CUPRINS

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

7. *PL/SQL* – Pachete

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

Ce fel de subprograme integrăm într-un pachet?

Importanţa pachetelor

- Atunci când este referenţiat un pachet (atunci când este apelată pentru prima dată o construcţie a pachetului), întregul pachet este încărcat în *SGA* (zona globală sistem) şi este pregătit pentru execuţie.
- Plasarea pachetului în *SGA* reprezintă avantajul vitezei de execuţie, deoarece *server*-ul nu mai trebuie să aducă informaţia despre pachet de pe disc, aceasta fiind deja în memorie.
  - Apelurile ulterioare ale unor construcţii din acelaşi pachet, nu solicită operaţii *I/O* de pe disc.
  - De aceea, ori de câte ori apare cazul unor proceduri şi funcţii 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 toţi utilizatorii).
- Subprogramelor *overload* pot deveni stocate prin intermediul pachetelor.

Diferenţa faţă de subprograme

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

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

7.1. Definirea pachetelor

- Un pachet conţine două parţi, fiecare fiind stocată separat în dicţionarul datelor:
  - specificaţia (*package specification*)
    - este partea „vizibilă“ a pachetului, adică interfaţa cu aplicaţiile sau cu alte unităţi program;
    - poate conţine declaraţii de tipuri, constante, variabile, excepţii, cursoare şi subprograme (care vor fi vizibile altor aplicaţii);
    - se declară prima (înaintea corpului pachetului).
  - corpul (*package body*)
    - este partea „ascunsă“ a pachetului, mascată de restul aplicaţiei;
    - conţine codul care implementează obiectele definite în specificaţie (cursoarele şi subprogramele) şi, de asemenea, obiecte şi declaraţii proprii;
    - poate conţine obiecte publice (declarate în specificaţie) sau private (nu sunt declarate în specificaţie).

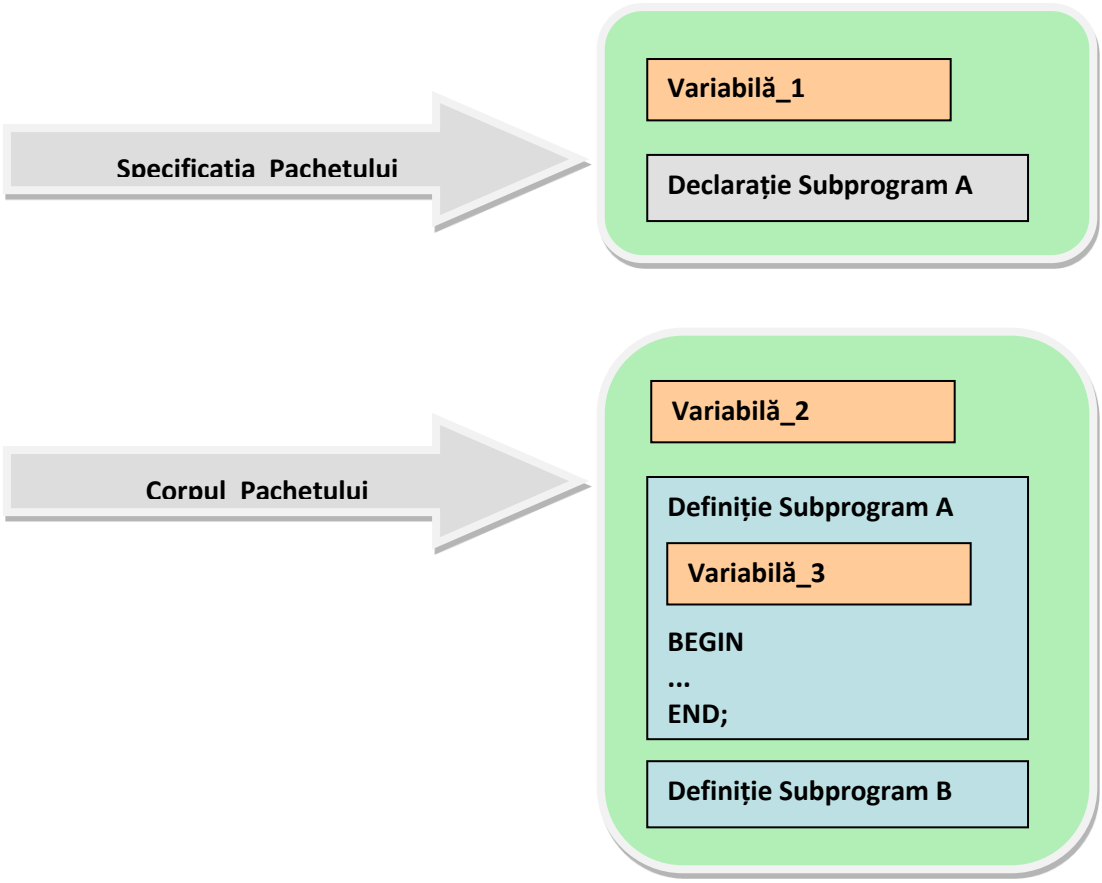

Fig. 7.1. Pachete *PL/SQL*

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

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

❖ Se pot defini pachete care cuprind doar partea de specificaţie?

❖ Se pot defini pachete care cuprind doar corpul pachetului?

❖ Ce situaţii impun definirea atât a specificaţiei, cât şi a corpului pachetului?

❖ Corpul pachetului poate fi schimbat fără a modifica specificaţia acestuia?

❖ Dacă specificaţia este schimbată, aceasta invalidează automat corpul pachetului, deoarece corpul depinde de specificaţie.

❖ Specificaţia şi corpul pachetului sunt unităţi compilate separat.

❖ Corpul poate fi compilat doar după ce specificaţia 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.

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

- Sintaxa:

```
CREATE PACKAGE nume_pachet
{IS | AS} - specificaţia pachetului
      -- declaraţii de tipuri şi obiecte publice,
      -- specificaţii de cursoare şi subprograme
END [nume_pachet];
/
CREATE PACKAGE BODY nume_pachet
{IS | AS} -- corpul pachetului
      /* declaraţii de obiecte şi tipuri private,
         corpurile cursoarelor şi subprogramelor precizate
         în specificaţie */
[BEGIN
      /* instrucţiuni de iniţializare, executate o singură
         dată, atunci când pachetul este invocat prima
         dată în sesiunea utilizatorului */
 ]
END [nume_pachet];
/
```

- Procesul de creare a specificaţiei şi corpului unui pachet urmează acelaşi algoritm ca cel întâlnit în crearea subprogramelor *PL/SQL* independente:
  - sunt verificate erorile sintactice şi semantice, iar modulul este depus în dicţionarul datelor;
  - sunt verificate instrucţiunile *SQL* individuale, adică 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, acestea devin obiecte în schema curentă.
  - În vizualizarea *USER_OBJECTS* din dicţionarul datelor, vor apărea două noi linii:

OBJECT_TYPE	OBJECT_NAME
PACKAGE	nume_pachet
PACKAGE BODY	nume_pachet

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

7.1.1. Specificaţia pachetului

- Specificaţia unui pachet cuprinde declararea procedurilor, funcţiilor, constantelor, variabilelor şi excepţiilor care pot fi accesibile aplicaţiilor, adică declararea obiectelor de tip *PUBLIC* din pachet.
  - Acestea pot fi utilizate în comenzi sau proceduri care nu aparţin pachetului.
    - Este necesar privilegiul *EXECUTE* asupra pachetului.

- ❖ Variabilele declarate în specificaţia unui pachet sunt globale pachetului şi sesiunii.
- ❖ Acestea sunt iniţializate (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}
   specificaţie_PL/SQL;
```

  - *Specificaţie_PL/SQL* poate include declaraţii de tipuri, variabile, cursoare, excepţii, funcţii, proceduri etc.
    - În secţiunea declarativă, un obiect trebuie declarat înainte de a fi referit.
  - Opţiunea *OR REPLACE* este specificată dacă există deja corpul pachetului.
  - Clauzele *IS* şi *AS* sunt echivalente, dar dacă se foloseşte *PROCEDURE BUILDER* este necesară opţiunea *IS*.
  - Clauza *AUTHID* specifică faptul ca 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 subprogramului sau a utilizatorului curent.

7.1.2. Corpul pachetului

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

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

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

❖ 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. Instanţierea pachetului

- Un pachet este instanţiat atunci când este apelat prima dată.
  - Pachetul este citit de pe disc, depus în memorie şi 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 instanţiat prima dată într-o sesiune, sunt necesare anumite iniţializări.
  - 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.
- Referinţa la o declaraţie sau la un obiect specificat în pachet se face prefixând numele obiectului cu numele pachetului.
  - În corpul pachetului, obiectele din specificaţie sunt referite fără a specifica numele pachetului.

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

7.2. Modificarea, ştergerea şi utilizarea pachetelor

Modificarea pachetului

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

  - ❖ Schimbarea corpului pachetului nu impune recompilarea construcţiilor dependente.
  - ❖ Schimbările în specificaţia pachetului necesită recompilarea fiecărui subprogram stocat care referenţiază pachetul.

- Folosind comanda *ALTER PACKAGE* se pot recompila explicit specificaţia 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*.

Funcţiile şi 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 specificaţiei, cât şi a corpului pachetului.
    - Este opţiune implicită.
  - Clauza *SPECIFICATION* determină recompilarea specificaţiei pachetului.
    - După modificarea specificaţiei unui pachet se poate dori recompilarea acesteia, pentru a verifica dacă apar erori de compilare.

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

- Atunci când se recompilează specificaţia pachetului, baza de date invalidează orice obiect local care depinde de acea specificaţie (ca de exemplu, proceduri care invocă proceduri sau funcţii din pachet).
- Corpul pachetului depinde de asemenea de specificaţie. 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 specificaţie.

Ştergerea pachetului

- Sintaxa:

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

  - Dacă în cadrul comenzii apare opţiunea *BODY*, atunci este şters doar corpul pachetului.
  - Dacă se omite opţiunea *BODY*, atunci sunt şterse atât specificaţia, cât şi corpul pachetului.
- Dacă pachetul este şters, 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 specificaţia pachetului, până când nu este recreat corpul pachetului.
- Pentru ca un utilizator să poată şterge un pachet trebuie ca pachetul să aparţină schemei utilizatorului sau utilizatorul să posede privilegiul sistem *DROP ANY PROCEDURE*.

Utilizarea pachetului

- Se realizează în funcţie 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)]
```

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

Exemplul 7.1 – vezi curs

Exemplul 7.2 – vezi curs

Comenzile *LCD - COMMIT, ROLLBACK, SAVEPOINT*

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

Invalidarea modulelor *PL/SQL*

- Dacă un subprogram independent apelează un subprogram definit într-un pachet, atunci apar următoarele situaţii:
  - atunci când corpul pachetului este modificat, dar specificaţia acestuia nu, subprogramul care referă o construcţie a pachetului rămâne valid;
  - dacă specificaţia pachetului este modificată, atunci subprogramul care referă o construcţie a pachetului, precum şi corpul pachetului devin invalide.
- Dacă un subprogram independent referit de un pachet se modifică, atunci întregul corp al pachetului devine invalid, dar specificaţia 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 specificaţia unui pachet este un tip de variabilă globală şi respectă aceleaşi reguli privind persistenţa ca şi 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).

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

7.3. Pachete predefinite

- *PL/SQL* conţine o serie de pachete predefinite utile în dezvoltarea aplicaţiilor.
- Pachetele predefinite adaugă noi funcţionalităţi limbajului, protocoale de comunicaţie, acces la fişierele sistemului etc.
- Exemple de pachete predefinite:
  - *STANDARD*
    - defineşte mediul *PL/SQL*
    - conţine funcţiile predefinite
  - *DBMS_STANDARD*
    - facilități ale limbajului utile pentru interacțiunea aplicației cu *server*-ul *Oracle*
  - *DBMS_OUTPUT*
    - permite afişarea de informaţii
  - *DBMS_DDL*
    - permite accesarea anumitor comenzi *LDD* din *PL/SQL*; în plus, oferă operaţii speciale de administrare
  - *DBMS_JOB*
    - permite planificarea şi gestiunea *job*-urilor
  - *DBMS_SQL*
    - oferă o interfaţă pentru a putea utiliza *SQL* dinamic
  - *DBMS_PIPE*
    - permite operaţii de comunicare între două sau mai multe sesiuni conectate la aceeaşi instanţă *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 utilităţi *DBA*, permite analiza obiectelor unei scheme, verifică dacă *server*-ul lucrează în mod paralel etc.
  - *DBMS_LOB*
    - oferă mecanisme de acces şi 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, ştergerea unor porţiuni din date *LOB*, deschiderea, închiderea şi regăsirea de informaţii din date *BFILE* etc.

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

- *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;
```

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

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 afişate pe ecran.
- Pachetul este utilizat preponderent:
  - în timpul testării şi depanării programelor;
  - pentru a afişa mesaje şi rapoarte în *SQL*DBA* sau *SQL*Plus*;
  - pentru a urmări paşii de execuţie a unui program.
- Subprograme definite în pachet:
  - *PUT* – depune text în *buffer* (pe o singură linie);
  - *NEW_LINE* – trimite conţinutul *buffer*-ului pe ecran (adaugă în *buffer* un sfârşit de linie);
  - *PUT_LINE* – depune text în *buffer* (*PUT*) şi trimite conţinutul *buffer*-ului pe ecran (*NEW_LINE*);
  - *GET_LINE* – citeşte din *buffer* o singură linie;
  - *GET_LINES* – citeşte din *buffer* mai multe linii (le depune într-o colecţie);
  - *DISABLE* – dezactivează apelurile procedurilor *PUT*, *NEW_LINE*, *PUT_LINE*, *GET_LINE*, *GET_LINES* şi elimină informaţia depusă în *buffer*;
  - *ENABLE* – activează apelurile procedurilor *PUT*, *NEW_LINE*, *PUT_LINE*, *GET_LINE*, *GET_LINES* şi 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 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 în vederea execuţiei;
  - se pot rula programe de întreţinere a sistemului în perioadele de timp în care acesta este mai puţin solicitat (de exemplu, noaptea) etc.

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

- Subprograme definite în pachet:
  - *SUBMIT* – adaugă un nou *job* în coada de aşteptare;
  - *REMOVE* – şterge un *job* specificat din coada de aşteptare;
  - *RUN* – execută imediat un *job* specificat;
  - *BROKEN* – dezactivează execuţia 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 execuţii a unui *job*;
  - *INTERVAL* – specifică intervalul de timp scurs dintre două execuţii 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 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*).

Exemplul 7.12 **– vezi curs**

- Vizualizarea *DBA_JOBS* din dicţionarul datelor furnizează informaţii referitoare la starea tuturor *job*-urilor din coada de aşteptare.
- Vizualizarea *DBA_JOBS_RUNNING* conţine informaţii despre *job*-urile care sunt în curs de execuţie.

Exemplul 7.13

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

SELECT *
FROM   DBA_JOBS_RUNNING;
```

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

7.3.4. Pachetul *UTL_FILE*

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

```
TYPE file_type IS RECORD (
   id       BINARY_INTEGER,
    -- handler intern
   datatype BINARY_INTEGER,
    -- tip fișier: char/nchar/binar
   byte_mode BOOLEAN
    -- a fost deschis ca fișier binar/text);
```

- Subprograme definite în pachetul *UTL_FILE*
  - funcţii
    - *FOPEN/FOPEN_NCHAR*
      - Deschide un fişier şi întoarce un *handler* care va fi utilizat în următoarele operaţii *I/O*.
      - Specificaţia funcţiei:

```
UTL_FILE.FOPEN/UTL_FILE.FOPEN_NCHAR (
   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 operaţii a fost deschis fişierul: 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 fişierul este deschis, altfel întoarce *FALSE*.
      - Specificaţia funcţiei:

```
UTL_FILE.IS_OPEN(file IN FILE_TYPE) RETURN BOOLEAN;
```

  - proceduri
    - *GET_LINE*
      - Citeşte o linie din fişierul deschis pentru citire şi o plasează într-un *buffer* de tip şir de caractere.
    - *PUT* şi *PUT_LINE*

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

- Permit scrierea textului din *buffer* în fişierul deschis pentru scriere sau adăugare.

▪ *PUTF*
- Este asemănătoare funcției *printf*().
- Este o procedură *PUT* cu format.

▪ *NEW_LINE*
- Scrie în fişier un caracter sfârşit de linie specific fişierelor sistemului de operare.

▪ *FCLOSE*
- Închide un fişier

▪ *FCLOSEALL*
- Închide toate *handler*-urile fişierului deschis.

- Utilizarea componentelor acestui pachet pentru procesarea fişierelor sistemului de operare poate declanşa excepţii, printre care:
  - *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*.

`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 şiruri de caractere.
- O comandă *SQL* dinamică este o instrucţiune *SQL* care conţine variabile ce se pot schimba în timpul execuţiei.
  - De exemplu, pot fi utilizate instrucţiuni *SQL* dinamice pentru:

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

- 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* 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 şterge 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 referinţele la obiecte sunt corecte;
  - asigurarea că există privilegiile referitoare la acele obiecte (*parse*);
  - obţinerea de valori pentru variabilele de legătură din comanda (*binding variables*);
  - executarea comenzii (*execute*);
  - selectarea linilor 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 stabileşte o zonă de memorie în care este procesată comanda *SQL*);
  - *PARSE* (stabileşte validitatea comenzii *SQL*, adică se verifică sintaxa instrucţiunii şi se asociază cursorului deschis);
  - *BIND_VARIABLE* (atribuie valoarea dată variabilei corespunzătoare din comanda *SQL* analizată)
  - *EXECUTE* (execută comanda *SQL* şi întoarce numărul de linii procesate);
  - *FETCH_ROWS* (regăseşte liniile care satisfac cererea);
  - *CLOSE_CURSOR* (închide cursorul specificat).

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

Exemplul 7.15 – vezi curs

Exemplul 7.16 – vezi curs

SQL Dinamic nativ

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

```
EXECUTE IMMEDIATE şir_dinamic
[[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 …] ];
```

  - *şir_dinamic* este un şir 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 ieşire (*OUT*) este o variabilă ce va conţine valoarea selectată de comanda *SQL* sau de blocul *PL/SQL*.
  - Clauza *INTO* este folosită pentru cereri care întorc o singură linie, iar clauza *USING* pentru a reţine argumentele de legătură.
  - Pentru procesarea unei cereri care întoarce mai multe linii sunt necesare instrucţiunile *OPEN…FOR*, *FETCH* şi *CLOSE*.
  - Prin clauza *RETURNING* sunt precizate variabilele care conţin rezultatele.

Exemplul 7.17 – vezi curs

Exemplul 7.18 – vezi curs

Exemplul 7.19 – vezi curs

Exemplul 7.20 – vezi curs

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

SQL Dinamic nativ versus pachetul _DBMS_SQL_

- Pentru execuţia dinamică a comenzilor *SQL* în *PL/SQL* există două tehnici:
  - utilizarea pachetului *DBMS_SQL*;
  - *SQL* dinamic nativ.
- Dacă s-ar face o comparaţie între SQL dinamic nativ şi funcţionalitatea pachetului *DBMS_SQL*, se poate sublinia că *SQL* dinamic nativ:
  - este mai uşor de utilizat;
  - solicită mai puţin 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.
- Faţă de *SQL* dinamic nativ pachetul *DBMS_SQL*:
  - suportă comenzi *SQL* mai mari de 32 *KB*;
  - suportă posibilităţile oferite de comanda *DESCRIBE* (procedura *DESCRIBE_COLUMNS*);
  - analizează validitatea unei comenzi *SQL* o singură dată (procedura *PARSE*), permiţând ulterior mai multe utilizări ale comenzii pentru diferite mulţimi de argumente.

7.3.6. Pachetul *DBMS_DDL*

- Permite accesul la anumite comenzi *LDD* 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, funcţie, declanşator, pachet, corp pachet).
- Sintaxa

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

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

- Instrucţiune 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 execuţie a comenzilor *SQL* care accesează obiectele analizate.
  - De exemplu, despre un tabel se pot obţine următoarele informaţii: numărul de linii, numărul de blocuri, lungimea medie a unei linii, numărul de valori distincte ale unei coloane, numărul elementelor *null* dintr-o coloană, distribuţia datelor (histograma) etc.
- Sintaxa

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

  - *Metoda* poate fi *COMPUTE*, *ESTIMATE* sau *DELETE*.
    - *DELETE* determină ştergerea statisticilor din dicţionarul datelor referitoare la obiectul analizat.
    - *COMPUTE* calculează statisticile referitoare la un obiect analizat şi le depune în dicţionarul 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 declanşează o eroare.
  - Parametrul *procent* reprezintă procentajul liniilor de estimat şi este ignorat dacă este specificat numărul liniilor de estimat (*număr_linii_estimate*). Implicit, ultimele patru argumente ale procedurii au valoarea *null*.
  - Argumentul *opţiune_metoda* poate avea forma:

```
[FOR TABLE] [FOR ALL INDEXES]
[FOR ALL [INDEXED] COLUMNS] [SIZE n]
```

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

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

- Instrucţiune echivalentă *SQL*

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

Exemplul 7.21 **– vezi curs**

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.

Bibliografie

1. Connolly T.M., Begg C.E., Database Systems: *A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
2. Dollinger R., Andron L., *Baze de date şi gestiunea tranzacţiilor*, Editura Albastră, Cluj-Napoca, 2004
3. Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2017
4. Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2017
5. Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2017
6. Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2017
7. Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2010
8. Oracle and/or its affiliates, *Pro*C/C++ Programmer's Guide*, 1996, 2014
9. Oracle University, *Oracle Database 11g*: *PL/SQL Fundamentals*, *Student Guide*, 2009
10. Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004

Copyright © 2012-2019 Lect. Dr. Gabriela Mihai. Toate drepturile rezervate.