



Curs 10

Funcții în PL/SQL

Cuprins

Funcții în PL/SQL

- 1. Crearea funcțiilor**
- 2. Funcții definite de utilizator**
- 3. Modificarea și suprimarea subprogramelor PL/SQL**
- 4. Module overload**

1. Crearea funcțiilor

○ **funcție** este *un subprogram care returnează exact o valoare.*

○ **procedura** este *o instrucțiune executabilă de sine statatoare, în timp ce o funcție poate exista doar ca o parte a unei instrucțiuni executabile.*

- O funcție este un bloc PL/SQL cu nume care accepta parametri **IN** optional și trebuie să returneze o singură valoare.
- Funcțiile sunt stocate în baza de date ca obiecte ale schemei pentru execuții repetate.
- O funcție poate fi apelată ca parte a unei expresii SQL sau a unei expresii PL/SQL.

1. Crearea funcțiilor

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 funcțiilor

Sintaxa pentru crearea funcțiilor

Antetul unei funcții 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 funcțiilor

```
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 puțin o
instrucțiune RETURN.

1. Crearea funcțiilor

° **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

```
GET_SAL

Code Dependencies Errors Grants

Save & Compile Find & Replace Undo Redo Download Source Drop

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

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

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

1. Crearea funcțiilor

Modalitati de apelare (sau executare) a functiilor cu parametri

Funcțiile pot fi apelate în felul următor:

- 1. Ca parte a expresiilor PL/SQL*** – se folosește o variabilă locală într-un bloc anonim pentru a păstra valoarea returnată de funcție
- 2. Ca parametru al unui alt subprogram*** – transmiterea funcțiilor între subprograme
- 3. Ca o expresie într-o instrucțiune SQL*** – apelarea unei funcții ca pe oricare funcție single-row într-o instrucțiune SQL

1. Crearea funcțiilor

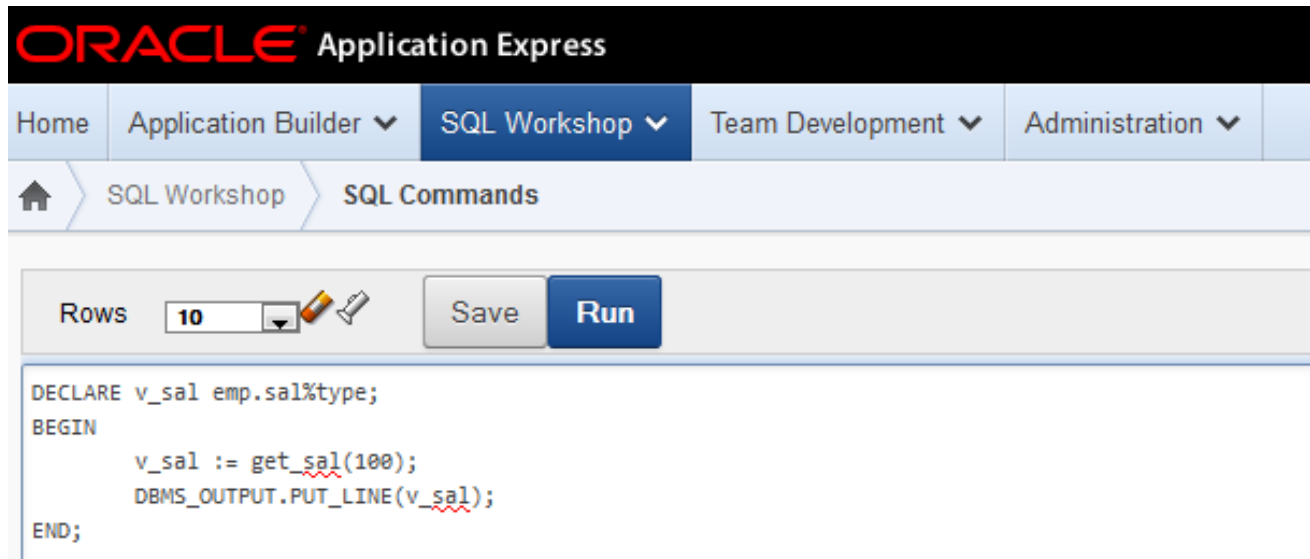
1. Apelarea ca o parte a unei expresii PL/SQL cu folosirea unei variabile locale pentru a stoca rezultatul obtinut:

```
DECLARE v_sal emp.sal%type;
```

```
BEGIN
```

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

```
...  
END;
```



The screenshot shows the Oracle Application Express interface. The top navigation bar includes 'Home', 'Application Builder', 'SQL Workshop', 'Team Development', and 'Administration'. The 'SQL Workshop' menu is expanded, showing 'SQL Commands'. Below the navigation, there is a 'Rows' dropdown set to '10', a 'Save' button, and a 'Run' button. The main area displays the following PL/SQL code:

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

1. Crearea funcțiilor

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 funcțiilor

Apelarea funcțiilor fara parametri

Majoritatea funcțiilor au parametri, dar nu toate.

USER si **SYSDATE** sunt funcții 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 funcțiilor

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 funcțiilor

Avantajele si restrictiile functiilor

Avantaje	Restrictii
<i>Funcțiile ne permit sa afisam temporar o valoare intr-un format nou</i>	<i>Tipurile de date PL/SQL nu se suprapun complet cu cele SQL</i>
<i>Permit noi caracteristici cum ar fi verificarea si analiza datelor</i>	<i>Dimensiunile PL/SQL nu sunt aceleasi cu dimensiunile SQL.</i> 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 funcțiilor

Diferente de sintaxa între proceduri și funcții

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 funcțiilor

Funcții

```
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 funcțiilor

Proceduri	Funcții
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

Funcții în PL/SQL

1. Crearea funcțiilor
2. Funcții definite de utilizator
3. Modificarea și suprimarea subprogramelor PL/SQL
4. Module overload

2. Funcții definite de utilizator

Funcții definite de utilizator

- *O funcție definită de utilizator este o funcție care este creată de către programator.*
- GET_DEPT_NAME și CALCULATE_TAX sunt exemple de funcții definite de utilizator, în timp ce UPPER, LOWER și LPAD sunt exemple de funcții sistem furnizate automat de Oracle.
- Majoritatea funcțiilor sistem (cum ar fi UPPER, LOWER și LPAD) sunt stocate în pachete denumite **SYS.STANDARD**.
- Aceste funcții sistem se mai numesc **funcții built-in**.

2. Funcții definite de utilizator

Avantajele funcțiilor in instructiunile SQL

- Prin folosirea funcțiilor in clauza WHERE a instructiunii SELECT crește eficiența prin eliminarea rândurilor nedorite înainte ca datele să fie transmise aplicației.
- De exemplu, in sistemul administrativ al unei facultăți, doriți să extrageți doar acei studenți ale căror nume sunt stocate in întregime cu majuscule. Acest lucru poate fi găsit pentru puține rânduri ale tabelului.

2. Funcții definite de utilizator

- Instrucțiunea folosită este:

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

- Fără funcția **UPPER** este nevoie să extrageți toate rândurile referitoare la studenți, să le transmiteți în rețea și să le eliminați pe cele nedorite prin aplicație.

- 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;
```

Funcții în SQL

Funcții în 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

Funcții create

SELECT empno, ename, sal, tax(sal)

FROM emp

WHERE deptno = 50;

Unde puteti folosi functii definite de utilizator in instructiunile SQL?

Funcțiile definite de utilizator se comporta ca și funcțiile 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;
```

Restricții la utilizarea funcțiilor în instrucțiunile SQL

Pentru a folosi o funcție definită de utilizator într-o instrucțiune **SQL**, funcția trebuie să respecte regulile și restricțiile limbajului **SQL**:

1. Funcția poate accepta doar tipurile de date valide ale **SQL**-ului în parametrii **IN** și trebuie să returneze un tip de date valid al **SQL**-ului.
2. Tipurile specifice **SQL**, cum ar fi tipurile **BOOLEAN** și **%ROWTYPE** nu sunt acceptate.
3. Limitele impuse de către **SQL** nu trebuie să fie depășite (PL/SQL permite variabile de tip **VARCHAR2** de până la 32 KB, dar **SQL** nu permite mai mult de 4 KB)

Restriții la utilizarea funcțiilor în instrucțiunile SQL (continuare)

4. Parametrii trebuie să fie specificați cu ajutorul notației pozitionale. Notatia (\Rightarrow) nu este permisă.

Exemplu:

```
SELECT empno, tax(sal)
```

```
FROM emp;
```

```
SELECT empno, tax(p_value => sal)
```

```
FROM emp;
```

A doua instrucțiune SELECT are ca efect o eroare.

Restricții la utilizarea funcțiilor în instrucțiunile SQL (continuare)

5. Funcțiile folosite într-o instrucțiune SELECT nu pot conține instrucțiuni DML
6. Funcțiile folosite într-o instrucțiune UPDATE sau DELETE nu pot conține o interogare sau o instr. DML pentru aceeași tabelă
7. Funcțiile folosite în orice instrucțiune SQL nu pot termina tranzacțiile (adică nu se pot executa operațiile COMMIT sau ROLLBACK)
8. Funcțiile folosite în orice instr. SQL nu pot utiliza comenzi DDL (de exemplu, CREATE TABLE) sau comenzi DCL (de exemplu ALTER SESSION) deoarece acestea implică COMMIT

2. Funcții 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

Funcții în PL/SQL

1. Crearea funcțiilor
2. Funcții definite de utilizator
3. Modificarea și suprimarea subprogramelor PL/SQL
4. Module overload

3. Modificarea și suprimarea subprogramelor PL/SQL

- Pentru a lua în considerare modificarea unei proceduri sau funcții, 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 și suprimarea subprogramelor PL/SQL

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

3. Modificarea și suprimarea subprogramelor PL/SQL

- Când este șters 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 aceleași.

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

Cuprins

Funcții în PL/SQL

1. Crearea funcțiilor
2. Funcții definite de utilizator
3. Modificarea și suprimarea subprogramelor PL/SQL
4. **Module overload**

4. Module overload

- În anumite condiții, două sau mai multe module pot să aibă aceleași nume, dar să difere prin lista parametrilor.
- Aceste module sunt numite **module overload** (supraîncărcate).
- Funcția **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* și execută modulul corespunzător.
- Toate programele *overload* trebuie să fie definite în același 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 secțiunea declarativă a unui bloc
 2. fie în interiorul unui pachet
- Supraîncărcarea funcțiilor sau procedurilor nu se poate face pentru funcții sau proceduri stocate, dar se poate face pentru subprograme locale, subprograme care apar în pachete sau pentru metode.

4. Module overload

Observații:

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

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

4. Module overload

Observații (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 diferențe (la apelare) între tipurile *IN* sau *OUT*.

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

4. Module overload

Supraîncărcarea funcțiilor sau procedurilor nu se poate face pentru funcții sau proceduri stocate

Exemplu:

- Să se creeze două funcții (locale) cu același nume care să calculeze media salariilor din departamentul 20.
- Prima funcție va avea un argument reprezentând numărul departamentului, iar cea de a doua va avea două argumente, unul reprezentând numărul departamentului, iar celălalt reprezentând jobul pentru care se calculează valoarea medie (adică funcția 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;
```

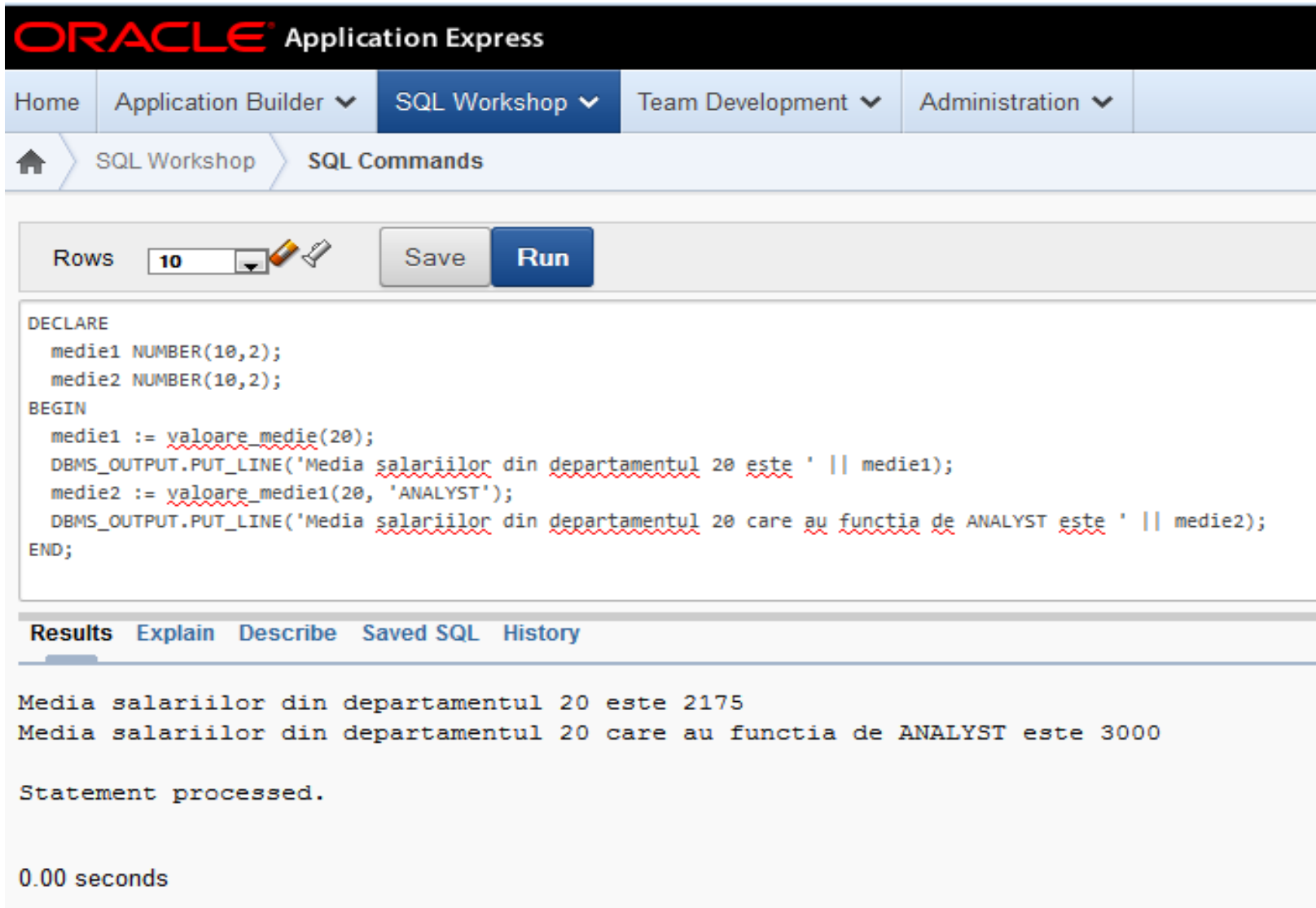
VALOARE_MEDIE1

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



The screenshot displays the Oracle Application Express interface. The top navigation bar includes 'Home', 'Application Builder', 'SQL Workshop', 'Team Development', and 'Administration'. The 'SQL Workshop' menu is expanded, showing 'SQL Commands'. Below the navigation, there is a toolbar with a 'Rows' dropdown set to '10', a 'Save' button, and a 'Run' button. The main area contains a PL/SQL script:

```
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;
```

Below the script, there are tabs for 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History'. The 'Results' tab is active, showing the output of the script:

```
Media salariilor din departamentul 20 este 2175
Media salariilor din departamentul 20 care au functia de ANALYST este 3000

Statement processed.

0.00 seconds
```



Întrebări?