



Universitatea
Politehnica
București



Facultatea de
Automatică și
Calculatoare



Catedra de
Calculatoare

Laborator 7

Funcții PL/SQL

Autori

Conf. Dr. Ing. Alexandru Boicea

As. Drd. Ing. Ciprian-Octavian Truică



Cuprins

- Funcții PL/SQL
- Funcții declarate în cadrul unui bloc
- Funcții stocate pe server
- Apelul funcțiilor din comenzi SQL
- Considerente asupra procedurilor și funcțiilor
- Informații din dicționarul bazei de date



Funcții PL/SQL

- O funcție este un subprogram care acceptă parametri, poate fi apelată dintr-un program apelant și returnează o valoare;
- În general, funcțiile sunt folosite pentru a procesa un set de date și a întoarce un rezultat;
- Funcțiile și procedurile au structuri asemănătoare;
- O funcție trebuie să întoarcă o valoare folosind clauza **return** (și mai multe folosind parametri de tip OUT și IN OUT) în timp ce o procedură poate să întoarcă una sau mai multe valori prin intermediul parametrilor de tip OUT sau IN OUT.



Funcții declarate în cadrul unui bloc

- Sunt funcții stocate în memorie împreună cu blocul PL/SQL în care sunt declarate și au o funcționalitate asemănătoare procedurilor;
- Durata de viață a unei astfel de funcții este doar pe perioada de existență a blocului;
- Aceste funcții pot să fie apelate doar în cadrul blocului în care sunt declarate;
- După terminarea execuției blocului funcțiile locale sunt șterse din memorie;
- Pentru a apela din nou o astfel de funcție trebuie să se execute din nou și blocul în care a fost declarată;
- Funcțiile pot fi folosite în blocul apelant, folosind operatorul de atribuire, în cadrul unor expresii și condiții;



Funcții declarate în cadrul unui bloc

- Sintaxa :

DECLARE

FUNCTION function_name [(parameter_name [IN|OUT|IN OUT]
parameter_type,...)]

RETURN return_data_type

{IS | AS}

[function_declaration_section]

BEGIN -- blocul funcției

function_executable_section;

[EXCEPTION

function_exception_section]

END [function_name]; -- sfârșitul declarării funcției

...



Funcții declarate în cadrul unui bloc

BEGIN

DECLARE

block_variables;

var return_data_type; -- variabilă de tipul întors de funcție

block_variables;

BEGIN

block_executable_section;

var := function_name[(parametrii)];

block_executable_section;

[EXCEPTION

block_exception_section]

END;

END;



Funcții declarate în cadrul unui bloc

- **function_name** – numele funcției
- **parameter_name** – numele unui parametru formal din lista de parametri
- **parameter_type** – reprezintă tipul parametrului formal, **se specifica fără precizie**
- **return_data_type** – tipul de date întors de funcție , **se specifica fără precizie**
- **function_declaration_section** – secțiunea de declarare a variabilelor locale folosite în funcție, poate să lipsească
- **function_executable_section** – secțiunea executabilă a funcției, trebuie să conțină cel puțin un **RETURN** care să întoarcă o variabilă care are tipul de date **return_data_type**
- **function_exception_section** – secțiune de tratare a excepțiilor din cadrul funcției, poate să lipsească, în schimb, dacă se folosește trebuie să conțină câte un **RETURN** pentru fiecare excepție tratată



Funcții declarate în cadrul unui bloc

- **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 funcției
- **var** – numele variabile în care funcția returnează o valoare, are tipul **return_data_type**



Funcții declarate în cadrul unui bloc

- Ex. 1. Să se scrie o funcție locală care primește ca parametru un identificador de departament și returnează numărul salariaților din departamentul respectiv.

```
set serveroutput on;
```

```
declare
```

```
function f_salariati(p_idDept in number) return number  
as
```

```
    nrAng number;
```

```
begin
```

```
    select count(distinct empno) into nrAng  
    from emp where deptno = p_idDept;
```

```
    return nrAng;
```

```
end;
```



Funcții declarate în cadrul unui bloc

```
begin
  declare
    numeDept dept.dname%type;
    idDept dept.deptno%type;
    nrAng number;
  begin
    idDept := &idDept;
    select dname into numeDept from dept where deptno = idDept;
    nrAng := f_salariati(idDept);
    dbms_output.put_line('Departamentul ' || numeDept
      || ' are ' || nrAng || ' salariati');
  exception
    when no_data_found then
      dbms_output.put_line('Departament inexistent');
  end;
end;
```



Funcții stocate pe server

- Funcțiile stocate sunt funcții create în dicționarul bazei de date și pot fi accesate ca orice obiect, dacă utilizatorul are suficiente privilegii.
- Sintaxa

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



Funcții stocate pe server

- **function_name** – numele funcției
- **parameter_name** – numele unui parametru formal din lista de parametri
- **parameter_type** – reprezintă tipul parametrului formal, **se specifica fără precizie**
- **return_data_type** – tipul de date întors de funcție , **se specifica fără precizie**
- **declaration_section** – secțiunea de declarare a variabilelor locale folosite în funcție
- **executable_section** – secțiunea executabilă a funcției, trebuie să conțină cel puțin un **RETURN** care să întoarcă o variabilă care are tipul de date **return_data_type**
- **exception_section** – secțiune de tratare a excepțiilor din cadrul funcției, poate să lipsească



Funcții stocate pe server

- **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)



Funcții stocate pe server

- Ex. 2. Să se scrie o funcție stocată, **f_puncte**, care calculează numărul de puncte acumulate de angajați în vederea numirii unui șef pe fiecare departament. Criteriile de punctaj și punctele aferente sunt următoarele:

A) vechimea în firmă

- Ani vechime $> 32 \Rightarrow 30$ puncte
- Ani vechime $\leq 32 \Rightarrow 15$ puncte

B) salariul maxim pe departament

- Salariu angajat = salariu maxim din departament $\Rightarrow 20$ puncte
- Salariu angajat $<$ salariu maxim din departament $\Rightarrow 10$ puncte

C) comision

- Comision angajat $> 0 \Rightarrow 10$ puncte
- Comision angajat $= 0 \Rightarrow 5$ puncte

Pentru stocarea punctelor acumulate de fiecare angajat, se va crea o tabelă **temp_puncte** care are următoarele coloane: idAng, nrPuncte, idDept.



Funcții stocate pe server

- Pasul I – crearea tabelii

```
create table temp_puncte  
(  
    idAng number(4),  
    nrPuncte number(3),  
    idDept number(2)  
);
```



Funcții stocate pe server

- Pasul II – crearea funcției stocate

```
create or replace function f_puncte(p_idAng in number)  
return number  
is  
    dataAng emp.hiredate%type;  
    idDept dept.deptno%type;  
    salariu emp.sal%type;  
    comision emp.comm%type;  
    salMax number;  
    puncte number := 0;  
begin  
    -- iau informatiile de care am nevoie  
    select hiredate, deptno, sal, nvl(comm, 0)  
        into dataAng, idDept, salariu, comision  
        from emp where empno = p_idAng;
```




Funcții stocate pe server

```
-- calculez puncte in functie de vechime
if months_between(sysdate, dataAng) > 32*12 then
    puncte := puncte + 30;
else
    puncte := puncte + 15;
end if;

-- iau salariul maxim din departament
select max(sal) into salMax from emp where deptno = idDept;
-- calculez puncte in functie de salariu
if salMax = salariu then
    puncte := puncte + 20;
else
    puncte := puncte + 10;
end if;
```



Funcții stocate pe server

```
-- calculez puncte in functie de comision
if comision > 0 then
    puncte := puncte + 10;
else
    puncte := puncte + 5;
end if;

return puncte;
end f_puncte;
```



Funcții stocate pe server

- Pasul III – crearea blocului apelant

```
set serveroutput on;
declare
    puncte number;
    cursor c_angajati is select empno, ename, deptno from emp;
    numeDept dept.dname%type;
    idMgr emp.mgr%type;
    numeAng emp.ename%type;
    dataAng emp.hiredate%type;
    aniVechime number;
    salMaxDept number;
    comision emp.comm%type;
begin
    -- sterg informatiile din tabelul
    delete from temp_puncte;
    -- calculez punctajul pentru fiecare angajat si
    -- inserez informatiile in tabelul temp_puncte
    for angajat in c_angajati
    loop
        puncte := f_puncte(angajat.empno);
        insert into temp_puncte values(angajat.empno, puncte, angajat.deptno);
    end loop;
```



Funcții stocate pe server

```
-- antetul listei
dbms_output.put_line(rpad('Departament', 20)||rpad('Salariu maxim', 20)
  ||rpad('Nume angajat', 20)||rpad('Ani vechime', 20)
  ||rpad('Comision', 20)||rpad('Puncte angajat', 20));
dbms_output.put_line(rpad('=', 120, '='));
```



Funcții stocate pe server

```
-- folosind un cursor selectez maximul de puncte pentru fiecare departament
for contor1 in (select max(nrPuncte) maxPuncte, idDept
                from temp_puncte group by idDept order by iddept)
loop
  -- folosind un cursor selectez toti angajatii care sunt din departamentul
  -- selectat de primul cursor si au maximul de puncte din departament
  for contor2 in (select idAng from temp_puncte
                  where nrPuncte = contor1.maxPuncte and idDept = contor1.idDept )
  loop
    -- selectez numele departamentului
    select dname into numeDept from dept where deptno = contor1.idDept;
    -- selectez salariul maxim din departament
    select max(sal) into salMaxDept from emp where deptno = contor1.idDept;
    -- selectez informatiile despre angajat
    select hiredate, nvl(comm, 0), ename into dataAng, comision, numeAng
    from emp where empno = contor2.idAng;
    -- calculez vechimea angajatului
    aniVechime := trunc(months_between(sysdate, dataAng)/12);
    -- afisez informatiile cerute
    dbms_output.put_line(rpad(numeDept, 20)||rpad(salMaxDept, 20)
                          ||rpad(numeAng, 20)||rpad(aniVechime, 20)
                          ||rpad(comision,20)||rpad(contor1.maxPuncte, 20));
  end loop;
end loop;
end;
```



Funcții stocate pe server

Observații:

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



Apelul funcțiilor din comenzi SQL

- Odată create și stocate în baza de date, funcțiile pot fi apelate în comenzi SQL, dar cu anumite restricții.
- Funcția **f_puncte** creată anterior, care returnează numărul de puncte obținute de un angajat în vederea întocmirii unui clasament pe departamente, poate fi folosită astfel:

```
select d.dname, e.ename, f_puncte(e.empno) punctaj
  from emp e inner join dept d on d.deptno = e.deptno
 where d.deptno = 10;
```

```
select d.dname, e.ename, f_puncte(e.empno) punctaj
  from emp e inner join dept d on d.deptno = e.deptno
 and f_puncte(e.empno) >= 40
 order by d.dname, punctaj desc;
```



Apelul funcțiilor din comenzi SQL

```
select d.dname Departament
      , (select max(sal) from emp where deptno = e.deptno) Salariu_Maxim
      , e.ename NumeAngajat
      , trunc(months_between(sysdate, e.hiredate)/12) Ani_Vechime
      , nvl(e.comm, 0) Comision
      , f_puncte(e.empno) Punctaj
from emp e inner join dept d on d.deptno = e.deptno
where f_puncte(e.empno) =
      (select max(f_puncte(empno)) from emp where deptno = e.deptno)
order by d.dname;
```




Apelul funcțiilor din comenzi SQL

- Informațiile obținute din ultimul select sunt echivalente cu cele obținute prin programul PL/SQL prezentat în exercițul 3;
- Se observă că dacă utilizatorul are cunoștințe de SQL, acesta poate să creeze fără dificultăți liste simple;
- Comanda se bazează pe un cursor implicit creat de SGBD;
- Faptul că o comandă SQL poate să facă același lucru ca un bloc PL/SQL nu înseamnă că este mai ușor să scrii comenzi SQL decât blocuri PL/SQL;
- Totuși, în dezvoltarea aplicațiilor, un dezvoltator va încerca întotdeauna, în momentul construirii unui bloc PL/SQL, să minimizeze numărul de cereri SQL, încercând să obțină toate informațiile necesare folosind cât mai puține cereri SQL.



Considerente asupra procedurilor și funcțiilor

- În general, o procedură este folosită pentru secvențe de cod mai complexe, în timp ce o funcție este folosită în cadrul procedurii pentru calcule repetate și mai simple.
- De exemplu, într-o aplicație de salarizare:
 - O procedură se pretează pentru implementarea algoritmului de calcul al salariului net, considerând salariul de bază, pontajele lunare, sporurile, creșterile salariale, etc.;
 - O funcție se pretează pentru calculul taxelor și impozitelor aferente;



Considerente asupra procedurilor și funcțiilor

- Din punct de vedere tehnic, sunt câteva mici diferențe între proceduri și funcții:
 - O funcție întotdeauna în mod direct o valoare în programul apelant și poate întoarce mai multe valori prin parametrii de tip OUT sau IN OUT;
 - O procedură nu returnează direct o valoare, dar poate întoarce mai multe valori în programul apelant prin parametrii de tip OUT sau IN OUT;
 - Funcțiile stocate pot fi apelate direct din comenzi SQL;
 - Procedurile stocate **NU** pot fi apelate direct din comenzi SQL;
 - Funcțiile pot fi folosite în operațiile de atribuire, în expresii și condiții (adică, atât în clauza WHERE cât și în condițiile structurilor de control);
 - Procedurile **NU** pot fi folosite în operațiile de atribuire, în expresii și condiții (adică, atât în clauza WHERE cât și în condițiile structurilor de control);
- Procedurile și funcțiile pot fi apelate direct din programe PL/SQL, sau din interfețe specifice, de exemplu *Oracle Forms Builder* sau *Oracle Reports Builder*
- Procedurile și funcțiile pot fi apelate și din limbaje de programare de nivel înalt (Java, C++, C#, Python, etc) folosindu-se API-uri dedicate.



Considerente asupra procedurilor și funcțiilor

- Funcțiile apelate din cereri SQL trebuie să îndeplinească următoarele condiții:
 - Când este folosită într-o cerere SELECT, funcția **NU** trebuie să conțină comenzi INSERT, UPDATE sau DELETE;
 - Când este folosită în comenzile INSERT, UPDATE sau DELETE, funcția **NU** trebuie să modifice conținutul tabelelor afectate de comanda SQL;
 - Când este folosită în comenzi ca SELECT, INSERT, UPDATE sau DELETE , funcția **NU** trebuie să conțină comenzi TCL (COMMIT sau ROLLBACK).



Considerente asupra procedurilor și funcțiilor

- Dacă într-o procedură sau funcție apare o excepție, tratată de sistem sau definită de utilizator, controlul este preluat de secțiunea EXCEPTION sau de blocul superior, iar parametrilor de tip OUT sau IN OUT nu li se transmite nicio valoare;
- În cazul în care apar excepții, parametrii actuali din procedura sau funcția apelată vor păstra aceleași valori pe care le-au avut înainte de a se face apelul;
- Execuția comenzilor CREATE OR REPLACE sau DROP pentru proceduri și funcții, precum și a comenzilor ALTER TABLE sau ALTER VIEW pentru tabele și view-uri, pot modifica starea altor obiecte din baza de date cu care sunt relaționate. (De exemplu dacă într-o procedură este declarat un cursor care lucrează pe o anumită tabelă, atunci ștergerea sau modificarea structurii tabelii poate invalida procedura care apelează cursorul respectiv).



Considerente asupra procedurilor și funcțiilor

- Probabilitatea apariției acestor efecte, ca urmare a relaționării obiectelor, este ridicată și de aceea serverul Oracle le tratează corespunzător în timpul execuției sau apelării obiectelor corespunzătoare;
- La apariția unei astfel de situații serverul modifică în dicționarul bazei de date starea obiectelor afectate ca DISABLE și este necesară o recompilare a lor pentru a reveni în starea ENABLE.



Considerente asupra procedurilor și funcțiilor

- Sistemul de gestionare ORACLE permite acordarea privilegiului de execuție asupra procedurilor și funcțiilor și altor utilizatori, acțiunea permisă proprietarului lor (utilizatorul care le-a creat) sau administratorului care are privilegii de DBA;

- Sintaxa comenzii este:

```
GRANT EXECUTE ON {procedure_name | function_name} TO  
user_name;
```

- Ștergerea privilegiilor se face folosind comanda REVOKE, sintaxa este:

```
REVOKE EXECUTE ON {procedure_name | function_name} FROM  
user_name;
```



Informații din dicționarul bazei de date

- Codul sursă al procedurilor și funcțiilor stocate se găsește în dicționarul bazei de date, în view-ul USER_SOURCE, care are următoarea structură

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS	INSERTABLE	UPDATABLE	DELETABLE
NAME	VARCHAR2(30)	Yes	(null)	1	Name of the object	NO	NO	NO
TYPE	VARCHAR2(12)	Yes	(null)	2	Type of the object: "TYPE", "TYP...	NO	NO	NO
LINE	NUMBER	Yes	(null)	3	Line number of this line of source	NO	NO	NO
TEXT	VARCHAR2(4000)	Yes	(null)	4	Source text	NO	NO	NO

- name – este numele obiectului (procedură, funcție, etc)
- type – reprezintă tipul obiectului
- line – este numărul liniei de cod
- text – este codul sursă de la linia respectivă



Informații din dicționarul bazei de date

- Dacă vrem să vedem toate liniile de cod ale funcției **f_puncte**, putem folosi următoarea cerere SQL:

```
select text from user_source where lower(name) = 'f_puncte' order by line;
```



Informații din dicționarul bazei de date

```
TEXT
-----
function f_puncte(p_idAng in number)
return number
is
  dataAng emp.hiredate%type;
  idDept dept.deptno%type;
  salariu emp.sal%type;
  comision emp.comm%type;
  salMax number;
  puncte number := 0;
begin
  -- iau informatiile de care am nevoie
  select hiredate, deptno, sal, nvl(comm, 0)
  into dataAng, idDept, salariu, comision
  from emp where empno = p_idAng;

  -- calculez puncte in functie de vechime
  if months_between(sysdate, dataAng) > 32*12 then
    puncte := puncte + 30;
  else
    puncte := puncte + 15;
  end if;

  -- iau salariul maxim din departament
  select max(sal) into salMax from emp where deptno = idDept;
  -- calculez puncte in functie de salariu
  if salMax = salariu then
    puncte := puncte + 20;
  else
    puncte := puncte + 10;
  end if;

  -- calculez puncte in functie de comision
  if comision > 0 then
    puncte := puncte + 10;
  else
    puncte := puncte + 5;
  end if;

  return puncte;
end f_puncte;

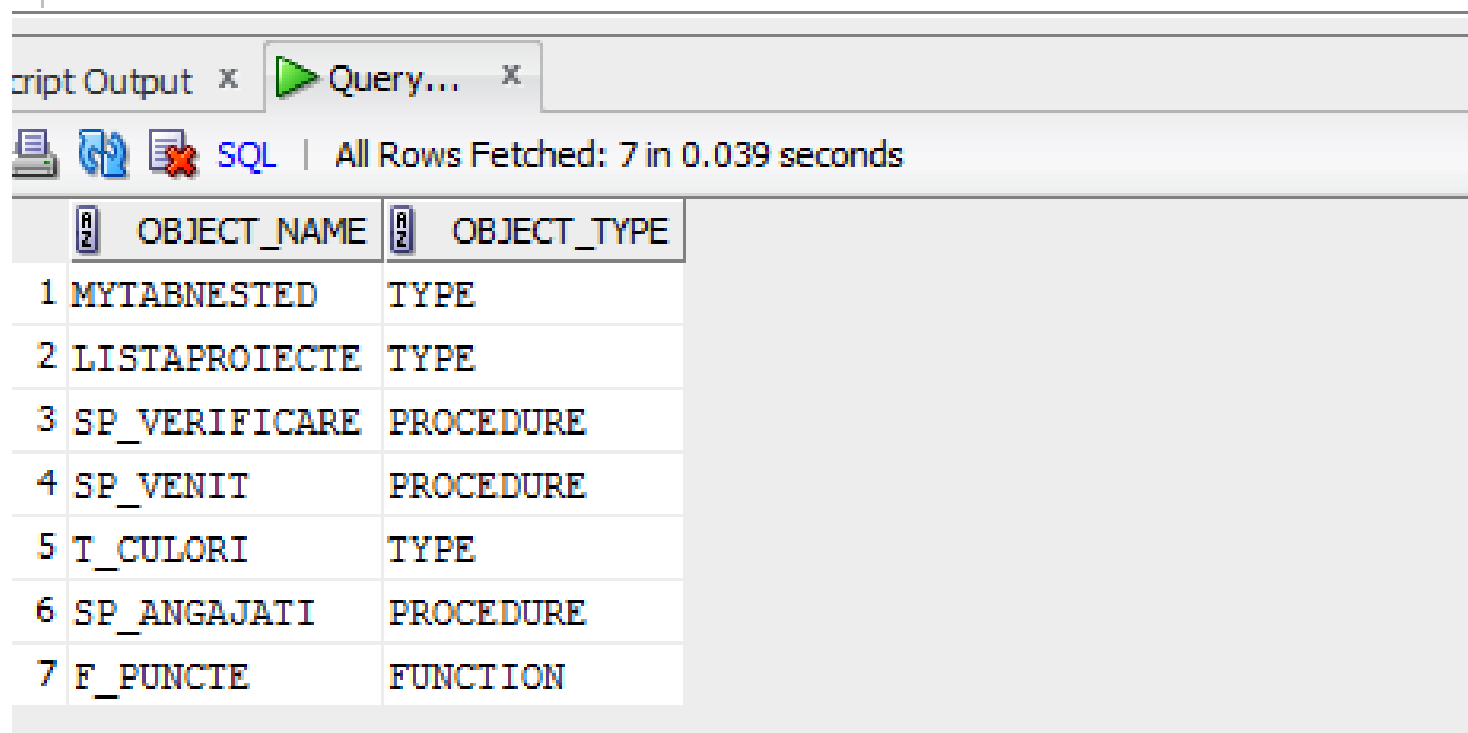
42 rows selected.
```



Informații din dicționarul bazei de date

- Pentru a vizualiza toate procedurile, funcțiile și tipurile de date create de utilizatorul curent, se poate folosi următoarea cerere SQL:

```
select object_name, object_type from user_objects  
where lower(object_type) in ('function', 'procedure', 'type');
```



The screenshot shows a SQL query execution window with a tab labeled "Query...". Below the tab, there are icons for a document, a hand, and a red X, followed by the text "SQL | All Rows Fetched: 7 in 0.039 seconds". The results are displayed in a table with two columns: "OBJECT_NAME" and "OBJECT_TYPE". The table contains 7 rows of data.

	OBJECT_NAME	OBJECT_TYPE
1	MYTABNESTED	TYPE
2	LISTAPROIECTE	TYPE
3	SP_VERIFICARE	PROCEDURE
4	SP_VENIT	PROCEDURE
5	T_CULORI	TYPE
6	SP_ANGAJATI	PROCEDURE
7	F_PUNCTE	FUNCTION



Informații din dicționarul bazei de date

- În SQL*Plus

```
SQL> col object_name for a30;  
SQL>  
SQL> select object_name, object_type from user_objects  
2      where lower(object_type) in ('function', 'procedure', 'type');  
  
OBJECT_NAME                OBJECT_TYPE  
-----  
F_PUNCTE                   FUNCTION  
SP_ANGAJATI                PROCEDURE  
T_CULORI                   TYPE  
SP_VENIT                   PROCEDURE  
SP_VERIFICARE              PROCEDURE  
LISTA PROIECTE             TYPE  
MYTABNESTED               TYPE  
  
7 rows selected.
```



Exercițiu

- Ex. 3. Să se scrie o funcție care calculează impozitul pe venit în funcție de gradul de salarizare (gradul se găsește în tabela SALGRADE), folosindu-se următorul algoritm:
 - Dacă grade = 1 atunci impozitul este 10% din venit
 - Dacă grade = 2 atunci impozitul este 15% din venit
 - Dacă grade = 3 atunci impozitul este 20% din venit
 - Dacă grade = 4 atunci impozitul este 25% din venit
 - Dacă grade = 5 atunci impozitul este 30% din venit
- Să se afișeze numele angajatului, dacă este șef de departament sau nu, gradul de salarizare, procentul, venitul și impozitul.



Exercițiu

- Pasul I – crearea funcției

```
create or replace
function f_impozit(p_idAngajat in emp.empno%type,
    p_grad OUT salgrade.grade%type, p_procent out number,
    p_venit out number, p_sef out varchar2)
return number
is
    isSef number;
    impozit number;
begin
    select sal+nvl(comm, 0) into p_venit from emp where empno = p_idAngajat;
    select grade into p_grad from salgrade
        where p_venit>= losal and p_venit<=hisal;
```



Exercițiu

```
if p_grad = 1 then p_procent := 10;
elsif p_grad = 2 then p_procent := 15;
elsif p_grad = 3 then p_procent := 20;
elsif p_grad = 4 then p_procent := 25;
else p_procent := 30;
end if;
select count(empno) into isSef from emp where mgr = p_idAngajat;
if isSef > 0 then p_sef := 'DA';
else p_sef := 'NU';
end if;
impozit := round(p_venit*p_procent/100);
return impozit;
end f_impozit;
```



Exercițiu

- Pasul II – crearea blocului apelant

```
set serveroutput on;
declare
    impozit number;
    grad number;
    procent number;
    venit number;
    sef varchar(2);
begin
    dbms_output.put_line(rpad('Nume',10)||rpad('Sef',10)
        ||rpad('Grad salarizare',20)||rpad('Procent', 10)||rpad('Venit',10)
        ||rpad('Impozit',10));
    dbms_output.put_line(rpad('=',70,'='));
    for contor in (select empno, ename from emp order by ename)
    loop
        impozit := f_impozit(contor.empno, grad, procent, venit, sef);
        dbms_output.put_line(rpad(contor.ename,10)||rpad(sef,10)
            ||rpad(grad,20)||rpad(procent, 10)||rpad(venit,10)
            ||rpad(impozit,10));
    end loop;
end;
```




Exercițiu

- Exercițiu rezolvat folosind o singură interogare

```
set serveroutput on;
declare
    impozit number;
    grad number;
    procent number;
    venit number;
    sef varchar(2);
begin
    dbms_output.put_line(rpad('Nume',10)||rpad('Sef',10)
        ||rpad('Grad salarizare',20)||rpad('Procent', 10)||rpad('Venit',10)
        ||rpad('Impozit',10));
    dbms_output.put_line(rpad('=',70,'='));
    for contor in (select empno, ename from emp order by ename)
    loop
```



Exercițiu

```
select
  (select grade from salgrade where e.sal+nvl(e.comm,0) >= losal
    and e.sal+nvl(e.comm,0) <= hisal) grad
, (case (select count(empno) from emp where mgr = e.empno)
  when 0 then 'NU' else 'DA' end) sef
, (case (select grade from salgrade
  where e.sal+nvl(e.comm,0) >= losal and e.sal+nvl(e.comm,0) <= hisal)
  when 1 then 10
  when 2 then 15
  when 3 then 20
  when 4 then 25
  else 30 end) procent
, e.sal+nvl(e.comm, 0) venit
, round(((case (select grade from salgrade
  where e.sal+nvl(e.comm,0) >= losal and e.sal+nvl(e.comm,0) <= hisal)
  when 1 then 0.1
  when 2 then 0.15
  when 3 then 0.2
  when 4 then 0.25
  else 0.3 end) * (e.sal+nvl(e.comm, 0)))) impozit
into grad, sef, procent, venit, impozit
from emp e where e.empno = contor.empno;
```



Exercițiu

```
dbms_output.put_line(rpad(contor.ename,10)||rpad(sef,10)
    ||rpad(grad,20)||rpad(procent, 10)||rpad(venit,10)
    ||rpad(impozit,10));
end loop;
end;
```