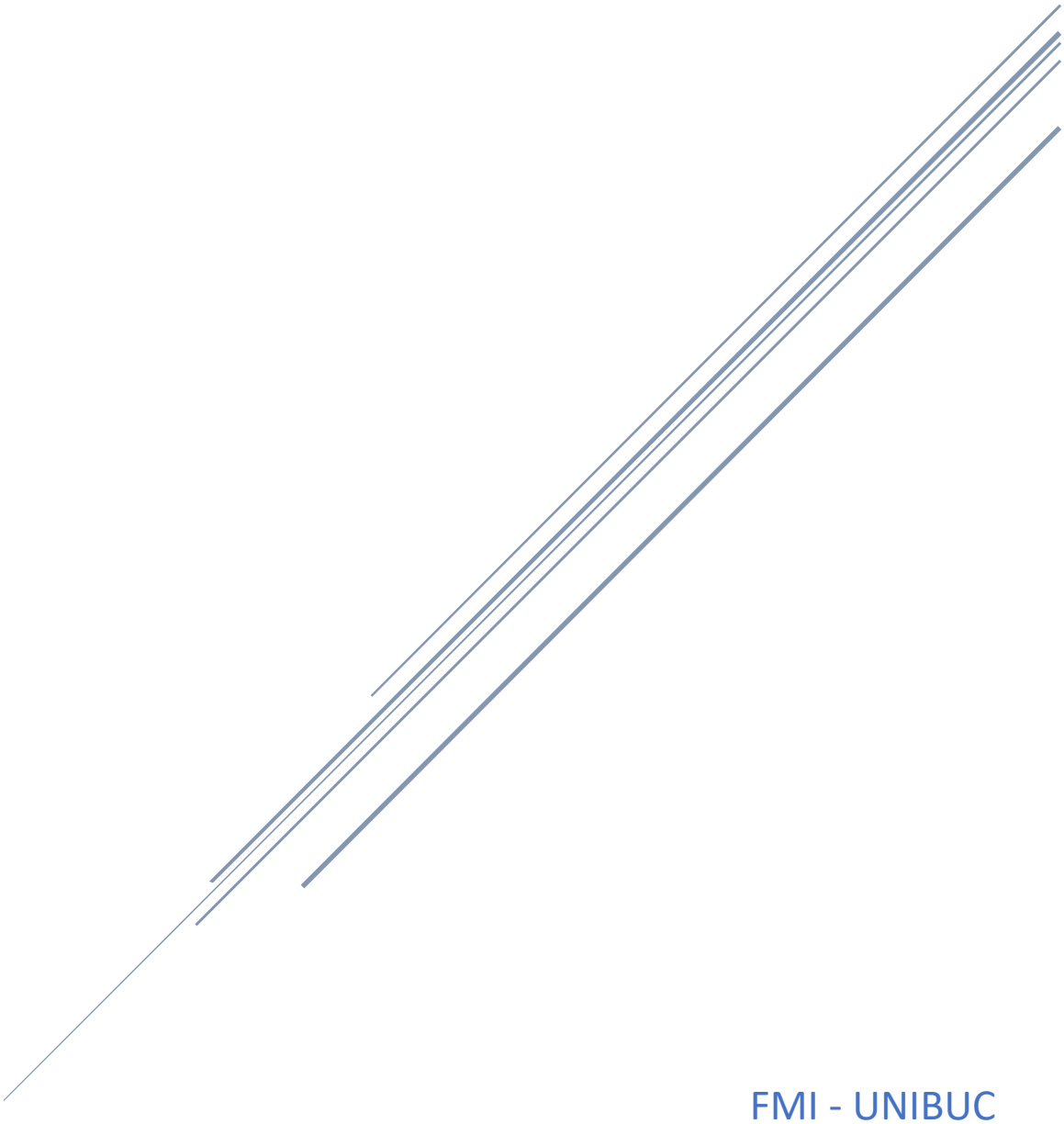
OPTIMIZAREA CODULUI PL\SQL

Iuga Paula – grupa 232



FMI - UNIBUC

SISTEME DE GESTIUNE A BAZELOR DE DATE

Cuprins

Motivația alegerii subiectului 2

Cum se optimizează codul PL\SQL în mod implicit 3

Proprietăți PL\SQL care îmbunătățesc timpul de execuție pt. comenzile SQL 6

FORALL în loc de FOR pentru structuri repetitve de INSERT, DELETE, UPDATE 6

Bulk collect în loc de select into repetitiv. 8

Optimizarea codului PL\SQL 10

A se ține cont de warninguri 10

A se folosi funcțiile din sistem pentru stringuri 12

Motivația alegerii subiectului

Optimizarea codului PL/SQL constituie o parte importantă atunci când scriem cod pentru diverse aplicații. Cu cât acestea sunt mai complexe, cu atât scrierea unui cod optimizat poate aduce schimbări semnificative în ceea ce priveste timpul de execuție și memoria folosită. Întrucât acest subiect nu a fost acoperit în cadrul cursului de Sisteme de gestiune a bazelor de date, consider utilă cercetarea acestuia.

Cele mai importante întrebări la care răspunde lucrarea de față sunt:

- Ce greseli sunt de evitat atunci cand vrem să scriem un cod optim?
- Cum putem eficientiza codul PL/SQL?

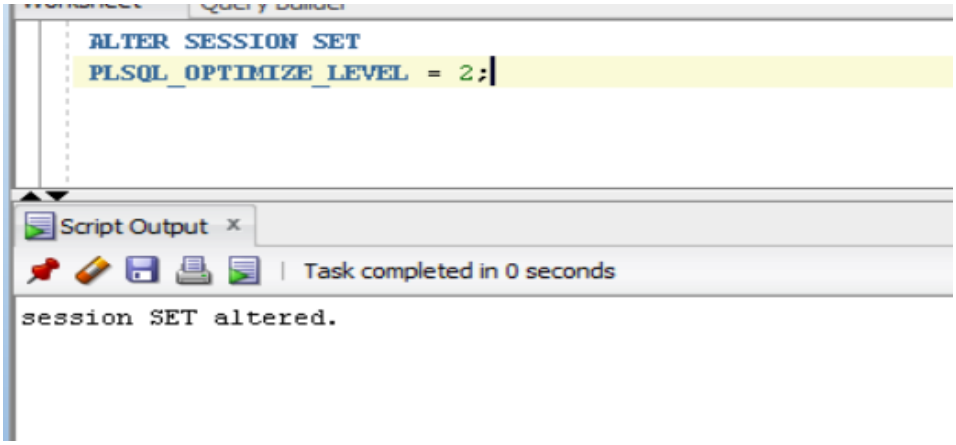
Cum se optimizează codul PL\SQL în mod implicit

Înainte de Oracle 10g, nu se aducea nicio optimizare implicită codului PL\SQL. Acum însă, compilatorul PL\SQL optimizează singur codul în momentul rulării.

De exemplu, codul este optimizat automat folosindu-se ceva ce se numește ***subprogram inlining*** (noțiune cunoscută de la C++ - foloseam inline ca directivă pt. compilator atunci când scriam funcții de tipul getter sau setter). Pe scurt, prin *subprogram inlining* se înțelege că apelul subrogramului este înlocuit de o copie a subprogramului apelat. Deci, atunci când un subprogram este apelat, tot codul subrogramului este inserat în locul unde se apelează acesta. Astfel, se înlătura ceea ce se numește ***function call overhead***. Pentru un subprogram apelat normal, rezultatul funcției este salvat într-o variabilă undeva în memorie iar apoi se redă accesul suprogramului care l-a apelat. Astfel, pentru subprograme mari, apelul direct cauzează probleme de timp dacă timpul de executare al subprogramului apelat este mai mic decât timpul necesar trecerii de la un subprogram de bază la cel apelat.

Cum spuneam mai sus, *subprogram inlining* este făcut automat de către compilator prin intermediul directivei **PLSQL_OPTIMIZE_LEVEL**, care, în mod implicitit este setata la valoarea 3.

Dacă nu dorim ca *subprogram inlining* să fie automat utilizat de compilator putem seta valoarea 2 pt PLSQL_OPTIMIZE_LEVEL.



Astfel, putem alege ca doar câteva subprograme să fie inline, folosind **PRAGMA INLINE.**

```
CREATE OR REPLACE FUNCTION calculeaza_suma(param_maxim NUMBER) RETURN NUMBER
IS
rez NUMBER;
BEGIN
rez := 0;
FOR i IN 2..param_maxim
LOOP
IF i MOD 2 = 0 THEN
rez := rez + i;
END IF;
END LOOP;

RETURN rez;

END calculeaza_suma;
/
```

```
DECLARE

 afisare NUMBER;
l_start  NUMBER;
 BEGIN
 l_start := DBMS_UTILITY.get_time;
 PRAGMA INLINE (calculeaza_suma, 'YES');
 afisare := calculeaza_suma(8) + calculeaza_suma(1000000) + calculeaza_suma(20000) + calculeaza_suma(8) + calculeaza_suma(8);
DBMS_OUTPUT.PUT_LINE('rezultatul este ' || afisare);
 DBMS_OUTPUT.put_line('time taken: ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');

 l_start := DBMS_UTILITY.get_time;
 afisare := calculeaza_suma(8) + calculeaza_suma(1000000) + calculeaza_suma(20000)+ calculeaza_suma(8) + calculeaza_suma(8);
DBMS_OUTPUT.PUT_LINE('rezultatul este ' || afisare);
 DBMS_OUTPUT.put_line('time taken: ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');


 END;
 /
```

```
Script Output x  Query Result x
Task completed in 0.468 seconds
anonymous block completed
rezultatul este 250100510060
time taken: 23 hsecs
rezultatul este 250100510060
time taken: 24 hsecs
```

- *Explicații cod:*

  După ce am eliminat *subprogram inliningul* automat, am creat o functie stocată care calculează suma numerelor până la un număr dat ca parametru. Apoi, în alt bloc am apelat de mai multe ori această funcție inline prin

  PRAGMA INLINE (calculeaza_suma, 'YES');

  Astfel în calculul primului „afisare”, apelul funcției calculeaza_suma devine inline, iar în cel de-al doilea calcul, apelul funcției calculeaza_suma este făcut

normal. Se observă o mică diferență de timp, în favoarea suprogramului inline. Pt. afișarea timpului am folosit funcția get_time din pachetul DBMS_UTILITY care întoarce timpul curent atunci când este apelată.

Același rezultat l-am obținut si creând funcția local într-un subbloc și apeland-o în blocul principal.

```
SET SERVEROUTPUT ON;
DECLARE
afisare NUMBER;
l_start  NUMBER;

FUNCTION calculeaza_suma(param_maxim NUMBER) RETURN NUMBER
IS
rez NUMBER;
BEGIN
rez := 0;
FOR i IN 2..param_maxim
LOOP
IF i MOD 2 = 0 THEN
rez := rez + i;
END IF;
END LOOP;

RETURN rez;

END calculeaza_suma;

BEGIN
 l_start := DBMS_UTILITY.get_time;
PRAGMA INLINE (p1, 'YES');
 afisare := calculeaza_suma(8) + calculeaza_suma(1000000) + calculeaza_suma(20000) + calculeaza_suma(8) + calculeaza_suma(8);
DBMS_OUTPUT.PUT_LINE('rezultatul este ' || afisare);
 DBMS_OUTPUT.put_line('time taken: ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');
```

```
 l_start := DBMS_UTILITY.get_time;
 afisare := calculeaza_suma(8) + calculeaza_suma(1000000) + calculeaza_suma(20000)+ calculeaza_suma(8) + calculeaza_suma(8);
DBMS_OUTPUT.PUT_LINE('rezultatul este ' || afisare);
 DBMS_OUTPUT.put_line('time taken: ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');
 END;
 /
```

```
Script Output × Query Result ×
Task completed in 0.469 seconds
anonymous block completed
rezultatul este 250100510060
time taken: 23 hsecs
rezultatul este 250100510060
```

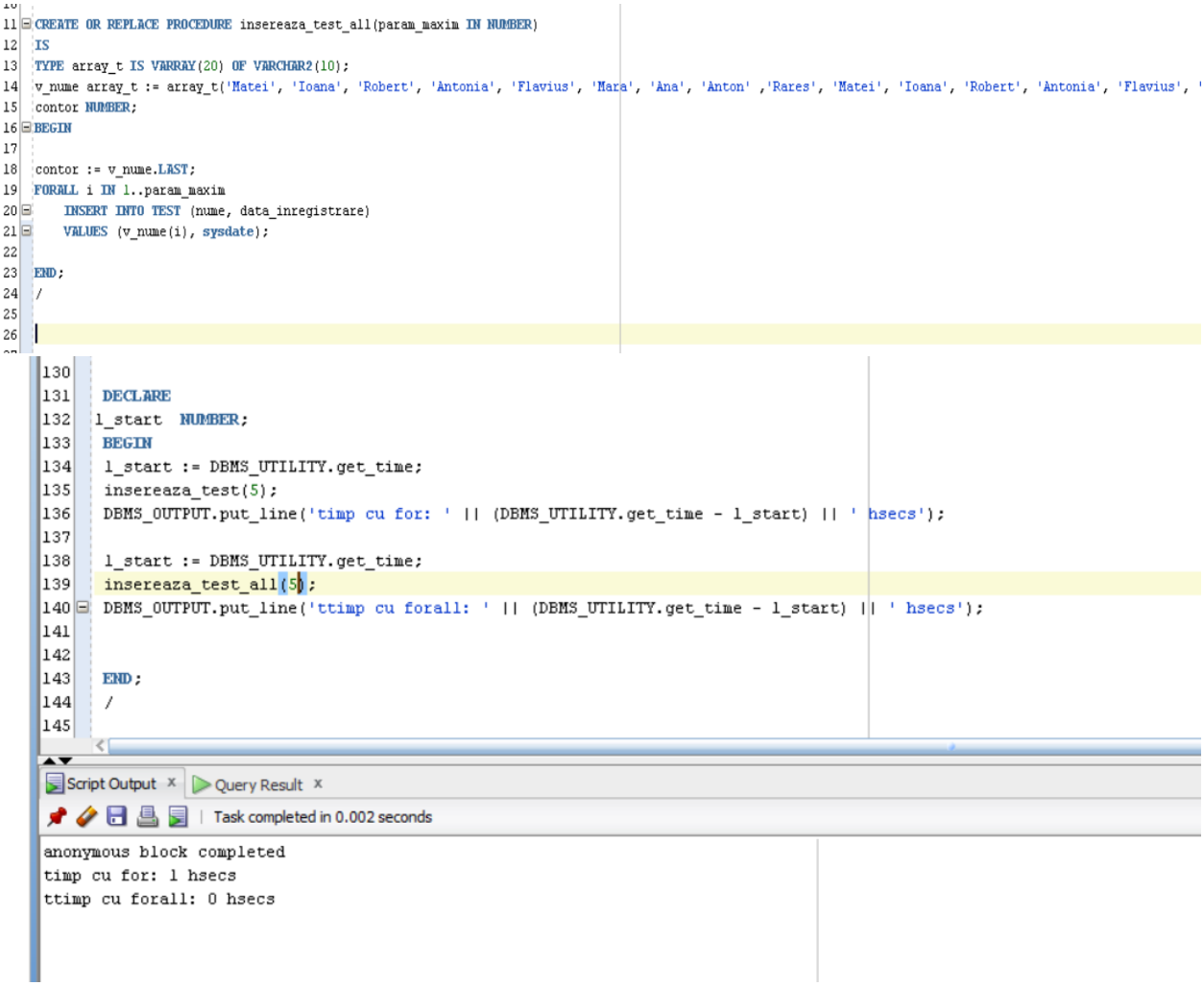
Proprietăți PL\SQL care îmbunătățesc timpul de execuție pt. comenzile SQL

FORALL în loc de FOR pentru structuri repetitve de INSERT, DELETE, UPDATE

For este un loop clasic care parcurge înregistrările una câte una. **Forall** nu este un loop obșinuit ci o doar o notație pentru **tanzacții LMD de tip BULK**, nu parcurge înregistrările una câte una. Este o variantă optimizată și mult mai rapidă decât forul clasic, atunci când este vorba de rulat comenzi LMD de mai multe ori.

```
77
78  CREATE TABLE TEST
79  (
80  Id INTEGER NOT NULL PRIMARY KEY,
81  Nume VARCHAR2(25),
82  data_inregistrare DATE
83  );
84
85  CREATE OR REPLACE TRIGGER pune_id_test
86  BEFORE INSERT ON test
87  FOR EACH ROW
88  BEGIN
89    SELECT id_info.NEXTVAL
90    INTO   :new.ID
91    FROM   dual;
92  END;
93  /
```

```
95  CREATE OR REPLACE PROCEDURE insereaza_test(param_maxim IN NUMBER)
96  IS
97  TYPE array_t IS VARRAY(20) OF VARCHAR2(10);
98  v_nume array_t := array_t('Matei', 'Ioana', 'Robert', 'Antonia', 'Flavius', 'Mara', 'Ana', 'Anton' ,'Rares', 'Matei', 'Ioana', 'Robert', 'Antonia', 'Flavius', 'Ma
99  contor NUMBER;
00  BEGIN
01
02  contor := v_nume.LAST;
03
04  FOR i IN 1..param_maxim LOOP
05     INSERT INTO TEST(nume,data_inregistrare)
06     VALUES (v_nume(i), sysdate);
07    END LOOP;
08  END;
09  /
```



```
CREATE OR REPLACE PROCEDURE insereaza_test_all(param_maxim IN NUMBER)
IS
TYPE array_t IS VARRAY(20) OF VARCHAR2(10);
v_nume array_t := array_t('Matei', 'Ioana', 'Robert', 'Antonia', 'Flavius', 'Mara', 'Ana', 'Anton' ,'Rares', 'Matei', 'Ioana', 'Robert', 'Antonia', 'Flavius', '
contor NUMBER;
BEGIN

contor := v_nume.LAST;
FORALL i IN 1..param_maxim
    INSERT INTO TEST (nume, data_inregistrare)
    VALUES (v_nume(i), sysdate);

END;
/
```

```
DECLARE
l_start  NUMBER;
BEGIN
l_start := DBMS_UTILITY.get_time;
insereaza_test(5);
DBMS_OUTPUT.put_line('timp cu for: ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');

l_start := DBMS_UTILITY.get_time;
insereaza_test_all(5);
DBMS_OUTPUT.put_line('ttimp cu forall: ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');


END;
/
```

Script Output | Query Result

Task completed in 0.002 seconds

```
anonymous block completed
timp cu for: 1 hsecs
ttimp cu forall: 0 hsecs
```

- *Explicații cod:*

Am creat un tabel test și un trigger care se declanșează atunci când dorim să inserăm linii în tabel, pt. a pune id-ul pe care îl ia dintr-o secvență. Se observă că triggreul nu afectează perfomanța, iar varinta cu for e mai infecientă decât cea cu forall.

Notă!

Pe teste mai mari s-ar vedea și mai bine eficiența forall versus for.

Bulk collect în loc de select into repetitiv.

Bulk collect utilizat cu select into introduce toate datele dintr-o dată în colecție. Astfel, este mult mai eficient decât a intoduce datele pe rand cu un for.

```
CREATE table test2
(
id INTEGER NOT NULL PRIMARY KEY,
numar1 NUMBER,
numar2 NUMBER
);

CREATE OR REPLACE PROCEDURE ins_test2
IS
BEGIN
FOR i in 1..50000 LOOP
INSERT INTO test2
VALUES(i, i+10, i+20);
END LOOP;
END;
/
```

```
CREATE OR REPLACE PROCEDURE salveaza_liniil
IS
 TYPE pers
   IS TABLE OF test2%ROWTYPE
      INDEX BY PLS_INTEGER;

  v_pers pers;
  BEGIN
  SELECT *
  BULK COLLECT INTO v_pers
  FROM test2;

  END;
  /


CREATE OR REPLACE PROCEDURE salveaza_linii2
IS
 TYPE pers
   IS TABLE OF test2%ROWTYPE
      INDEX BY PLS_INTEGER;

  l_pers test2%ROWTYPE;
  v_pers pers;
  BEGIN
  FOR i IN(SELECT *
           FROM test2)
  LOOP
  v_pers(v_pers.count) := i;
  END LOOP;
  END;
```

```
195
196   DECLARE
197    l_start  NUMBER;
198  BEGIN
199  ins_test2;
200   l_start := DBMS_UTILITY.get_time;
201 salveaza_liniil;
202  DBMS_OUTPUT.put_line('timp cu bulk collect: ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');
203
204  l_start := DBMS_UTILITY.get_time;
205 salveaza_linii2;
206  DBMS_OUTPUT.put_line('ttimp fara bulck collect ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');
207
208
209   END;
210   /
211
```

Script Output | Query Result

Task completed in 4.728 seconds

```
50,000 rows deleted.
anonymous block completed
timp cu bulk collect: 2 hsecs
ttimp fara bulck collect 20 hsecs
```

- Explicații cod:
  Am creat un tabel nou test2, iar cu ajutorul procedurii ins_test2 am introdus multe date

în tabel. Apoi am apelat procedurile salveaza_linii1 (care folosește bulk collect pt. a salva într-un tabel indexat datele din tabelul test2) și salveaza_linii2(care folosește un ciclu cursor cu subcereri ca mai apoi să insereze datele extrase în tabelul indexat). Se observă o diferența mare de timp în avantajul procedurii cu bulk collect .

Optimizarea codului PL\SQL

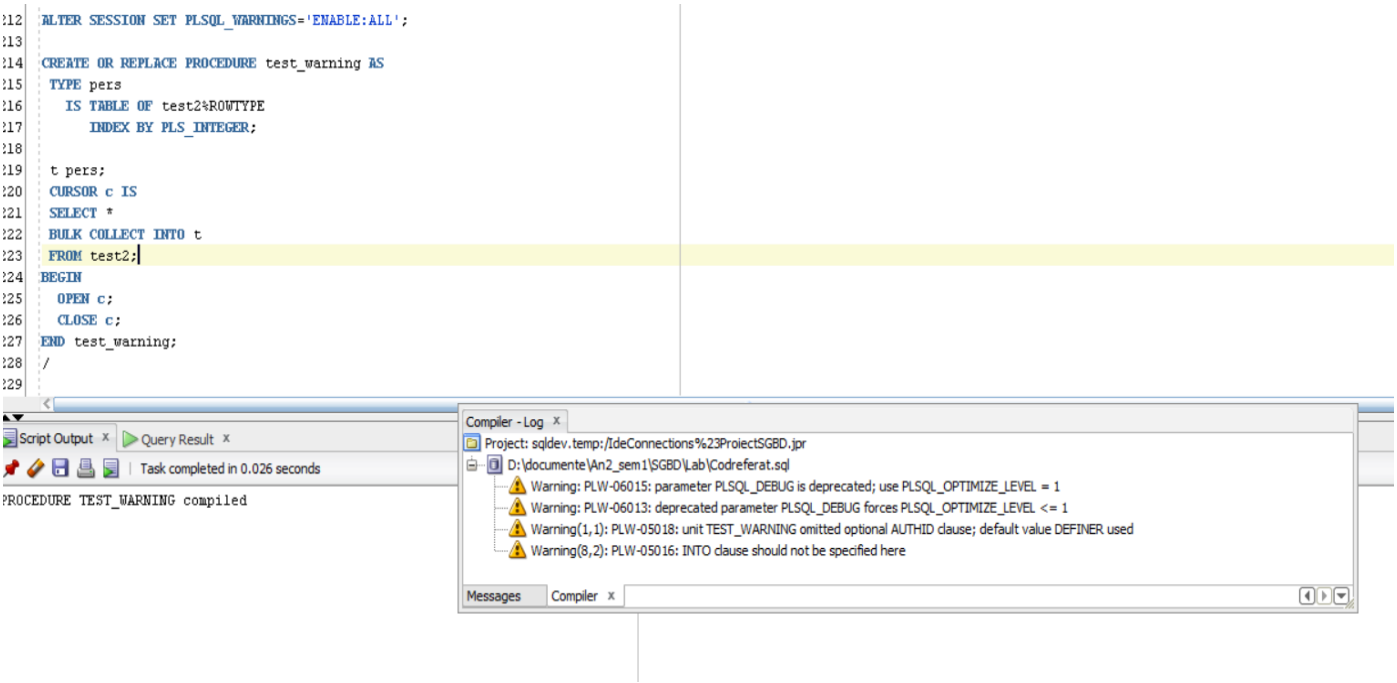
A se ține cont de warninguri

Pentru o mai bună funcționare și optimizare a codului PL\SQL trebuie sa ținem cont de warningurile aruncate de compilator.

Putem porni warningurile prin comanda:

```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:ALL';
```

Chiar dacă rulează, un program incorect poate duce la o prefomanță slabă.



Explicații cod:

În exemplul de mai sus, am folosit un bulk collect înntr-un cursor:

- Warning(8,2): PLW-05016: INTO clause should not be specified here

A se folosi funcțiile din sistem pentru stringuri

Funcțiile pt. stringuri predefinite sunt optimizate și mult mai eficiente decât variantele lor implementate personal de către programator.

```
CREATE OR REPLACE FUNCTION ltrim_string(str VARCHAR2) RETURN VARCHAR2 IS
rez VARCHAR2(400);
v_length NUMBER;
v_out VARCHAR2(20);
ok NUMBER;

BEGIN
rez := '';
v_length := LENGTH(str);
FOR i IN 1..v_length
LOOP
v_out  := substr(str,i,1);
IF  v_out <> ' ' THEN
rez := rez || v_out;

END IF;
END LOOP;
RETURN rez;
END;
/
```

```
SET SERVEROUTPUT ON;
DECLARE
  l_start  NUMBER;
BEGIN
 l_start := DBMS_UTILITY.get_time;
 DBMS_OUTPUT.PUT_LINE('sirul cu ltrim implementat ' || ltrim_string('     ananaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  DBMS_OUTPUT.put_line('timp: ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');

  l_start := DBMS_UTILITY.get_time;
 DBMS_OUTPUT.PUT_LINE('sirul cu ltrim din sistem ' || ltrim('    ananaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  DBMS_OUTPUT.put_line('timp: ' || (DBMS_UTILITY.get_time - l_start) || ' hsecs');

END;
/
```

Script Output × Query Result ×

Task completed in 0.001 seconds

```
anonymous block completed
sirul cu ltrim implementat ananaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
timp: 0 hsecs
sirul cu ltrim din sistem ananaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
timp: 0 hsecs
```