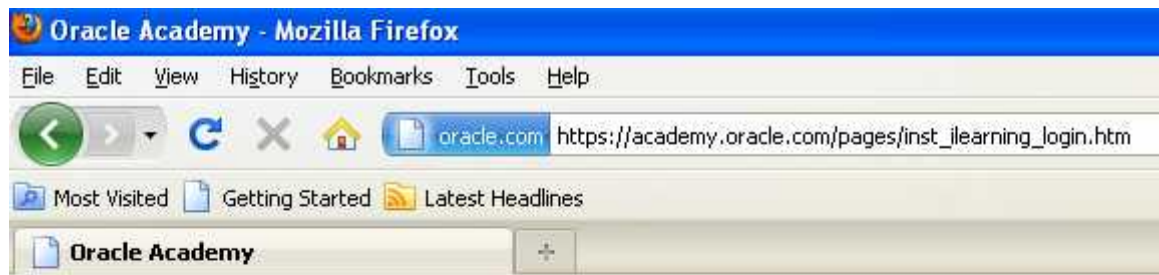


Mini tutorial de PL/SQL

Laboratoarele se bazează pe cursurile online oferite de către Oracle Academy. Pentru testarea exercițiilor în PL/SQL se va folosi aplicația proprietară Oracle: Oracle Application Express.

Accesul la cursurile online valabile studenților de la Informatică anul II se face la adresa web: <http://academy.oracle.com/>. Se alege linkul Introduction to computer science (stânga jos) -> Student Sign In (colțul din dreapta sus), iar pe pagina de iLearning se va scrie un user și o parolă furnizate în cadrul orelor de laborator.





Oracle iLearning Sign In

Username for Oracle iLearning

Password

Oracle Academy Instructors

Please type your user name and current iLearning password at the l

If you type an incorrect password, iLearning will prompt you, and will iLearning profile.

Once in iLearning, you may update your e-mail address and passwo
Please make sure to keep this profile up to date with your most curre

Pagina iLearning

Accesul la Oracle Application Express se face la adresa web: <https://iacademy3.oracle.com/>.

Conturile sunt:

Workspace: RO_P1268_S {useri de la 01 la 30}

User: RO_P1268_S {nr folosit mai sus} _PLSQL

Password: va fi comunicată

Exemplu:

School: RO_P1268_S03

User: RO_P1268_S03_PLSQL

Password: *****



Oracle Application Express

Pentru testarea exemplor in PL/SQL se va folosi SQL Workshop -> SQL Commands.

Introducere în PL/SQL

Acest mini tutorial realizează o scurtă introducere a limbajului de programare PL/SQL folosit în cadrul laboratoarelor de SGBD conform lecțiilor virtuale oferite de Academy Oracle.

PL/SQL este un limbaj de programare procedural folosit de Oracle pentru bazele de date relaționale. Pe scurt PL/SQL este:

- o extensie pentru limbajul SQL ce permite combinarea declarațiilor SQL cu un limbaj de programare;
- este un limbaj proprietar Oracle și poate fi folosit doar cu o bază de date Oracle;
- este un limbaj procedural;
- este un limbaj de generația a treia 3GL.

Exemplu:

Considerăm tabela *enrollments* cu notele studenților. Dacă vrem să selectăm notele atunci avem următoarea interogare:

```
Select stu_id, final_numeric_grade, final_letter_grade  
From enrollments;
```

Dacă vrem în continuare să modificăm datele în funcție de o anumită condiție după principiul DACĂ este adevărată expresia ATUNCI execută instrucțiunea1, ALTFEL execută instrucțiunea2, nu putem doar cu declarații SQL. Aici intervine PL/SQL care folosește variabile, constante, tipuri de date, cursoare, structuri de control și proceduri și funcții. În continuare vom oferi drept exemplu de utilizare a limbajului PL/SQL pentru modificarea notelor studenților din valori numerice cu valori de tip caracter:

Declare

```
v_new_letter_grade varchar2(1);  
Cursor c_enrollments IS
```

```
Select stu_id, final_numeric_grade from enrollments where class_id=1;
```

Begin

```
For c1 in c_enrollments
```

```
Loop
```

```
    If c1.final_numeric_grade between 66 and 75 then v_new_letter_grade:= 'A';
```

```
    Elsif c1.final_numeric_grade between 56 and 65 then v_new_letter_grade:= 'B';
```

```
    Elsif c1.final_numeric_grade between 56 and 65 then v_new_letter_grade:= 'C';
```

```
    Elsif c1.final_numeric_grade between 56 and 65 then v_new_letter_grade:= 'D';
```

```
    Else
```

```
        V_new_letter_grade:='F';
```

```
    End if;
```

```
Update enrollments
```

```
    Set final_letter_grade=v_new_letter_grade where class_id=1 and stu_id=c1.stu_id;
```

```
End loop;
```

```
Commit;
```

End;

După cum se poate observa limbajul folosește variabile definite în partea declarativă; folosește cursoare pentru situații când avem mai mult de o înregistrare în urma unui select; structuri de control: FOR, IF; și declarații SQL.

Prin urmare avantajele PL/SQL sunt:

- integrarea construcțiilor procedurale cu SQL;
- integrarea structurilor de control cu SQL ceea ce oferă un control mult mai bun al instrucțiunilor SQL și al execuțiilor acestora;
- modularizarea programelor deoarece unitatea de bază într-un program PL/SQL este **blocul** considerat a fi un modul iar programul poate fi alcătuit dintr-o secvență de astfel de blocuri sau putem avea blocuri imbricate;
- performanță ridicată prin combinarea logică a declarațiilor SQL reducându-se numărul de apeluri la baza de date;
- portabilitate, programele PL/SQL pot rula oriunde un server Oracle funcționează indiferent de SO și platforma folosită;
- manipularea excepțiilor, limbajul PL/SQL oferă posibilitatea captării excepțiilor eficient reducând astfel posibilitatea apariției erorilor.

Crearea blocurilor PL/SQL

Reprezintă unitatea de bază în PL/SQL. Există 3 tipuri de blocuri: blocul anonim, procedura și funcția, ultimele două sunt subprograme.

Un bloc PL/SQL conține trei părți:

1. **Partea Declarativă** (opțională): începe cu DECLARE și se termină când partea executabilă începe. Conține declarații de variabile, cursoare și excepții definite de utilizator (există și excepții predefinite).
2. **Partea Executabilă** (obligatorie): începe cu BEGIN și se termină cu END. End se termină cu punct și virgulă. Partea executabilă poate conține oricâte blocuri sunt necesare.
3. Partea de captare a **Excepțiilor** (opțională): partea această este inclusă în partea executabilă. Începe cu EXCEPTION.

a. Blocul anonim

```
Exemplu:  [ Declare .....]
          Begin .....
          [ Exception ..... ]
          End;
```

Blocul anonim nu poate fi invocat deoarece nu are un nume și nu mai există după ce este executat.

Exemple de blocuri anonime:

```
Begin
  DBMS_OUTPUT.PUT_LINE('un prim exemplu');
End;
```

```
-----
Declare
  v_date DATE := sysdate;
Begin
  DBMS_OUTPUT.PUT_LINE(v_date);
End;
```

```
-----
Declare
  v_country_id varchar2(2);
Begin
  select country_id into v_country_id from countries
  where country_id='a';
Exception
  WHEN NO_DATA_FOUND THEN
  Dbms_Output.Put_Line('eroare');
End;
```

Observații!

Pentru exemplul cu exception avem o excepție predefinită: NO_DATA_FOUND.

De asemenea se poate deduce ca și regulă de declarare a variabilelor: folosirea literei *v* și apoi cât mai explicit denumiri pentru variabile.

Funcția de afișare este DBMS_OUTPUT.PUT_LINE iar ca argumente pentru aceasta se folosesc variabile declarate și NU câmpuri din baza de date.

De asemenea în cadrul codului PL/SQL rezultatul în urma unei interogări SQL se păstrează tot într-o variabilă declarată.

Exe.:

```
Begin
  Select 2*2 from dual,
End;
```

Este greșit, iar eroarea va fi: PLS-00428: an INTO clause is expected in this SELECT statement.

Corect este:

```
Declare
  nr      number;
Begin
```

```
select 2*2 INTO nr from dual;  
End;
```

b. Declararea subprogrameelor: procedurilor și funcțiilor

Subprogramele sunt stocate în baza de date și pot fi invocate ori de câte ori este necesar.

1. Procedură pentru a afișa data curentă

```
CREATE PROCEDURE print_date IS  
  v_date varchar2(30);  
BEGIN  
  Select to_char(sysdate, 'Mon DD, YYYY') into v_date from dual;  
  dbms_output.put_line(v_date);  
END;
```

Cuvintele cheie sunt: **CREATE PROCEDURE** <nume> **IS** . Ea poate fi apoi apelată prin numele ei.

2. Funcție pentru a număra caracterele dintr-un string

```
CREATE FUNCTION num_characters (p_string IN VARCHAR2)  
  RETURN INTEGER IS  
  v_num_characters integer;  
BEGIN  
  Select length(p_string) into v_num_characters from dual;  
  RETURN v_num_characters;  
END;
```

Cuvintele cheie sunt: **CREATE FUNCTION** <nume> (argument IN <tip data>) **RETURN** <tip> **IS**.
Funcțiile față de proceduri returnează o valoare de un anumit tip de dată.

Folosirea variabilele in PL/SQL

Variabilele pot fi folosite pentru a stoca temporar datele, pentru manipularea valorilor stocate și pentru reutilizare.

Ele sunt folosite în cadrul declarațiilor SQL astfel:

```
Select first_name, department_id INTO v_emp_fname, v_emp_deptno FROM ...
```

unde v_emp_fname și v_emp_deptno sunt variabilele în care se vor stoca numele și departamentul;

sau

pot fi folosite ca parametri în subprograme PL/SQL și pentru a reține output-ul subprogrameelor:

```
CREATE FUNCTION num_characters (p_string IN VARCHAR2) RETURN INTEGER IS  
  v_num_characters integer
```

Toate variabilele trebuie declarate în secțiunea declarativă a oricărui block PL/SQL!

Inițializarea variabilelor

Declare

```
count INTEGER := 0;
```

Begin

```
count :=count + 1;
DBMS_OUTPUT.PUT_LINE(count);
End;
```

După cum se poate vedea variabila count a fost declarată în secțiunea declarativă și totodata inițializată cu valoarea 0.

Alte exemple de declarare și inițializare a variabilelor:

```
data_azi      DATE;
număr        NUMBER(2) NOT NULL :=2;
constanta    CONSTANT NUMBER :=10;
nume         VARCHAR2(30) DEFAULT 'Ema';
```

Atribuirea valorilor în secțiunea executabilă

```
Declare
v_nume varchar2(20);
Begin
    DBMS_OUTPUT.PUT_LINE('Numele meu este: ' || v_nume);
    v_nume := 'Ema';
    DBMS_OUTPUT.PUT_LINE('Numele meu este: ' || v_nume);
End;
```

Va afișa:

```
Numele meu este:
Numele meu este: Ema
```

SAU

```
CREATE FUNCTION num_characters (p_string IN VARCHAR2)RETURN INTEGER IS
    v_num_characters integer;
Begin
    Select length (p_string) into v_num_characters from dual;
    Return v_num_characters;
End;
Declare
    v_length_of_string integer;
Begin
    --atribuirea variabilei v_length_of_string valoarea returnată de funcție
    v_length_of_string := num_characters('Oracle Academy');
    DBMS_OUTPUT.PUT_LINE(v_length_of_string);
End;
```

! În cadrul blocurilor PL/SQL pentru comentariile pe o singură liniie se folosește -- , iar pentru cele pe mai multe linii /* */

Variabilele au următoarele proprietăți:

- pot avea maxim 30 de caractere
- trebuie să înceapă cu o literă
- pot include simboluri precum \$, _ și #
- nu pot conține spații
- sunt case insensitive

Tipurile de date in PL/SQL

Există cinci categorii de tipuri de date în PL/SQL.

- Scalar - dețin o singură valoare: character, number, date și boolean('Bacau', 234, 1-01-2010, true)
- Compus - conțin elemente interne care sunt fie scalare(recorduri) fie compuse(recorduri și tabele): table, record, nested table(tabel imbricat), varray
- obiecte mari(LOB) - valoarea lor specifică adresa obiectelor (precum imaginile): CLOB – character large object(cărți), BLOB – binary large object(poze), BFILE – binary file(filme), NCLOB – national language character large objec(caractere chinezești)
- referință: pointeri
- obiect: similar clasei din C++ și Java

Folosirea tipul SCALAR

Pentru variabile de tip caracter

- CHAR: v_dept CHAR(7)
- VARCHAR2: v_emp_job VARCHAR2(9)
- LONG: v_name LONG

Pentru variabile de tip numeric

- NUMBER: v_dept_sal NUMBER(9,2) :=0
- PLS_INTEGER
- BINARY_INTEGER: v_count Constant Binary_Integer:= 0
- BINARY_FLOAT

! INTEGER este același lucru cu NUMBER(38,0)

Pentru variabile de tip dată

- DATE: v_data DATE := sysdate+1
- TIMESTAMP: v_date TIMESTAMP
- TIMESTAMP WITH TIMEZONE – pentru zone cu diferențe de fus orar

Pentru variabile boolean

- BOOLEAN cu cele trei valori posibile: true, false și NULL v_valid BOOLEAN:=true

Atributul %TYPE

Atributul poate fi folosit pentru a declara o variabilă conform unei alte variabile declarate anterior sau a unei coloane dintr-o tabelă. Este folosit în special atunci când declarăm variabile pentru a stoca valori rezultate dintr-o tabelă din baza de date. Variabila creată cu acest atribut va avea același tip de dată conform variabilei/coloanei după care este declarată.

Pentru a declara o variabilă cu %TYPE atributul trebuie prefixat cu numele tabelii și numele coloanei conform căreia declarăm variabila.

Exemplu: create table salariat (nume varchar2(30), id_salariat number(6));

Declare

v_id_salariat salariat.id_salariat%TYPE;

Begin

select id_salariat INTO v_id_salariat from salariat where nume like 'John'

End;

SAU

Declare

v_cont NUMBER(7);

```
v_cont_debit v_cont%TYPE;
```

```
...
```

În aceste mod, variabila `v_id_salariat` va avea în mod automat același tip ca și coloana din tabela `salariat`. Atributul `%Type` este folosit pentru cazurile când schimbăm tipul de dată pentru o coloană anume, iar în acest caz variabila creată pentru stocarea valorilor din coloana respectivă nu ar mai corespunde ca tip de dată (data type mismatch). Astfel nu mai trebuie să modificăm apoi și tipul variabilei respective. Acesta este preluat automat indiferent de schimbare.

Funcții SQL în PL/SQL

Sunt disponibile în PL/SQL următoarele funcții:

- single-row character
- single-row number
- date
- data-type conversion

Nu sunt disponibile în PL/SQL:

- funcția `decode`
- funcțiile de grupare (group functions)

Exemple:

```
-- obținerea lungimii unui șir
```

```
v_lung INTEGER(5);
```

```
v_sir VARCHAR2(70):= 'exemplu';
```

```
v_lung := LENGTH(v_sir);
```

```
-- rotunjirea unui număr la 0 decimale
```

```
Declare
```

```
    v_median_age NUMBER(6,2);
```

```
Begin
```

```
    DBMS_OUTPUT.PUT_LINE(ROUND(v_median_age, 0));
```

```
End;
```

```
-- calcularea numărului de luni dintre două date calendaristice
```

```
Declare
```

```
    v_no_months PLS_INTEGER :=0;
```

```
Begin
```

```
    v_no_months := MONTHS_BETWEEN('31-ian-09', '31-ian-10');
```

```
    DBMS_OUTPUT.PUT_LINE(v_no_months);
```

```
End;
```

În PL/SQL se pot efectua două tipuri de conversii, explicite și implicite.

!!NU uitați, conversiile implicite pot încetini codul, puteți pierde controlul asupra programului deoarece prin conversie implicite faceți doar presupuneri despre cum Oracle manipulează datele, iar în caz că Oracle schimbă regulile de conversie, codul nu va mai funcționa. Este indicat să folosiți conversii explicite: `TO_NUMBER()`, `TO_CHAR()`, `TO_DATE()`.

BLOCURI IMBRICATE ȘI SCOPUL VARIABILELOR

Blocurile din PL/SQL pot conține oricâte subblocuri.

Exemplu:


```
Declare
  V_outer-variable VARCHAR2(20) := 'global variable';
Begin
  Declare
    V_inner_variable VARCHAR2(20) := 'local variable';
  Begin
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  End;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
End;
```

În exemplul de mai sus avem un bloc exterior – părinte – și un bloc interior – copil. Variabila `v_outer_variable` este declarată în blocul părinte și variabila `v_inner_variable` este declarată în blocul copil. Scopul unei variabile este în cadrul blocului sau blocurilor în care ea este accesibilă, și anume unde poate fi denumită și folosită.

În PL/SQL **scopul unei variabile este în blocul unde a fost declarată plus blocurile imbricate în cadrul blocului.**

O variabilă este **locală** pentru blocul unde a fost declarată și **globală** pentru toate subblocurile acelui bloc:
`v_outer_variable` este locală pentru blocul outer și globală pentru blocul inner.

Nu pot exista două variabile cu același nume în cadrul aceluiași bloc însă două variabile pot avea același nume dacă sunt din blocuri diferite.

Vizibilitatea unei variabile este porțiunea de program unde variabila poate fi accesată fără a folosi un calificator.

Un **calificator** este o etichetă dată unui bloc. Acest calificator este folosit pentru a accesa variabilele care au scop în cadrul altui bloc dar nu au vizibilitate.

```
<<outer>>
Declare
  v_ume_tata VARCHAR2(20):='Filip';
  v_zi_nastere DATE:='20-apr-1960';
Begin
  Declare
    v_ume_copil VARCHAR2(20):='Mihai';
    v_zi_nastere DATE:='12-ian-1993';
  Begin
    DBMS_OUTPUT.PUT_LINE('Tatal este ' || v_ume_tata);
    DBMS_OUTPUT.PUT_LINE('Zi de nastere ' || outer.v_zi_nastere);
    DBMS_OUTPUT.PUT_LINE('Copilul este ' || v_ume_copil);
    DBMS_OUTPUT.PUT_LINE('Zi de nastere ' || v_zi_nastere);
  End;
End;
Se va afișa: Tatal este Filip
             Zi de nastere 20-apr-1960
             Copilul este Mihai
             Zi de nastere 12-ian-1993
```

După cum se poate observa din exemplul de mai sus, avem patru variabile, două declarate în blocul etichetat outer și două declarate în blocul copil – imbricat blocului outer.

Variabila **v_ nume_tata** are scopul în ambele blocuri și vizibilitatea tot în ambele blocuri și este variabilă locală pentru blocul outer și globală pentru blocul imbricat – copil.

Variabila **v_ nume_copil** are scopul doar în blocul copil, unde a fost declarată și vizibilitatea la fel, și este variabilă locală pentru blocul interior.

Variabila **v_zi_nastere** declarată în **blocul outer** are scopul în ambele blocuri dar are vizibilitate doar în blocul outer deoarece în blocul copil există o variabilă cu același nume, iar variabila din outer nu poate fi accesată decât prin calificatorul outer (pentru a ști exact că facem referire la această variabilă). Este locală pentru blocul outer și globală pentru blocul copil.

Variabila **v_zi_nastere** din **blocul copil** are scopul și vizibilitatea doar în blocul copil. Pentru a fi accesată nu este nevoie de calificator. Este locală pentru blocul unde a fost declarată.

Scopul excepțiilor în blocurile imbricate

O **excepție** este o secțiune de program care „prinde” erorile pentru a înlătura afârșitul brusc al programului.

O excepție poate fi prinsă/manevrată sau propagată în mediul de execuție.

Putem avea secțiuni de manevrare a excepțiilor în cadrul blocurilor interioare și dacă excepția este tratată cu succes atunci blocul copil își termină execuția iar blocul părinte și-o continuă în mod normal:

```
Begin -- outer
    ...
    Begin -- inner
        ...
    Exception
        When exception_name Then -- excepția este tratată aici
    End;
    ...
End;
```

Sau putem să lăsăm excepția să se propage în cadrul blocului părinte (de fapt se caută în blocurile succesive secțiunea de tratare a erorii până se găsește una):

```
Begin -- outer
    ...
    Begin -- inner
        ...
    End;
    ... -- cum blocul interior nu a tratat eroarea, atunci aceasta este propagată în blocul exterior iar
        -- codul rămas din blocul exterior este sărit
    Exception
        When exception_name then -- blocul părinte va trata eroarea
    End;
```

!În continuare, pentru a înțelege mai bine teoria, este necesar să recapitulați JOIN-urile, funcțiile din SQL și sintaxa **DML (data modeling language)** pentru a insera, modifica, șterge și reuni datele dintr-o bază de date și **DDL (data definition language)** pentru crearea, modificarea și ștergerea obiectelor din baza de date.

Regăsirea și prelucrarea datelor în PL/SQL

Declarații SQL care pot fi folosite în PL/SQL sunt:

- SELECT

- INSERT, UPDATE și DELETE
- COMMIT, ROLLBACK și SAVEPOINT

! Nu se poate folosi DDL – create table, alter table, drop table și grant, revoke direct în PL/SQL.

Sintaxa SELECT în PL/SQL este:

```
SELECT lista_câmpuri INTO {variabilă1, variabilă2, . . . | nume_record}
                        FROM tabelă
                        [WHERE condiție]
```

Clauza INTO este obligatorie și este folosită pentru a specifica numele variabilelor care vor deține valorile returnate în urma selectului. Trebuie specificată câte o variabilă pentru fiecare câmp din tabelă selectat și ordinea variabilelor trebuie să corespundă cu ordinea câmpurilor din lista selectată.

De asemenea clauza select trebuie să returneze doar un rând, cele care vor returna mai mult de un rând din tabelă vor genera eroare.

Exemplu:

Declare

```
v_salary employees.salary%TYPE;
```

Begin

```
SELECT salary INTO v_salary FROM employees;
DBMS_OUTPUT.PUT_LINE('Salariul este ' || v_salary);
```

End;

Selectul de mai sus va returna mai mult de un rând, și anume toate salariile pentru toți angajații, iar variabila v_salary nu poate reține decât o singură valoare. Prin urmare se va genera eroare.

! Nu uitați:

- să terminați fiecare instrucțiune SQL cu punct și virgulă,
- fiecare valoare trebuie stocată în variabile folosind clauza INTO,
- specificați același număr de variabile câte câmpuri selectați,
- declarați variabilele corespunzătoare câmpurilor din tabelă folosind atributul %TYPE,
- nu folosiți aceeași denumire pentru variabile ca și câmpurile selectate (puteți pune litera v în fața numelui).

Insert în PL/SQL este folosit astfel:

Begin

```
INSERT INTO copy_emp (employee_id, first_name, last_name, email, hire_date, job_id, salary)
VALUES (99, 'Ruth', 'Cores', 'RCORES', sysdate, 'AD-ASST', 4000);
```

End;

- o nouă înregistrare este adăugată în tabela copy_emp

Update:

Declare

```
v_sal_increase employees.salary%TYPE :=800;
```

Begin

```
UPDATE copy_emp
SET salary=salary + v_sal_increase
WHERE job_id = 'ST_CLERK';
```

End;

- modifică salariul celor care sunt stock clerks

Ștergerea datelor:

Declare

```
v_deptno employees.department_id%TYPE := 10;
```

Begin

```
DELETE FROM copy_emp  
WHERE department_id = v_deptno;
```

End;

- șterge rândurile care aparțin departamentului 10

Merge:

Begin

```
MERGE INTO copy_emp c  
USING employees e  
ON (e.employee_id = c.employee_id)  
When MATCHED THEN  
UPDATE SET  
    c.first_name = e.first_name,  
    c.last_name = e.last_name,  
    c.email      = e.email,  
    ...  
WHEN NOT MATCHED THEN  
INSERT VALUES (e.employee_id, e.first_name, . . . e.department_id);
```

End;

Folosirea clauzelor de control a tranzacțiilor (operații în baza de date) în PL/SQL

Acstea sunt: COMMIT, ROLLBACK, SAVEPOINT.

Exemple:

Begin

```
INSERT INTO pairtable VALUES (1,2);  
COMMIT;
```

End;

-- **COMMIT** este folosit pentru a face modificările permanente

Begin

```
INSERT INTO pairtable VALUES (3,4);  
ROLLBACK;  
INSERT INTO pairtable VALUES (5,6);  
COMMIT;
```

End;

-- **ROLLBACK** este folosit pentru a anula orice modificare realizată în baza de date după ultimul COMMIT realizat. În exemplul de mai jos doar ultimul insert are loc.

Begin

```
INSERT INTO pairtable VALUES (3,4);  
SAVEPOINT my_sp_1;  
INSERT INTO pairtable VALUES (9,10);  
SAVEPOINT my_sp_2;  
INSERT INTO pairtable VALUES (11,12);
```

```
ROLLBACK to my_sp_1;  
INSERT INTO pairtable VALUES (13,14);  
COMMIT;
```

End;

-- SAVEPOINT este folosit pentru a marca puncte intermediare în procesul tranzacției. Doar ROLLBACK poate fi folosit cu SAVEPOINT. În exemplu prin ROLLBACK se anulează toate tranzacțiile până la savepoint my_sp_1. În final vor fi introduse doar primul insert și cel de după rollback.

CURSorul IMPLICIT

De fiecare dată când o instrucțiune SQL urmează să fie executată, serverul Oracle alocă o zonă de memorie privată pentru a stoca declarația SQL și datele implicate. Această zonă de memorie se numește **cursor implicit**.

Cu ajutorul atributelor cursorului implicit putem afla câte rânduri au fost procesate de către declarația SQL.

Există două tipuri de cursoare:

- cursorul implicit, este automat denumit SQL

- cursorul explicit, definit de către programator pentru interogări ce returnează mai mult de o înregistrare

Atributele pentru cursorul implicit sunt:

- SQL%FOUND – atribut boolean evaluat cu true dacă interogarea returnează cel puțin o înregistrare

- SQL%NOTFOUND – atribut boolean evaluat la true dacă interogarea nu returnează nicio valoare

- SQL%ROWCOUNT – este o valoare întreagă ce reprezintă numărul de înregistrări afectate de interogare

Exemplu:

Declare

```
v_deptno copy_emp.department_id%TYPE:=50;
```

Begin

```
Delete from copy_emp where department_id = v_deptno;
```

```
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || 'înregistrări șterse.');
```

End;

Structuri de control

Structura Condițională IF

```
IF condiție THEN
```

```
    Instrucțiune;
```

```
[ELSIF condiție THEN
```

```
    Instrucțiune;]
```

```
[ELSE
```

```
    Instrucțiune;]
```

```
END IF;
```

Declare

```
V_varsta NUMBER:=13;
```

Begin

```
IF v_varsta < 18 THEN
```

```
    DBMS_OUTPUT.PUT_LINE ('Sunt copil');
```

```
ELSIF v_varsta < 40 THEN
```

```
    DBMS_OUTPUT.PUT_LINE ('Sunt încă tânăr');
```

```
ELSE
```

```
    DBMS_OUTPUT.PUT_LINE('Sunt mereu tânăr!');
```

```
END IF;
```

End;

!ATENȚIE: atunci când avem NULL în condiția din IF se comporta ca și FALSE, deci se va trece direct pe ramura următoare din IF. De exemplu: x :=5 și y:= NULL atunci IF x!=y va da NULL deoarece NULL este nedeterminat. La fel pentru x:= NULL și y:=NULL dă tot NULL și instrucțiunile nu se execută.

Structura CASE

Exista două tipuri de structuri CASE: Case-ul obișnuit și 2 tipuri de expresii CASE care sunt funcții ce returnează una din mai multe valori într-o variabilă.

CASE

```
CASE [selector]
    WHEN expresie1 THEN instrucțiuni;
    WHEN expresie2 THEN instrucțiuni;
    . . .
    [ELSE instrucțiuni;]
END CASE;
```

Exemplu:

Declare

```
v_in NUMBER;
```

Begin

```
CASE v_in
    WHEN 1 THEN
        Select * from employees;
    WHEN 2 THE
        Select * from employees_copie;
    ELSE
        DBMS_OUTPUT.PUT_LINE('nu se selectează nimic');
END CASE;
```

End;

Expresie CASE tip1:

```
variabilă := CASE selector
    WHEN valoare1 THEN rezultat1
    WHEN valoare2 THEN rezultat2
    . . .
    [ELSE rezultat n]
```

End;

Exemplu:

Declare

```
v_out VARCHAR2(15) := 50;
```

```
v_in NUMBER;
```

Begin

```
v_out := CASE v_in
    WHEN 1 THEN 'mic'
    WHEN 50 THE 'mijlociu'
    ELSE 'altă valoare'
```

```
END;  
DBMS_OUTPUT.PUT_LINE(v_out);  
End;
```

Expresie CASE de căutare – tip2:

```
Case  
  WHEN condiție_de_căutare1 THEN rezultat1  
  WHEN condiție_de_căutare2 THEN rezultat2  
  ...  
  [ELSE rezultat n]
```

End;

- nu are selector iar clauza WHEN conține o condiție de căutare care returnează o valoare booleană și nu o expresie care poate returna o valoare de orice tip

Exemplu:

Declare

```
v_out VARCHAR2(15) := 50;  
v_in NUMBER;
```

Begin

```
v_out := CASE      -- nu mai avem selector  
  WHEN v_in = 1 THEN 'mic'  
  WHEN v_in = 50 THEN 'mijlociu'  
  ELSE 'altă valoare'  
END;  
DBMS_OUTPUT.PUT_LINE(v_out);
```

End;

Structurile REPETITIVE: FOR, WHILE, LOOP

LOOP – structura repetitivă de bază

```
LOOP  
  Instrucțiune1;  
  ...  
  EXIT [WHEN condiție];  
End LOOP;
```

Exemplu:

Declare

```
v_counter NUMBER(3) := 1;
```

Begin

LOOP

```
INSERT into salariat values (v_counter, 'Leonte', 'Bacau');  
v_counter := v_counter + 1;  
EXIT WHEN v_counter > 5;
```

End loop;

End;

Putem avea oricâte clauze EXIT dorim. Este indicată folosirea lui WHEN pentru a nu avea o buclă infinită.

Bucla WHILE

WHILE condiție LOOP /*dacă condiția este adevărată se trece la execuția instrucțiunilor. Poate returna

true, false și NULL – caz în care nu se execută while */

```
Instrucțiuni;
...
End LOOP;
```

Declare

```
v_counter NUMBER:=1;
```

Begin

```
While v_counter <= 3 LOOP
```

```
    INSERT into salariat values (v_counter, 'Leonte', 'Bacau');
```

```
    v_counter:=v_counter+1;
```

```
End LOOP;
```

End;

Bucula FOR

```
FOR contor IN [REVERSE]      /*contor nu se declară este declarat implicit și este incrementat automat.
                               Se folosește reverse pentru a decremента contorul. */
```

```
    Limita_minimă . . limita_maximă LOOP -- limitele pot fi și expresii cu rezultat valoare numerică
    Instrucțiuni;
```

```
...
End LOOP;
```

Begin

```
FOR v_counter IN 1..3 LOOP
```

```
    INSERT into salariat values (v_counter, 'Leonte', 'Bacau');
```

```
End LOOP;
```

End;

! Structurile repetitive pot fi imbricate una în alta. Atunci structurile sunt etichetate <<etichetă>> astfel încât putem specifica explicit din care buclă se iese: End LOOP <<etichetă>>.

Cursorul Explicit

Este folosit atunci când în urma interogării rezultă mai mult de o înregistrare.

Exemplu: se declară un cursor explicit pentru a obține numele și zile naționale pentru țările din Asia

Declare

```
CURSOR wf_holiday_cursor IS
```

```
    Select country_name, national_holiday_date
```

```
    From wf_countries where region_id IN (30, 34, 35);
```

```
v_country_name      wf_countries.country_name%TYPE;
```

```
v_holiday          wf_countries.nationa_holiday_date%TYPE;
```

Begin

```
OPEN wf_holiday_cursor;          --întâi se deschide cursorul
```

```
LOOP                              -- buclă pentru parcurgerea tuturor înrgistrărilor obținute
```

```
    FETCH wf_holiday_cursor INTO v_country_name, v_holiday;
```

```
    EXIT WHEN wf_holiday_cursor%NOTFOUND;
```

```
    DBMS_OUTPUT.PUT_LINE(v_country_name || ' ' || v_holiday);
```

```
End LOOP;
```

```
CLOSE wf_holiday_cursor;        --se închide cursorul
```


End;

Pașii pentru folosirea unui cursor:

1. declare: **declare nume_cursor IS instrucțiune_select;** (selectul nu va mai conține clauza INTO)
2. open: **open nume_cursor;** - cursorul este poziționat pe prima înregistrare
3. fetch – parcurgerea fiecărei înregistrări: **fetch nume_cursor INTO variabilă1, variabilă2, . . .**
4. close: **close nume_cursor;**

Structura Record în cursor

Este folosită pentru a ușura munca când vrem să selectăm toate câmpurile dintr-un tabel.

Declare

```
CURSOR wf_holiday_cursor IS
    Select *
    From wf_countries where region_id IN (30, 34, 35);
v_holiday_record wf_holiday_cursor%ROWTYPE;
```

Begin

```
OPEN wf_holiday_cursor;
LOOP
    FETCH wf_holiday_cursor INTO v_holiday_record;
    EXIT WHEN wf_holiday_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_holiday_record.v_country_name );
End LOOP;
CLOSE wf_holiday_cursor;
```

End;

Alte atribute ale cursorului explicit sunt: %ISOPEN, %NOTFOUND, %FOUND, %ROWCOUNT

Cursor FOR LOOP

For nume_record IN nume_cursor LOOP -- nume_record este un cursor declarat **implicit**.

Instrucțiuni;

...

End LOOP;

Exemplu:

Declare

```
CURSOR wf_holiday_cursor IS
    Select *
    From wf_countries where region_id IN (30, 34, 35);
```

Begin

```
FOR v_holiday_record IN wf_cursor LOOP
    -- v_holiday_record a fost implicit declarat ca wf_holiday_cursor%ROWTYPE
    EXIT WHEN wf_holiday_cursor%NOTFOUND; -- sau wf_holiday_cursor%ROWCOUNT>5
    DBMS_OUTPUT.PUT_LINE(v_holiday_record.v_country_name );
End LOOP;
-- nu se folosește OPEN și CLOSE cursor
```

End;

!În loc să folosim cursor se mai poate face și așa:

```
FOR v_holiday_record IN (Select * from wf_countries where region_id IN (30, 34, 35); ) LOOP
```

Cursor cu parametri

```
CURSOR nume_cursor [(nume_parametru tip, . . .)] IS  
    Instructiune select; -- fără clauza INTO
```

Exemplu:

Declare

```
CURSOR wf_holiday_cursor(p_region NUMBER) IS  
    Select * from wf_countries where region_id =p_region ;
```

Begin

```
FOR v_holiday_record IN wf_cursor(30) LOOP  
EXIT WHEN wf_holiday_cursor%ROWCOUNT>5  
    DBMS_OUTPUT.PUT_LINE(v_holiday_record.v_country_name );  
End LOOP;
```

End;

Cursor FOR UPDATE

```
CURSOR nume_cursor IS
```

```
    Select . . .from . . .
```

```
    FOR UPDATE [OF referință_coloană] [NOWAIT | WAIT n];
```

-referință_coloana este coloana ale cărei rânduri le blocam

- dacă rândurile au fost deja blocate nowait returnează imediat eroare. Dacă se omite nowait atunci serverul Oracle va aștepta până când rândurile sunt disponibile

- wait n – așteaptă n secunde și returnează eroare dacă altă sesiune blochează rândurile

Când declarăm un cursor FOR UPDATE fiecare rând este blocat când deschidem cursorul. Acest lucru previne modificarea cursorului de către alți utilizatori cât timp cursorul este deschis, dar nu previne citirea rândurilor. Ne permite de asemenea să modificăm noi cursorul prin clauza WHERE CURRENT OF pentru a ne referi la cea mai recentă FETCH înregistrare: UPDATE employees SET salary = . . . WHERE CURRENT OF emp_cursor. Această clauză este folosită doar cu UPDATE și DELETE.

Declare

```
CURSOR wf_holiday_cursor IS  
    Select * from wf_countries where region_id= 30  
    FOR UPDATE of region_id NOWAIT;
```

Begin

```
FOR v_holiday_record IN wf_cursor LOOP  
    UPDATE wf_countries  
        SET region_id=31  
        WHERE CURRENT OF wf_holiday_cursor;
```

```
End LOOP;
```

```
COMMIT;
```

End;

! De asemenea există cursoare imbricate și se deschid pe rând și se închid în ordinea descrescătoare deschiderii lor.

Tratarea EXCEPTIILOR

Exception handler – este porțiunea de program care definește acțiunile ce au loc când o excepție apare
EXCEPTION

```
WHEN exception1 [OR exception2] THEN -- NO_DATA_FOUND OR TOO_MANY_ROWS
Instrucțiuni;
...
[WHEN OTHERS THEN
    Instrucțiuni; ...]
```

Cele de mai sus sunt excepții Oracle implicite. Există și excepții non-predefinite. În acest sens orice programator poate declara explicit un anumit timp de excepție pentru a prinde o eroare nedefinită. Pentru aceasta trebuie:

1. declarate în DECLARE cu tipul EXCEPTION
2. asociem un număr de eroare standard Oracle ORA-n cu funcția PRAGMA EXCEPTION_INIT
3. tratarea erorii în secțiunea EXCEPTION

Exemplu:

Declare

```
e_insert_except EXCEPTION;      (1)
PRAGMA EXCEPTION_INIT(e_insert_except, -01400);      (2)
```

Begin

```
Insert into departments (department_id, department_name) values (280, NULL);
```

EXCEPTION

```
WHEN e_insert_except      (3)
    THEN DBMS_OUTPUT.PUT_LINE('insert failed');
```

End;

Putem lansa o eroare și în secțiunea BEGIN astfel:

Begin

```
IF ... THEN RAISE v_eroare;
```

EXCEPTION

```
WHEN v_eroare THEN ...
```

End;

PROCEDURI

```
CREATE [OR REPLACE] PROCEDURE nume [parametri] IS | AS
    [variabile, cursoare];
```

Begin

```
Instrucțiuni SQL și PL/SQL;
```

[Exception

```
    WHEN exception_nume;]
```

End [nume];

Exemplu:

```
CREATE OR REPLACE PROCEDURE add_salariat IS
```

```
    v_id salariat.salariat_id%TYPE:=20;
```

```
    v_nume salariat.salariat_nume%TYPE:='Mihaela';
```

Begin

```
INSERT INTO salariat values(v_id, v_nume);
```

```
DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT ); --folosit pentru a testa dacă insertul are loc
```

End;

Invocarea Procedurilor

! Nu se poate invoca o procedură în cadrul unei instrucțiuni SQL precum SELECT.

```
Begin
    add_salariat;
End;
```

Crearea procedurilor cu PARAMETRI

```
CREATE OR REPLACE PROCEDURE add_salariat
    (v_id IN salariat.salariat_id%TYPE, -- este numit parametru formal
    v_numename IN salariat.salariat_numename%TYPE ) IS
Begin
    INSERT INTO salariat values(v_id, v_numename);
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
End;

Begin
    add_salariat(20, 'Mihaela'); -- parametru actual
End;
```

Modul parametrilor este specificat în declararea parametrilor formali după numele acestora și înainte de tipul lor. Un IN parametru oferă valori pentru prelucrare (pot fi doar citați în cadrul procedurii, nu pot fi modificați), un parametru OUT returnează o valoare apelantului, iar un IN OUT parametru oferă o valoare de intrare care apoi poate fi returnată ca valoare modificată.

FUNCTII

```
CREATE [OR REPLACE] FUNCTION nume [parametri [mode] tip] RETURN tip IS | AS
    [variabile];
Begin
    Instrucțiuni SQL și PL/SQL;
    RETURN expresie;
End [nume];
```

Apelul se face: **variabilă := nume_funcție(parametri); SAU nume_funcție(parametri);**

! Pentru a șterge o procedură sau o funcție stocată se folosește DROP PROCEDURE|FUNCTION nume;