.

Exemplul 7.7 – vezi explicaţii curs

```
SELECT STANDARD.ABS (-1) , ABS (-1)
FROM DUAL;
```



- ❖ Se pot defini funcţii cu acelaşi nume ca şi cele predefinite?
- ❖ Funcţia din exemplul 7.8 se poate defini?

Exemplul 7.8 – vezi explicaţii curs

```
CREATE OR REPLACE FUNCTION ABS(x NUMBER)
RETURN VARCHAR2
IS
BEGIN
  IF X<0 THEN RETURN 'Rezultatul intors este: '||-1*X;
  ELSE RETURN 'Rezultatul intors este: '||X;
  END IF;
END;
/

SELECT ABS (-1)
FROM DUAL;
```

7.3.2. Pachetul *DBMS_OUTPUT*

- Trimite mesaje text dintr-un bloc *PL/SQL* într-o zonă privată de memorie, din care mesajele pot fi afișate pe ecran.
- Pachetul este utilizat preponderent:
 - în timpul testării și depanării programelor;
 - pentru a afișa mesaje și rapoarte în *SQL*DBA* sau *SQL*Plus*;
 - pentru a urmări pașii de execuție a unui program.
- Subprograme definite în pachet:
 - *PUT* – depune text în *buffer* (pe o singură linie);
 - *NEW_LINE* – trimite conținutul *buffer*-ului pe ecran (adaugă în *buffer* un sfârșit de linie);
 - *PUT_LINE* – depune text în *buffer* (*PUT*) și trimite conținutul *buffer*-ului pe ecran (*NEW_LINE*);
 - *GET_LINE* – citește din *buffer* o singură linie;
 - *GET_LINES* – citește din *buffer* mai multe linii (le depune într-o colecție);
 - *DISABLE* – dezactivează apelurile procedurilor *PUT*, *NEW_LINE*, *PUT_LINE*, *GET_LINE*, *GET_LINES* și elimină informația depusă în *buffer*;
 - *ENABLE* – activează apelurile procedurilor *PUT*, *NEW_LINE*, *PUT_LINE*, *GET_LINE*, *GET_LINES* și permite specificarea dimensiunii *buffer*-ului.

Exemplul 7.9 – vezi curs

Exemplul 7.10 – vezi curs

Exemplul 7.11 – vezi curs

7.3.3. 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 în vederea execuției;
 - se pot rula programe de întreținere a sistemului în perioadele de timp în care acesta este mai puțin solicitat (de exemplu, noaptea) etc.

- Subprograme definite în pachet:
 - *SUBMIT* – adaugă un nou *job* în coada de așteptare;
 - *REMOVE* – șterge un *job* specificat din coada de așteptare;
 - *RUN* – execută imediat un *job* specificat;
 - *BROKEN* – dezactivează execuția unui *job* și îl setează *broken* (implicit, orice *job* este *not broken*, iar un *job* marcat *broken* nu se execută);
 - *WHAT* – descrie un *job* specificat;
 - *NEXT_DATE* – specifică momentul următoarei execuții a unui *job*;
 - *INTERVAL* – specifică intervalul de timp scurs dintre două execuții consecutive ale unui *job*;
 - *CHANGE* – modifică argumentele *WHAT*, *NEXT_DATE*, *INTERVAL*.
- Fiecare dintre subprogramele pachetului are argumente specifice.
 - Procedura *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*).

Exemplul 7.12 – vezi curs

- Vizualizarea *DBA_JOBS* din dicționarul datelor furnizează informații referitoare la starea tuturor *job*-urilor din coada de așteptare.
- Vizualizarea *DBA_JOBS_RUNNING* conține informații despre *job*-urile care sunt în curs de execuție.

Exemplul 7.13

```
SELECT JOB, LOG_USER, NEXT_DATE, BROKEN, WHAT
FROM   DBA_JOBS;

SELECT *
FROM   DBA_JOBS_RUNNING;
```

7.3.4. Pachetul *UTL_FILE*

- Pachetul *UTL_FILE* permite programului *PL/SQL* scrierea în fișiere text definite la nivelul sistemului de operare, respectiv citirea din aceste fișiere.
- Tipuri de date definite pachetul *UTL_FILE*
 - o *FILE_TYPE*
 - Specificația tipului:

```
TYPE file_type IS RECORD (  
    id          BINARY_INTEGER,  
    datatype BINARY_INTEGER);
```
- Subprograme definite în pachetul *UTL_FILE*
 - o funcții
 - *FOPEN*
 - Deschide un fișier și întoarce un *handler* care va fi utilizat în următoarele operații *I/O*.
 - Specificația funcției:

```
UTL_FILE.FOPEN (  
    location      IN VARCHAR2,  
    filename      IN VARCHAR2,  
    open_mode     IN VARCHAR2,  
    max_linesize  IN BINARY_INTEGER DEFAULT 1024)  
RETURN file_type;
```
 - Parametrul *open_mode* este un string care specifică pentru ce operații a fost deschis fișierul: *r* (*read text*), *w* (*write text*), *a* (*append text*), *rb* (*read byte mode*), *wb* (*write byte mode*) sau *ab* (*append byte mode*).
 - *IS_OPEN*
 - Întoarce valoarea *TRUE* dacă fișierul este deschis, altfel întoarce *FALSE*.
 - Specificația funcției:

```
UTL_FILE.IS_OPEN(file IN FILE_TYPE)  
RETURN BOOLEAN;
```
 - o proceduri
 - *GET_LINE*
 - Citește o linie din fișierul deschis pentru citire și o plasează într-un *buffer* de tip șir de caractere.
 - *PUT* și *PUT_LINE*
 - Permit scrierea textului din *buffer* în fișierul deschis pentru scriere sau adăugare.

- *PUTF*
 - Este asemănătoare funcției *printf()*.
 - Este o procedură *PUT* cu format.
 - *NEW_LINE*
 - Scrie în fișier un caracter sfârșit de linie specific fișierelor sistemului de operare.
 - *FCLOSE*
 - Închide un fișier
 - *FCLOSEALL*
 - Închide toate *handler*-urile fișierului deschis.
- Utilizarea componentelor acestui pachet pentru procesarea fișierelor sistemului de operare poate declanșa excepții, printre care:
 - *INVALID_PATH* – numele sau locația fișierului sunt invalide;
 - *INVALID_MODE* – parametrul *OPEN_MODE* (prin care se specifică dacă fișierul este deschis pentru citire, scriere, adăugare) este invalid;
 - *INVALID_FILEHANDLE* – *handler*-ul de fișier obținut 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*.

Exemplul 7.14 – **vezi curs**

7.3.5. Pachetul *DBMS_SQL*

- Permite utilizarea dinamică a comenzilor *SQL* în proceduri stocate sau în blocuri anonime.
 - Comenzile dinamice nu sunt încorporate 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* comenzile date ca exemplu anterior nu pot fi executate static.
 - ❖ Pachetul *DBMS_SQL* permite, de exemplu, ca într-o procedură stocată să fie utilizată o comandă *DROP TABLE*.
 - ❖ Utilizarea pachetului *DBMS_SQL* 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 tipurile de date specifice *PL/SQL*.
- Orice comandă *SQL* trebuie să treacă prin anumite etape (unele putând fi evitate):
 - analizarea sintactică;
 - validarea;
 - asigurarea că toate referințele la obiecte sunt corecte;
 - asigurarea că există privilegiile referitoare la acele obiecte (*parse*);
 - obținerea de valori pentru variabilele de legătură din comanda (*binding variables*);
 - executarea comenzii (*execute*);
 - selectarea liniilor rezultatului;
 - încărcarea liniilor rezultatului (*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* (leagă valoarea dată de variabila corespunzătoare din comanda *SQL* analizată)
 - *EXECUTE* (execută comanda *SQL* și întoarce numărul de linii procesate);
 - *FETCH_ROWS* (regăsește o linie pentru un cursor specificat, iar pentru mai multe linii folosește un *LOOP*);
 - *CLOSE_CURSOR* (închide cursorul specificat).

Exemplul 7.15 – vezi curs

Exemplul 7.16 – vezi curs

SQL Dinamic nativ

- Comanda de bază utilizată pentru procesarea dinamică nativă a comenzilor *SQL* și a blocurilor *PL/SQL* este *EXECUTE IMMEDIATE*, care are următoarea sintaxă:

```
EXECUTE IMMEDIATE șir_dinamic  
[[BULK COLLECT] INTO {def_variabila [, def_variabila ...] |  
record} ]  
[USING [IN | OUT | IN OUT] argument_bind  
[, [IN | OUT | IN OUT] argument_bind ...] ]  
[ {RETURNING | RETURN}  
[BULK COLLECT] INTO argument_bind [, argument_bind ...] ];
```

- *șir_dinamic* este un șir de caractere care reprezintă o comandă *SQL* (fără caracter de terminare “;”) sau un bloc *PL/SQL* (fără caracter de terminare “/”).
- *def_variabila* reprezintă variabila în care se stochează valoarea coloanei selectate.
- *record* reprezintă înregistrarea în care se depune o linie selectată.
- *argument_bind*, dacă se referă la valori de intrare (*IN*) este o expresie (comandă *SQL* sau bloc *PL/SQL*), iar dacă se referă la valori de ieșire (*OUT*) este o variabilă ce va conține valoarea selectată de comanda *SQL* sau de blocul *PL/SQL*.
- Clauza *INTO* este folosită pentru cereri care întorc o singură linie, iar clauza *USING* pentru a reține argumentele de legătură.
- Pentru procesarea unei cereri care întoarce mai multe linii sunt necesare instrucțiunile *OPEN...FOR*, *FETCH* și *CLOSE*.
- Prin clauza *RETURNING* sunt precizate variabilele care conțin rezultatele.

Exemplul 7.17 – vezi curs

Exemplul 7.18 – vezi curs

Exemplul 7.19 – vezi curs

Exemplul 7.20 – vezi curs

SQL Dinamic nativ versus pachetul *DBMS_SQL*

- Pentru execuția dinamică a comenzilor *SQL* în *PL/SQL* există două tehnici:
 - utilizarea pachetului *DBMS_SQL*;
 - *SQL* dinamic nativ.
- Dacă s-ar face o comparație între *SQL* dinamic nativ și funcționalitatea pachetului *DBMS_SQL*, se poate sublinia că *SQL* dinamic nativ:
 - este mai ușor de utilizat;
 - solicită mai puțin cod;
 - este mai rapid;
 - poate încărca liniile direct în înregistrări *PL/SQL*;
 - suportă toate tipurile acceptate de *SQL* static în *PL/SQL*, inclusiv tipuri definite de utilizator.
- Față de *SQL* dinamic nativ pachetul *DBMS_SQL*:
 - suportă comenzi *SQL* mai mari de 32 KB;
 - permite încărcarea înregistrărilor (procedura *FETCH_ROWS*);
 - acceptă comenzi cu clauza *RETURNING* pentru reactualizarea și ștergerea de linii multiple;
 - suportă posibilitățile oferite de comanda *DESCRIBE* (procedura *DESCRIBE_COLUMNS*);
 - analizează validitatea unei comenzi *SQL* o singură dată (procedura *PARSE*), permițând ulterior mai multe utilizări ale comenzii pentru diferite mulțimi de argumente.

7.3.6. Pachetul *DBMS_DDL*

- Permite accesul la anumite comenzi *DDL* care pot fi folosite în subprograme *PL/SQL* stocate.
 - De exemplu, prin intermediul acestui pachet pot utilizate în *PL/SQL* comenzile *ALTER* sau *ANALYZE*.

Procedura *ALTER_COMPILE*

- Permite recompilarea programului modificat (procedură, funcție, declanșator, pachet, corp pachet).

- Sintaxa

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

- Instrucțiune echivalentă *SQL*

```
ALTER PROCEDURE | FUNCTION | PACKAGE [nume_schema.]nume  
COMPILE [ PACKAGE | SPECIFICATION | BODY ];
```

Vezi Curs SGBD6 PL/SQL - Exemplul 6.19 (recompilare subprograme invalide)

Procedura *ANALYZE_OBJECT*

- Permite colectarea statisticilor pentru obiecte de tip *table*, *cluster* sau *index* care vor fi utilizate pentru optimizarea planului de execuție a comenzilor *SQL* care accesează obiectele analizate.
 - De exemplu, despre un tabel se pot obține următoarele informații: numărul de linii, numărul de blocuri, lungimea medie a unei linii, numărul de valori distincte ale unei coloane, numărul elementelor *null* dintr-o coloană, distribuția datelor (histograma) etc.

- Sintaxa

```
ANALYZE_OBJECT (tip_obiect, nume_schema, nume_obiect,  
metoda, număr_linii_estimate, procent, opțiune_metoda,  
nume_partiție);
```

- *Metoda* poate fi *COMPUTE*, *ESTIMATE* sau *DELETE*.
 - *DELETE* determină ștergerea statisticilor din dicționarul datelor referitoare la obiectul analizat.
 - *COMPUTE* calculează statisticile referitoare la un obiect analizat și le depune în dicționarul datelor.
 - *ESTIMATE* estimează statistici.
- 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 o eroare.
- 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 au 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.
- Instrucțiune echivalentă *SQL*

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

Exemplul 7.21 – **vezi curs**

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)