Universitatea
Politehnica
București

Facultatea de
Automatică și
Calculatoare

Catedra de
Calculatoare

Laborator 6
Proceduri PL/SQL

Autori

Conf. Dr. Ing. Alexandru Boicea

As. Drd. Ing. Ciprian-Octavian Truică

Cuprins

- Subprograme
- Proceduri declarate în cadrul unui bloc
- Proceduri stocate
- Parametrii unei proceduri
- Cursoare în proceduri
- Tipuri colecție ca parametri (Opțional)
- Cursoare ca parametric (Opțional)

Subprograme

- Un subprogram este un bloc PL/SQL cu nume (spre deosebire de blocurile anonime) care poate primi parametri şi poate fi invocat dintr-un anumit mediu (de exemplu, SQL*Plus, Oracle Forms, Oracle Reports etc.)
- Subprogramele sunt bazate pe structura de bloc PL/SQL.
- Similar unui bloc PL/SQL ele conţin o parte declarativă facultativă, o parte executabilă obligatorie şi o parte de tratare de excepţii facultativă.
- Exista 2 tipuri de subprograme:
  - proceduri;
  - funcţii (trebuie să conţină cel puţin o comandă RETURN);
- Subprogramele pot fi :
  - locale (în cadrul altui bloc PL/SQL sau subprogram)
  - stocate (create cu comanda CREATE) – odată create, procedurile şi funcţiile sunt stocate în baza de date de aceea ele se numesc subprograme stocate.

Proceduri PL/SQL

- O procedură este un subprogram care execută un set de instrucțiuni și nu returnează direct o valoare către programul apelant;
- Rezultatele obținute prin procesarea datelor pot fi folosite în programul apelant, în funcție de modul de declarare a parametrilor.

Proceduri declarate în cadrul unui bloc

- O procedură stocată poate fi declarată în cadrul unui bloc PL/SQL;
- Durata de viață al unei astfel de proceduri este numai pe perioada existenței blocului;
- După terminarea execuției blocului procedura se pierde, deoarece este stocată doar în memorie, nu este stocată permanent în SGBD;

Proceduri declarate în cadrul unui bloc

- Sintaxa

```
DECLARE
        PROCEDURE procedure_name [(parameter_name [IN|OUT|IN OUT]
                                                    parameter_type,...)]
        {IS|AS}
                [procedure_declaration_section]
        BEGIN
                procedure_executable_section;
        [EXCEPTION
                procedure_exception_section]
        END [procedure_name];
        ...
```

Proceduri declarate în cadrul unui bloc

```
BEGIN
        [DECLARE
                block_variables;]
        BEGIN
                block_executable_section;
                 procedure_name [(parameters)];
                block_executable_section;
        [EXCEPTION
                block_exception_section]
        END;
END;
```

Proceduri declarate în cadrul unui bloc

- **procedure_name** – numele procedurii
- **parameter_name** – numele unui parametru formal din lista de parametri
- **parameter_type** – reprezintă tipul parametrului formal, se dă fără precizie
  - **GREȘIT: nume_param varchar(30)**
  - **CORECT: nume_param varchar**
- **procedure_declaration_section** – secțiunea de declarare a variabilelor locale folosite în procedură, poate să lipsească
- **procedure_executable_section** – secțiunea executabilă a procedurii
- **procedure_exception_section** – secțiune de tratare a excepțiilor din cadrul procedurii, poate să lipsească
- **block_variables** – secțiunea de declarare a variabilelor locale folosite în bloc
- **block_executable_section** – secțiunea executabilă a blocului
- **block_exception_section** – secțiunea de tratare a excepțiilor din cadrul blocului, este opțională
- **IN|OUT|IN OUT** – specifică dacă parametrul poate fi referit sau modificat în interiorul sau exteriorul procedurii

Proceduri declarate în cadrul unui bloc

- Ex1. Exemplu de procedură declarată într-un bloc.

```
set verify off
set serveroutput on
declare
  procedure procFaraParam
  as
  begin
    dbms_output.put_line('Am apelat o procedura fara parametri');
  end;
begin
  -- nu este nevoie de un bloc imbricat fiindca nu am o sectiune de
  declaratii
  -- daca blocul necesita variabile,
  -- atunci TREBUIE sa se foloseasca un bloc imbricat
  procFaraParam();
  -- pentru proceduri care NU au parametri parantezele nu sunt obligarotii
  --procFaraParam;
end;
/
```

Proceduri declarate în cadrul unui bloc

- Ex. 2. Să se scrie o procedură declarată în cadrul unui bloc care întoarce salariu maxim pentru un ID de departament și o funcție introduse de la tastatură. Salariu maxim să fie returnat folosindu-se o variabilă scalară. Să se traducă joburile în limba română, în cadrul procedurii.

Proceduri declarate în cadrul unui bloc

```
set serveroutput on;
declare
  procedure salariu (deptId in number,
    functie in out varchar2, salariuMaxim out number)
  is --merge si AS in loc de IS
    salmax number; -- imediar dupa IS sau AS se fac declaratiile
  begin
    select max(sal) into salmax from emp
      where deptno = deptId and lower(job) = lower(functie) group by deptno;
    salariuMaxim := salmax;
    -- se putea insera direct in variabila salariuMaxim trimisa ca parametru
    --select max(sal) into salariuMaxim from emp
    --  where deptno = deptId and lower(job) = lower(functie) group by deptno;
    functie := case upper(functie)
                 when 'ANALYST' then 'Analist'
                 when 'SALESMAN' then 'Vanzator'
                 when 'CLERK' then 'Functionar'
                 when 'PRESIDENT' then 'Presedinte'
                 when 'MANAGER' then 'Director'
                 else 'Nu avem functia'
               end;
  exception
    when no_data_found then
      dbms_output.put_line('Nu a fost gasit nici o inregistrare');
  end;
```

Proceduri declarate în cadrul unui bloc

```
begin
  declare
    numeDepartament emp.ename%type;
    idDept emp.deptno%type;
    functie varchar(40);
    salMax number;
  begin
    idDept := &idDepartament;
    functie := '&numeFunctie';

    select dname into numeDepartament from dept where deptno = idDept;
    salariu(idDept, functie, salMax);
    -- folosind operatorul de asociere (discutat la cursoare, merge la fel)
    --salariu(deptId => idDept, salariuMaxim => salMax, functie => functie);
    --salariu(idDept, salariuMaxim => salMax, functie => functie);
    --salariu(idDept, functie, salariuMaxim => salMax);
    dbms_output.put_line('In departamentul '||numeDepartament
      ||' salariu maxim pentru functia '||functie
      ||' este '||salMax);
    exception
      when no_data_found then
        dbms_output.put_line('Departament inexistent');
  end;
end;
```

Proceduri stocate

- O procedură stocată este creată ca un obiect în dicționarul de date de la nivelul SGBD-ului și poate fi folosită oricând;
- Sintaxa

```
CREATE [OR REPLACE] PROCEDURE procedure_name [(parameter_name
        [IN|OUT|IN OUT] parameter_type, ... )]
    [AUTHID {DEFINER|CURRENT_USER}]
    [PRAGMA AUTONOMOUS_TRANSACTION]
    {IS|AS}
        [declaration_section]
    BEGIN
        execution_section;
    [EXCEPTION
        exception_section;]
    END [procedure_name];
```

Proceduri stocate

- **procedure_name** – numele procedurii
- **parameter_name** – numele unui parametru formal din lista de parametri
- **parameter_type** – reprezintă tipul parametrului formal, se specifica fără precizie
- **declaration_section** – secțiunea de declarare a variabilelor locale folosite în procedură
- **executable_section** – secțiunea executabilă a procedurii
- **exception_section** – secțiune de tratare a excepțiilor din cadrul procedurii, poate să lipsească

Proceduri stocate

- **IN|OUT|IN OUT** – specifică dacă parametrul poate fi referit sau modificat în interiorul sau exteriorul procedurii
- **[AUTHID {DEFINER|CURRENT_USER}]** – specifică dacă o procedură stocată se execută cu drepturile celui care a creat-o (valoarea implicită) sau ale utilizatorului curent
- **[PRAGMA AUTONOMOUS_TRANSACTION]** – specifică că execuția procedurii suspendă tranzacția curentă care se reia după terminarea execuției procedurii, adică într-o tranzacție imbricăm o altă tranzacție cu propriile sale comenzi TCL (COMMIT și ROLLBACK)

Proceduri stocate

- Ex. 3. Să se creeze o procedură stocată care calculează veniturile angajaților cu o vechime de peste 20 de ani în firmă, dintr-un anumit departament.

```
create or replace procedure sp_venit(idDept in number,
  venit in out number, dataRef in date default sysdate)
 is
  numeDept dept.dname%type;
 begin
  select dname into numeDept from dept where deptno = idDept;
  select nvl(sum(sal+nvl(comm, 0)),0) into venit from emp
    where deptno = idDept and add_months(hiredate, 240) < dataRef;
  dbms_output.put_line(rpad(numeDept,20)||rpad(venit,20));
 exception
  when no_data_found then
    dbms_output.put_line('Nu a fost gasita nicio inregistrare!');
 end sp_venit;
 /
```

Proceduri stocate

- Procedura stocată va fi apelată din următorul bloc anonim:

```
set verify off
set serveroutput on
declare
  dataRef date default sysdate;
  venit number := 0;
  total number := 0;
begin
  for rand in (select distinct deptno from dept order by deptno)
  loop
    sp_venit(rand.deptno, venit, dataRef);
    -- merge si fara sa dam data fiindca are default in lista
    -- de parametri pe care ii primeste procedura stocata
    -- sp_venit(rand.deptno, venit);
    -- sp_venit(idDept => rand.deptno, venit => venit);
    -- sp_venit(venit => venit, idDept => rand.deptno);
    -- sp_venit(rand.deptno, venit => venit);
    total := total + nvl(venit, 0);
  end loop;
  dbms_output.put_line('Total '||total);
end;
/
```

Proceduri stocate

Observații:

- Procedura se va crea prima și dacă nu are erori de compilare se va afișa mesajul: **Procedure created;**
- În caz contrar se va afișa mesajul: **Warning: Procedure created with compilation errors;**
- Pentru a vedea erorile de compilare se va folosi comanda **show errors;**
- Dacă în dicționarul de date există o procedură cu un nume, atunci nu se mai poate crea alta cu același nume;
- Pentru a suprascrie o procedură existentă se folosește comanda **CREATE OR REPLACE**, folosiți comanda aceasta cu atenție deoarece puteți să înlocuiți din greșeală o procedură deja existentă, scrisă de un alt dezvoltator, care are același nume dar are altă funcționalitate;
- Pentru a șterge o procedură stocată se folosește comanda DDL drop: **DROP PROCEDURE procedure_name.**

Parametrii unei proceduri

- Parametrii sunt variabile disponibile atât pentru programul principal (apelant) cât și pentru procedură (subprogram) și care determină funcționalitatea și rezultatele procesării datelor;
- Parametrii unei proceduri sunt opționali și sunt declarați în momentul creării procedurii;
- Un nume de parametru trebuie să înceapă cu o literă, să nu conțină spații, și să aibă lungimea maximă de 30 de caractere;
- Parametrii actuali reprezintă valorile efective transmise procedurii în momentul apelului sau rezultatele întoarse de procedură (în cazul definirii unor parametri OUT sau IN OUT), în timp ce parametrii formali sunt cei care recepționează valorile parametrilor actuali, fiind referiți conform logicii procedurii.

Parametrii unei proceduri

- Sintaxă:

  parameter_name [IN|OUT [NOCOPY]|IN OUT [NOCOPY]]
  parameter_type [{:= | DEFAULT } {expresion|value}]
- **parameter_name** – numele parametrului
- **parameter_name** – tipul parametrului
- **NOCOPY** – specifică că parametrul se transmite prin referință (adresă) nu prin valoare și este valabil doar pentru **OUT** și **IN OUT** care se transmit implicit prin valoare
- **:= |DEFAULT** – pentru a da o valoare implicită
- **expresion | value** – se pot atribui expresii sau valori parametrilor

Parametrii unei proceduri

- Parametrii unei proceduri pot fi transmiși în două moduri, prin referință sau prin valoare;
- În primul caz un pointer către parametrul actual este transmis parametrului formal corespunzător și nu se copiază efectiv, ceea ce duce la creșterea performanțelor mai ales la parametrii de tip colecție;
- La transmiterea prin valoare are loc copierea valorii parametrului actual la cel formal;
- Până la versiunea 8i doar parametrii de intrare (IN) erau transmiși prin referință (se întâmplă în continuare acest lucru), iar cei de ieșire (OUT) prin valoare;
- Folosindu-se NOCOPY se pot transmite parametrii de ieșire (OUT) și cei de intrare-ieșire (IN OUT) prin referință;
- Când un parametru este transmis prin referință, orice modificare asupra parametrului actual va modifica și parametrul formal, deoarece ambii au pointeri către aceeași valoare.

Parametrii unei proceduri

- Există trei tipuri de parametri:
  - IN – parametrul poate fi referit în interiorul procedurii, dar nu poate fi modificat;
  - OUT – parametrul nu poate fi referit în interiorul procedurii, dar poate fi modificat și poate fi referit în afara procedurii (în programul apelant);
  - IN OUT – parameterul poate fi referit în interiorul procedurii, poate fi modificat și poate fi referit în afara procedurii (în programul apelant).
- Parametrul IN este tipul implicit;
- Parametrul OUT este inițializat cu NULL, iar procedura atribuie parametrului o valoare care poate fi referită în afara ei;
- Un parametru IN OUT poate avea (sau nu) o valoare inițială, valoarea inițială poate fi modificată în procedură și orice modificare poate fi returnată în programul apelant.

Parametrii unei proceduri

- Parametrii sunt declarați ca tipuri de date dar fără lungime de tip sau de precizie, adică un parametru poate fi declarat ca VARCHAR2 dar fără o componentă de lungime(de ex. VARCHAR2(30) nu este permis);
- Parametrii pot avea o valoare implicită similară variabilelor, deci se poate folosi operatorul de atribuire (:=) sau opțiunea DEFAULT;
- Dacă un parametru are o valoare implicită nu este nevoie să se includă în apel;
- Se poate folosi operatorul de asociere (=>) pentru a trimite valori efective unui parametru, în acest caz ordinea nu contează;
- Se indică folosirea operatorului de asociere în cazul procedurilor cu mulți parametri mai ales pentru că o astfel de scriere este mult mai clară și nu există posibilitatea strecurării unei erori în momentul când se apelează procedura.

Cursoare în proceduri

- Cursoarele declarate într-o procedură urmează aceleași reguli ca într-un bloc obișnuit;
- O procedură poate conține mai multe cursoare înlănțuite sau imbricate.

Cursoare în proceduri

- Ex. 4. Să se scrie o procedură nestocată, care utilizează un cursor, pentru calculul numărului de zile de concediu pentru toți angajații, după următorul algoritm:
- a) manageri de departament
  - Vechimea < 32 de ani primesc 20 zile de concediu
  - Vechimea >= 32 de ani primesc 22 zile de concediu
- b) angajatii care nu sunt sefi :
  - Vechimea < 32 de ani primesc 15 zile de concediu
  - Vechimea >= 32 de ani primesc 22 zile de concediu

Cursoare în proceduri

```
set serveroutput on;
declare
  procedure concediu
  as
    cursor c_concediu is select * from emp order by deptno;
    randConcediu c_concediu%rowtype;
    type r_concediu is record (
      numeDept dept.dname%type,
      numeAng emp.ename%type,
      dataAng emp.hiredate%type,
      sef varchar2(2),
      aniVechime number,
      zileConcediu number
    );
    concediu r_concediu;
    manager emp.mgr%type;
```

Cursoare în proceduri

```
begin
  open c_concediu;
  loop
    fetch c_concediu into randConcediu;
    exit when c_concediu%notfound;

    select dname into concediu.numeDept from dept
      where deptno = randConcediu.deptno;
    concediu.dataAng := randConcediu.hiredate;
    concediu.numeAng := randConcediu.ename;
    concediu.aniVechime := trunc(months_between(sysdate,
                randConcediu.hiredate)/12);

    begin
      select mgr into manager from emp
        where mgr = randConcediu.empno;
      concediu.sef := 'DA';
    exception
      when no_data_found then concediu.sef := 'NU';
      when too_many_rows then concediu.sef := 'DA';
    end;
```

Cursoare în proceduri

```
        if concediu.aniVechime<32 and concediu.sef = 'DA' then
          concediu.zileConcediu := 20;
        elsif concediu.aniVechime>=32 and concediu.sef = 'DA' then
          concediu.zileConcediu := 22;
        elsif concediu.aniVechime<32 and concediu.sef = 'NU' then
          concediu.zileConcediu := 15;
        else
          concediu.zileConcediu := 17;
        end if;

        dbms_output.put_line(rpad(concediu.numeDept, 20)
          ||rpad(concediu.numeAng, 20)||rpad(concediu.dataAng, 20)
          ||rpad(concediu.sef, 20)||rpad(concediu.aniVechime, 20)
          ||rpad(concediu.zileConcediu, 20));


    end loop;
  end;
begin
  concediu;
end;
```

Cursoare în proceduri

- Ex. 5. Să se scrie un program PL/SQL cu o procedură care distribuie salariul șefului de departament la subalternii lui, în funcție de vechimea în companie, astfel:
  - grupa 1 – subalternii cu o vechime <= 31 de ani să primească un un fond de premiere egal cu 30% din salariul șefului
  - grupa 2 – subalternii cu o vechiime > 31 de ani să primească un fond de premiere egal cu 70% din salariul șefului
- Fondul de premiere se distribuie în mod egal la toți subalternii dintr-o grupă.

Cursoare în proceduri

```
set serveroutput on;
declare
  procedure distributie(sal in number, vechime in number, nrangl in number,
    nrang2 in number, prima out number, grupa out number)
  is
  begin
    if vechime <= 31 then
      prima := round(0.3*sal/nrangl);
      grupa := 1;
    else
      prima := round(0.7*sal/nrang2);
      grupa := 2;
    end if;
  end;
```

Cursoare în proceduri

```
begin
  declare
    nrangl number := 0;
    nrang2 number := 0;
    salSef emp.sal%type;
    prima number;
    grupa number;
    numeSef varchar2(20);
    vechime number;
  begin
    dbms_output.put_line(rpad('Nume sef', 20)||rpad('Salariu sef', 20)
      ||rpad('Nume subaltern',20)||rpad('Vechime',20)||rpad('Prima', 20)
      ||rpad('Grupa',20));
    for sef in (select distinct mgr from emp where mgr is not null)
    loop
      nrAngl := 0;
      nrAng2 := 0;
      select ename, sal into numeSef, salSef from emp where empno=sef.mgr;
```

Cursoare în proceduri

```
    for angajat in (select distinct empno from emp where mgr = sef.mgr)
    loop
      select trunc(months_between(sysdate, hiredate)/12) into vechime
        from emp where empno = angajat.empno;
      if vechime<=31 then
        nrAng1:= nrAng1+1;
      else
        nrAng2 := nrAng2+1;
      end if;
    end loop;

    for angajat in (select distinct ename,
        trunc(months_between(sysdate, hiredate)/12) vechime
        from emp where mgr = sef.mgr)
    loop
        distributie(salSef,angajat.vechime, nrAng1, nrAng2, prima, grupa);
        dbms_output.put_line(rpad(numeSef, 20)||rpad(salSef, 20)
          ||rpad(angajat.ename,20)||rpad(angajat.vechime,20)||rpad(prima, 20)
          ||rpad(grupa,20));
    end loop;
  end loop;
 end;
end;
```

Tipul Colecție ca parametri (Opțional)

- Pentru a folosi un tip colecție ca parametru pentru o procedură stocată acesta trebuie să fie definit la nivelul bazei de date folosindu-se instrucțiunea DDL *create*;
- Tipurile colecție care pot fi declarate la nivelul bazei de date sunt:
  - Varray
  - Nested table

Tipul Colecție ca parametri (Opțional)

- Ex. 6. Să se creeze un vector la nivelul dicționarului de date. Într-un bloc, să se definească și inițializeze o variabilă de tipul nou creat și să se apeleze o procedură care afișează valorile elementelor. Să se șteargă tipul din dicționarul de date.
- Rezolvare:
- Pasul I – crearea vectorului în dicționarul de date

```
create or replace type v_culori is varray(6) of varchar2(20);
```

Tipul Colecție ca parametri (Opțional)

- Pasul II – crearea blocului și a procedurii

```
set serveroutput on;
declare
  procedure itereaza(culori in v_culori)
  is
  begin
    for contor in culori.first .. culori.last
    loop
      dbms_output.put_line(culori(contor));
    end loop;
  end;
begin
  declare
    culori v_culori;
  begin
    culori := v_culori('Rosu', 'Galben', 'Albastru', 'Verde', 'Maro', 'Mov');
    itereaza(culori);
  end;
end;
```

Tipul Colecție ca parametri (Opțional)

- Pasul III - ștergerea din baza de date a tipului v_culori

```
drop type v_culori;
```

Tipul Colecție ca parametri (Opțional)

- Ex. 7. Să se refacă ex. 6 folosind un nested table.
- Rezolvare:
- Pasul I – crearea tipului nested table în dicționarul de date

```
create or replace type t_culori is table of varchar2(20);
```

- Pasul II – crearea blocului

Tipul Colecție ca parametri (Opțional)

```
set serveroutput on;
declare
  procedure itereaza(culori in t_culori)
  is
  begin
    for contor in culori.first .. culori.last
    loop
      dbms_output.put_line(culori(contor));
    end loop;
  end;
begin
  declare
    culori t_culori;
  begin
    culori := t_culori('Rosu', 'Galben', 'Albastru', 'Verde', 'Maro', 'Mov');
    itereaza(culori);
  end;
end;
```

Tipul Colecție ca parametri (Opțional)

- Pasul III - ștergerea tipului din dicționarul de date

```
drop type t_culori;
```

Cursoare ca parametri(Opțional)

- O procedură stocată poate să primească ca parametru un cursor;
- Acesta trebuie să fie de tipul REF CURSOR

Cursoare ca parametri(Opțional)

- Ex. 8. Folosire cursor ca parametru IN:

```
set serveroutput on;
declare
  procedure  test_ref (c_dept in sys_refcursor) is
    type r_dept is record (dname dept.dname%type, loc dept.loc%type);
    dept_rec r_dept;
  begin
    dbms_output.put_line(rpad('Nume',20)||rpad('Locatie', 20));
    loop
      fetch c_dept into dept_rec;
      exit when c_dept%notfound;
      dbms_output.put_line(rpad(dept_rec.dname,20) || rpad(dept_rec.loc,20));
    end loop;
  end;
begin
  declare
    type dept_ref is ref cursor;
    c_dept dept_ref;
  begin
    open c_dept for select dname, loc from dept;
    test_ref(c_dept);
    if c_dept%isopen then
      dbms_output.put_line('Cursorul este deschis, trebuie inchis!');
      close c_dept;
    end if;
  end;
end;
```

Cursoare ca parametri(Opțional)

- Ex. 9. Folosire cursor ca parametru OUT:

```
set serveroutput on;
declare
  procedure  test_ref (c_dept out sys_refcursor) is
  begin
    open c_dept for select dname, loc from dept;
  end;
begin
  declare
    type dept_ref is ref cursor;
    c_dept dept_ref;
    type r_dept is record (dname dept.dname%type, loc dept.loc%type);
    dept_rec r_dept;
  begin
    test_ref(c_dept);
    dbms_output.put_line(rpad('Nume',20)||rpad('Locatie', 20));
    loop
      fetch c_dept into dept_rec;
      exit when c_dept%notfound;
      dbms_output.put_line(rpad(dept_rec.dname,20) || rpad(dept_rec.loc,20));
    end loop;
    if c_dept%isopen then
      dbms_output.put_line('Cursorul este deschis, trebuie inchis!');
      close c_dept;
    end if;
  end;
end;
```

Cursoare ca parametri(Opțional)

- Ex. 10. Folosire cursor ca parametru IN OUT:

```
set serveroutput on;
declare
  procedure  test_ref (ref_cursor in out sys_refcursor) is
    type r_dept is record (dname dept.dname%type, loc dept.loc%type);
    dept_rec r_dept;
  begin
    dbms_output.put_line(rpad('Nume',20)||rpad('Locatie', 20));
    loop
      fetch ref_cursor into dept_rec;
      exit when ref_cursor%notfound;
      dbms_output.put_line(rpad(dept_rec.dname,20) || rpad(dept_rec.loc,20));
    end loop;
    if ref_cursor%isopen then
      close ref_cursor;
    end if;
    open ref_cursor for select ename, sal from emp;
  end;
```

Cursoare ca parametri(Opțional)

```
begin
  declare
    type c_ref_cursor is ref cursor;
    ref_cursor c_ref_cursor;
    type r_emp is record (ename emp.ename%type, sal emp.sal%type);
    emp_rec r_emp;
  begin
    open ref_cursor for select dname, loc from dept;
    dbms_output.put_line(rpad('Nume',20)||rpad('Locatie', 20));
    test_ref(ref_cursor);
    dbms_output.put_line(rpad('Nume',20)||rpad('Salariu', 20));
    loop
      fetch ref_cursor into emp_rec;
      exit when ref_cursor%notfound;
      dbms_output.put_line(rpad(emp_rec.ename,20) || rpad(emp_rec.sal,20));
    end loop;
    if ref_cursor%isopen then
      close ref_cursor;
    end if;
  end;
end;
```