

TEMĂ

Cuprins:

1. Curs 8.
 - 1.1 Se poate defini o variabilă globală a cărei valoare să poată fi citită/modificată de programe din aceeași sesiune?
 - 1.2 Se poate defini o variabilă globală a cărei valoare să poată fi citită/modificată de programe din sesiuni diferite?
 - 1.3 Funcții și proceduri care folosesc ca parametrii tipuri de date complexe (înregistrări și colecții).
 - 1.4 Funcții care întorc tipuri de date complexe (înregistrări și colecții).
2. Curs 9.
 - 2.1 FUNCȚIE cu COUNT(*) vs FUNCȚIE cu SELECT 1 și excepții.
 - 2.2 Trigger LMD care să fie declanșat de acțiuni pe un tabel imbricat (nested table).
3. Curs 10.
 - 3.1 Trigger compound.
 - 3.2 Exemplul 8.6 din curs SELECT COUNT(*) vs SELECT 1 și excepții.

1. Curs 8

1.1 Se poate define o variabilă globală a cărei valoare să poată fi citită/modificată de programe din aceeași sesiune?

Știm că în PL/SQL, variabilele globale se pot defini doar cu ajutorul pachetelor. Vom crea un pachet `variabile_globale`, care va conține o variabilă numită `numar`, inițializată cu 0 și un bloc PL/SQL în care vom încerca să modificăm această variabilă.

Rezolvare:

```
CREATE OR REPLACE PACKAGE variabile_globale
```

```
IS
```

```
    numar PLS_INTEGER := 0;
```

```
END;
```

```
/
```

```
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE('Inainte de modificare: ' || variabile_globale.numar);
```

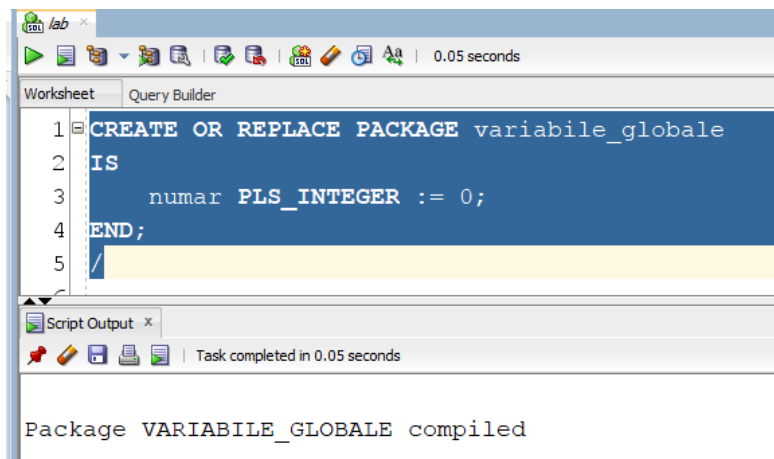
```
    variabile_globale.numar := 7;
```

```
    DBMS_OUTPUT.PUT_LINE('Dupa modificare: ' || variabile_globale.numar);
```

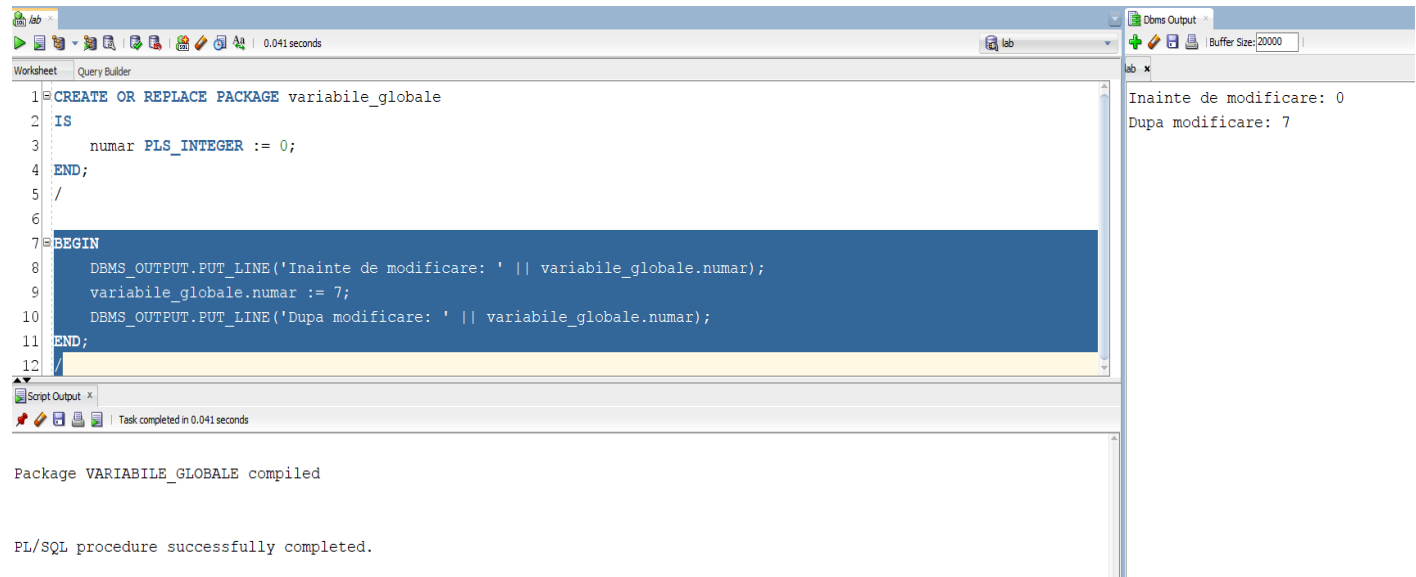
```
END;
```

```
/
```

Print-Screen:



Am creat cu succes pachetul care conține variabila noastră globală inițializată cu 0, variabila `numar`. Acum vom încerca să modificăm această variabilă din blocul nostru PL/SQL.



The screenshot shows the Oracle SQL Developer interface. The main window displays a PL/SQL script in the 'Worksheet' tab. The script is as follows:

```
1 CREATE OR REPLACE PACKAGE variabile_globale
2 IS
3     numar PLS_INTEGER := 0;
4 END;
5 /
6
7 BEGIN
8     DBMS_OUTPUT.PUT_LINE('Inainte de modificare: ' || variabile_globale.numar);
9     variabile_globale.numar := 7;
10    DBMS_OUTPUT.PUT_LINE('Dupa modificare: ' || variabile_globale.numar);
11 END;
12 /
```

The 'Script Output' tab at the bottom shows the results of the execution:

```
Package VARIABLE_GLOBALE compiled

PL/SQL procedure successfully completed.
```

The 'DBMS Output' window on the right shows the output of the script:

```
Inainte de modificare: 0
Dupa modificare: 7
```

După cum putem vedea, am reușit să modificăm valoarea variabilei din 0 în 7.

1.2 Se poate defini o variabilă globală a cărei valoare să poată fi citită/modificată de programe din sesiuni diferite?

Rezolvare:

BEGIN

DBMS_OUTPUT.PUT_LINE('Inainte de modificare: ' || variabile_globale.numar);

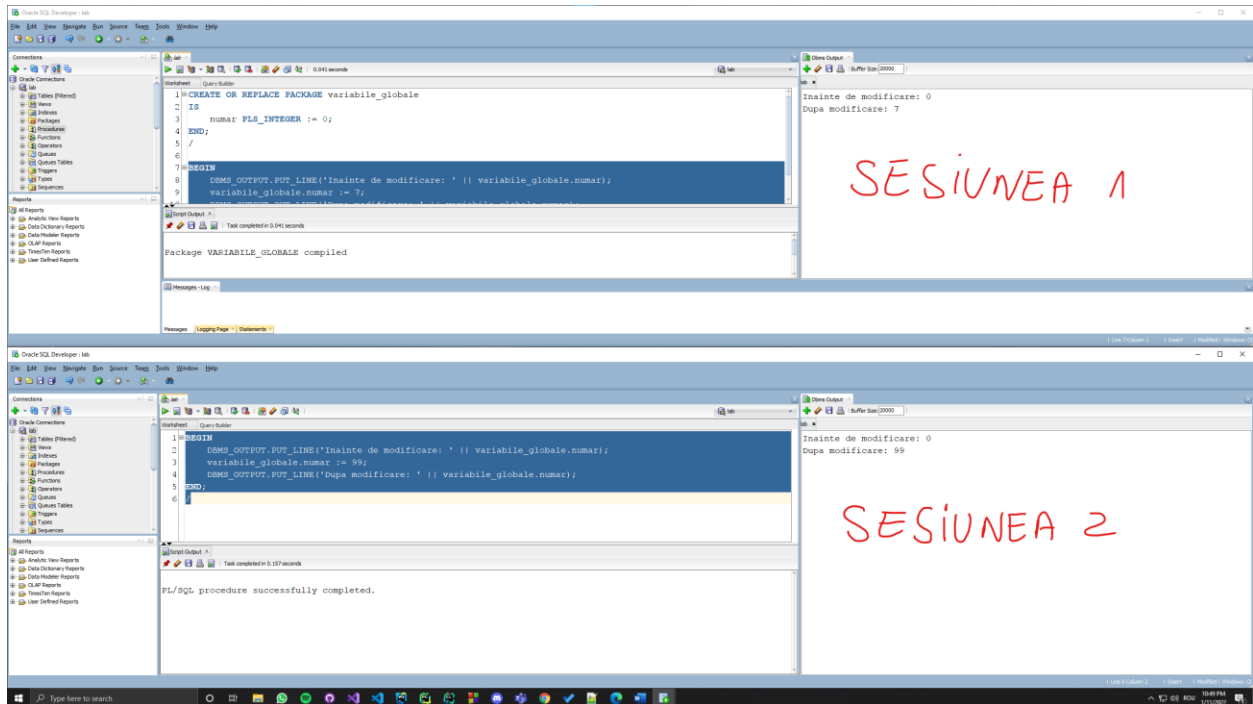
variabile_globale.numar := 99;

DBMS_OUTPUT.PUT_LINE('Dupa modificare: ' || variabile_globale.numar);

END;

/

Print-Screen:



Observăm că variabila `numar`, din pachetul `variabile_globale` s-a modificat cu succes în Sesiunea 2. De asemenea modificarea variabilei din Sesiunea 1, nu a fost vizibilă în Sesiunea 2.

Observăm că modificările au fost făcute, mai exact, și sesiunea 2 a recunoscut pachetul. În schimb, putem vedea faptul că modificarea variabilei dintr-o sesiune nu a fost vizibilă și în cealaltă sesiune.

1.3 Funcții și proceduri care folosesc ca parametrii tipuri de date complexe (înregistrări și colecții).

Funcție care primește ca parametru înregistrare.

Rezolvare:

```
CREATE TABLE tab_par (
```

```
  a NUMBER,
```

```
  b NUMBER
```

```
)
```

```
/
```

```
insert into tab_par values ( 10, 5);
```

```
CREATE OR REPLACE TYPE tab_numbers AS
```

```
TABLE OF NUMBER
```

```
/
```

```
CREATE OR REPLACE FUNCTION fun_par (
```

```
    r IN tab_par%rowtype
```

```
) RETURN tab_numbers IS
```

```
    retval tab_numbers := NEW tab_numbers();
```

```
BEGIN
```

```
    retval.extend(2);
```

```
    retval(1) := r.a;
```

```
    retval(2) := r.b;
```

```
    RETURN retval;
```

```
END;
```

```
/
```

```
DECLARE
```

```
    r tab_par%rowtype;
```

```
    n tab_numbers;
```

```
BEGIN
```

```
    SELECT * INTO r
```

```
    FROM tab_par
```

```
    WHERE ROWNUM = 1;
```

```
    n := fun_par(r);
```

```
    dbms_output.put_line('n = ' || n(1));
```

```
    dbms_output.put_line('n = ' || n(2));
```

```
END;
```

```
/
```

Print-Screen:

The screenshot displays the Oracle SQL Developer environment. The main window shows a PL/SQL function named `FUN_PAR` being edited. The function takes a row of data from a table `tab_par` and returns a table of numbers. The function is defined as follows:

```
14  r IN tab_par%rowtype
15  ) RETURN tab_numbers IS
16    retval tab_numbers := NEW tab_numbers();
17  BEGIN
18    retval.extend(2);
19    retval(1) := r.a;
20    retval(2) := r.b;
21    RETURN retval;
22  END;
23  /
24
25  DECLARE
26    r tab_par%rowtype;
27    n tab_numbers;
28  BEGIN
29    SELECT * INTO r
30    FROM tab_par
31    WHERE ROWNUM = 1;
32    n := fun_par(r);
33    dbms_output.put_line('n = ' || n(1));
```

The right-hand pane shows the output of the function execution:

```
n = 10
n = 5
```

The bottom status bar indicates that the function was compiled successfully and the PL/SQL procedure was completed.

Rezolvare:

```
CREATE OR REPLACE PACKAGE employee_details AS

  TYPE details IS RECORD (
    p_name  VARCHAR2(40),
    p_emp_id NUMBER
  );

  TYPE table_employees IS
    TABLE OF details;

  PROCEDURE get_employees (
    p_deptno IN employees.department_id%TYPE,
    p_sal    IN employees.salary%TYPE,
    emp_rec  OUT table_employees
  );

END employee_details;

/
```

```
CREATE OR REPLACE PACKAGE BODY employee_details AS
```

```
    PROCEDURE get_employees (
```

```
        p_deptno IN employees.department_id%TYPE,
```

```
        p_sal IN employees.salary%TYPE,
```

```
        emp_rec OUT table_employees
```

```
    ) IS
```

```
    BEGIN
```

```
        SELECT first_name, employee_id BULK COLLECT INTO emp_rec
```

```
        FROM employees
```

```
        WHERE department_id = p_deptno
```

```
        AND salary > p_sal;
```

```
    END get_employees;
```

```
END employee_details;
```

```
/
```

```
DECLARE
```

```
    l_table_rec_type employee_details.table_employees;
```

```
BEGIN
```

```
    dbms_output.put_line(' Apelam get_employees ');
```

```
    employee_details.get_employees(30, 1000, l_table_rec_type);
```

```
    FOR l_rec IN 1..l_table_rec_type.count LOOP dbms_output.put_line('Detalii angajat: '
```

```
        || l_table_rec_type(l_rec).p_name
```

```
        || ' '
```

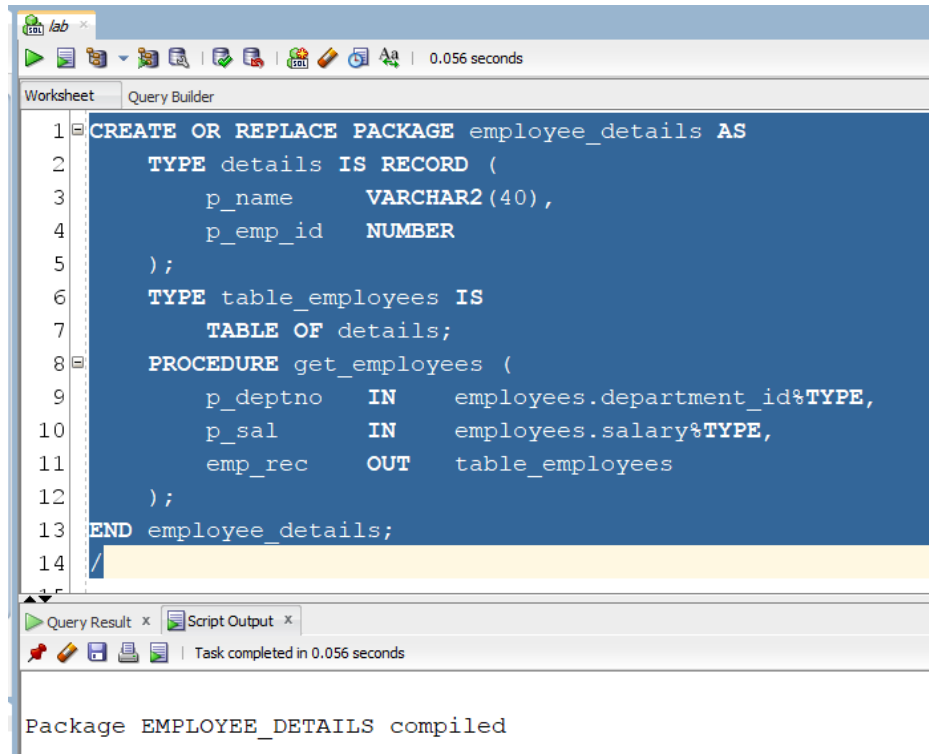
```
        || l_table_rec_type(l_rec).p_emp_id);
```

```
    END LOOP;
```

```
END;
```

```
/
```

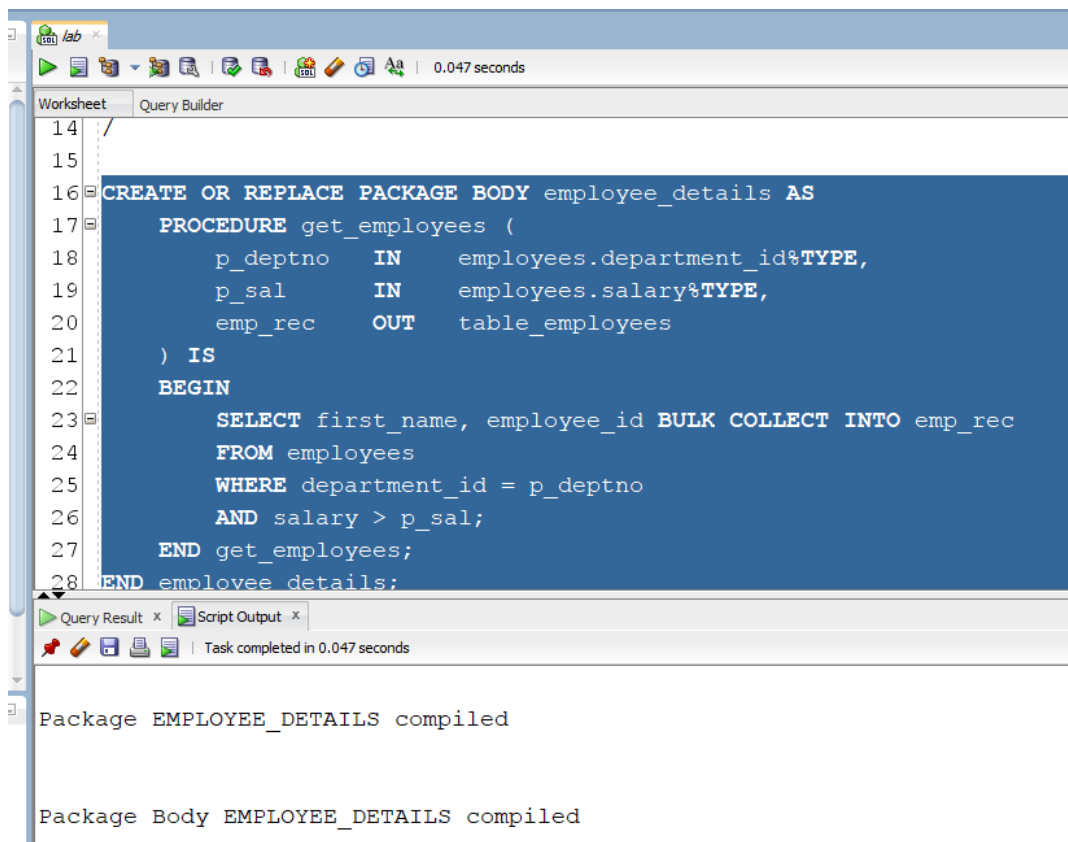
Print-Screen:



The screenshot shows the SQL Developer interface with the 'Query Builder' tab active. The SQL editor contains the following code:

```
1 CREATE OR REPLACE PACKAGE employee_details AS
2     TYPE details IS RECORD (
3         p_name      VARCHAR2(40),
4         p_emp_id     NUMBER
5     );
6     TYPE table_employees IS
7         TABLE OF details;
8     PROCEDURE get_employees (
9         p_deptno     IN     employees.department_id%TYPE,
10        p_sal         IN     employees.salary%TYPE,
11        emp_rec       OUT    table_employees
12    );
13 END employee_details;
14 /
```

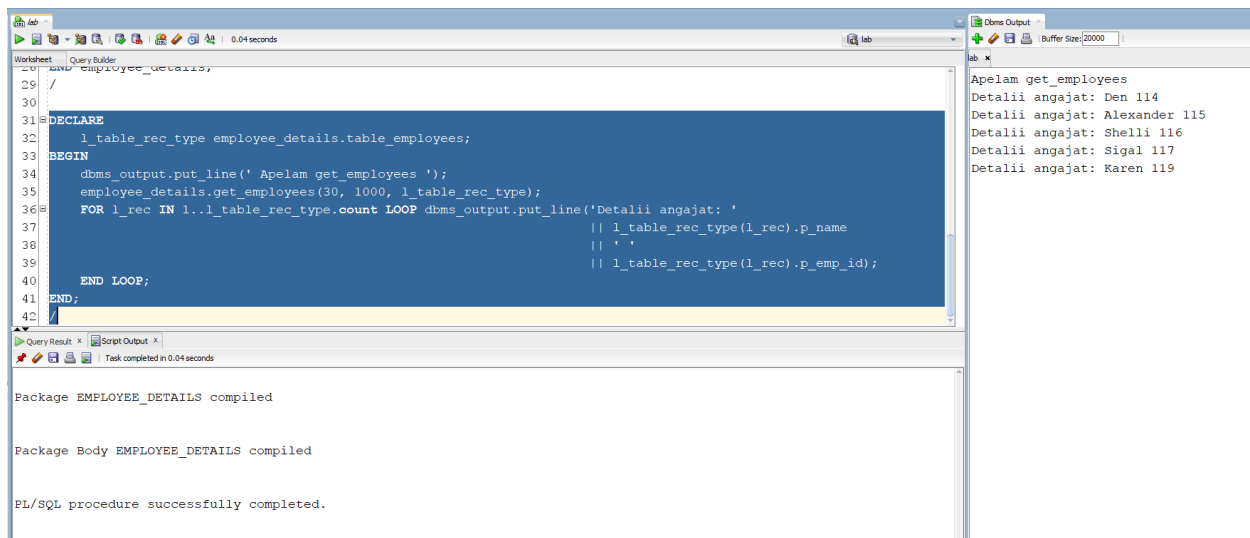
The status bar at the bottom indicates 'Task completed in 0.056 seconds'. Below the editor, the message 'Package EMPLOYEE_DETAILS compiled' is displayed.



The screenshot shows the SQL Developer interface with the 'Query Builder' tab active. The SQL editor contains the following code:

```
14 /
15
16 CREATE OR REPLACE PACKAGE BODY employee_details AS
17     PROCEDURE get_employees (
18         p_deptno     IN     employees.department_id%TYPE,
19         p_sal         IN     employees.salary%TYPE,
20         emp_rec       OUT    table_employees
21     ) IS
22     BEGIN
23         SELECT first_name, employee_id BULK COLLECT INTO emp_rec
24         FROM employees
25         WHERE department_id = p_deptno
26         AND salary > p_sal;
27     END get_employees;
28 END employee_details;
```

The status bar at the bottom indicates 'Task completed in 0.047 seconds'. Below the editor, the message 'Package EMPLOYEE_DETAILS compiled' is displayed. At the bottom of the window, the message 'Package Body EMPLOYEE_DETAILS compiled' is also visible.



1.4 Funcții care întorc tipuri de date complexe (înregistrări și colecții).

Rezolvare:

Funcție care returnează un tablou imbricat:

-- Produsele preferate pentru un client citit de la tastatura

-- Se afiseaza numele si prenumele clientului, produsele preferate si de cate ori le-a comandat.

```
CREATE OR REPLACE TYPE tab_imb IS
```

```
TABLE OF NUMBER(10);
```

```
/
```

```
CREATE OR REPLACE FUNCTION afis_produce_preferate (my_id_client IN client.id_client%TYPE)
```

```
RETURN tab_imb IS
```

```
TYPE tab_ind IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
```

```
v_contor          tab_ind;
```

```
v_index_max       PLS_INTEGER;
```

```
id                PLS_INTEGER;
```

```
v_id_feluri_preferate tab_imb;
```

```
ok                BINARY_INTEGER := 0;
```

```
BEGIN
```

```

v_id_feluri_preferate := tab_imb();
FOR i IN (SELECT p.id_produc -- CURSOR IMPLICIT
        FROM produs p, continut_comanda cont, comanda cmd
        WHERE my_id_client = cmd.id_client
        AND cmd.id_comanda = cont.id_comanda
        AND cont.id_produc = p.id_produc) LOOP BEGIN
    v_contor(i.id_produc) := v_contor(i.id_produc) + 1; -- simulez un vector de frecventa, unde tin de
cate ori a fost comandat produsul respectiv
    IF ok = 1 THEN
        IF v_contor(i.id_produc) > v_contor(v_index_max) THEN
            v_index_max := i.id_produc;
        END IF;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN -- intr-un tabel indexat daca nu exista indexul respectiv
    -- se arunca exceptia no data found
        v_contor(i.id_produc) := 1; -- initializam numarul de aparitii cu 1
        IF ok = 0 THEN -- daca e primul produs selectat de cursor
            ok := 1;
            v_index_max := i.id_produc;
        END IF;
END;
END LOOP;
IF ok = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Acest client nu a comandat nimic in viata lui');
ELSE
    id := v_contor.first; -- prima valoare din tabelul indexat (vector frecventa)
    LOOP
        EXIT WHEN id IS NULL;

```

```

        IF v_contor(id) = v_contor(v_index_max) THEN
            v_id_feluri_preferate.extend;
            v_id_feluri_preferate(v_id_feluri_preferate.LAST) := id;
        END IF;

        id := v_contor.NEXT(id);
    END LOOP;

END IF;

RETURN v_id_feluri_preferate;

END;

/

```

DECLARE

```

v_id_client_citit    client.id_client%TYPE := &client_id;
my_tab              tab_imb;
id                  PLS_INTEGER;
v_nume_produs       produs.nume%TYPE;
v_nume              client.nume%TYPE;
v_prenume           client.prenume%TYPE;
v_contor            PLS_INTEGER := 0;

```

BEGIN

```

my_tab := afis_produce_preferate(v_id_client_citit);

SELECT c.nume, c.prenume INTO v_nume, v_prenume
FROM client c

WHERE c.id_client = v_id_client_citit;

DBMS_OUTPUT.PUT_LINE('Clientul ' || v_nume || ' ' || v_prenume || ' are urmatoarele produse
favorite:');

id := my_tab.FIRST; -- prima valoare din tabelul indexat (vector frecventa)

LOOP

    EXIT WHEN id IS NULL;

```

```

SELECT p.nume INTO v_nume_produus

FROM produs p

WHERE p.id_produus = my_tab(id);

-- afisam produsele cu numarul maxim de aparitii

v_contor := v_contor + 1;

DBMS_OUTPUT.PUT_LINE(v_contor || '. ' || v_nume_produus);

id := my_tab.NEXT(id);

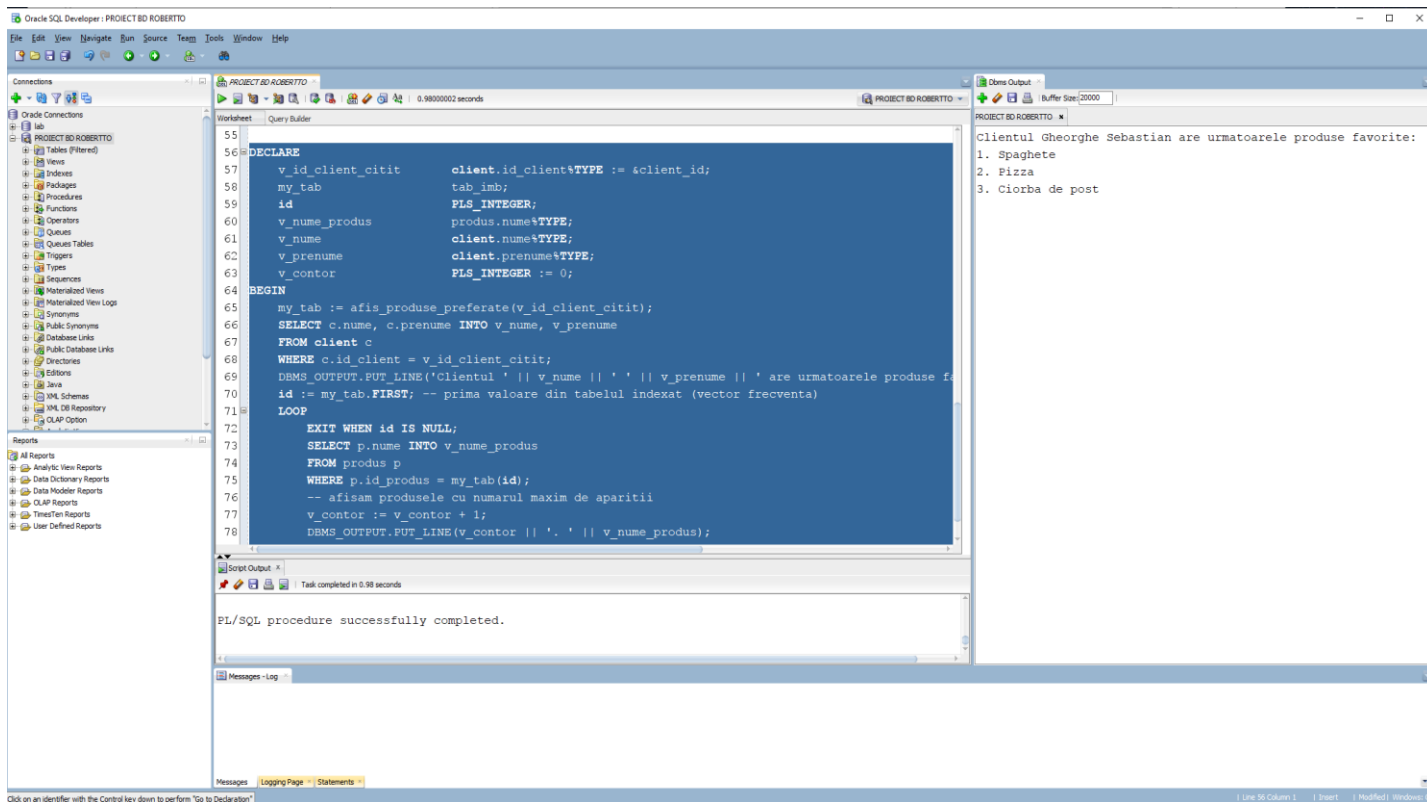
END LOOP;

END;

/

```

Print-Screen:



Funcție care returnează record-uri:

```

CREATE OR REPLACE FUNCTION get_clienti(my_id_client IN client.id_client%TYPE)

RETURN client%ROWTYPE

AS

```

```
l_client_record client%ROWTYPE;

BEGIN

SELECT * INTO l_client_record

FROM client

WHERE id_client = my_id_client;

RETURN l_client_record;

END;

/
```

```
DECLARE

r_clienti client%ROWTYPE;

BEGIN

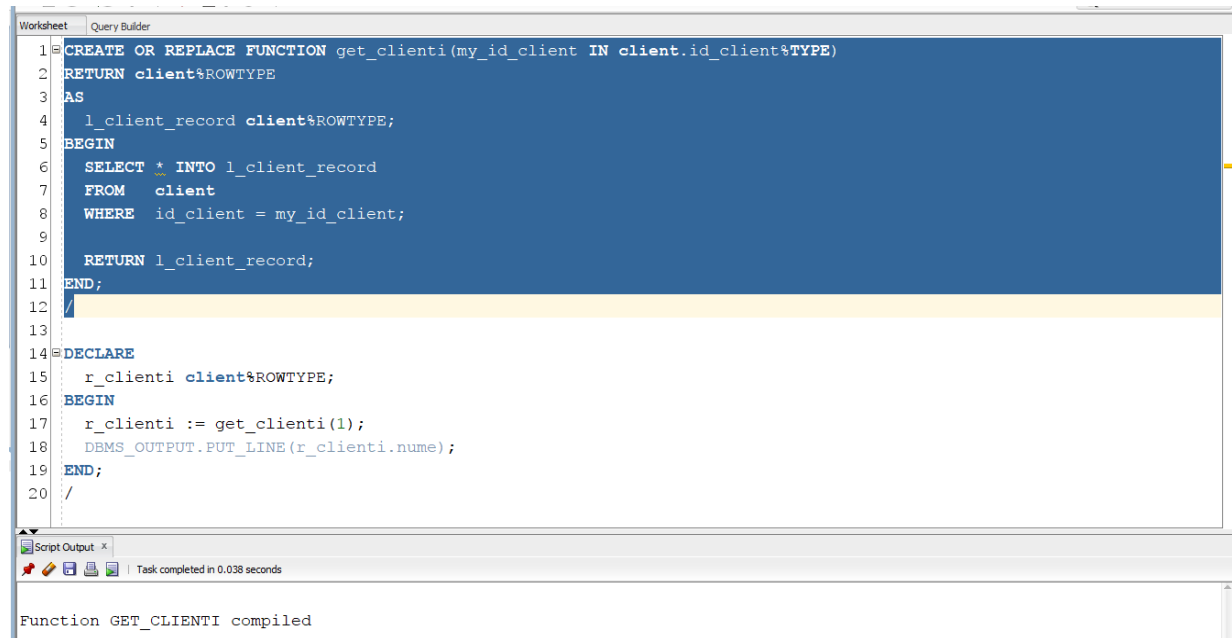
r_clienti := get_clienti(1);

DBMS_OUTPUT.PUT_LINE(r_clienti.num);

END;

/
```

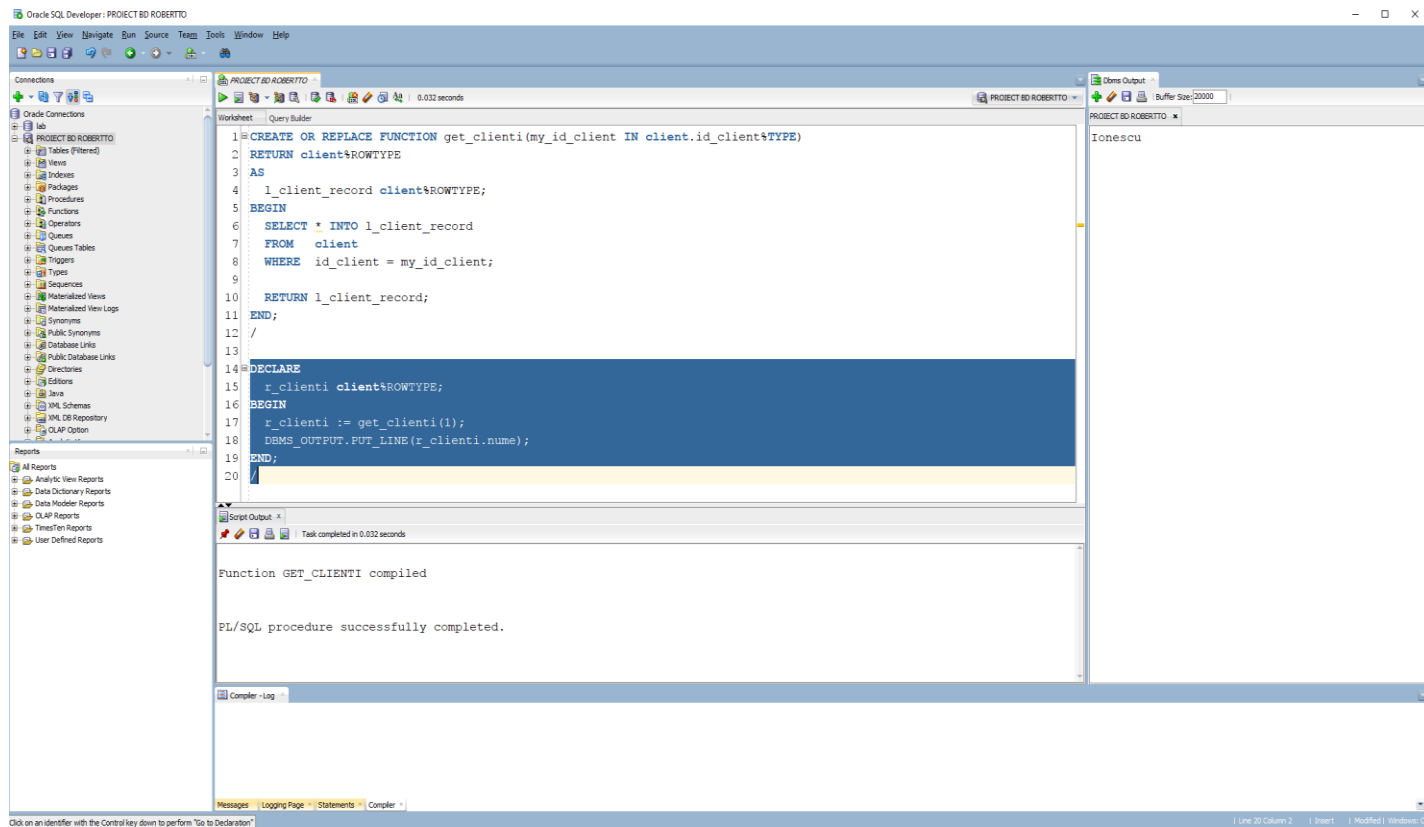
Print-Screen:



The screenshot displays the SQL Developer environment. The main window is the 'Query Builder' tab, which contains a script editor with the following SQL code:

```
1 CREATE OR REPLACE FUNCTION get_clienti(my_id_client IN client.id_client%TYPE)
2 RETURN client%ROWTYPE
3 AS
4 l_client_record client%ROWTYPE;
5 BEGIN
6 SELECT * INTO l_client_record
7 FROM client
8 WHERE id_client = my_id_client;
9
10 RETURN l_client_record;
11 END;
12 /
13
14 DECLARE
15 r_clienti client%ROWTYPE;
16 BEGIN
17 r_clienti := get_clienti(1);
18 DBMS_OUTPUT.PUT_LINE(r_clienti.num);
19 END;
20 /
```

Below the script editor is the 'Script Output' window, which shows the message: 'Task completed in 0.038 seconds'. At the bottom of the interface, a status bar indicates 'Function GET_CLIENTI compiled'.



2. Curs 9

2.1 FUNCȚIE cu COUNT(*) vs FUNCȚIE cu SELECT 1 și excepții.

Pentru a vedea diferențele între cele două vom folosi funcția **sysimestamp** cu care vom vedea timpul la care a început/sfârșit fiecare instrucțiune, **COUNT(*)**, respectiv **SELECT 1**.

Rezolvare:

```
CREATE OR REPLACE FUNCTION subprogram_count(v_min_salary employees.salary%TYPE,
v_max_salary employees.salary%TYPE)
```

```
RETURN BOOLEAN
```

```
IS
```

```
rezultat NUMBER;
```

```
BEGIN
```

```
SELECT COUNT(*) INTO rezultat
```

```
FROM employees e, departments d, jobs j, job_history j1, locations l, countries c, regions r
```

```
WHERE e.department_id = d.department_id
AND e.employee_id = j1.employee_id
AND j1.job_id = j.job_id
AND d.location_id = l.location_id
AND l.country_id = c.country_id
AND c.region_id = r.region_id
AND e.salary BETWEEN v_min_salary AND v_max_salary;
```

```
IF rezultat > 0 THEN
```

```
    RETURN TRUE;
```

```
ELSE
```

```
    RETURN FALSE;
```

```
END IF;
```

```
END subprogram_count;
```

```
/
```

```
DECLARE
```

```
    ok    BOOLEAN;
```

```
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE('Inainte de subprogram cu COUNT(*)');;
```

```
    DBMS_OUTPUT.PUT_LINE(systimestamp);
```

```
    ok := subprogram_count(0, 7900);
```

```
    DBMS_OUTPUT.PUT_LINE('Dupa subprogram cu COUNT(*)');;
```

```
    DBMS_OUTPUT.PUT_LINE(systimestamp);
```

```
    DBMS_OUTPUT.NEW_LINE();
```

```
    IF ok = TRUE THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Rezultat: TRUE !');
```

```
    ELSE
```

```

        DBMS_OUTPUT.PUT_LINE('Rezultat: FALSE !');

    END IF;

END;

/

CREATE OR REPLACE FUNCTION subprogram_select(v_min_salary employees.salary%TYPE,
v_max_salary employees.salary%TYPE)

RETURN BOOLEAN

IS

    rezultat NUMBER;

BEGIN

    SELECT 1 INTO rezultat

    FROM employees e, departments d, jobs j, job_history j1, locations l, countries c, regions r

    WHERE e.department_id = d.department_id

    AND e.employee_id = j1.employee_id

    AND j1.job_id = j.job_id

    AND d.location_id = l.location_id

    AND l.country_id = c.country_id

    AND c.region_id = r.region_id

    AND e.salary BETWEEN v_min_salary AND v_max_salary;

    RETURN TRUE;

EXCEPTION

    WHEN NO_DATA_FOUND THEN

        RETURN FALSE;

    WHEN TOO_MANY_ROWS THEN

        RETURN TRUE;

END subprogram_select;

/

```



```
DECLARE

    ok    BOOLEAN;

BEGIN

    DBMS_OUTPUT.PUT_LINE('Inainte de subprogram cu SELECT 1:');

    DBMS_OUTPUT.PUT_LINE(systimestamp);

    ok := subprogram_select(0, 7900);

    DBMS_OUTPUT.PUT_LINE('Dupa subprogram cu SELECT 1:');

    DBMS_OUTPUT.PUT_LINE(systimestamp);

    IF ok = TRUE THEN

        DBMS_OUTPUT.PUT_LINE('Rezultat: TRUE !');

    ELSE

        DBMS_OUTPUT.PUT_LINE('Rezultat: FALSE !');

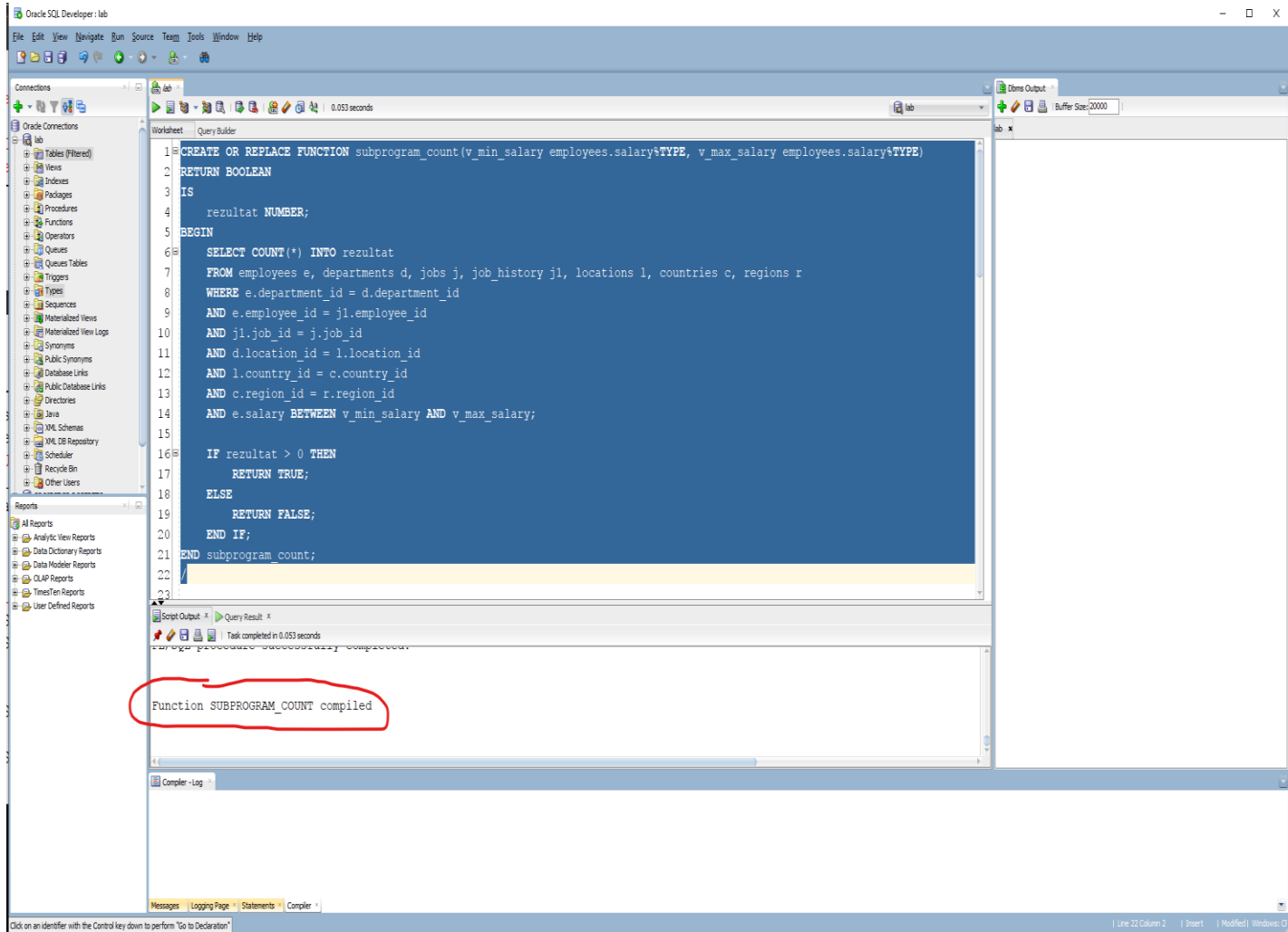
    END IF;

END;

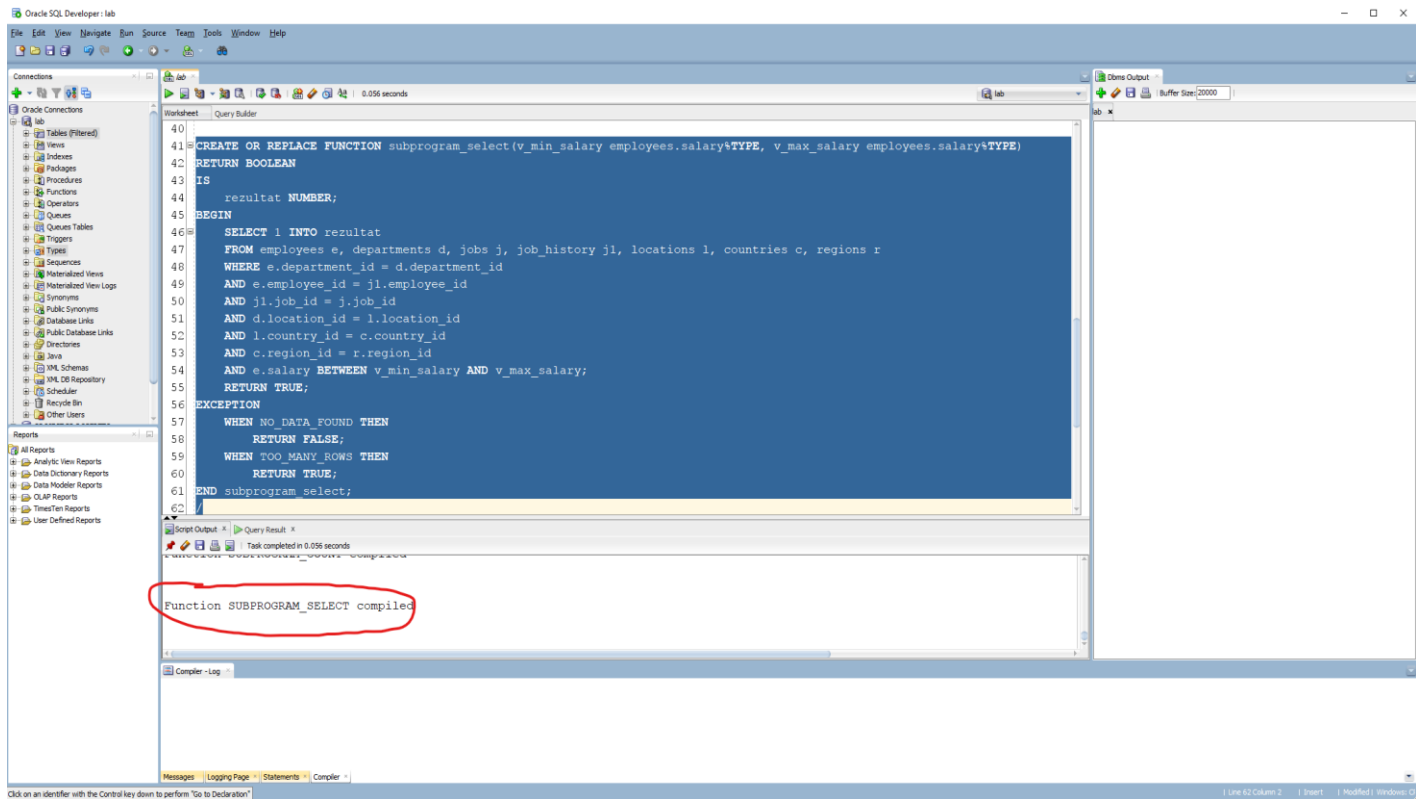
/
```

Print-Screen:

Vom testa cazurile când avem minim 2 înregistrări (pentru SELECT 1 va intra pe excepția *TOO_MANY_ROWS*), o singură înregistrare (SELECT 1 nu intră pe nicio excepție) și nicio înregistrare (SELECT 1 intră pe excepția *NO_DATA_FOUND*).

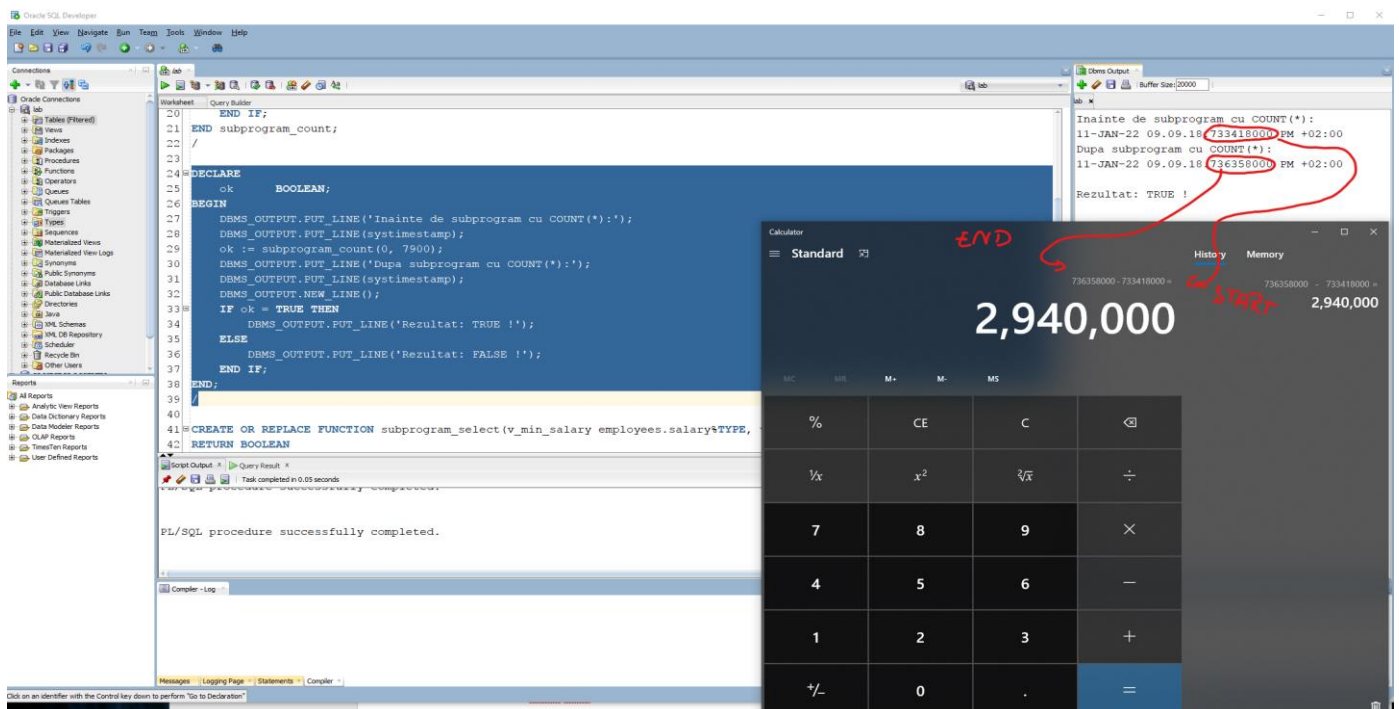


Am creat funcția cu numele `subprogram_count` cu succes. Acum vom crea funcția cu numele `subprogram_select`.



Am creat funcția cu numele `subprogram_select` cu succes. Acum vom rula blocurile PL/SQL la fiecare pentru a putea face o comparație între cele două.

Testăm primul caz când salariul minim este 0 și maxim 7900, care conține 3 înregistrări pentru `COUNT(*)`.



Observăm că a durat **2.940.000** unități (ordinul milisecundelor), subprogramul cu *COUNT(*)*. Acum vom face calculul pentru subprogramul cu *SELECT 1* și excepții.

Testăm primul caz când salariul minim este 0 și maxim 7900, care conține 3 înregistrări pentru *SELECT 1* și excepții.

The screenshot shows the Oracle SQL Developer interface. The main window displays a PL/SQL procedure named `subprogram_select` with an exception handler. The procedure is executed, and the output window shows the results of the execution. A calculator window is open in the foreground, showing the calculation `3140000 - 33534000 = 314,000`. The calculator also shows the result `2,940,000` for the `COUNT(*)` query.

Observăm că subprogramul cu *SELECT 1* și excepții a durat **314.000** unități (ordinul milisecundelor), **INTRĂ PE EXCEPȚIA TOO_MANY_ROWS!** Deci s-a executat cu **2.626.000** de unități mai repede decât subprogramul cu *COUNT(*)*.

The screenshot shows a calculator window with the result `2,626,000` displayed. The calculator also shows the calculation `2940000 - 314000 = 2,626,000`. The calculator interface includes a numeric keypad and a history window.

Testăm al doilea caz când salariul minim este 7900 și maxim 7900, care conține o singură înregistrare pentru COUNT(*)).

Oracle SQL Developer interface showing the execution of a PL/SQL procedure. The procedure is named `subprogram_count` and is being executed with parameters 7900 and 7900. The output window shows the following messages:

```
Inainte de subprogram cu COUNT(*):  
11-JAN-22 09.31.31.689577000 PM +02:00  
Dupa subprogram cu COUNT(*):  
11-JAN-22 09.31.31.992233000 PM +02:00  
Rezultat: TRUE !
```

The calculator window shows the calculation $2,656,000$.

Observăm că a durat **2.656.000** unități (ordinul milisecundelor), subprogramul cu `COUNT(*)`. Acum vom face calculul pentru subprogramul cu `SELECT 1` și excepții.

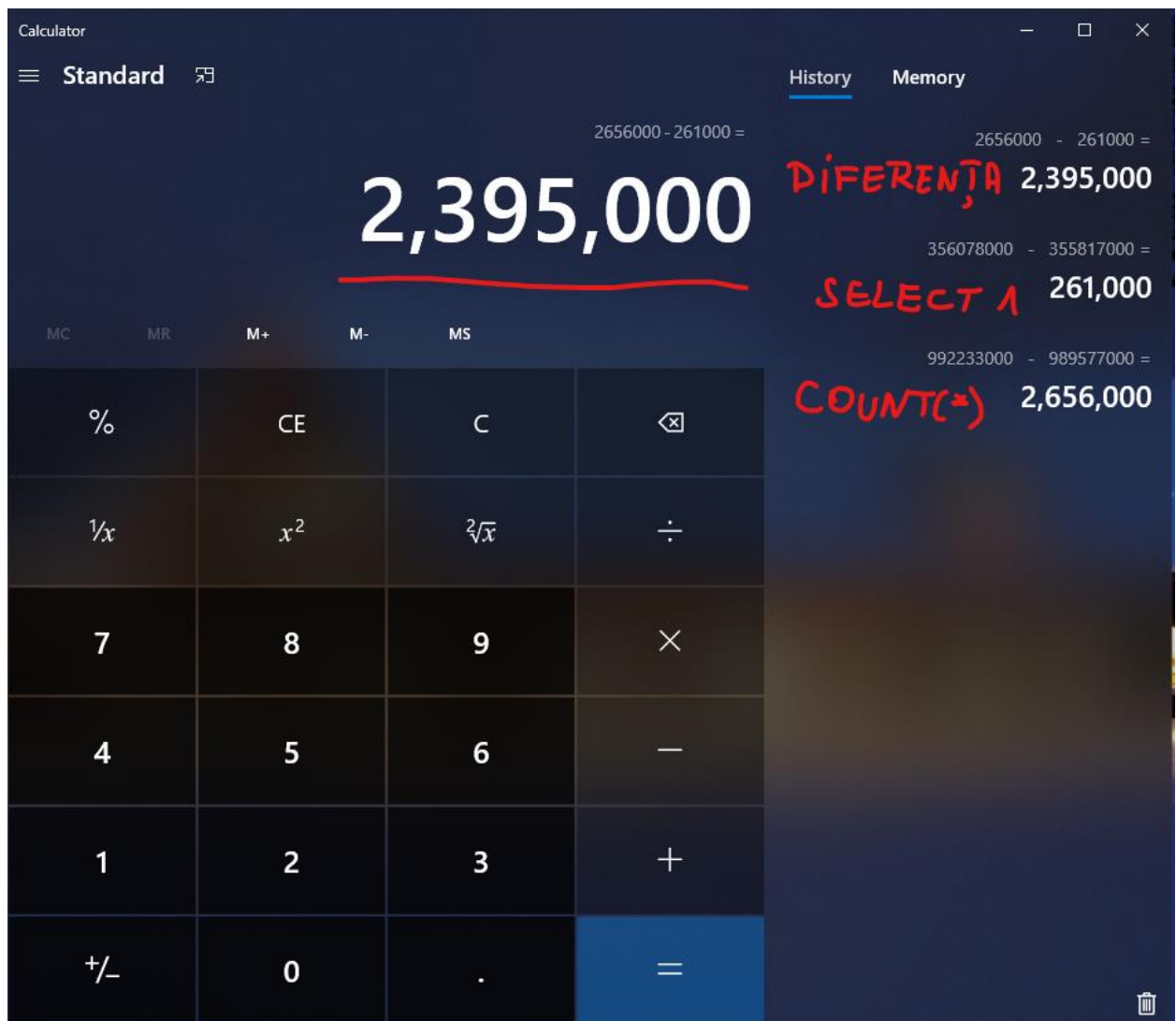
Testăm al doilea caz când salariul minim este 7900 și maxim 7900, care conține o singură înregistrare pentru `SELECT 1` și excepții.

Oracle SQL Developer interface showing the execution of a PL/SQL procedure. The procedure is named `subprogram_select` and is being executed with parameters 7900 and 7900. The output window shows the following messages:

```
Inainte de subprogram cu SELECT 1:  
11-JAN-22 09.33.30.655817000 PM +02:00  
Dupa subprogram cu SELECT 1:  
11-JAN-22 09.33.30.656078000 PM +02:00  
Rezultat: TRUE !
```

The calculator window shows the calculation $261,000$.

Observăm că subprogramul cu *SELECT 1 și excepții* a durat **261.000** unități (ordinul milisecundelor), **NU INTRĂ PE BLOCUL DE EXCEPȚII!** Deci s-a executat cu **2.395.000** de unități mai repede decât subprogramul cu *COUNT(*)*.



Testăm al treilea caz când salariul minim este 0 și maxim 0, care conține 0 înregistrări pentru *COUNT(*)*.

Oracle SQL Developer

Connections

Oracle Connections

Tables (Filtered)

Views

Indexes

Packages

Procedures

Functions

Operators

Queues

Queues Tables

Triggers

Types

Sequences

Materialized Views

Materialized View Logs

Synonyms

Public Synonyms

Database Links

Database Links

Deviations

Java

XML Schemas

XML DB Repository

Scheduler

Recycle Bin

Other Users

Reports

All Reports

Analytics View Reports

Data Dictionary Reports

Data Modeler Reports

CLAP Reports

TimeTen Reports

User Defined Reports

Worksheet - Query Builder

0.037 seconds

```

19 RETURN FALSE;
20 END IF;
21 END subprogram_count;
22 /
23
24 DECLARE
25     ok BOOLEAN;
26 BEGIN
27     DBMS_OUTPUT.PUT_LINE('Inainte de subprogram cu COUNT(*)');
28     DBMS_OUTPUT.PUT_LINE(sysdate);
29     ok := subprogram_count(0, 0);
30     DBMS_OUTPUT.PUT_LINE('Dupa subprogram cu COUNT(*)');
31     DBMS_OUTPUT.PUT_LINE(sysdate);
32     DBMS_OUTPUT.PUT_LINE('');
33     IF ok = TRUE THEN
34         DBMS_OUTPUT.PUT_LINE('Rezultat: TRUE !');
35     ELSE
36         DBMS_OUTPUT.PUT_LINE('Rezultat: FALSE !');
37     END IF;
38 END;
39
40
41 CREATE OR REPLACE FUNCTION subprogram_select(v_min_salary employees.salary%TYPE, v_max
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

PL/SQL procedure successfully completed.

Messages Logging Page Statements Compiler

Click on an identifier with the Control key down to perform 'Go to Declaration'

Items Output

Buffer Size: 20000

Inainte de subprogram cu COUNT(*):
11-JAN-22 09.43.56.02513000 PM +02:00
Dupa subprogram cu COUNT(*):
11-JAN-22 09.43.56.027679000 PM +02:00
Rezultat: FALSE !

Calculator

Standard

2,466,000

27679000 - 25213000 = 2466000

History Memory

END

Observăm că a durat **2.466.000** unități (ordinul milisecundelor), subprogramul cu *COUNT(*)*. Acum vom face calculul pentru subprogramul cu *SELECT 1* și excepții.

Testăm al treilea caz când salariul minim este 0 și maxim 0, care conține 0 înregistrări pentru *SELECT 1* și excepții.

Oracle SQL Developer

Connections

Oracle Connections

Tables (Filtered)

Views

Indexes

Packages

Procedures

Functions

Operators

Queues

Queues Tables

Triggers

Types

Sequences

Materialized Views

Materialized View Logs

Synonyms

Public Synonyms

Database Links

Database Links

Deviations

Java

XML Schemas

XML DB Repository

Scheduler

Recycle Bin

Other Users

Reports

All Reports

Analytics View Reports

Data Dictionary Reports

Data Modeler Reports

CLAP Reports

TimeTen Reports

User Defined Reports

Worksheet - Query Builder

0.033 seconds

```

56 EXCEPTION
57 WHEN NO_DATA_FOUND THEN
58     RETURN FALSE;
59 WHEN TOO_MANY_ROWS THEN
60     RETURN TRUE;
61 END subprogram_select;
62 /
63
64 DECLARE
65     ok BOOLEAN;
66 BEGIN
67     DBMS_OUTPUT.PUT_LINE('Inainte de subprogram cu SELECT 1:');
68     DBMS_OUTPUT.PUT_LINE(sysdate);
69     ok := subprogram_select(0, 0);
70     DBMS_OUTPUT.PUT_LINE('Dupa subprogram cu SELECT 1:');
71     DBMS_OUTPUT.PUT_LINE(sysdate);
72     IF ok = TRUE THEN
73         DBMS_OUTPUT.PUT_LINE('Rezultat: TRUE !');
74     ELSE
75         DBMS_OUTPUT.PUT_LINE('Rezultat: FALSE !');
76     END IF;
77 END;
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

PL/SQL procedure successfully completed.

Messages Logging Page Statements Compiler

Click on an identifier with the Control key down to perform 'Go to Declaration'

Items Output

Buffer Size: 20000

Inainte de subprogram cu COUNT(*):
11-JAN-22 09.43.56.02513000 PM +02:00
Dupa subprogram cu COUNT(*):
11-JAN-22 09.43.56.027679000 PM +02:00
Rezultat: FALSE !

Inainte de subprogram cu SELECT 1:
11-JAN-22 09.48.09.140229000 PM +02:00
Dupa subprogram cu SELECT 1:
11-JAN-22 09.48.09.140474000 PM +02:00
Rezultat: FALSE !

Calculator

Standard

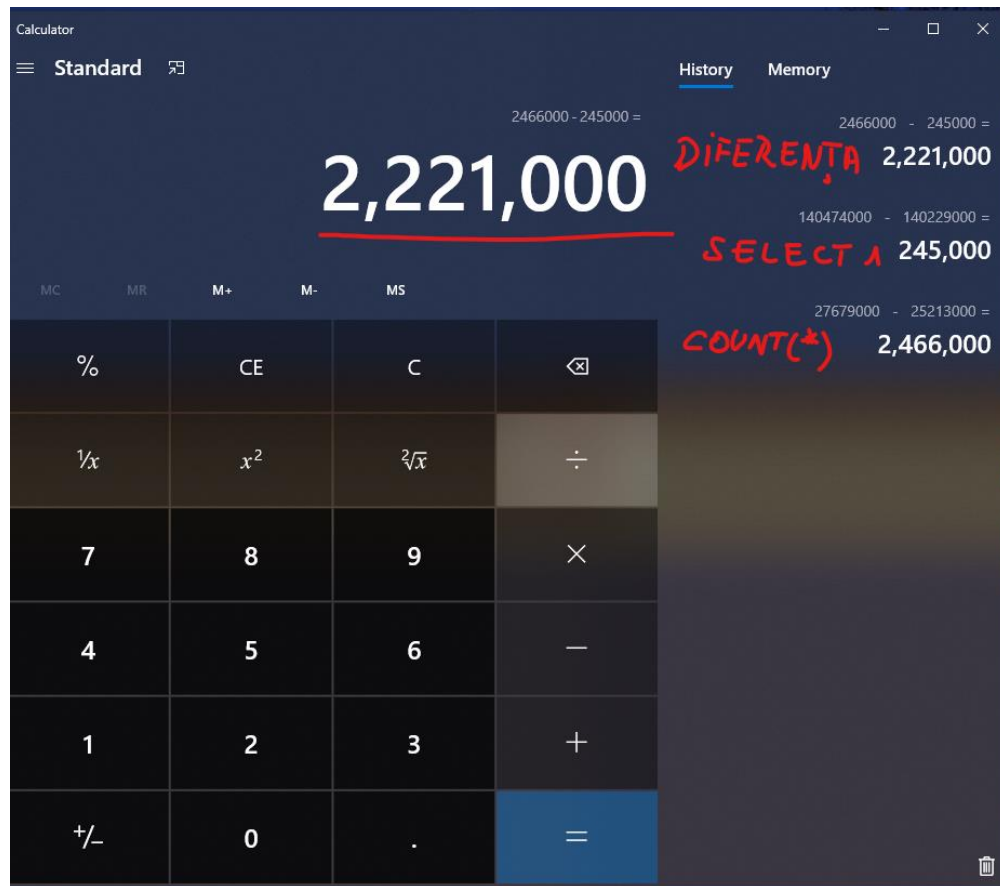
245,000

140474000 - 140229000 = 245000

History Memory

END

Observăm că subprogramul cu *SELECT 1 și excepții* a durat **245.000** unități (ordinul milisecundelor), **INTRĂ PE EXCEPȚIA NO_DATA_FOUND!** Deci s-a executat cu **2.221.000** de unități mai repede decât subprogramul cu *COUNT(*)*.



Concluzie:

- *SELECT 1* folosind excepții este mai rapid, indiferent de caz (nu intră pe excepții, intră pe *NO_DATA_FOUND/TOO_MANY_ROWS*) decât *COUNT(*)* din punct de vedere al timpului.

2.2 Trigger LMD care să fie declanșat de acțiuni pe un tabel imbricat (nested table).

Vom crea un tabel imbricat pe care îl vom folosi pentru a stoca prenumele meu, un tabel_robertto în care vom stoca id-ul prenumelui și lista de prenume pentru acel id.

Rezolvare:

```
CREATE OR REPLACE TYPE tab_imb IS
```

```
TABLE OF VARCHAR2(30);
```

```
/
```



```
CREATE TABLE tabel_robertto (
```

```
    id      NUMBER(10),
```

```
    coloana_imb  tab_imb
```

```
)
```

```
NESTED TABLE coloana_imb STORE AS coloana_imb_tab;
```

```
INSERT INTO tabel_robertto VALUES (1, tab_imb('Karloss'));
```

```
CREATE OR REPLACE TRIGGER trigger_tab_imb BEFORE
```

```
    UPDATE ON tabel_robertto
```

```
    FOR EACH ROW
```

```
BEGIN
```

```
    IF updating THEN
```

```
        IF :new.coloana_imb.count <> :old.coloana_imb.count THEN
```

```
            dbms_output.put_line('S-a schimbat numarul de elemente!');
```

```
        ELSE
```

```
            FOR i IN 1..:new.coloana_imb.count LOOP
```

```
                IF :new.coloana_imb(i) <> :old.coloana_imb(i) THEN
```

```
                    dbms_output.put_line('Am actualizat elementul: ' || i);
```

```
                END IF;
```

```
            END LOOP;
```

```
        END IF;
```

```
    END IF;
```

```
END trigger_tab_imb;
```

```
/
```

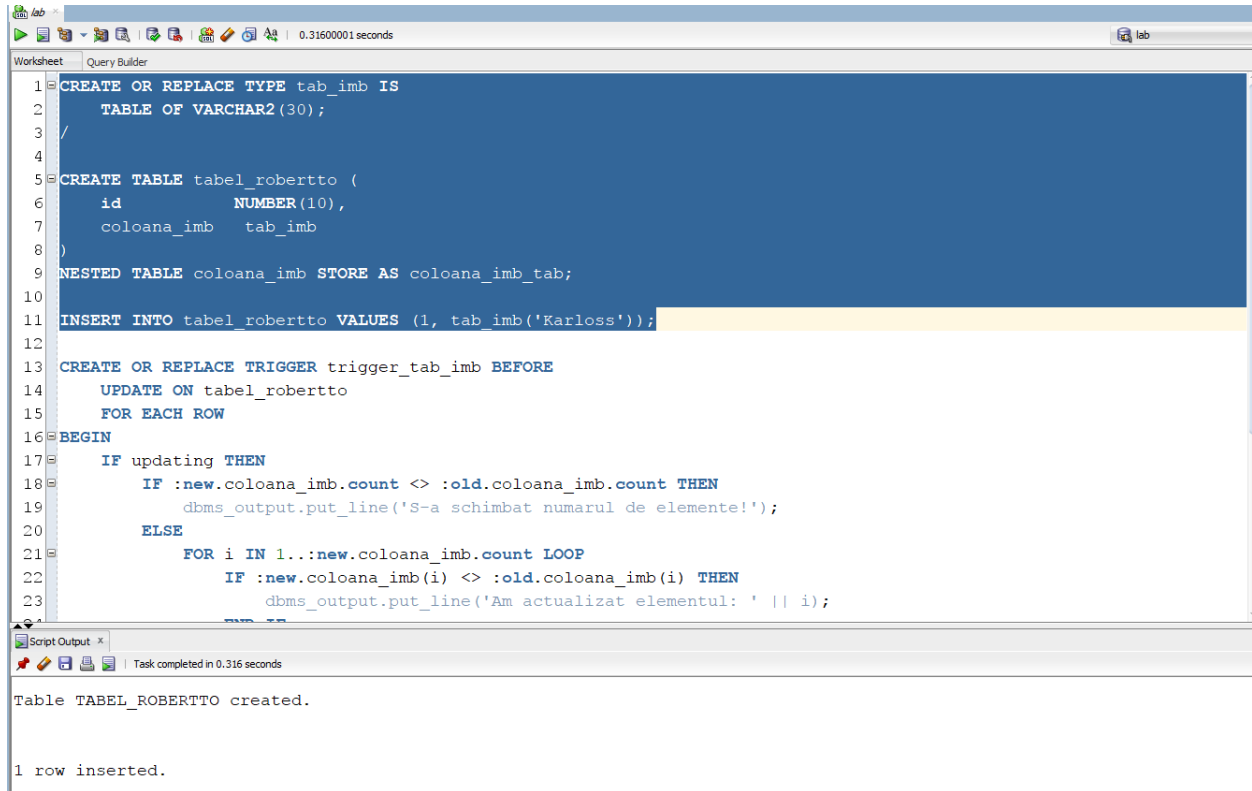
```
UPDATE tabel_robertto
```

```
SET id = 1, coloana_imb = tab_imb('Paullo');
```

UPDATE tabel_robertto

SET id = 1, coloana_imb = tab_imb('Paullo', 'Robertto');

Print-Screen:



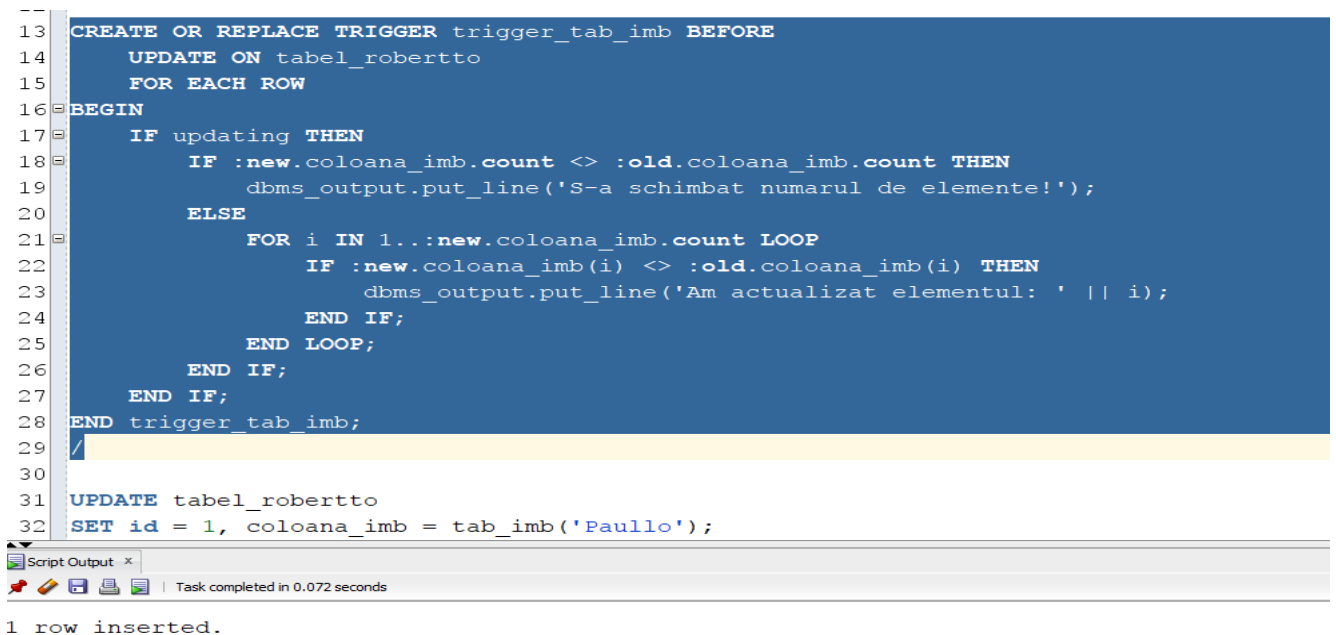
The screenshot shows the SQL Developer interface with a script editor containing the following SQL code:

```
1 CREATE OR REPLACE TYPE tab_imb IS
2   TABLE OF VARCHAR2(30);
3 /
4
5 CREATE TABLE tabel_robertto (
6   id          NUMBER(10),
7   coloana_imb tab_imb
8 )
9 NESTED TABLE coloana_imb STORE AS coloana_imb_tab;
10
11 INSERT INTO tabel_robertto VALUES (1, tab_imb('Karloss'));
12
13 CREATE OR REPLACE TRIGGER trigger_tab_imb BEFORE
14   UPDATE ON tabel_robertto
15   FOR EACH ROW
16 BEGIN
17   IF updating THEN
18     IF :new.coloana_imb.count <> :old.coloana_imb.count THEN
19       dbms_output.put_line('S-a schimbat numarul de elemente!');
20     ELSE
21       FOR i IN 1..:new.coloana_imb.count LOOP
22         IF :new.coloana_imb(i) <> :old.coloana_imb(i) THEN
23           dbms_output.put_line('Am actualizat elementul: ' || i);
24         END IF;
25       END LOOP;
26     END IF;
27   END IF;
28 END trigger_tab_imb;
29 /
```

The script output pane shows the following messages:

```
Table TABEL_ROBERTTO created.

1 row inserted.
```



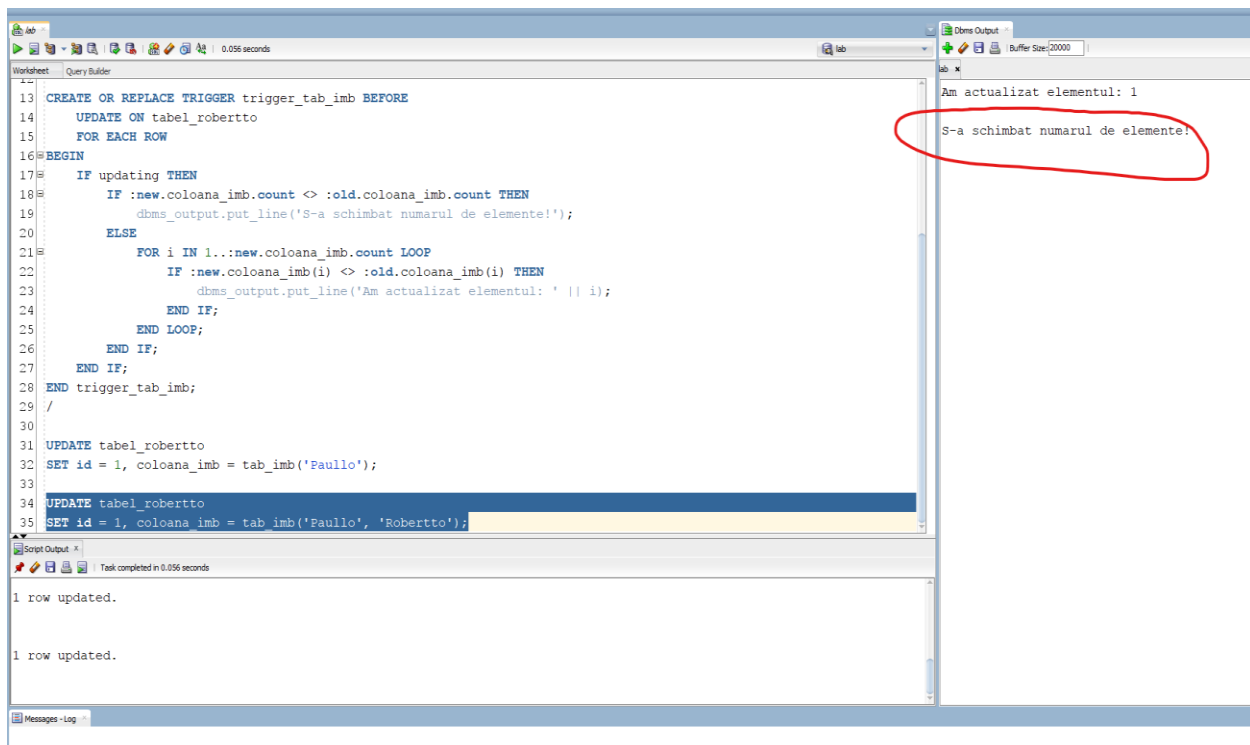
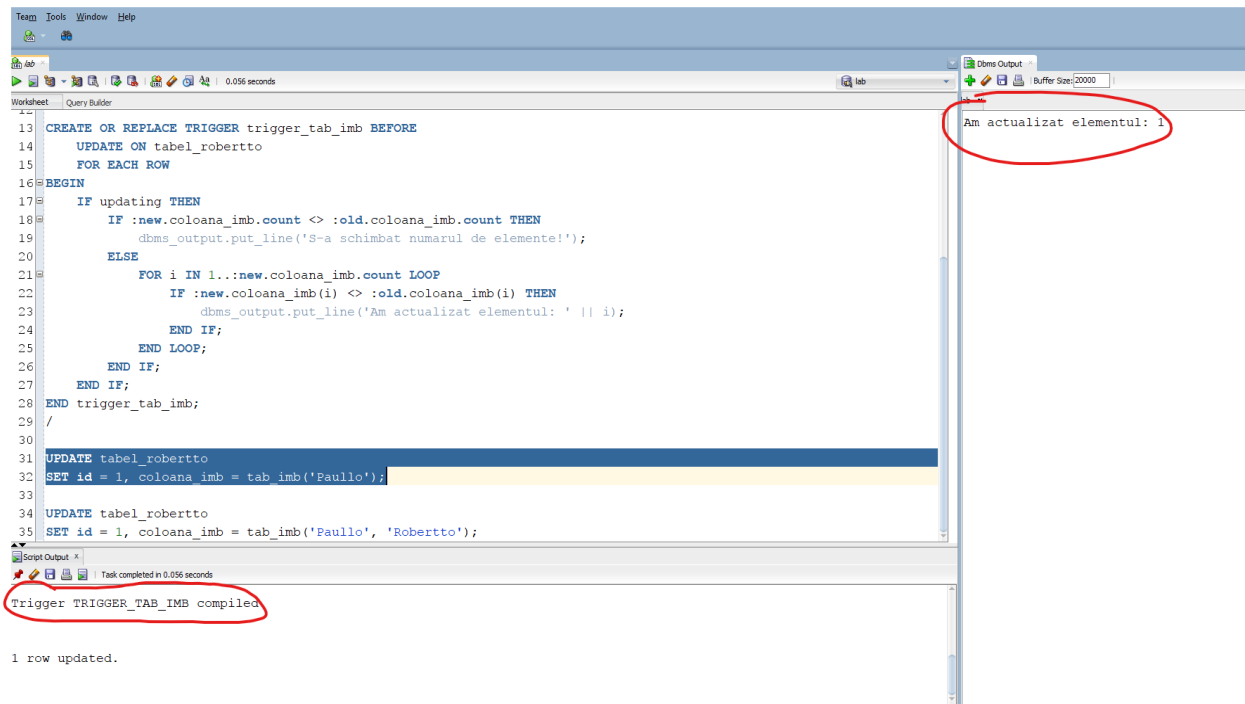
The screenshot shows the SQL Developer interface with a script editor containing the following SQL code:

```
13 CREATE OR REPLACE TRIGGER trigger_tab_imb BEFORE
14   UPDATE ON tabel_robertto
15   FOR EACH ROW
16 BEGIN
17   IF updating THEN
18     IF :new.coloana_imb.count <> :old.coloana_imb.count THEN
19       dbms_output.put_line('S-a schimbat numarul de elemente!');
20     ELSE
21       FOR i IN 1..:new.coloana_imb.count LOOP
22         IF :new.coloana_imb(i) <> :old.coloana_imb(i) THEN
23           dbms_output.put_line('Am actualizat elementul: ' || i);
24         END IF;
25       END LOOP;
26     END IF;
27   END IF;
28 END trigger_tab_imb;
29 /
30
31 UPDATE tabel_robertto
32 SET id = 1, coloana_imb = tab_imb('Paullo');
```

The script output pane shows the following messages:

```
1 row inserted.
```

Trigger TRIGGER_TAB_IMB compiled



Inițial avem un singur prenume pentru id-ul 1. Când am actualizat coloana_imb cu un singur prenume, Karlross a devenit Paullo, dar când am încercat să adăugăm încă un prenume pe lângă Paullo pentru id-ul 1, observăm că s-a modificat numărul de elemente din tabelul nostru imbricat!

3. Curs 10

3.1 Trigger compound.

Ordinea de execuție a triggerului va fi: Before statement, before each row, after each row, after statement.

Rezolvare:

```
CREATE OR REPLACE TRIGGER trig_produs_in_meniu FOR
  INSERT OR UPDATE OF id_produs ON continut_comanda
COMPOUND TRIGGER
  v_id_restaurant meniu.id_restaurant%TYPE;
  BEFORE STATEMENT IS BEGIN
    dbms_output.put_line('Verificare daca produs exista in meniu');
  END BEFORE STATEMENT;
  BEFORE EACH ROW IS BEGIN
    SELECT
      id_restaurant
    INTO v_id_restaurant
    FROM
      meniu m
    WHERE
      m.id_produs = :new.id_produs;

    dbms_output.put_line('Aces produs se afla doar in meniul restaurantului ' || v_id_restaurant);
  EXCEPTION
    WHEN no_data_found THEN
      raise_application_error(-20065, 'Acest produs nu exista in niciun meniu');
    WHEN too_many_rows THEN
      dbms_output.put_line('Acest produs se afla in meniurile mai multor restaurante');
  END BEFORE EACH ROW;
  AFTER EACH ROW IS BEGIN
```

```
        dbms_output.put_line('Produsul cu id '
                               || :new.id_produs
                               || ' in comanda');
    END AFTER EACH ROW;
AFTER STATEMENT IS BEGIN
    dbms_output.put_line('Verificare completa. Pofta buna!');
END AFTER STATEMENT;
END trig_produs_in_meniu;
/
```

```
insert into continut_comanda (id_produs, id_comanda, numar_produce)
values (6, 4, 2); --nu se poate insera un produs care nu apare in niciun meniu
```

```
delete from continut_comanda
where id_produs = 6 and id_comanda = 4;
```

```
insert into continut_comanda (id_produs, id_comanda, numar_produce)
values (1, 6, 2); --in meniul carui restaurant apare produsul (doar in restaurant 4)
```

```
delete from continut_comanda
where id_produs = 1 and id_comanda = 6;
```

```
insert into continut_comanda (id_produs, id_comanda, numar_produce)
values (4, 3, 2); --produs care se afla in mai multe meniuri
```

```
delete from continut_comanda
where id_produs = 4 and id_comanda = 3;
```

Print-Screen:

The screenshot shows the SQL Developer interface with a query window titled "PROJECT BD ROBERTTO". The query is a PL/SQL trigger definition:

```
1 CREATE OR REPLACE TRIGGER trig_produs_in_meniu FOR
2   INSERT OR UPDATE OF id_produs ON continut_comanda
3 COMPOUND TRIGGER
4   v_id_restaurant meniu.id_restaurant%TYPE;
5   BEFORE STATEMENT IS BEGIN
6     dbms_output.put_line('Verificare daca produs exista in meniu');
7   END BEFORE STATEMENT;
8   BEFORE EACH ROW IS BEGIN
9     SELECT
10      id_restaurant
11     INTO v_id_restaurant
12     FROM
13      meniu m
14     WHERE
15      m.id_produs = :new.id_produs;
16
17     dbms_output.put_line('Aces produs se afla doar in meniul restaurantului ' || v_id_restaurant);
18 EXCEPTION
19   WHEN no_data_found THEN
20     raise_application_error(-20065, 'Acest produs nu exista in niciun meniu');
21   WHEN too_many_rows THEN
22     dbms_output.put_line('Acest produs se afla in meniurile mai multor restaurante');
23   END BEFORE EACH ROW;
24 AFTER EACH ROW IS BEGIN
```

Below the query window, the "Script Output" window shows the message: "Trigger TRIG_PRODUS_IN_MENIU compiled".

The screenshot shows the SQL Developer interface with a query window titled "PROJECT BD ROBERTTO". The query is a PL/SQL script:

```
25 dbms_output.put_line('Produsul cu id '
26   || :new.id_produs
27   || ' in comanda');
28 END AFTER EACH ROW;
29 AFTER STATEMENT IS BEGIN
30   dbms_output.put_line('Verificare completa. Pofta buna!');
31 END AFTER STATEMENT;
32 END trig_produs_in_meniu;
33 /
34
35 insert into continut_comanda (id_produs, id_comanda, numar_produce)
36 values (6, 4, 2); --nu se poate insera un produs care nu apare in niciun meniu
37
38 delete from continut_comanda
39 where id_produs = 6 and id_comanda = 4;
40
41 insert into continut_comanda (id_produs, id_comanda, numar_produce)
```

Below the query window, the "Script Output" window shows the message: "Trigger TRIG_PRODUS_IN_MENIU compiled".

The "Dbms Output" window shows the message: "Verificare daca produs exista in meniu".

The "Error Report" window shows the following error:

```
Error starting at line : 35 in command -
insert into continut_comanda (id_produs, id_comanda, numar_produce)
values (6, 4, 2)
Error report -
ORA-20065: Acest produs nu exista in niciun meniu
ORA-06512: at "ROBERTTO.TRIG_PRODUS_IN_MENIU", line 18
ORA-04088: error during execution of trigger 'ROBERTTO.TRIG_PRODUS_IN_MENIU'
```

PROJECT BD ROBERTTO 0.077 seconds

Worksheet Query Builder

```
31 END AFTER STATEMENT;
32 END trig_produs_in_meniu;
33 /
34
35 insert into continut_comanda (id_produs, id_comanda, numar_produce)
36 values (6, 4, 2); --nu se poate insera un produs care nu apare in niciun meniu
37
38 delete from continut_comanda
39 where id_produs = 6 and id_comanda = 4;
40
41 insert into continut_comanda (id_produs, id_comanda, numar_produce)
42 values (1, 6, 2); --in meniul carui restaurant apare produsul (doar in restaurant 4)
43
44 delete from continut_comanda
45 where id_produs = 1 and id_comanda = 6;
46
47 insert into continut_comanda (id_produs, id_comanda, numar_produce)
```

Script Output X Task completed in 0.077 seconds

Error starting at line : 35 in command -

```
insert into continut_comanda (id_produs, id_comanda, numar_produce)
values (6, 4, 2)
Error report -
ORA-20065: Acest produs nu exista in niciun meniu
ORA-06512: at "ROBERTTO.TRIG_PRODUS_IN_MENIU", line 18
ORA-04088: error during execution of trigger 'ROBERTTO.TRIG_PRODUS_IN_MENIU'
```

1 row inserted.

PROJECT BD ROBERTTO x Buffer Size:20000

Verificare daca produs exista in meniu

Verificare daca produs exista in meniu

Aces produs se afla doar in meniul restaurantului 4

Produsul cu id 1 in comanda

Verificare completa. Pofta buna!

PROJECT BD ROBERTTO 0.066 seconds

Worksheet Query Builder

```
35 insert into continut_comanda (id_produs, id_comanda, numar_produce)
36 values (6, 4, 2); --nu se poate insera un produs care nu apare in niciun meniu
37
38 delete from continut_comanda
39 where id_produs = 6 and id_comanda = 4;
40
41 insert into continut_comanda (id_produs, id_comanda, numar_produce)
42 values (1, 6, 2); --in meniul carui restaurant apare produsul (doar in restaurant 4)
43
44 delete from continut_comanda
45 where id_produs = 1 and id_comanda = 6;
46
47 insert into continut_comanda (id_produs, id_comanda, numar_produce)
48 values (4, 3, 2); --produs care se afla in mai multe meniuri:
49
50 delete from continut_comanda
51 where id_produs = 4 and id_comanda = 3;
```

Script Output X Task completed in 0.066 seconds

values (6, 4, 2)

Error report -

```
ORA-20065: Acest produs nu exista in niciun meniu
ORA-06512: at "ROBERTTO.TRIG_PRODUS_IN_MENIU", line 18
ORA-04088: error during execution of trigger 'ROBERTTO.TRIG_PRODUS_IN_MENIU'
```

1 row inserted.

PROJECT BD ROBERTTO x Buffer Size:20000

Verificare daca produs exista in meniu

Verificare daca produs exista in meniu

Aces produs se afla doar in meniul restaurantului 4

Produsul cu id 1 in comanda

Verificare completa. Pofta buna!

Verificare daca produs exista in meniu

Acest produs se afla in meniurile mai multor restaurante

Produsul cu id 4 in comanda

Verificare completa. Pofta buna!

3.2 Exemplul 8.6 din curs *SELECT COUNT(*)* vs *SELECT 1* și excepții.

Exemplul 8.6

```
-- coloana este populata cu null sau cu 0?
ALTER TABLE categorii
ADD nr_produce NUMBER DEFAULT 0;

-- coloana este populata cu null sau cu 0?
UPDATE categorii c
SET   nr_produce =
      (SELECT COUNT(*)
       FROM   produse
       WHERE  id_categorie = c.id_categorie);

CREATE OR REPLACE VIEW info_categorii_produce
AS
SELECT p.*, c.denumire AS categ_denumire, nivel,
       id_parinte, nr_produce
FROM   produse p, categorii c
WHERE  p.id_categorie = c.id_categorie;

CREATE OR REPLACE TRIGGER actualizeaza_info
INSTEAD OF INSERT OR DELETE OR UPDATE
ON info_categorii_produce
FOR EACH ROW

DECLARE
  v_nr NUMBER(1);
BEGIN
  IF INSERTING THEN
    SELECT COUNT(*) INTO v_nr
    FROM   categorii
    WHERE  id_categorie = :NEW.id_categorie;

    IF v_nr = 0 THEN
      INSERT INTO categorii
      VALUES (:NEW.id_categorie, :NEW.categ_denumire,
              :NEW.nivel, :NEW.id_parinte, 1);

      INSERT INTO produse
      VALUES (:NEW.id_produc, :NEW.denumire,
              :NEW.descriere, :NEW.stoc_curent,
              :NEW.stoc_impus, :NEW.pret_unitar,
              :NEW.greutate, :NEW.volum, :NEW.tva,
              :NEW.id_zona, :NEW.id_um,
              :NEW.id_categorie, SYSDATE, SYSDATE,
              :NEW.activ);
    ELSE
      INSERT INTO produse
      VALUES (:NEW.id_produc, :NEW.denumire,
              :NEW.descriere, :NEW.stoc_curent,
              :NEW.stoc_impus, :NEW.pret_unitar,
```

După cum am precizat mai sus, o variantă mai optimă, din punct de vedere al timpului este *SELECT 1* INTO variabilă, declanșând excepțiile *NO_DATA_FOUND* când nu există nicio categorie identică cu cea nouă și *TOO_MANY_ROWS* când există mai multe categorii identice. Pe noi nu ne interesează câte sunt, *COUNT(*)* cu adevărat, ci doar dacă este sau nu vreo categorie identică, de aceea este mai optim să folosim excepțiile.

Popescu Paullo Robertto Karlross

Grupa 231

Temă SGBD #9