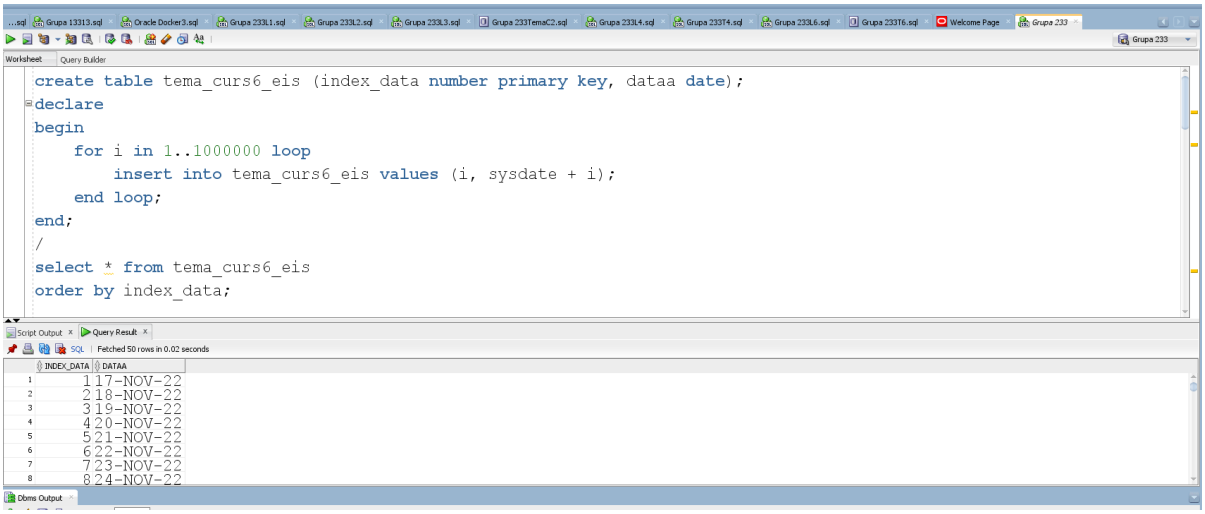
Enescu Irina Stefania
Grupa 233

Tema Curs SGDB
(17 noiembrie 2022)

1. De ce am face FETCH ... BULK COLLECT INTO intr-un cursor si nu am face direct BULK COLLECT intr-o colectie?

Introducem intr-un tabel un numar semnificativ de date.

```
create table tema_curs6_eis (index_data number primary key, dataa date);
declare
begin
   for i in 1..1000000 loop
      insert into tema_curs6_eis values (i, sysdate + i);
   end loop;
end;
/
select * from tema_curs6_eis
order by index_data;
```

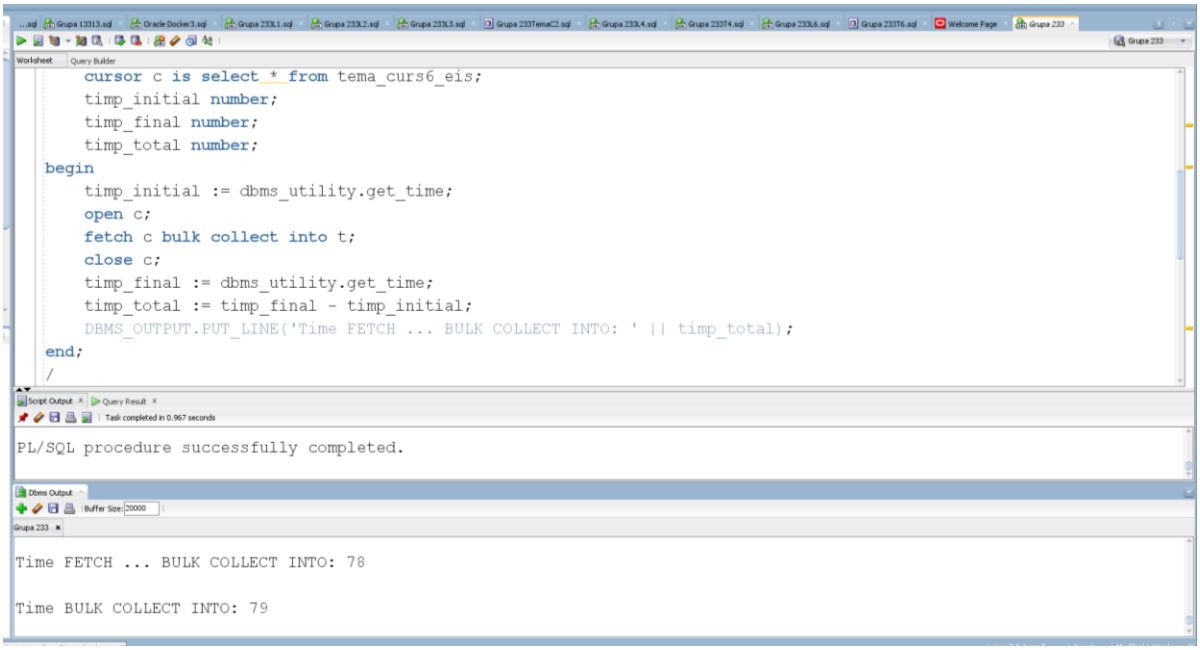


Rulam o comanda cu FETCH ... BULK COLLECT INTO si una cu BULK COLLECT:

```
declare
   type tablou_imbricat is table of tema_curs6_eis%rowtype;
   t tablou_imbricat;
   cursor c is select * from tema_curs6_eis;
   timp_initial number;
   timp_final number;
   timp_total number;
begin
   timp_initial := dbms_utility.get_time;
```

```
   open c;
   fetch c bulk collect into t;
   close c;
   timp_final := dbms_utility.get_time;
   timp_total := timp_final - timp_initial;
   DBMS_OUTPUT.PUT_LINE('Time FETCH ... BULK COLLECT INTO: ' || timp_total);
end;
/

declare
   type tablou_imbricat is table of tema_curs6_eis%rowtype;
   t tablou_imbricat;
   timp_initial number;
   timp_final number;
   timp_total number;
begin
   timp_initial := dbms_utility.get_time;
   select * bulk collect into t from tema_curs6_eis;
   timp_final := dbms_utility.get_time;
   timp_total := timp_final - timp_initial;
   DBMS_OUTPUT.PUT_LINE('Time BULK COLLECT INTO: ' || timp_total);
end;
/
```



Observam ca timpul de rulare din cazul FETCH ... BULK COLLECT INTO este mai mic decat timpul de rulare din cazul BULK COLLECT INTO. Bineinteles, cu cate numarul de date selectate este mai mare, cu atat diferenta dintre cele doua cazuri va fi mai semificativa.

FETCH ... BULK COLLECT INTO	BULK COLLECT INTO
Operatia returneaza in timp scurt mai multe inregistrari sub forma unei multimi. Acest lucru se efectueaza o singura data cu ajutorul cursorului.	Operatia returneaza cate o linie de fiecare data cand apelam instructiunea, fapt ce implica mai mult timp necesar salvarii inregistrarilor.
Operatia ne permite limitarea numarului de inregistrari returnate. Astfel, in cazul in care vrem sa extragem doar 5 inregistrari, o putem face fara a aloca memorie suplimentara salvarii celorlalte.	Operatia nu ne permite limitarea numarului de inregistrari. Astfel, in cazul in care vrem sa extragem doar 5 inregistrari, trebuie sa alocam memorie suplimentara salvarii tuturor.

Prin urmare, facem FETCH ... BULK COLLECT INTO intr-un cursor in locul unui BULK COLLECT intr-o colectie pentru a optimiza codul, salvand astfel memorie si timp de rulare.

2. Verificati de ce nu folosesc colectie si folosesc cursor cu intoarcere date in colectie in exemplul 5.6 din curs (Cursul 5 – Cursoare):

```
DECLARE
   TYPE tab_imb IS TABLE OF categorii%ROWTYPE;
   v_categorii tab_imb;
   CURSOR c IS
      SELECT *
      FROM   categorii
      WHERE  id_parinte IS NULL;
BEGIN
   OPEN c;
   FETCH c BULK COLLECT INTO v_categorii;
   CLOSE c;
   FOR i IN 1..v_categorii.LAST LOOP
      DBMS_OUTPUT.PUT_LINE(v_categorii(i).id_categorie || ' '|| v_categorii(i).denumire);
   END LOOP;
END;
/
```

Raspunsul se gaseste explicat detaliat si demonstrat in exercitiul 1. Practic, exercitiul 2 este un exemplu concret al exercitiului 1.
Facem FETCH ... BULK COLLECT INTO intr-un cursor in locul unui BULK COLLECT intr-o colectie pentru a optimiza codul, salvand astfel memorie si timp de rulare.

3. Este mai bine sa folosim un CURSOR + CURRENT OF sau sa folosim o COLECTIE + FORALL (spre exemplu in contextul unui update)? Utilizati exemplul 5.12 din curs.

```
DECLARE
   CURSOR c IS
      SELECT id_produs
      FROM produse
      WHERE id_categorie IN (SELECT id_categorie
                             FROM  categorii
                             WHERE denumire = 'Placi de retea Wireless')
      FOR UPDATE OF pret_unitar NOWAIT;
```

```
BEGIN
   FOR i IN c LOOP
      UPDATE  produse
      SET pret_unitar = pret_unitar*0.95
      WHERE CURRENT OF c;
   END LOOP;
   COMMIT;
END;
/
```

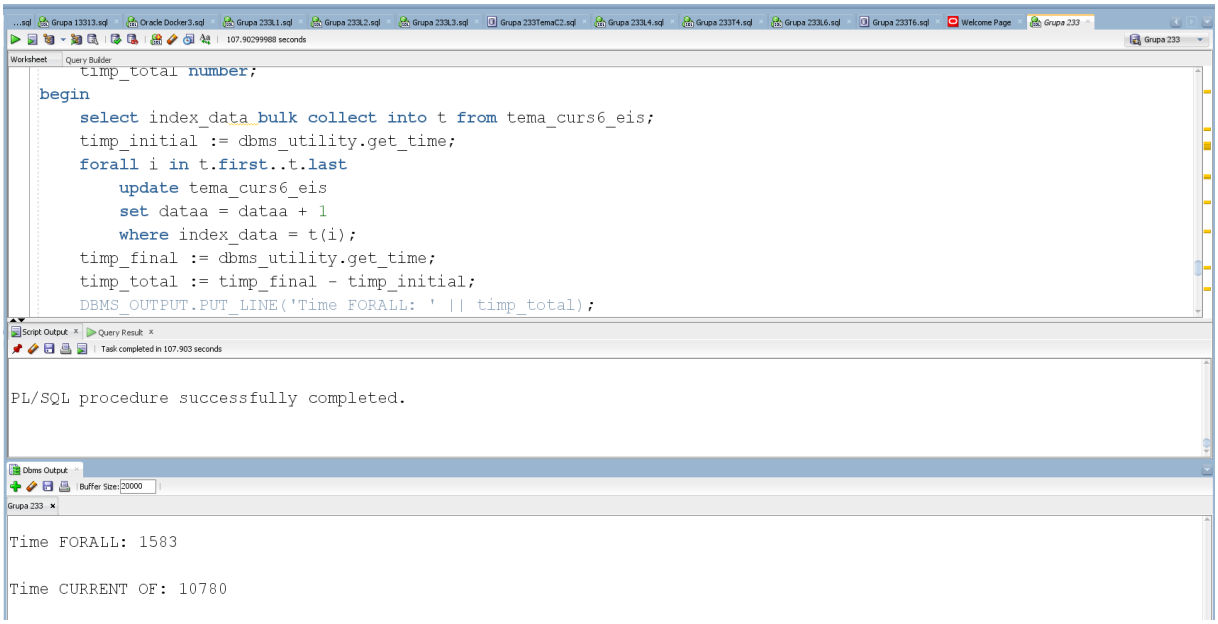
Procedam ca la exercitiul 1, ajutandu-ne de tabelul creat acolo.

Construim un exemplu pentru current of si unul pentru forall:

```
declare
   type tablou_imbricat is table of tema_curs6_eis.index_data%type;
   t tablou_imbricat;
   timp_initial number;
   timp_final number;
   timp_total number;
begin
   select index_data bulk collect into t from tema_curs6_eis;
   timp_initial := dbms_utility.get_time;
   forall i in t.first..t.last
      update tema_curs6_eis
      set dataa = dataa + 1
      where index_data = t(i);
   timp_final := dbms_utility.get_time;
   timp_total := timp_final - timp_initial;
   DBMS_OUTPUT.PUT_LINE('Time FORALL: ' || timp_total);
end;
/

declare
   cursor c is select index_data from tema_curs6_eis for update of dataa;
   timp_initial number;
   timp_final number;
   timp_total number;
begin
   timp_initial := dbms_utility.get_time;
   for i in c loop
      update tema_curs6_eis
      set dataa = dataa + 1
      where current of c;
   end loop;
   timp_final := dbms_utility.get_time;
   timp_total := timp_final - timp_initial;
   DBMS_OUTPUT.PUT_LINE('Time CURRENT OF: ' || timp_total);
```

```
end;
/
```



Observam ca timpul de rulare din cazul FORALL este mai mic decat timpul de rulare din cazul CURRENT OF. Bineinteles, cu cate numarul de date selectate este mai mare, cu atat diferenta dintre cele doua cazuri va fi mai semificativa.

CURSOR + CURRENT OF	COLECTIE + FORALL
Operatia face update pe ultima linie din cursor.	Operatia face update pe fiecare linie dupa index.
Operatia are loc direct pe ultima linie din cursor, nefiind necesara conditia where ce specifica indexul curent.	Operatia are loc pe linia identificata din conditia where ce specifica indexul curent.

Asadar, facem CURRENT OF intr-un cursor in locul unui FORALL intr-o colectie pentru a optimiza codul, salvand astfel timp de rulare.

4. Ce varianta a exemplului 5.14 din curs este optima?

```
DECLARE
   TYPE tip_cursor IS REF CURSOR RETURN produse%ROWTYPE;
   c tip_cursor;
   v_optiune NUMBER(1) := &p_optiune;
   i produse%ROWTYPE;
BEGIN
   IF v_optiune= 1 THEN
      OPEN c FOR
         SELECT * FROM  produse p
         WHERE EXISTS (SELECT 1
                       FROM facturi_produse pf, facturi f
                       WHERE p.id_produs = pf.id_produs
```

```
                        AND pf.id_factura = f.id_factura
                        AND TO_CHAR(data,'q') = 1);
  ELSIF v_optiune = 2 THEN
    OPEN c FOR
      SELECT * FROM  produse p
          WHERE id_produs IN (SELECT id_produs
                                FROM facturi_produse pf, facturi f
                                WHERE pf.id_factura = f.id_factura
                                        AND TO_CHAR(data,'q') = 2);
  ELSIF v_optiune = 3 THEN
    OPEN c FOR
      SELECT DISTINCT p.* FROM  produse p, facturi_produse pf, facturi f
      WHERE p.id_produs = pf.id_produs
              AND pf.id_factura = f.id_factura
              AND TO_CHAR(data,'q') = 3;
  ELSE
    OPEN c FOR
      SELECT * FROM  produse p
      WHERE id_produs IN (SELECT id_produs
                            FROM facturi_produse);
  END IF;
  LOOP
    FETCH c INTO i;
    EXIT WHEN c%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(i.id_produs||' ' ||i.denumire);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Nr produse vandute: '||c%ROWCOUNT);
  CLOSE c;
END;
/
```

Varianta 1 (EXISTS):
In momentul in care utilizam operatorul EXIST ne oprim la prima valoare de True returnata din subcerere, ignorand restul valorilor si salvand astfel timp de rulare.

Varianta 2 (IN):
In momentul in care utilizam operatorul IN returnam toate valorile din subcererea corespunzatoare, ulterior cautand valoarea pe care o dorim in toate inregistrarile.

Varianta 3 (JOIN):
In momentul in care utilizam JOIN returnam toate valorile care corespund intersectiei tabelelor cu ajutorul conditiilor puse in clauza where.

Concluzie: Varianta implementata cu join este cea mai eficienta deoarece se bazeaza pe conditiile din clauza where, nefiind necesara o subcerere pentru a filtra rezultatele (precum la exists si in). A doua cea mai eficienta varianta este cea implementata cu exists deoarece nu returneaza toate valorile pentru a face selectia, ci doar cat este nevoie (pana gaseste o valoare True). Varianta ineficienta este cea

implementata cu in deoarece are nevoie de o subcerere ce intrerupe cererea principala si returneaza toate valorile pentru a face selectia.