Tehnici de programare cu baze de date

#10 PL/SQL
Functii în PL/SQL

Adrian Runceanu
www.runceanu.ro/adrian

2022

Curs 10

Funcţii în PL/SQL

Cuprins

Funcţii în PL/SQL

1. Crearea funcţiilor

O ***functie*** este *un subprogram care returneaza exact o valoare*.

O ***procedura*** este *o instructiune executabila de sine statatoare, in timp ce o functie poate exista doar ca o parte a unei instructiuni executabile*.

- O functie este un bloc PL/SQL cu nume care accepta parametri IN optional si trebuie sa returneze o singura valoare.
- Functiile sunt stocate in baza de date ca obiecte ale schemei pentru executii repetate.
- O functie poate fi apelata ca parte a unei expresii SQL sau a unei expresii PL/SQL.

1. Crearea funcţiilor

1. In expresiile SQL o functie trebuie sa indeplineasca reguli specifice pentru a controla efectele secundare.

- Efectele secundare care trebuie evitate sunt:
  - Orice tip de DML sau DDL
  - COMMIT sau ROLLBACK
  - Afectarea variabilelor globale

2. In expresiile PL/SQL identificatorul de functie se comporta ca o variabila a carei valoare depinde de parametrii care ii sunt transmisi.

1. Crearea funcţiilor

Sintaxa pentru crearea functiilor

Antetul unei functii este asemanator cu cel al unei proceduri, cu doua diferente:

- Tipul parametrilor trebuie sa fie doar IN
- Clauza RETURN este folosita in locul parametrului de tip OUT

1. Crearea funcţiilor

```
CREATE [OR REPLACE] FUNCTION
   function_name
[(parameter1 [mode1] datatype1, ...)]
RETURN datatype IS|AS
[local_variable_declarations; …]
BEGIN
        -- actions;
        RETURN expression;
END [function_name];
```

Blocul PL/SQL trebuie sa aibă cel putin o instructiune RETURN.

1. Crearea funcţiilor

- ***Observatii:***
  - O functie este un subprogram PL/SQL care returneaza o singura valoare.
  - Trebuie sa folositi o instructiune RETURN pentru a rezulta o valoare cu un tip de date care este continut in declararea tipului functiei
  - Puteti crea functii noi cu ajutorul instructiunii CREATE [OR REPLACE] FUNCTION care poate avea declarata o lista de parametri, trebuie sa returneze exact o valoare si trebuie sa defineasca operatiile care se executa in blocul PL/SQL.

Exemple:

1)

- Crearea functiei

```
CREATE OR REPLACE FUNCTION get_sal
(p_id emp.empno%TYPE)
RETURN NUMBER IS
   v_sal emp.sal%TYPE := 0;
BEGIN
   SELECT sal INTO v_sal
   FROM emp
   WHERE empno = p_id;
   RETURN v_sal;
END get_sal;
```

- Apelarea functiei

```
... v_sal := get_sal(100);
```

Exemple:
1)
- Crearea functiei

GET_SAL

Code Dependencies Errors Grants

Save & Compile | Find & Replace | Undo | Redo | Download Source | Drop

```
1 create or replace FUNCTION get_sal (p_id emp.empno%TYPE)
2 RETURN NUMBER IS
3     v_sal emp.sal%TYPE := 0;
4 BEGIN
5     SELECT sal INTO v_sal
6     FROM emp
7     WHERE empno = p_id;
8     RETURN v_sal;
9 END get_sal;
```

2) Se poate folosi RETURN din sectiunea executabila si/sau sectiunea EXCEPTION

- Crearea functiei

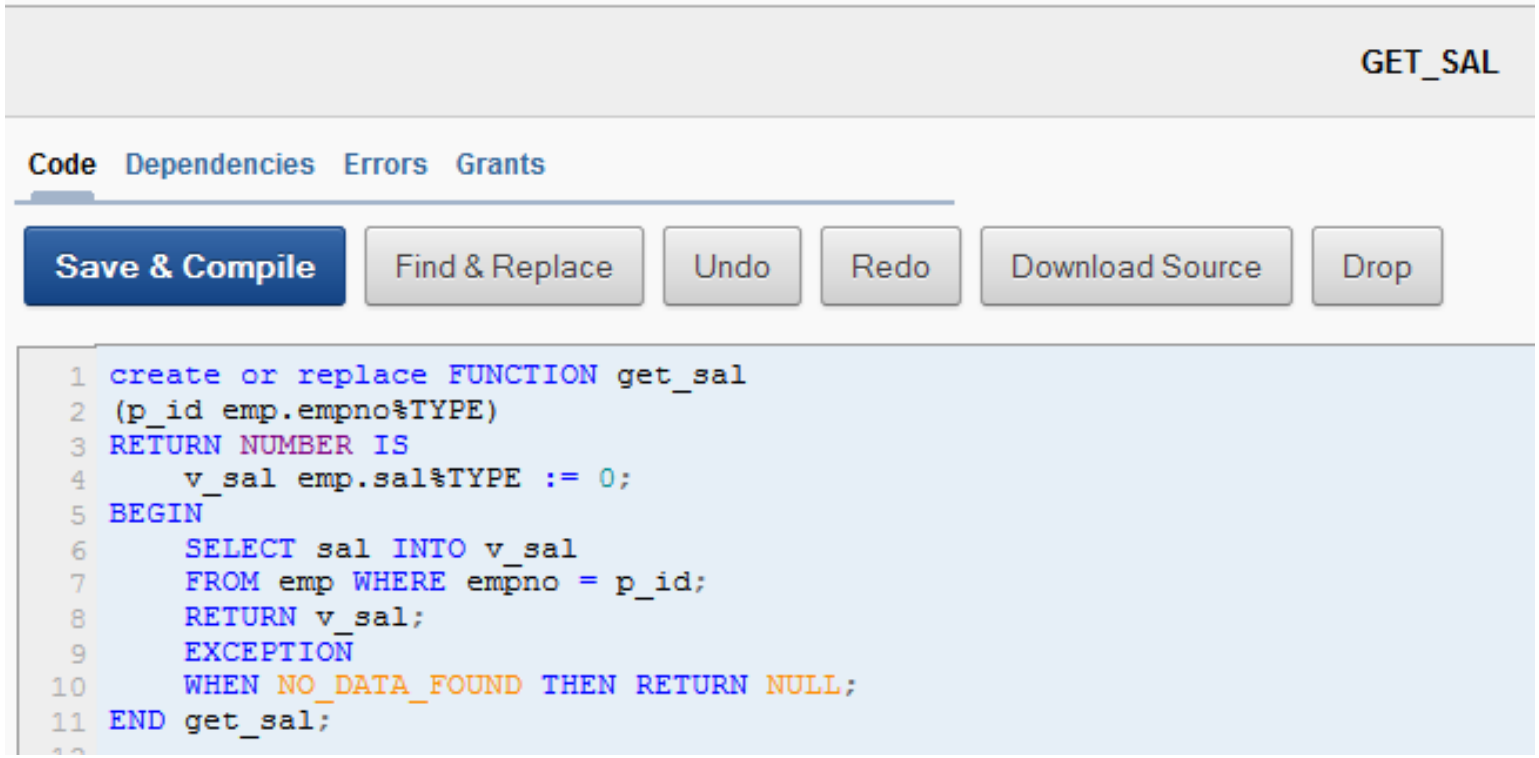
```
CREATE OR REPLACE FUNCTION get_sal
(p_id emp.empno%TYPE)
RETURN NUMBER IS
   v_sal emp.sal%TYPE := 0;
BEGIN
   SELECT sal INTO v_sal
   FROM emp WHERE empno = p_id;
   RETURN v_sal;
   EXCEPTION
   WHEN NO_DATA_FOUND THEN RETURN NULL;
END get_sal;
```

- Apelarea functiei ca o expresie cu un parametru care nu este bun

```
... v_sal := get_sal(999);
```

2) Se poate folosi RETURN din sectiunea executabila si/sau sectiunea EXCEPTION

- Crearea functiei



- Apelarea functiei ca o expresie cu un parametru care nu este bun

... v_sal := get_sal(999);

1. Crearea funcţiilor

Modalitati de apelare (sau executare) a functiilor cu parametri

Functiile pot fi apelate in felul urmator:

1. Ca parte a expresiilor PL/SQL – se foloseste o variabila locala intr-un bloc anonim pentru a pastra valoarea returnata de functie

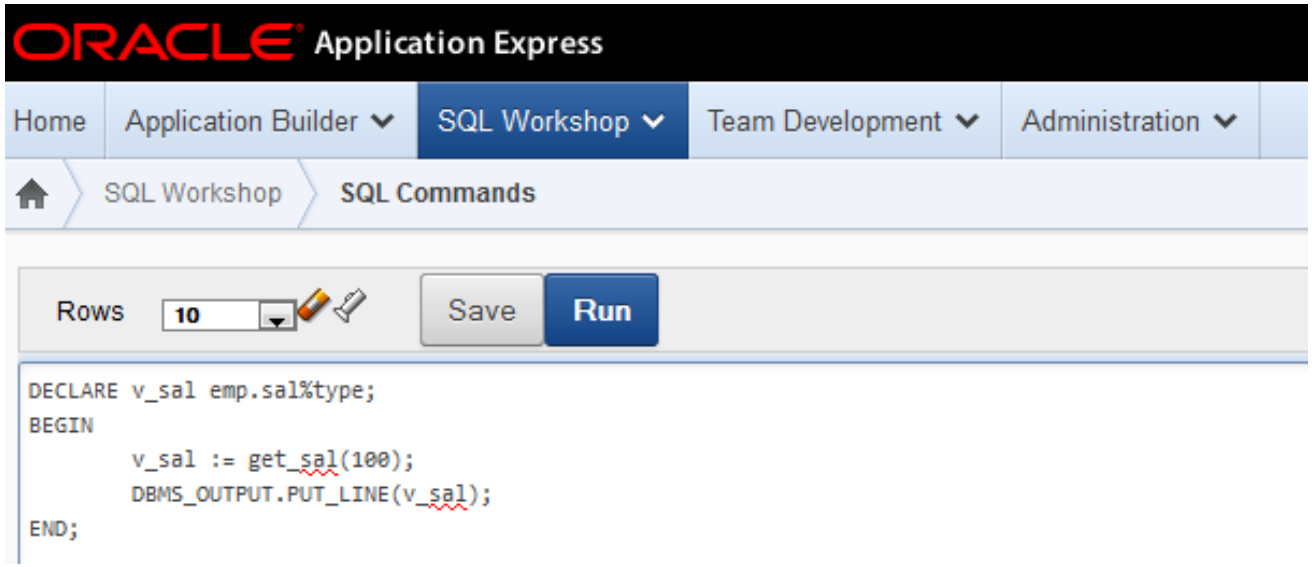
2. Ca parametru al unui alt subprogram – transmiterea functiilor intre subprograme

3. Ca o expresie intr-o instructiune SQL – apelarea unei functii ca pe oricare functie single-row intr-o instructiune SQL

1. Crearea funcţiilor

- *1. Apelarea ca o parte a unei expresii PL/SQL cu folosirea unei variabile locale pentru a stoca rezultatul obtinut*:

```
DECLARE v_sal emp.sal%type;
BEGIN
   v_sal := get_sal(100);
   ...
END;
```



1. Crearea funcţiilor

2. Folosirea ca un parametru al altui subprogram

... DBMS_OUTPUT.PUT_LINE(get_sal(100));

3. Ca o expresie intr-o instructiune SQL

```
SELECT job_id, get_sal(empno)
FROM emp;
```

1. Crearea funcţiilor

Apelarea functiilor fara parametri

Majoritatea functiilor au parametri, dar nu toate.

USER si SYSDATE sunt functii sistem fara parametri.

1. Apelarea ca parte a unei expresii PL/SQL folosind o variabila locala pentru a obtine rezultatul:

```
DECLARE v_today DATE;
BEGIN
  v_today := SYSDATE;
  ...
END;
```

1. Crearea funcţiilor

2. Folosirea ca parametru al altui subprogram

```
...
DBMS_OUTPUT.PUT_LINE(USER);
```

3. Folosirea intr-o instructiune SQL

```
SELECT job_id, SYSDATE-hiredate
FROM emp;
```

1. Crearea funcţiilor

Avantajele si restrictiile functiilor

Avantaje	Restrictii
Functiile ne permit sa afisam temporar o valoare intr-un format nou	*Tipurile de date PL/SQL nu se suprapun complet cu cele SQL*
Permit noi caracteristici cum ar fi verificarea si analiza datelor	*Dimensiunile PL/SQL nu sunt aceleasi cu dimensiunile SQL.* De exemplu o variabila VARCHAR2. PL/SQL poate ajunge pana la 32KB, iar o coloana SQL de tip VARCHAR2 poate fi de maxim 4 KB.

1. Crearea funcţiilor

Diferente de sintaxa intre proceduri si functii

Proceduri:

CREATE [OR REPLACE] PROCEDURE name
[parameters] IS|AS (Mandatory)
Variables, cursors, etc. (Optional)
BEGIN (Mandatory)
SQL and PL/SQL statements;
EXCEPTION (Optional)
WHEN exception-handling actions;
END [name]; (Mandatory)

1. Crearea funcţiilor

Functii

CREATE [OR REPLACE] FUNCTION name
[parameters] (Mandatory)
RETURN datatype IS|AS (Mandatory)
Variables, cursors, etc. (Optional)
BEGIN (Mandatory)
SQL and PL/SQL statements;
RETURN ...; (One Mandatory, more optional)
EXCEPTION (Optional)
WHEN exception-handling actions;
END [name]; (Mandatory)

1. Crearea funcţiilor

Proceduri	Functii
Se executa ca o instructiune PL/SQL	Se apeleaza ca parte a unei expresii
Nu contine clauza RETURN in antet	Trebuie sa contina clauza RETURN in antet
Poate returna valori in parametrii de iesire	Trebuie sa returneze o singura valoare
Poate contine o instructiune RETURN fara o valoare	Trebuie sa contina o instructiune RETURN

- Ambele pot avea 0 sau mai multi parametri IN care pot fi transmisi de la mediul apelant.
- Ambele au o structura de bloc standard ce include o sectiune de manipulare a exceptiilor.

Caracteristici proceduri

- Puteti crea o procedura pentru a retine o serie de operatii pentru o executie ulterioara.
- O procedura nu trebuie neaparat sa returneze o valoare.
- O procedura poate apela o functie.
- O procedura care contine un singur parametru OUT poate fi rescrisa ca o functie care returneaza o valoare.

Caracteristici functii

- Functiile se pot folosi atunci cand dorim sa transmitem o singura valoare mediului apelant.
- Valoarea este returnata cu ajutorul instructiunii RETURN.
- Functiile utilizate in instructiunile SQL nu pot folosi parametri de tip OUT si IN OUT.
- Totusi, o functie care are parametri de tip OUT poate fi apelata de o procedura PL/SQL sau de un bloc anonim.

Cuprins

Funcţii în PL/SQL

1. Crearea funcţiilor
2. Funcţii definite de utilizator
3. Modificarea şi suprimarea subprogramelor PL/SQL
4. Module overload

2. Funcţii definite de utilizator

Functii definite de utilizator

- *O functie definita de utilizator este o functie care este creata de catre programator.*
- GET_DEPT_NAME si CALCULATE_TAX sunt exemple de functii definite de utilizator, in timp ce UPPER, LOWER si LPAD sunt exemple de functii sistem furnizate automat de Oracle.
- Majoritatea functiilor sistem (cum ar fi UPPER, LOWER si LPAD) sunt stocate in pachete denumite SYS.STANDARD.
- Aceste functii sistem se mai numesc **functii built-in**.

2. Funcţii definite de utilizator

Avantajele functiilor in instructiunile SQL

- Prin folosirea functiilor in clauza WHERE a instructiunii SELECT creste eficienta prin eliminarea randurilor nedorite inainte ca datele sa fie transmise aplicatiei.
- De exemplu, in sistemul administrativ al unei facultati, doriti sa extrageti doar acei studenti ale caror nume sunt stocate in intregime cu majuscule. Acest lucru poate fi gasit pentru putine randuri ale tabelei.

2. Funcţii definite de utilizator

- Instructiunea folosita este:

```
SELECT * FROM students
  WHERE student_name = UPPER(student_name);
```

- Fara functia UPPER este nevoie sa extrageti toate randurile referitoare la studenti, sa le transmiteti in retea si sa le eliminati pe cele nedorite prin aplicatie.

- In instructiunile SQL functiile pot manipula valorile de tip data calendaristica.
- De exemplu, pentru un eveniment social de la sfarsitul semestrului, vreti sa afisati (doar ca amuzament) numele fiecarui profesor cu caracterele inversate – de exemplu “Adrian Runceanu” devine “unaecnuR nairdA”.

- Puteti crea o functie definita de utilizator numita REVERSE_NAME care face acest lucru, iar apoi codul:

```
SELECT name, reverse_name(name) FROM teachers;
```

- Functiile definite de utilizator pot extinde SQL acolo unde operatiile sunt prea complexe, prea incomode sau nu se pot realiza in mod obisnuit cu SQL.
- De asemenea, functiile ne ajuta sa depasim in mod repetat scrierea aceluiasi cod.
- De exemplu, daca vreti sa calculati cat timp a muncit un angajat pentru un proiect rotunjit la un numar intreg de luni, puteti crea o functie definita de utilizator numita HOW_MANY_MONTHS pentru a face acest lucru.
- Apoi se poate folosi instructiunea:

```
SELECT employee_id, how_many_months(hire_date)
  FROM employees;
```

Functii in SQL

Functii in expresii SQL – Exemple

```
CREATE OR REPLACE FUNCTION
   tax(p_value IN NUMBER)
RETURN NUMBER IS
BEGIN
   RETURN (p_value * 0.08);
END tax;
```

EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
124	Mourgos	5800	464
141	Rajs	3500	280
142	Davies	3100	248
143	Matos	2600	208
144	Vargas	2500	200

Functii create

```
SELECT empno, ename, sal, tax(sal)
FROM emp
WHERE deptno = 50;
```

Unde puteti folosi functii definite de utilizator in instructiunile SQL?

Functiile definite de utilizator se comporta ca si functiile single-row de tip **built-in** – cum ar fi **UPPER**, **LOWER**, **LPAD**.

Ele pot fi folosite in:

1. Lista de coloane a unei interogari SELECT
2. Expresii conditionale in clauzele WHERE si HAVING
3. Clauzele ORDER BY si GROUP BY
4. Clauza VALUES din instructiunea INSERT
5. Clauza SET din instructiunea UPDATE

Adica, *pot fi folosite oriunde putem scrie o valoare a unei expresii*.

Exemplu care arata functia definita de utilizator cu numele TAX, care poate fi folosita in patru locuri intr-o singura instructiune SELECT:

```
SELECT empno, tax(sal)
FROM emp
WHERE tax(sal) > (SELECT
                  MAX(tax(sal))
                  FROM emp
                  WHERE department_id = 20)
ORDER BY tax(sal) DESC;
```

Restrictii la utilizarea functiilor in instructiunilor SQL

Pentru a folosi o functie definite de utilizator intr-o instructiune SQL, functia trebuie sa respecte regulile si restrictiile limbajului SQL:

1. Functia poate accepta doar tipurile de date valide ale SQL-ului in parametrii IN si trebuie sa returneze un tip de date valid al SQL-ului.
2. Tipurile specifice SQL, cum ar tipurile BOOLEAN si %ROWTYPE nu sunt acceptate
3. Limitele impuse de catre SQL nu trebuie sa fie depasite (PL/SQL permite variabile de tip VARCHAR2 de pana la 32 KB, dar SQL nu permite mai mult de 4 KB)

Restrictii la utilizarea functiilor in instructiunilor SQL (continuare)

4. Parametrii trebuie sa fie specificati cu ajutorul notatiei pozitionale. Notatia (=>) nu este permisa.

Exemplu:

```
SELECT empno, tax(sal)
FROM emp;
SELECT empno, tax(p_value => sal)
FROM emp;
```

A doua instructiune SELECT are ca efect o eroare.

Restrictii la utilizarea functiilor in instructiunilor SQL (continuare)

5. Functiile folosite intr-o instructiune SELECT nu pot contine instructiuni DML
6. Functiile folosite intr-o instructiune UPDATE sau DELETE nu pot contine o interogare sau o instr. DML pentru aceeasi tabela
7. Functiile folosite in orice instructiune SQL nu pot termina tranzactiile (adica nu se pot executa operatiile COMMIT sau ROLLBACK)
8. Functiile folosite in orice instr. SQL nu pot utiliza comenzi DDL (de exemplu, CREATE TABLE) sau comenzi DCL (de exemplu ALTER SESSION) deoarece acestea implica COMMIT

2. Funcţii definite de utilizator

Exemplu 1:

```
CREATE OR REPLACE FUNCTION dml_call_sql(p_sal NUMBER)
  RETURN NUMBER IS
BEGIN
  INSERT INTO employees(employee_id, last_name, email,
                        hire_date, job_id, salary)
  VALUES(1, 'Frost', 'jfrost@company.com',
         SYSDATE, 'SA_MAN', p_sal);
  RETURN (p_sal + 100);
END dml_call_sql;
```

```
UPDATE employees
  SET salary = dml_call_sql(2000)
WHERE employee_id = 174;
```

```
ORA-04091:  table USVA_TEST_SQL01_S01.EMPLOYEES is mutating, trigger/function may not see it
```

Exemplu 2

Urmatoarea functie interogheaza tabela emp:

```
CREATE OR REPLACE FUNCTION query_max_sal (p_dept_id NUMBER)
  RETURN NUMBER IS
  v_num NUMBER;
 BEGIN
   SELECT MAX(salary) INTO v_num FROM employees
     WHERE department_id = p_dept_id;
   RETURN (v_num);
 END;
```

Cand folosim instr. DML, aceasta returneaza un mesaj de “mutating table” asemanator cu mesajul din exemplu anterior.

```
UPDATE employees SET salary = query_max_sal(department_id)
  WHERE employee_id = 174;
```

Cuprins

Funcţii în PL/SQL

1. Crearea funcţiilor
2. Funcţii definite de utilizator
3. Modificarea şi suprimarea subprogramelor PL/SQL
4. Module overload

3. Modificarea şi suprimarea subprogramelor PL/SQL

- Pentru a lua în considerare modificarea unei proceduri sau funcţii, recompilarea acestora se face prin comanda:

ALTER {***FUNCTION*** | ***PROCEDURE***}
[*schema.*]*nume* ***COMPILE****;*

- Comanda recompilează doar procedurile catalogate standard.
- Procedurile unui pachet se recompilează într-o altă manieră.

3. Modificarea şi suprimarea subprogramelor PL/SQL

- Ca şi în cazul tabelelor, funcţiile şi procedurile pot fi suprimate cu ajutorul comenzii *DROP*.
- Aceasta presupune eliminarea subprogramelor din dicţionarul datelor.
- *DROP* este o comandă ce aparţine limbajului de definire a datelor(DDL), astfel că se execută un *COMMIT* implicit atât înainte, cât şi după comandă.

3. Modificarea și suprimarea subprogramelor PL/SQL

- Când este şters un subprogram prin comanda *DROP*, automat sunt revocate toate privilegiile acordate referitor la acest subprogram.
- Dacă este utilizată sintaxa *CREATE OR REPLACE*, privilegiile acordate asupra acestui obiect (subprogram) rămân aceleaşi.

DROP {***FUNCTION*** | ***PROCEDURE***}
[*schema.*]*nume;*

Cuprins

Funcţii în PL/SQL

1. Crearea funcţiilor
2. Funcţii definite de utilizator
3. Modificarea şi suprimarea subprogramelor PL/SQL
4. Module overload

4. Module overload

- În anumite condiţii, două sau mai multe module pot să aibă aceleaşi nume, dar să difere prin lista parametrilor.
- Aceste module sunt numite module *overload* (supraîncărcate).
- Funcţia *TO_CHAR* este un exemplu de modul *overload*.

4. Module overload

- În cazul unui apel, compilatorul compară parametri actuali cu listele parametrilor formali pentru modulele *overload* şi execută modulul corespunzător.
- Toate programele *overload* trebuie să fie definite în acelaşi bloc *PL/SQL* (bloc anonim, modul sau pachet).
- Nu poate fi definită o versiune într-un bloc, iar altă versiune într-un bloc diferit.

4. Module overload

- Modulele *overload* pot să apară în programele *PL/SQL*:

  1. fie în secţiunea declarativă a unui bloc
  2. fie în interiorul unui pachet

- Supraîncărcarea funcţiilor sau procedurilor nu se poate face pentru funcţii sau proceduri stocate, dar se poate face pentru subprograme locale, subprograme care apar în pachete sau pentru metode.

4. Module overload

Observaţii:

1. Două programe *overload* trebuie să difere, cel puţin, prin tipul unuia dintre parametri. Două programe nu pot fi *overload* dacă parametri lor formali diferă numai prin subtipurile lor şi dacă aceste subtipuri se bazează pe acelaşi tip de date.

2. Nu este suficient ca lista parametrilor programelor *overload* să difere numai prin numele parametrilor formali.

4. Module overload

Observaţii (continuare):

3. Nu este suficient ca lista parametrilor programelor *overload* să difere numai prin tipul acestora (*IN*, *OUT*, *IN OUT*). *PL/SQL* nu poate face diferenţe (la apelare) între tipurile *IN* sau *OUT*.

4. Nu este suficient ca funcţiile *overload* să difere doar prin tipul datei returnate (tipul datei specificate în clauza *RETURN* a funcţiei).

4. Module overload

Supraîncărcarea funcţiilor sau procedurilor nu se poate face pentru funcţii sau proceduri stocate

Exemplu:

- Să se creeze două funcţii (locale) cu acelaşi nume care să calculeze media salariilor din departamentul 20.
- Prima funcţie va avea un argument reprezentând numarul departamentului, iar cea de a doua va avea două argumente, unul reprezentând numarul departamentului, iar celălalt reprezentând jobul pentru care se calculează valoarea medie (adică funcţia va calcula media salariilor pentru un anumit tip de job).

4. Module overload

```
create or replace function "VALOARE_MEDIE"
(v_deptno in emp.deptno%TYPE)
    return NUMBER is
    medie NUMBER(7,2);
BEGIN
    SELECT AVG(sal)
    INTO   medie
    FROM   emp
    WHERE  deptno = v_deptno;
    RETURN medie;
END;
```

4. Module overload

```
create or replace function "VALOARE_ MEDIE1"
(v_deptno in NUMBER, v_job in VARCHAR2)
RETURN NUMBER IS
    medie NUMBER(10,2);
BEGIN
    SELECT AVG(sal)
    INTO   medie
    FROM   emp
    WHERE  deptno = v_deptno AND job = v_job;
    RETURN medie;
END;
```

4. Module overload

```
DECLARE
  medie1 NUMBER(10,2);
  medie2 NUMBER(10,2);
BEGIN
  medie1 := valoare_medie(20);
  DBMS_OUTPUT.PUT_LINE('Media salariilor
din departamentul 20 este ' || medie1);
  medie2 := valoare_medie1(20, 'ANALYST');
  DBMS_OUTPUT.PUT_LINE('Media salariilor
din departamentul 20 care au functia de
ANALYST este ' || medie2);
END;
```

4. Module overload

VALOARE_MEDIE

Code Dependencies Errors Grants

Save & Compile | Find & Replace | Undo | Redo | Download Source | Drop

```
1  create or replace function "VALOARE_MEDIE" (v_deptno in emp.deptno%TYPE)
2      return NUMBER is
3      medie NUMBER(7,2);
4  BEGIN
5      SELECT AVG(sal)
6      INTO   medie
7      FROM   emp
8      WHERE  deptno = v_deptno;
9      RETURN medie;
10 END;
```

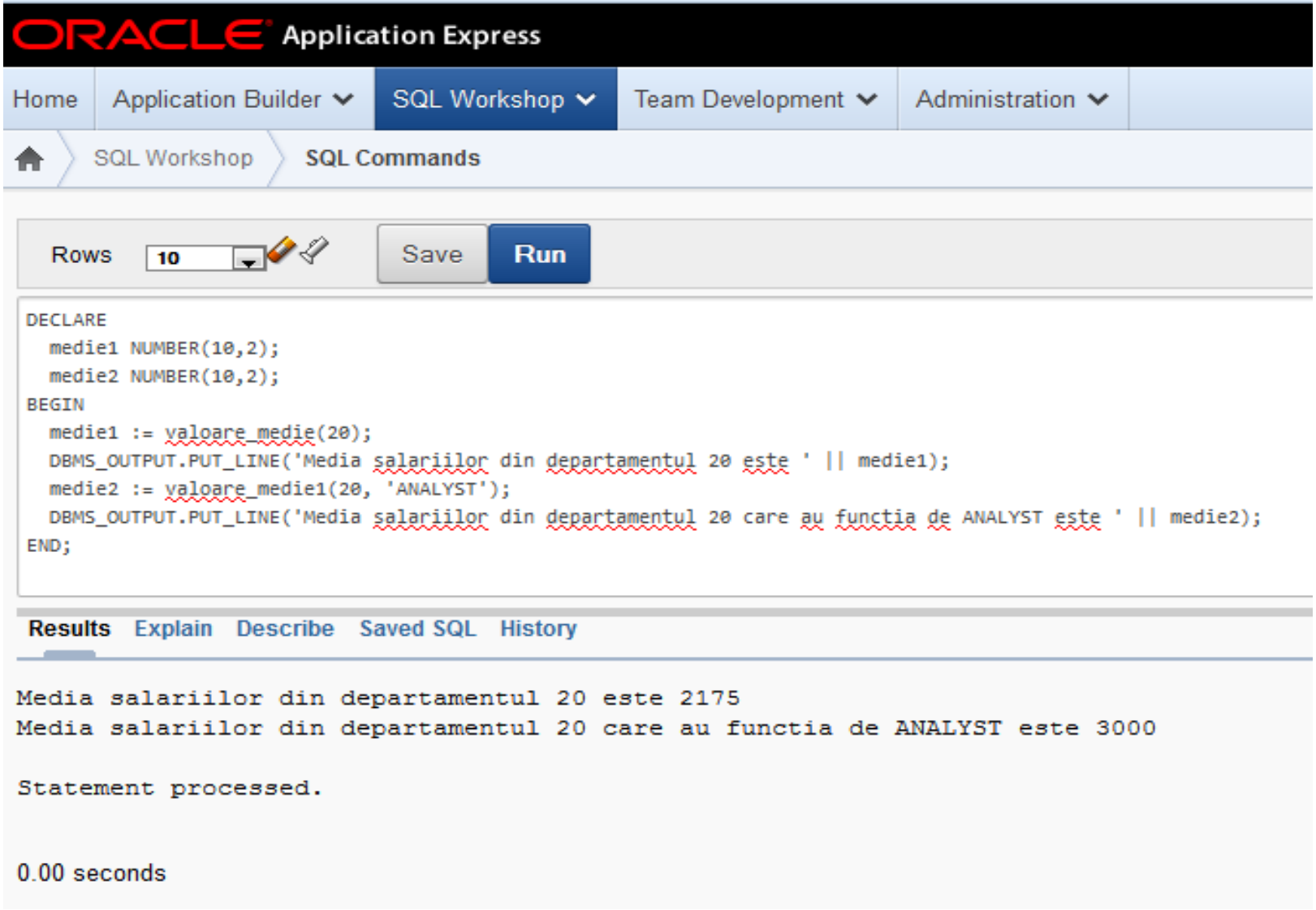
VALOARE_MEDIE1

Code Dependencies Errors Grants

Save & Compile | Find & Replace | Undo | Redo | Download Source | Drop

```
1  create or replace function "VALOARE_MEDIE1" (v_deptno in NUMBER,v_job in VARCHAR2)
2  RETURN NUMBER IS
3      medie NUMBER(10,2);
4  BEGIN
5      SELECT AVG(sal)
6      INTO   medie
7      FROM   emp
8      WHERE  deptno = v_deptno AND job = v_job;
9      RETURN medie;
10 END;
```

4. Module overload



Întrebări?